commodore

Commodore Canada's
Tech/News Periodical

The Transactor

VOLUME 3
Issue #4

Bits & Pieces

Suppressed '?'

In the latest Midnight Software Gazette a POKE was published to suppress the question mark that follows an INPUT command prompt. Try this short program:

```
10 POKE 16, 1          (BASIC 2.0: POKE 14, 1)
20 INPUT "DATA ";A$
30 PRINT A$
```

Note that line 20 prompts for 'DATA ' with no "?" following. But when you hit RETURN after typing some characters, line 30 prints this string on the same line. This is a residual affect of the POKE in line 10. You might be able to use this to your advantage but to get a line feed between lines 20 and 30 you'll have to do an extra PRINT. Subsequent INPUT commands will also have the "?" suppressed. Get it back with POKE 16, 0. The program then becomes:

```
10 POKE 16, 1         (BASIC 2.0: POKE 14, 1)
20 INPUT "DATA ";A$
10 POKE 16, 0         (BASIC 2.0: POKE 14, 0)
40 PRINT
50 PRINT A$
```

By the way, the Midnight Software Gazette is available FREE by sending a self-address STAMPED envelope to:

CIPUG
635 Maple
Mt. Zion, Illinois
U.S.A.
62549

The Midnight publishes some great editorials and reviews, the latest news, other info sources, and interesting facts about PET, CBM and now VIC! And for the price, it can't be beat!

The Transactor is produced on the CBM 8032 with WordPro IV Plus and the NEC Spinwriter

Index Transactor #4

Cassette Notes

Jeff Kriss of Toronto has submitted the POKEs for turning the cassette motors on or off for BASIC 4.0 machines. It seems they're not quite the same as before. Now you need an extra POKE to turn them off.

```
Cassette #1:  OFF   POKE 249, 52
                    POKE 59411, 61

              ON    POKE 249, 0

Cassette #2:  OFF   POKE 250, 52
                    POKE 59456, 61

              ON    POKE 250, 0
```

Ernest Blaschke of Toronto has this friendly bit of information:

"When loading a program or reading a data-file from tape, quite often I forget to press the cassette deck STOP button after the tape has stopped moving. This can result in dire consequences when later, in the program, a file is opened for writing on tape, and yet the cassette is still on "PLAY" rather than "PLAY & RECORD". As a safeguard against this happening, I now routinely include a line in my program as follows:

```
10 IF (PEEK (59408) AND 16) = 0 THEN
   PRINT "STOP TAPE" : WAIT 59408, 16
```

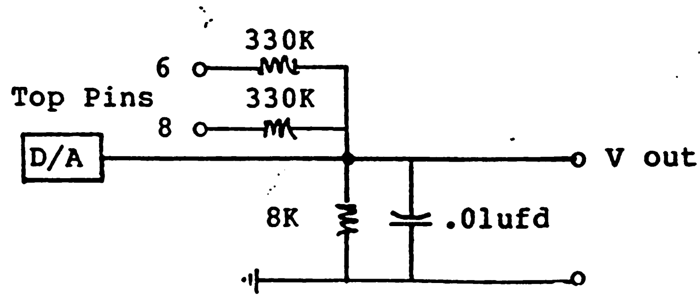
Anyone using two tape drives will need these two lines:

```
10 P9 = PEEK(241) : P8 = 59408
20 IF (PEEK (P8) AND 16 * P9) = 0 THEN
   PRINT "STOP TAPE #";P9 : WAIT P8, 16 * P9
```

This will eliminate any potential problems. Presumably the 59408 location may have changed with the new ROMs ?"

The above is for BASIC 1.0 ROMs. For BASIC 2.0 & 4.0 the 59408 location stays the same. Change the 241 in line 10 to a 212.

The circuit below can be added to the Poor Man's D/A Converter (Volume 2, Issue #11) or simply used by itself. Pins 6 and 8 of the User Port (top pins) are connected to the tape read pins on the cassette ports. Due to numerous main logic board variations, it would be too difficult to say which pin belongs to which cassette. But for the price of two 330K resistors, it would be a shame not to hook up both.



The 8K resitor and the .01 microfd cap. are already in the D/A. If you already have this D/A circuit built, simply add these parts. Now when the cassette is being read, the signal will also be sent to your amplifier... an audio cassette monitor!

Weekday Calculator

This neat little subroutine returns the day of the week for any date given in DAY/MONTH/YEAR format. Of course you could change it around for YEAR/MONTH/DAY... just alter the order of the variables following the INPUT statement. The program does not check for date validity... but that's no problem. Just do some testing for day greater than 31 some months, 30 other months and 28 for February. For leapyears, do an extra test of YEAR/4=INT(YEAR/4) in the case of Feb. 29.

```
100 INPUT "DD, MM, YYYY";D,M,Y
110 K = INT( (60+ (100/M) )/100 )
120 F = 365 * Y + D + 31 * (M-1) - INT(.4*M+2.3) * (1-K)
130 F = F + INT((Y-K/4) - INT(.75*( INT((Y-K) / 100+1)))
140 F = F - INT(F/7) * 7
150 PRINT MID$( "SATSUNMONTUEWEDTHUFRI", F * 3 + 1, 3 )
```

Steve's BBS

Steve Punter of WordPro fame (and fortune I hasten to add), has developed a Bulletin Board System for use with PET/CBMs. Much like WordPro, the system has several great features; User LOG and daily LOG, upload/download capabilities for programs, WordPro files and SEQ files, optional protection on messages and programs, optional password sign-on, formatted messages, bulletin section and much more... plus all the editing functions a SYSOP could ever ask for! Steve runs his own system at 416-624-5431. Operating hours are:

Mon-Fri: 8 PM. - 9 AM.
Weekends: All Day!

Give it a try! (mention how you found out about it) Steve's system runs TV, movie, and restaurant reviews plus numerous provocative discussions and debates by regular columnists. Any ASCII terminal or terminal program can be used, but to up/download programs you'll need Steve's own terminal program which is FREE of charge (see your Commodore dealer).

The Bulletin Board Host System will soon be distributed by Commodore and available from any authorized Commodore dealer. A simple circuit schematic is included to modify the Commodore 8010 Modem for auto-answering capability. Steve even plans to make the system compatable with the DATAPAC network (available early 1982).

Disk User Notes

Henry Troup of Mississauga has this valuable information for BASIC 4.0 programmers with disk units:

"Mixing BASIC 2.0 OPEN to disk and BASIC 4.0 DOPEN commands can be hazardous to your health! The full OPEN command is:

```
OPEN lf, dv, sa, fn
```

```
where:  lf  is the logical file number
        dv  is the device number
        sa  is the secondary address
        fn  is the filename
```

But the BASIC 4.0 command is:

```
DOPEN#lf, fn
```

Notice that only 'lf' and 'fn' are declared by the programmer ('dv' <u>defaults</u> to 8). While there is convenience in allowing the machine to choose the secondary address, there is danger in mixing the two forms. If DOPEN has used a secondary address, there is absolutely nothing to stop you from re-using it in a subsequent OPEN. There never was before either (when DOPEN didn't exist), but at least you could see the secondary addresses selected.

The only mechanism the disk drive has to tell two files apart is the secondary address: if two open files have the same SA, they are considered the same file. This can cause all kinds of havoc with your files.

What's the cure? Don't mix OPEN to disk with DOPEN. Use one or the other, but if you choose the OPEN command for disk I/O (which is still supported by BASIC 4.0), be sure that different secondary addresses are selected for files that will be open simultaneously. If you want to see what DOPEN is doing in terms of secondary addresses, see my article "FILESTATUS" in Transactor #10, Volume 2.

One last note... a string variable to specify the filename in a DOPEN command, the variable must enclosed in round brackets or parenthesis. The same goes for variables used to specify logical file number, drive number, device or unit number, and record length.

```
                    100 DOPEN#8, "SOME FILE", d1, u9

using variables:    100 LF=8 : FN$="SOME FILE" : DR=1 : DV=9
                    110 DOPEN#(LF), (FN$), D(DR), U(DV)
```

SYS 'EM!

Two useful SYS addresses to note:

SYS 64790
SYS 54386

The first does a jump to 'warm start'... kinda like turning the machine off and back on again, but without that nasty power interruption. The second can be extremely handy when you want to send an M.L.M. memory dump to the printer. It seems that breaking to the monitor with SYS 4 cancels any CMD status you may have set up previously.

Extra Linefeeds Anyone ?

In BASIC 2.0, the PRINT# command always wanted to send a Linefeed (CHR$(10)) after the Carriage Return (CHR$(10)). As a lot of us disk users know, this was a pain! But not always... some printers that don't automatically do a line advance require that linefeed character to be sent (eg. LIST to printer). So when Commodore decided to alter this for BASIC 4.0, some careful thinking was necessary. The engineers decided that logical file numbers of 128 or greater would send the LF, while numbers below 128 would not. With PRINT# to the disk, you would usually opt to suppress LFs, while you could OPEN128,4 to do double spacing, or follow that with CMD128 to LIST to a printer without a hardware line advance.

A Most Harmless Bug!

Jim Butterfield (who else?!) wins the award for discovering the most insignifigant DOS bug, although he'll get absolutely nothing for it! He found that after using APPEND# to add a small bit of data to a very small SEQ file, that the block count was unjustifiably increased from 1 block to 2. This wasn't possible since the total amount of data was less than 60 bytes, which is nowhere near the 254 byte capacity of a block. The answer? A bug. It seems that DOS just assumes that the result of an append will increase any file size by at least 1 block. But the 'blocks free' count didn't change, indicating that the disk hadn't really used another block but just incremented the block count that's stored in the directory along with the filename.

APPEND#ing large amounts of data won't cause this problem. Evidently it only happens when the results of the 'append' do NOT warrant the use of an extra block. When extra blocks are required for the appended data, the DOS correctly increments the block count before updating the directory.

The same bug may surface after a CONCAT of two files, depending (of course) on the size of the file being concatenated (ie. the file that is added, NOT the file that is added to). Apparently the DOS uses the same routines to perform this operation.

The solution? There isn't one.. nor is one necessary! Even a COLLECT won't restore proper block count, BUT, this bug will cause absolutely no damage or ill side effects on your diskette! Thanks again Jim.

CONCATenating Programs

The preceding item brings to mind another question frequently posed to Commodore. "Why will the CONCAT command concatenate two SEQ files but fail to work on two programs ?" The answer ?: CONCAT will not join two program files because it can't merge two programs. What if there were a line in each file that has the same line number ? The disk was not designed to deal with this type of situation.

But you say, "I could make sure that all line numbers in the file to be concatenated are higher than the line numbers in the first file". Well... that's not really the problem. All BASIC program files (PET/CBM) end with three binary zeros. This is so the LIST command knows when to stop listing. GOTO and GOSUB also look for these zeros when searching for a line. If the line is not found before encountering '00 00 00', an ?UNDEF'D STATEMENT ERROR occurs. If you could concatenate two program files, the three zeros that belong to the first programwould reside in memory ahead of code that was concatenated. LIST, GOTO and GOSUB would never look past this point.

For those doing a lot of program merging, it might be best to consider one of many 'toolkit' or programmers aid ROMs that include this feature.

Sound Off!

No this is not the complaints department, but rather a neat trick out of St. Catherines Ontario. Have you ever been playing a game with sound, and then STOPped the game while the sound is activated ? The scenario is usually a frantic programmer looking through memory maps or trying to remember that POKE to turn it off. Before you turn the power off, try this (12" screens only): use CRSR right until you get to that point on the screen that rings the bell. After the jingle, CB2 sound will be de-activated.

Collect

One disk command that doesn't get nearly as much attention as it should is COLLECT. BASIC 2 users will know this as the disk Verify or Validate command.

Collect causes the disk to throw away the old BAM (Block Availability Map) and rebuild a new one. The process starts with the first directory entry. The disk picks up the track and sector co-ordinates of the first block of the first file and begins tracing the block chain. During the trace, the disk re-allocates each block back into the BAM. Collect is complete once all directory entries (PRG, SEQ, REL and USR) have been examined.

Improperly closed files are thrown away by the Collect operation. An improperly closed file is indicated by an asterisk (*) preceding the file type in a directory listing. This can occur in any number of ways; no DCLOSE or CLOSE command after recording a file; DISK FULL occurring before the file is CLOSEd; hitting STOP while saving a program; or a power failure while storing data.

Regardless of how it happens, unclosed files should NOT be SCRATCHED! As you know, SCRATCH does not erase blocks, it merely de-allocates them from the BAM. This means that the old data is left behind (including track & sector chain pointers) but in blocks that are now available for re-use.

Consider this: You pull out a full or almost full diskette. The diskette has no improperly closed files. Now you want to save a couple of programs on this diskette but there's not enough room. So you SCRATCH 4 or 5 old files that are no longer needed. With more than enough space you SAVE your first new program... no problem. Now you go to save the second program and for some reason the operation is aborted (DISK FULL, STOP key, etc.) leaving this file improperly closed! Chances are that the last block to be written points at a block that was previously used by one of your old files. This block would contain old track & sector pointers which might point at other blocks that are now in use by (quite possibly) the program that you just saved successfully. SCRATCHing this unclosed file would then go de-allocating blocks that were just written PLUS blocks that belong to your other program. Another SAVE at this point could be hazardous. The disk might choose to re-use those free blocks that belong to the other program, thus replacing parts of the first program with parts of the second... YUK!

A COLLECT after the aborted SAVE would have avoided all problems. The unclosed PRG file would be discarded, and the integrity of the other files preserved. Some believe that reported problems with write & replace (using the '@' symbol) are connected somehow to the presence of unclosed files, but no proof is available.

Collect has only one drawback. Any blocks allocated by the block-allocate (B-A) command will be freed by Collect as these will not belong to a chain as with other files. Subsequent B-A & B-W commands will use these blocks, possibly overwriting valid data. However, with the advent of Relative files, direct access should be fading from use.

Otherwise, it's never too soon for a Collect. If your block count doesn't add up or you suspect another undesirable condition, use Collect to be safe.

"Now let's see, for optimum results add 1.29987143 cups of flour."

Keyword Abbreviations

PET/CBM/VIC keywords include all commands in BASIC: LOAD, POKE, NEXT, GOSUB, MID$, to name but a few. Each keyword has a respective number or 'token' (eg. END is a 128, FOR is 129 and so on). As commands are entered, the operating system scans or parses the characters typed and compares them against the keyword table in ROM. When it 'sees' a keyword that it recognizes, PET crunches it into its respective token. In direct mode, this token is then passed to the operating system to be executed. When writing a program line, the token is stored in RAM for later execution. By doing this, PET can use a single byte to represent a command, thus optimizing on memory space and maximizing on speed during execution.

But just like PRINT, which is abbreviated with a "?", all BASIC commands and statements can be abbreviated. Thanks to a "bug" (?!) in the operating system, all keywords can be entered by typing the first letter followed by the shifted second letter. Depending of what mode you're in, the latter will show up as either a graphic character or a capital letter. If they are entered into program lines, you'll see that the LIST command uncrunches the tokens into their expanded versions.

This can be extremely useful in circumstances such as: often used commands like CATALOG (eg. cAd0), LIST (lI) and DSAVE (dS) can be entered quickly with a minimum of typing effort; program lines that, for one reason or another, contain more code than can fit on a line and; after displaying the directory, the cursor can be moved up beside the filename where any number of commands could be issued without the need for retyping the filename or moving it over to accommodate the expanded keyword. Here you could give dL for DLOAD, sC for SCRATCH, reN for RENAME, cO for COPY, and more just by typing the abbreviation on top of the block count (you'll also have to erase the file type or place a colon after the filename else ?SYNTAX ERROR).

There are a few exceptions. The abbreviation for PRINT is not "pR"... that belongs to PRINT#. There is no abbreviation for INPUT, but INPUT# is "iN". Words such as TO, IF, OR and ON also cannot be abbreviated, nor can reserved variables such as ST, TI, TI$, DS or DS$... but lets not be too lazy since they're only two letters anyways. Other keywords have the same second letter: LET, LEN and LEFT$; READ, RESTORE, RETURN and RENAME; GOTO and GOSUB. The shortest of theses sets will be abbreviated with the shifted second letter, the others with the shifted third letter. The TAB (tA) and SPC (sP) functions will also give you the opening bracket, so watch that you don't add in a second one!

There are a few rules to remember, but with practise you'll find using abbreviations most enjoyable!

From The Editor - Karl J. Hildon

This issue is probably our biggest ever! Many thanks to all contributing writers, especially Dave Hook, Ted Evers, Glen Pearce, Jim Butterfield and Greg Yob. Don't miss Gregs' two articles on the IEEE bus and the User Port, reprinted by permission from Kilobaud Mag. Jim Butterfield's latest "SuperChart" appears again this issue with updates for screen control characters and VIC-20 colour controls. Next issue we plan a special VIC bonus section which may develope into a new Commodore Canada magazine... any ideas for a name ?

	Subscription Fees		
	Canada	U.S.A.	All other foreign
The Best of The Transactor Volume 1	$10.00	$10.00	$12.00
The Transactor Volume 2	$15.00	$17.00	$19.00
The Transactor Volume 3	$10.00	$11.00	$13.00

The Best of The Transactor Volume 1 is Volume 1 back issues together in one bind.

All 12 of The Transactor Volume 2 back issues will be available for a limited time only. After supplies run out, The Best of The Transactor Vol. 2 will replace the back issues, cost unchanged.

A subscription to The Transactor Volume 3 will cover 6 issues, back issues included.

NOTE: Pre-payment required. Invoices cannot be issued for subscription fees.

Commodore Business Machines
3370 Pharmacy Ave.
AGINCOURT, Ontario
M1W 2K4
Atn: The Transactor

Keyed Random Access For The PET/CBM

Glen Pearce
Commodore Johannesburg

Since the advent of Relative Files and the large storage capacity of the CBM 8050 Disk, some form of 'K.R.A' (Keyed Random Access) would be useful to make full use of these facilities. Here is a version that meets most of the specifications of K.R.A, but is relatively (excuse the pun!) easy to use. It works as follows:-

An ordinary sequential file is used to store a 'key-file' of all records held within a system (eg. Stock, Accounts, Clients, etc.). This key-file would normally contain the first 10 characters of a Customer's name (Part #, Account #, etc.) followed by the Relative Record Number of the record containing the remaing data for that Customer.

Right - now all you have to do is search through this key-file until you find the record you're looking for; retrieve the relative record number and you have access to the main record. The only problem in doing this in BASIC is time - especially if you have 500 to 1000 records or more!

Here is a machine-code routine which will do the above significantly faster (it searches through 500 ten-character record keys in approximately 4 seconds). This routine may only be used with BASIC 4.0 and DOS 2.0. Here's how you use it:-

The length of each record in the key-file (SEQ) is not important and it may contain any valid ASCII characters (for safety's sake, stick to alpha-numerics only). To seperate the record-key from the associated relative record number, a delimiter must be used. In this case the delimiter is a '#' symbol. Therefore, a record in the SEQ key-file should look something like:

```
SMITH# 1234
```

The space between the delimiter and the rel/rec number is the sign of the number and can be suppressed if space-saving on the disk is necessary.

It is important that each record in the key-file be seperated by a Carriage Return - CHR$(13). This shouldn't present any problem as the PET/CBM automatically sends this character after each PRINT# command.

The K.R.A. machine code program must be located at the top of memory and protected in the usual way:

```
POKE 53, 127 : POKE 52, 0 : CLR
```

...must be the first statement in your program.

This program also allows you to do a form of 'pattern-matching'. Say, for instance, you don't know the exact spelling of a record-key in the key-file. All you do is enter the first few characters of the record-key and allow the program to search for that. When a 'match' is found in the file, the attached rel/rec number will be returned. You could then retrieve that relative record and display it. If it is NOT the correct record, simply tell the program to continue searching the key-file until it finds another match and so on. If NO match is found, a relative record number of 0 (zero) will be returned by the K.R.A. routine.

Here is an example of a BASIC program using the routine:

```
100 A$="" : A=0 : REM INITIALIZE VARIABLES BEFORE
    USING K.R.A.
110 INPUT "ENTER SEARCH-STRING";A$
120 DOPEN#2,"KEY-FILE" : IF DS <> 0 THEN PRINT DS$:
    STOP
130 SYS 32512, 2, A$, A
140 IF A = 0 THEN DCLOSE#2 : STOP : REM NO MATCH
150 REM RETRIEVE THE ASSOCIATED RELATIVE RECORD
160 REM AT THIS STAGE, IF THE REL/REC IS NOT CORRECT
170 REM YOU COULD 'GOTO 130' TO LOOK FOR ANOTHER MATCH
```

Any string and numeric variable may be used, but should be declared before the SYS 32512 to the routine. (In the above example 'A$' would have been initialized by the INPUT statement anyways). The '2' used after the first comma in the SYS command is the logical file number used in the DOPEN statement. It is important to check the DISK STATUS word (DS) after opening the file.

Adding records to the key-file could be a problem once the file gets large. Make use of the APPEND# command in BASIC 4.0 to simply append new record-keys to the file.

Another suggestion is to have seperate key-files. For alphabetic keys there would be 26 titled 'A' to 'Z'; for numeric keys, 10 labelled '0' to '9'; or combine for alphanumeric and have 36 seperate key files. Now you could simply check the first character of the search string (ie. LEFT$(A$,1)) and open that particular file. This would reduce your key-file size to approximately 100 records per file in a 2000 record system, thereby making your search times even faster!

__Editor's Note__

Glen's K.R.A. routine could be a perfect partner for the BMB Stringthing published in Volume 3, Issue #1. Only one problem... they both want to live at the same place in memory. For those with assemblers, either routine could be reassembled lower in memory ($7D00). Don't forget to change the SYS numbers and also the POKEs to lower top of memory farther down.

For those without assemblers, it will probably be easier to move K.R.A. down rather than Stringthing. Simply change each occurence of $7F in the source listings (127 in the BASIC loader) to $7D (decimal 125). This means that K.R.A. will start at $7D00. Remember that the BME Stringthing requires a 256 byte buffer which has been slated for the $7E00 page and followed by the program at $7F02. Therefore K.R.A. must go an extra page lower... but no problem. Now enter K.R.A. with SYS 32000 and the POKEs to protect it in high memory become: POKE 53, 125 : POKE 52, 0 : CLR . You'll also have to change the parameters of the FOR/NEXT loop in the loader to FOR I = 32000 to 32255...

One last thing to watch... both K.R.A. and Stringthing use locations 0 and 1 in zero page for work space. This won't harm the operation of either routine but the Stringthing returns the results of Position Search into $00. This result is then PEEKed by the programmer. If, for any reason, you'll need this value after a call to K.R.A., then you'd better save it (ie. PS=PEEK(0)) or K.R.A. will clobber it!

```
30 REM ****************************************************
40 REM *                                                  *
50 REM *   BASIC LOADER FOR MACHINE CODE ISAM ROUTINE     *
60 REM *                GLEN PEARCE  20/8/81              *
70 REM *                                                  *
80 REM ****************************************************
90 REM
100 POKE53,127:CLR:REM LOWER MEMTOP TO PROTECT PROGRAM
110 FOR I = 32512 TO 32767 : READ J : POKE I, J :NEXT : END
200 DATA  32,  73, 127,  32,  45, 201, 165,  18, 240,   3
210 DATA  76,   0, 191, 165,  17, 133, 210,  32,  82, 127
220 DATA 166, 210,  32, 198, 255, 160,   0,  32, 228, 255
230 DATA 166, 150, 208,  66, 201,  13, 240, 243, 209,   1
240 DATA 208,  18, 200, 196,   0, 144, 236,  32, 228, 255
250 DATA 166, 150, 208,  46, 201,  35, 240,  90, 208, 243
260 DATA  32, 228, 255, 166, 150, 208,  33, 201,  13, 240
270 DATA 210, 208, 243,  32, 245, 190,  32, 152, 189, 160
280 DATA   0,  96,  32,  73, 127, 177,  68, 133,   0, 200
290 DATA 177,  68, 133,   1, 200, 177,  68, 133,   2,  96
300 DATA  32,  73, 127, 169,   0, 133,  95, 133,  96, 133
310 DATA   7, 162, 144,  32, 122, 205, 160,   0, 165,  94
320 DATA 145,  68, 200, 165,  95,  41, 127, 145,  68, 200
330 DATA 165,  96, 145,  68, 200, 165,  97, 145,  68, 200
340 DATA 165,  98, 145,  68,  32, 204, 255,  96,  32,  73
350 DATA 127, 169,   0, 133,  95, 133,   7,  32, 195, 127
360 DATA 201,  13, 240,  23, 166, 150, 208, 188, 133,  96
370 DATA  32, 195, 127, 201,  13, 240,  10, 166, 150, 208
380 DATA 175,  32, 213, 127,  76, 170, 127, 162, 144,  32
390 DATA 122, 205,  76, 116, 127,  32, 228, 255, 201,  13
400 DATA 240,  10, 201,  48, 144, 245, 201,  58, 176, 241
410 DATA  41,  15,  96, 133,   0, 165,  95,  72, 165,  96
420 DATA  72,   6,  96,  38,  95,   6,  96,  38,  95, 104
430 DATA 101,  96, 133,  96, 104, 101,  95, 133,  95,   6
440 DATA  96,  38,  95, 165,   0, 101,  96, 133,  96, 169
450 DATA   0, 101,  95, 133,  95,  96
```

LINE# LOC CODE LINE

```
0000                  ;*********************************************
0000                  ;*  SEARCH THRU A SEQ FILE FOR A KEY RECORD AND  *
0000                  ;*   THEN RETRIEVE AN ATTACHED REL/REC NUMBER.   *
0000                  ;*                                               *
0000                  ;*    GLEN PEARCE                  22/08/81      *
0000                  ;*    COMMODORE,  JOHANNESBURG,  SOUTH AFRICA    *
0000                  ;*********************************************
0000                  ;
0000                  ; ## CONSTANTS FROM PET BASIC (BASIC 4.0) ##
0000                  GETCHR = $FFE4          ;GET A CHARACTER
0000                  CLRCHN = $FFCC          ;CLOSE I/O CHANNELS
0000                  COIN   = $FFC6          ;SET INPUT DEVICE
0000                  CHKCOM = $BEF5          ;CHK FOR COMMA
0000                  FRMEVL = $BD98          ;EVALUATE EXPRESSION
0000                  FACINT = $C92D          ;CONVERT FL/P TO INT
0000                  SNERR  = $BF00          ;PRINT SYNTAX ERROR
0000                  ;
0000                  ; ## PAGE ZERO VARIABLES ##
0000                  LENGTH = $00            ;TEMP STORE OF STR LENGTH
0000                  WORK1  = $01            ;TEMP WORK AREA
0000                  CHKINT = $11            ;CHECK FOR INTEGER
0000                  CURFIL = $D2            ;CURRENT FILE NUMBER
0000                  VARPNT = $44            ; PNTR TO CURRENT VARIABLE
0000                  FAC    = $5E            ;MAIN FLT/PNT ACCUMULATOR
0000                  ;
0000                         * = $7F00
7F00                  ;
7F00  20 49 7F        FIND   JSR EVALEX       ;CHK SYNTAX OF COMMAND
7F03  20 2D C9               JSR FACINT       ;IN BASIC LINE & EXTRACT LFN
7F06  A5 12                  LDA CHKINT+1     ;AND SEARCH STRING
7F08  F0 03                  BEQ ISINTG
7F0A  4C 00 BF               JMP SNERR        ;EXIT IF SYNTAX ERROR
7F0D  A5 11           ISINTG LDA CHKINT
7F0F  85 D2                  STA CURFIL       ;SET UP LFN FOR READ
7F11  20 52 7F               JSR FNDEXP       ;FIND SRCH STRING
7F14  A6 D2                  LDX CURFIL
7F16  20 C6 FF               JSR COIN         ;SET I/O FOR READ
7F19                  ;
7F19  A0 00           GET10  LDY #0
7F1B  20 E4 FF        GET11  JSR GETCHR       ;GET CHAR FROM FILE
7F1E  A6 96                  LDX $96          ;CHK STATUS BYTE FOR EOF
7F20  D0 42                  BNE DONE1
7F22  C9 0D                  CMP #13          ;CHK FOR C/RET
7F24  F0 F3                  BEQ GET10        ;MOVE TO NEXT RECORD
7F26  D1 01                  CMP (WORK1)Y     ;COMPARE TO EQUIVALENT
7F28  D0 12                  BNE CLRSTR       ;CHAR OF SEARCH STRING
7F2A  C8                     INY
7F2B  C4 00                  CPY LENGTH       ;IF NUMBR OF CHARS CHK'D
7F2D  90 EC                  BCC GET11        ;EQUALS LEN OF SEARCH STRING
7F2F  20 E4 FF        FNDDEL JSR GETCHR       ;THEN MATCH IS MADE
7F32  A6 96                  LDX $96
7F34  D0 2E                  BNE DONE1
7F36  C9 23                  CMP #'#          ;FIND DELIMITER & THEN GO
7F38  F0 5A                  BEQ RELNUM       ;AND READ IN REL/NO.
7F3A  D0 F3                  BNE FNDDEL
```

```
LINE# LOC CODE LINE

7F3C  20 E4 FF   CLRSTR JSR GETCHR      ;DISCARD REST OF STRING
7F3F  A6 96             LDX $96
7F41  D0 21             BNE DONE1
7F43  C9 0D             CMP #13
7F45  F0 D2             BEQ GET10       ;GO AND CHK NEXT STRING
7F47  D0 F3             BNE CLRSTR
7F49             ;
7F49  20 F5 BE   EVALEX JSR CHKCOM      ;CHK FOR COMMA
7F4C  20 98 BD          JSR FRMEVL      ;& EVALUATE EXPRESSION
7F4F  A0 00             LDY #0
7F51  60                RTS
7F52             ;
7F52  20 49 7F   FNDEXP JSR EVALEX      ;FIND SRCH STRING
7F55  B1 44             LDA (VARPNT)Y   ;SET UP STRING PNTRS
7F57  85 00             STA LENGTH      ;IN TEMP WORK AREAS
7F59  C8                INY
7F5A  B1 44             LDA (VARPNT)Y
7F5C  85 01             STA WORK1
7F5E  C8                INY
7F5F  B1 44             LDA (VARPNT)Y
7F61  85 02             STA WORK1+1
7F63  60                RTS
7F64             ;
7F64  20 49 7F   DONE1  JSR EVALEX      ;IF NO MATCH FOUND THEN
7F67  A9 00             LDA #0          ;RETURN A REL/NO. OF ZERO
7F69  85 5F             STA $5F
7F6B  85 60             STA $60
7F6D  85 07             STA $07         ;SET VARIABLE TYPE TO NUMERIC
7F6F  A2 90             LDX #$90
7F71  20 7A CD          JSR $CD7A       ;CONVERT HEX TO FL/P
7F74  A0 00      DONE2  LDY #0
7F76  A5 5E             LDA FAC         ;TRANSFER BCD VALUE OF
7F78  91 44             STA (VARPNT)Y   ;REL/NO. TO NUMERIC VAR
7F7A  C8                INY             ;SPECIFIED IN SYS CMD
7F7B  A5 5F             LDA FAC+1
7F7D  29 7F             AND #$7F        ;STRIP OFF SIGN
7F7F  91 44             STA (VARPNT)Y
7F81  C8                INY
7F82  A5 60             LDA FAC+2
7F84  91 44             STA (VARPNT)Y
7F86  C8                INY
7F87  A5 61             LDA FAC+3
7F89  91 44             STA (VARPNT)Y
7F8B  C8                INY
7F8C  A5 62             LDA FAC+4
7F8E  91 44             STA (VARPNT)Y
7F90  20 CC FF          JSR CLRCHN      ;CLEAR ALL I/O CHANS AND
7F93  60                RTS             ;EXIT PROGRAM
7F94             ;
7F94  20 49 7F   RELNUM JSR EVALEX      ;FIND VARIABLE FOR REL/NO.
7F97  A9 00             LDA #0
7F99  85 5F             STA $5F
7F9B  85 07             STA $07
7F9D  20 C3 7F          JSR NEWDIG      ;READ IN REL/NO. AND CONVERT
7FA0  C9 0D             CMP #13         ;IT TO A 2-BYTE HEX DIGIT
```

```
LINE# LOC CODE LINE

7FA2  F0 17            BEQ PUTVAR
7FA4  A6 96            LDX $96
7FA6  D0 BC            BNE DONE1
7FA8  85 60            STA $60
7FAA  20 C3 7F  NXTDIG JSR NEWDIG
7FAD  C9 0D            CMP #13
7FAF  F0 0A            BEQ PUTVAR
7FB1  A6 96            LDX $96
7FB3  D0 AF            BNE DONE1
7FB5  20 D5 7F         JSR ASCHEX
7FB8  4C AA 7F         JMP NXTDIG
7FBB  A2 90     PUTVAR LDX #$90
7FBD  20 7A CD         JSR $CD7A
7FC0  4C 74 7F         JMP DONE2
7FC3            ;
7FC3  20 E4 FF  NEWDIG JSR GETCHR      ;GET NEXT REL/NO. DIGIT
7FC6  C9 0D            CMP #13
7FC8  F0 0A            BEQ ENDDIG
7FCA  C9 30            CMP #$30        ;CHK FOR NUMERIC
7FCC  90 F5            BCC NEWDIG
7FCE  C9 3A            CMP #$3A
7FD0  B0 F1            BCS NEWDIG
7FD2  29 0F            AND #$0F        ;MASK OUT THE FOUR MSB'S
7FD4  60        ENDDIG RTS
7FD5            ;
7FD5  85 00     ASCHEX STA LENGTH      ;HANDLE ASC - HEX CONVERSION
7FD7  A5 5F            LDA $5F
7FD9  48               PHA
7FDA  A5 60            LDA $60
7FDC  48               PHA
7FDD  06 60            ASL $60
7FDF  26 5F            ROL $5F
7FE1  06 60            ASL $60
7FE3  26 5F            ROL $5F
7FE5  68               PLA
7FE6  65 60            ADC $60
7FE8  85 60            STA $60
7FEA  68               PLA
7FEB  65 5F            ADC $5F
7FED  85 5F            STA $5F
7FEF  06 60            ASL $60
7FF1  26 5F            ROL $5F
7FF3  A5 00            LDA LENGTH
7FF5  65 60            ADC $60
7FF7  85 60            STA $60
7FF9  A9 00            LDA #0
7FFB  65 5F            ADC $5F
7FFD  85 5F            STA $5F
7FFF  60        RETN   RTS
8000                   .END

ERRORS = 0000
```

ROM Sockets

S. Donald, Rossland B.C.

For those of you with the old 8k PET and 24 pin ROMs who envy the three empty sockets in the newer machines, good news.

The 'upgrade' ROMs for these machines only occupy four of the seven sockets and a simple cut and hack operation on your main board will enable you to use two of the freed sockets. All three sockets may be used by the simple addition of one more IC.

Furthermore, if you want to only use one socket for the Toolkit, or the Word Pro 3, you don't even have to pull the board from the case.

A word of advice, however. If you are not reasonably expert in handling this type of operation (soldering directly to the IC pins), or live and work in a high 'static electricty' environment, don't try it.

This modification requires two sequences of events:

1. Change the bank select lines to the emptied ROM sockets, and
2. Change the bank access to the external PET data bus.

Both these operations may be done with the main board still in the case if only one socket is to be enabled. If you want two sockets operational, you have to pull the board to get at a trace on the underside.

Change Bank Select Lines

The 'bank' addresses of the three freed sockets has to be changed from C, D, and F, (in hexadecimal notation; 12, 13, and 15, in decimal), to 9, A, and B, or whatever. The three bank select lines of interest originate at IC G2, pin 14 (select C or, 12), pin 15 (select D, or 13), and pin 16 (select F, or 15). They run a short distance toward the front of the board on the underside of the card, then surface near socket H4. They run across upper surface of the board toward the power supply for several inches then return to the underside of the board to connect to pin 20 of the appropiate socket. These three traces are to be cut just above H5. Be very sure that the traces are completely cut and that you remove all the metal scrap that is generated.

Now carefully solder three wires to IC G2 pin 10 (select 9), pin 11 (select A, or 10), and pin 13 (select B, or 11). Run these wires to the solder dots on the ROM ends of the traces just cut. Simple. But if you try to get the machine to recognize ROMs plugged into these sockets, it will insist that there is nothing there!

Data Bus Access

The problem lies in the design of the data bus. The PET presumes that all addresses between the screen memory and the four ROMs of the operating system are external to the machine. When accessing these addresses, it enables the external data bus drivers. These drivers take data from the outside world and place it on the internal bus. In the meantime the ROM you have just installed is trying to do the same thing. That doesn't work well at all. The solution here is quite simple; don't allow the external data bus drivers to be activated when your ROM Socket is being addressed.

The IC which controls this action is G4, a 74LS21. Two of the input lines to this chip are not used in the original model and may be 'stolen' to enable two of the freed sockets. The trace that ties the two pins of interest (pins 4 and 5) together is on the underside of the board. If only one socket is to be used (say for WordPro) you do not have to seperate them and the board can be left in place during the alteration.

These pins are held at logic '1' ('high') by a resistor at IC G3. The trace of interest is on the upper surface of the board, and goes from the resistor to IC G3, pin 9, and IC G4, pin 5. Cut the trace near G4, remove the scrap metal, and run a wire from G4 pin 5 to the appropiate bank select wire installed in part 1, above. To use a second socket, you have to remove the main board, cut the trace connecting IC G4 pins 4 and 5 together, and run a second wire from pin 4 to another bank select line.

The third socket may be used, but you have to instal another IC. Drop me a line and I'll send you a schematic. My address is Box 481, Rossland B.C., V0G 1Y0.

If you are like me and have the Toolkit hung on the side of the PET at the expansion port, you can even have two ROMs with same address, selectable with an external switch. The bank select signal goes to the switch and is routed to the appropiate ROM. The unselected ROM must have the bank select line pulled high with a 1k resistor to the +5 volt power supply line. The circuit is left as a exercise, but don't forget to switch the external data bus drivers at the same time.

<u>4032 Program Conversions</u>

Joe Ferrari,
Commodore Canada

The addition of some new features to the 40 column PET has brought about some problems with program compatibility between the 4032 nine inch and 4032 twelve inch CRT display machines. In some cases the changes required to programs for proper operation on the 'FAT FORTY' may be trivial, and in other cases the conversion may be a little more difficult. In the following text I will attempt to cover as many areas where possible failure can occur and what changes need implementing.

<u>LEVEL 1: Programs Loading Below BASIC (<$0400)</u>

Standard BASIC programs should work without any modification, unless they employ PEEKs and POKEs or if the program loads into memory below BASIC. The latter problem can be a bit tricky to spot unless you know specifically what to look for. If the program does load below BASIC (say $033A) but does not use locations $03E9-03F9, one method that will correct the problem is:

1) LOAD the program (don't execute)
2) enter the monitor (SYS 4)
3) display hex 03E9 - 03F9
4) modify the display as follows:

```
.:  03E9 10 10 09 10 00 00 00 00
.:  03F1 00 00 00 00 00 00 00 00
.:  03F9 00 ...[don't change]...
```

5) resave the program via the monitor

Tape Unit #2

Another area where the standard BASIC rpogram can fail is in the utilization of the second cassette unit for sequential file access. If any program calls files from Tape Unit #2, unpredictable effects can result depending on the data coming in to the buffer. In this case nothing can be done to resolve the problem unless the data can be handled from Cassette #1. This would require all associated OPEN commands to be modified for device 1. The 12" 4032 uses parts of the second cassette buffer for other reasons that can't be interfered with.

PEEKs & POKEs

Decimal location 151, which is often used to check if a particular key has been pressed, is still the same on the 12 inch, but the value of the keys have changed and therefore expected values for certain keys will return false information. The following table will assist in the conversion of a program with this problem.

KEY	OLDV	NEWV	KEY	OLDV	NEWV
@	15	64	S	40	83
A	48	65	T	62	84
B	30	66	U	61	85
C	31	67	V	23	86
D	47	68	W	56	87
E	63	69	X	24	88
F	39	70	Y	54	89
G	46	71	Z	32	90
H	38	72			
I	53	73	0	10	48
J	45	74	1	26	49
K	37	75	2	18	50
L	44	76	3	25	51
M	29	77	4	42	52
N	22	78	5	34	53
O	60	79	6	41	54
P	52	80	7	58	55
Q	64	81	8	50	56
R	55	82	9	57	57

When POKEs to this problem area are used for saving byte variables (or any data for whatever purpose), they must be moved to a free spot elsewhere in memory. If space is free just below $03E9, then this could be a good area for relocating the byte variables.

LEVEL 2: BASIC Programs Containing Machine Language

BASIC programs using machine language utilities that reside in the second cassette buffer can work properly provided they don't use the taboo area of the buffer (ie decimal 1001-1017). Again, if the utility uses this area, the space must be relinquished to the PET operating system in order to obtain successful operation of the program. Usually, in the case of small machine language utilities, it shouldn't be too difficult to understand and relocate to an area of memory that is free.

LEVEL 3: Machine Language Programs

This will be the most difficult area to troubleshoot. If you are going to attempt modifying this type of program, be prepared to spend a good deal of time. Making the necessary changes to get the program working will most likely require a considerable amount of effort, which I personally don't recommend. In most cases the author should be contacted and he/she should facilitate the changes. If you are really desperate, here are a few helpful hints that may assist you:

1) use Supermon or Extramon to locate any absolute occurences of memory addresses from $03E9 to $03F9 and re-assign new values

2) check hi-low tables for references to the same address locations and, if any, re-assign new values

3) seek all immediate operations involving $03 and $E9-F9... if any, look at code where occurence takes place and evaluate

4) check all JSR & JMP occurences into the $E000 ROM. All other ROMs can be ignored since they are identical.

Factory CRT Setup

One other problem that may be encountered is screen setup. If the user decides to enter 'text mode' with "PRINT CHR$(14)", the top line of the screen may run off the upper edge and not be visible. To restore 'graphics mode' enter "PRINT CHR$(142)". One easy solution to this problem is to use "POKE 59468, 14". This will put the PET in text mode without opening up pixel lines between text.

CONCLUSION

The changes required to existing softwaremay be a problem now but, at the same time, these changes bring the 4032 to a closer compatibility with the 8032 model. Features such as repeat keys, scroll up and down, the bell, and more have been implemented. These changes make the 4032 a much more desirable product. I hope the information in this report will help support the 12 inch 4032. If anyone encounters a problem that I have not covered, please let me know.

Editor's Note

The new 12" monitors have an adjustment for screen height. At the factory, the machines are turned on, and this adjustment is used to set the top line of text just under the top of the CRT face. However, unlike the 8032 which comes up in text mode, the 4032 comes up in graphics mode. Therefore, when text mode is set on the 4032, the top and bottom lines get pushed off the screen. You'll also note that when graphics mode is entered on the 8032, a 'rectangular' display results with a gap at the top and bottom of the screen.

If you want to use text mode on the 4032, you can adjust the screen height with very little effort and bring the top and bottom lines of text back onto the glass. Undo the machines main housing screws under both sides of the keyboard, open the 'lid', and set the stand in place. If you look up underneath the monitor, you'll see the bottom of the video circuit board. There are two adjustments accessible from here. One is marked 'Screen Height'. Fill the screen with characters and enter text mode (CHR$(14)). With a small screwdriver you can adjust Screen Height to get the full display.

According to these calculations, we're only going to be able to make three payments on this thing.

Half a Dialogue - Inputting

Jim Butterfield, Toronto

Asking a program to go and get input from the user is a subtle thing for beginners. When you write your first programs, it's hard to look ahead and see the program independently communicating with the user. "If the program needs a value, I'll program it in right now ... " It takes a level of sophistication to imagine a program accepting working values at a later time, when it runs, and using different values supplied by the user in different runs.

There are three fundamental ways of checking what the user is doing at the keyboard: INPUT, GET, and a PEEK. We'll talk about each, and its uses.

INPUT.

The INPUT statement does a lot of work for you. It's certainly one of the most powerful statements in Basic. Some of us would like to see it more powerful, and some would like to see it less sophisticated; for the moment, we'll have to accept it as it is.

When you give the command INPUT in a program, a prompting question mark is printed and the cursor begins to flash. Your program is held in suspended animation; it will not resume operation until RETURN is pressed. There's no code which allows something like:

```
INPUT M:IF (NO REPLY IN 15 SECONDS) GOTO...
```

Your code will hang on the INPUT statement forever if the user doesn't reply.

When the user presses RETURN, INPUT takes the information from the screen. It doesn't matter if the user wandered back and forth, changing, deleting and inserting; INPUT looks only at the screen which is the result of his actions. In fact, if there's something on the screen that the user didn't type, INPUT will take that too. This can be useful for prompting: you can arrange to type a sample response on the screen, and the user will be able to press RETURN to have that response entered. As INPUT takes the information from the screen, it trims away all leading and trailing spaces; other than that it takes the whole line, even though it may not need it.

Now INPUT starts to plow through the line, digging out the information you need for your program. If it's looking for a number it will not like to find a string, and will ask, REDO FROM START. If it's looking for a string, it won't mind a number at all: it will accept it as a string.

Road Signs for INPUT.

Whether INPUT is looking for a number or a string, it will stop its search when it finds one of three things; comma, colon, or end of line. If it finds a comma it will assume that more information will be needed later in the INPUT statement; if it finds a colon or end of line it assumes that there is no more useful input from the user. If it needs more, it will ask for it.

Suppose you need to input a string that contains a comma or a colon, such as ULYSSES M PHIPPS, PHD. or ATTENTION: JOHN, MARY. Since INPUT normally stops at the comma or colon character, we need to do something. The answer is easy: the user must put the desired input in quotes: "ATTENTION: JOHN, MARY" and the whole thing, commas, colon and all, will be received as a single string.

Keep in mind that the INPUT statement allows prompting. INPUT "YOUR NAME";N$ causes the computer to type YOUR NAME? and wait for input. That's a good human interface; help the user along.

If a user presses RETURN without supplying any information on the screen, programs on the PET/CBM will stop. There are several ways to prevent this from happening; the easiest is to add a "canned reply" to the input prompt message. When you are writing the INPUT statement prompt (such as YOUR NAME) add two extra spaces and, say, an asterisk character; then type three Cursor-Lefts (they will print as an odd-looking reversed bar) and close the quotes on the prompt. Finish the INPUT statement in the usual way: a semicolon behind the prompt and then the name of the variable to be input. Now: the asterisk or whatever will print to the right of the prompt and question mark. Unless the user overtypes it, this character will be received from the screen as his input - and the program won't stop.

One last comment: don't forget that INPUT can accept several values. You can say INPUT N$,A$,C$ and allow the user to type JOE BLOW, CITY HALL, DENVER. It's often better to use separate input statements: users can respond better when prompted for each piece of information.

GET and PEEK: a preview

GET isn't as clever as INPUT, but it has valuable uses. First of all, it doesn't wait; if a key isn't ready in the keyboard buffer, the GET statement lets Basic continue. Secondly, keystrokes received with GET don't affect the screen unless you, the programmer, decide to allow them to do so. This means that you have much more control over what the user can do.

There's a PEEK location (PEEK(151) on most PET/CBMs, PEEK(515) on Original ROMs, and PEEK(197) on the VIC that tells you whether a key is being held down or not. This can be useful to avoid the situation where a user needs to press the same key repeatedly to cause some action; you can program so that the key repeats its action if it is held down.

We'll talk in more detail about the GET and PEEK next time around. They are more fun in some ways that the INPUT statement... but they call for quite a bit more programming work to be done.

Editor's Note

Jim's next article was made available to The Transactor just shortly after this one. Rather than splitting them between two issues, we've decided to include it here in Issue #4.

Half a dialogue - Reading keys   Jim Butterfield, Toronto

We've already discussed the INPUT statement. When you do an INPUT, the program pauses and waits for the user to compose a line on the screen. When the user presses RETURN, the program resumes and uses the information entered.

This is often useful and convenient; but when we use INPUT, we don't have complete control over the user. If the user doesn't answer, the program is stopped forever, and other jobs will not take place. The user might also do undesirable things like clearing the screen, and might even stop the program if he presses RETURN without any input on the screen.

We can deal with the user on a more elemental level by using the GET command.

GET.

GET takes one character directly from the keyboard buffer; the character does not go via the screen. It's usually a good idea to echo the character to the screen so that the user can see what he's typing (GET X$:PRINT X$;). There is a GET numeric (GET X) which gets a single numeric digit, but it's rare since the program will stop if the user inadvertantly presses an alphabetic key.

GET doesn't wait. If there's no character in the input buffer, GET returns with a null string. We can wait for a key to be pressed with a line like:

```
300 GET X$:IF X$="" GOTO 300
```

You can see that if we get no character, we go back and try again. More sophisticated versions of the same program might allow us to wait for up to 10 seconds for the user to type a key.

GET receives everything typed at the keyboard. Even cursor movements or insert and delete keys are received as single character strings. The RUN/STOP key and the SHIFT are about the only keys that GET won't receive directly.

Screen control keys - cursor move, reverse, home, etc. - are picked up directly by GET and don't influence the screen when typed. If you want them actioned, you'll have to arrange for it yourself, again by echoing the character with a PRINT. On the other hand, GET is an excellent way to prevent a user from clearing the screen or doing other things that you don't want. The easiest way to identify such characters is by their ASC ascii value, but the obvious also works: GET X$:IF X$="[HOME]" GOTO... The Reverse-S symbol will appear where I have typed [HOME].

Sometimes there are left-over characters in the keyboard buffer. The user might have touched the keyboard accidentally, or the last key pressed might have "bounced" and been registered twice. You can strip out such characters with simple coding like GET X$,X$,X$,X$. If the keyboard buffer contains up to four characters, they will be cleared out; if there were none, GET still doesn't hold anything up.

Remember that GET takes characters from the keyboard buffer. For one key depression, no matter if you tap a key quickly or hold it down for five minutes, only one character will go into the buffer and GET will find it there only once.

PEEK.

The value of PEEK(151) will tell you whether or not a key is being held down. If you find 255 there, no key is being pressed - except maybe the SHIFT key which doesn't register there. If there is any value other than 255 in PEEK(151), somebody's holding down a key.

Special note: for Original ROM PETs, the place to check is PEEK(515). And on the VIC, check location PEEK(197); a value of 64 means that no key is being pressed.

It's possible to figure out which key is pressed based on the value you find in the PEEK location, but I don't recommend it. Different keyboards are "decoded" in different ways, and what works on one machine won't necessarily work on another. The best way to sort out which key is pressed is to use the PEEK together with the GET statement.

The trick is this: if GET says that there is no character in the keyboard buffer and PEEK says that someone is holding a key down, it's safe to assume that the key being held down is the last one you received with GET. Timing is important here, since a key could be touched in the split second between two Basic statements. I recommend the following kind of sequence:

```
300 X=PEEK(151)
310 GET X$:IF X$<>"" THEN X1=ASC(X$):GOTO 330
320 IF X=255 GOTO [...NO KEY ACTIVE]
330  .... KEY ACTIVITY
```

This kind of test is very good for movement games, where you are directing something (a ball, a paddle, a tank) around the screen based on whether a key is held down or not.

Summary.

GET is more elementary than INPUT. You'll need to do more work with GET, but you'll have more control over the user input.

Use the PEEK where it's necessary to find out if a key is being held down or not ... it can give you a nice interface, especially where the user would otherwise pound repeatedly on a key.

WORD COUNT 9 David A. Hook, Barrie Ontario

Purpose:

After slaving over the composition of an article, most writers are required to count the words, as the basis for payment for their work. I am told that many commercial word-processors include this function. Neither WordPro 3 nor WordPro 4 contain this feature.

Although my writing efforts are infrequent, my wife has done a lot of freelance work. Currently she is working on a complete rewrite of a BASIC text to be used in Grade 9. This project involves a 40% reduction in word count. Thus, this program was created.

An initial effort was accomplished using BASIC. For a WordPro file with 2200 words, the time to perform the count was a shade over 21 minutes. This was acceptable, since other tasks (non-computer) could be performed while the CBM was busy.

However, we've all heard the praises sung for the speed of Machine Language. The logic aspect was fairly straightforward and already de-bugged in Basic. The results are before you in this article.

The same WordPro file was counted in 12.67 seconds!!

The program works with either WordPro 3 or WordPro 4 files and with Basic 2.0 and Basic 4.0 (Regular-, Fat-40 and 80-column machines). The WordPro file is read from Drive #0 of the disk unit. DOS 2.1 is not necessary, although I have not included an error-checking routine (except for Basic 4.0).

Procedure:

First, type in the BASIC listing exactly as given below. Be very careful to include all the spaces specified, especially in Line 8 of the program. There is one after the CLR/HOME, 13 before the title and 12 following.

Now SAVE this part as "WC.BAS". After VERIFYing, reset the machine for the next step:

For those who wish their own Assembly, skip to Step "b" below.

a) For the "non-Assembler"-crowd here's the method for you. Type in 'SYS4' to get into the M.L. monitor. Then enter the following line, right after the displayed "." (at the present cursor position):

```
.M 0624 06BC  <RETURN>
```

The screen will fill with a display much like that shown in the 'HEX DUMP' listing below. Your task is to carefully change all of the displayed figures to match the listing (top half). Simply type in the proper values, remembering to hit 'RETURN' at the end of each line.

For the remainder, do the same again after typing this line:

.M 06BC 0733 <RETURN>

After making the required changes, this should be SAVEd, using the monitor, as follows:

.S "0:WC.ML",08,0624,0733 <RETURN>

.X <RETURN> (exit the monitor)

You may VERIFY this normally, if you wish.

Now skip to Step "c" below.

b) The source code for the program has been included. This code will work with either MAE or ASM/TED assemblers.

If you choose to relocate the machine-language "start address", remember that there are three references in the Basic portion. Be sure that these get corrected, too.

c) If you're still with me, only two things remain to be done:

Simply reLOAD "WC.BAS" first, then reLOAD "WC.ML". Use the normal BASIC SAVE command now, and both pieces will be linked together.

Remember that any changes to the Basic portion now will also move the machine language. Do so at your own risk.

<u>Operation:</u>

Before you RUN the program, be sure you know the file name of the WordPro file to be counted. Put this diskette into Drive #0, and you are ready to go.

The program self-adjusts for 40- or 80-column operation. This assumes that you will only be counting 40-column files on a 40-column machine, and 80-column files on an 80-column machine. Thus, the correction is based on the machine in use, not the file being read.

The program ignores WordPro format commands (and anything on the same line as a format command).

If you have used the "--" characters as a dash, there should be no preceding or following blanks. If you use a series of "-", (as I sometimes do for underlining) the count may not be correct.

If you've entered everything correctly, the word count total should have appeared on the screen, after 2-25 seconds. Disk activity should end and the "READY" prompt should now be displayed.

Since none of us ever make any misteaks (??), you should be ready to count every WordPro file within reach. In our house, this program has had a real workout. I hope it proves useful to you too.

This is the usual place to acknowledge Jim Butterfield. I blame him for getting me into this all-consuming habit...er hobby!

WORD COUNT 9

LISTING

```
0 REM    WORD COUNT 9 -- WORDPRO 3
1 REM     AS OF NOVEMBER 29, 1981
2 REM
3 REM  (C) DAVID A. HOOK, 58 STEEL ST.
4 REM BARRIE, ONTARIO, CANADA, L4M 2E9
5 REM
6 REM        ALL RIGHTS RESERVED
7 REM
8 PRINT"] R               - WORD COUNT ML          "
9 PRINT"QQQPLACE PROGRAM DISK IN DRIVE #0
10 PRINT"QQHIT A KEY WHEN READY  █";
11 GETZ$:IFZ$=""THEN11
12 PRINT"  OK"
13 INPUT"QPROGRAM NAME   *███";F$
14 OPEN1,8,15,"I0":CLOSE1
15 OPEN2,8,2,"0:"+F$+",P,R"
16 IFDSTHENZ$=DS$:GOTO21
17 SYS1582
18 PRINT"SQQQQQQQQQQWORD COUNT = ";
19 PRINTPEEK(1572)+256*PEEK(1573)
20 Z$="DONE"
21 PRINT"QQ"Z$:CLOSE2:END
READY.
```

WORD COUNT 9
HEX DUMP

```
C*
    PC   IRQ  SR AC XR YR SP
.;  B780 E455 34 33 38 36 FA
.
.:  0624 45 01 99 22 11 12 22 5A
.:  062C 24 3A A2 09 A9 00 9D 24
.:  0634 06 CA 10 FA A9 28 A2 60
.:  063C 8E 00 84 AE 00 80 E0 60
.:  0644 F0 01 0A 8D 28 06 A2 02
.:  064C 20 C6 FF 20 06 07 20 06
.:  0654 07 A2 00 8E 2B 06 8E 29
.:  065C 06 8E 27 06 18 AD 24 06
.:  0664 6D 26 06 8D 24 06 AD 25
.:  066C 06 69 00 8D 25 06 8E 26
.:  0674 06 AE 27 06 EC 28 06 F0
.:  067C D8 20 06 07 EE 27 06 AD
.:  0684 2C 06 AC 2D 06 A2 00 C9
.:  068C 7A D0 09 8E 26 06 20 21
.:  0694 07 4C 55 06 C9 1F D0 0B
.:  069C AE 2B 06 F0 03 EE 26 06
.:  06A4 4C 92 06 AE 29 06 D0 21
.:  06AC AE 27 06 CA D0 13 C0 20
.:  06B4 F0 0F C0 6F F0 0B C9 20
.:  06BC F0 04 C9 6F D0 03 EE 26
.:  06C4 06 C9 20 F0 AC C9 6F F0
.:  06CC A8 A2 FF 8E 29 06 C9 20
.:  06D4 F0 16 C9 6F F0 12 C9 2D
.:  06DC D0 04 C0 2D F0 0F 8E 2B
.:  06E4 06 E8 8E 2A 06 4C 75 06
.:  06EC AE 2A 06 D0 84 AE 2B 06
.:  06F4 F0 08 EE 26 06 A2 00 8E
.:  06FC 2B 06 A2 FF 8E 2A 06 4C
.:  0704 75 06 AE 2C 06 8E 2D 06
.:  070C 20 E4 FF D0 02 09 40 8D
.:  0714 2C 06 A5 96 F0 06 20 CC
.:  071C FF A2 F8 9A 60 AE 28 06
.:  0724 CA 86 B4 20 06 07 A6 B4
.:  072C CA EC 27 06 B0 F3 60 44
.
```

```
                             .LS
                   ;***************************************
                   ;*                                     *
                   ;*     WORDCOUNT.SRC9 -- WORDPRO 3     *
                   ;*                                     *
                   ;*        AS OF NOVEMBER 29, 1981      *
                   ;*                                     *
                   ;*  (C) DAVID A. HOOK, 58 STEEL STREET *
                   ;*      BARRIE, ONTARIO  L4M 2E9       *
                   ;*        CANADA (705) 726-8126        *
                   ;*                                     *
                   ;*          ALL RIGHTS RESERVED        *
                   ;*                                     *
                   ;***************************************
                   ;
                   ; VARIABLES
                   ;
                   CHANNEL     .DE $02        ;DISK CHANNEL NUMBER
                   ENDLIN      .DE $1F        ;END OF LINE
                   BLANK       .DE $20        ;BLANK
                   LENGTH      .DE $28        ;NORMAL = 40 CHARS.
                   DASH        .DE $2D        ;SINGLE DASH
                   SHFSPC      .DE $60        ;SHIFTED SPACE
                   FORSPC      .DE $6F        ;FORCED SPACE
                   FORCMD      .DE $7A        ;FORMAT COMMAND
                   ST          .DE $96        ;STATUS BYTE
                   SAVX        .DE $B4        ;KEEP R(X)
                   ;
                   SCREEN      .DE $8000      ;SCREEN MEMORY
                   IMAGES      .DE $8400      ;SCREEN IMAGES (40 COL.)
                   ;
                   ; BASIC ROUTINES
                   ;
                   SETINP      .DE $FFC6      ;SET INPUT DEVICE
                   CLRCHN      .DE $FFCC      ;RESTORE DEFAULT I/O DEV.
                   WRT         .DE $FFD2      ;PRINT CHARACTER
                   GETCHR      .DE $FFE4      ;GET CHARACTER
                   ;
                               ;.OS      (DON'T STORE CODE)
                   ;
                               .BA $0624
                   ;
0624-              WORDTOT     .DS 2          ;# WORDS (TOTAL)
0626-              LINETOT     .DS 1          ;# WORDS (CURRENT LINE)
0627-              CHARTOT     .DS 1          ;# CHARACTERS (CUR. LINE)
0628-              LINLEN      .DS 1          ;LENGTH OF WORDPRO LINE
                   ;
0629-              LINFLG      .DS 1          ;LINE START FLAG
062A-              BLNKFLG     .DS 1          ;BLANK FLAG
062B-              WORDFLG     .DS 1          ;WORD FLAG
                   ;
062C-              CURCHAR     .DS 1          ;CURRENT CHARACTER
062D-              LASTCHAR    .DS 1          ;LAST CHARACTER
                   ;
062E- A2 09        START       LDX #LASTCHAR-WORDTOT       ;INITIALIZE LOCS.
0630- A9 00                    LDA #0
0632- 9D 24 06     LOOP        STA WORDTOT,X
```

```
0635- CA                           DEX
0636- 10 FA                        BPL LOOP
                  ;
0638- A9 28       SETLEN           LDA #LENGTH  ;40/80 COLUMN ?
063A- A2 60                        LDX #SHFSPC
063C- 8E 00 84                     STX IMAGES
063F- AE 00 80                     LDX SCREEN
0642- E0 60                        CPX #SHFSPC
0644- F0 01                        BEQ FORTY
                  ;
0646- 0A          EIGHTY           ASL A
0647- 8D 28 06    FORTY            STA LINLEN
                  ;
064A- A2 02       SETCHN           LDX #CHANNEL         ;SET CHANNEL FOR INPUT
064C- 20 C6 FF                     JSR SETINP
                  ;
064F- 20 06 07    LOADADR          JSR GET      ;IGNORE LOAD ADDRESS
0652- 20 06 07                     JSR GET
                  ;
0655- A2 00       LINESTRT         LDX #0       ;START A WORDPRO LINE
0657- 8E 2B 06                     STX WORDFLG
065A- 8E 29 06                     STX LINFLG
065D- 8E 27 06                     STX CHARTOT
                  ;
0660- 18          ADDLINE          CLC  ;SUM PREV. LINE INTO TOTAL
0661- AD 24 06                     LDA WORDTOT
0664- 6D 26 06                     ADC LINETOT
0667- 8D 24 06                     STA WORDTOT
066A- AD 25 06                     LDA WORDTOT+1
066D- 69 00                        ADC #0
066F- 8D 25 06                     STA WORDTOT+1
                  ;
0672- 8E 26 06                     STX LINETOT
                  ;
0675- AE 27 06    CHKLINE          LDX CHARTOT  ;IS LINE DONE ?
0678- EC 28 06                     CPX LINLEN
067B- F0 D8                        BEQ LINESTRT
                  ;
067D- 20 06 07                     JSR GET
0680- EE 27 06                     INC CHARTOT
0683- AD 2C 06                     LDA CURCHAR
0686- AC 2D 06                     LDY LASTCHAR
0689- A2 00                        LDX #0
068B- C9 7A                        CMP #FORCMD  ; WORDPRO FORMAT COMMAND ?
068D- D0 09                        BNE NOTFORMAT
                  ;
068F- 8E 26 06    FORMAT           STX LINETOT  ;ZERO LINE COUNT
0692- 20 21 07    FINISH           JSR GETREST  ;IGNORE REST OF LINE
0695- 4C 55 06                     JMP LINESTRT
                  ;
0698- C9 1F       NOTFORMAT        CMP #ENDLIN  ;END OF LINE SYMBOL ?
069A- D0 0B                        BNE MORE
                  ;
069C- AE 2B 06                     LDX WORDFLG
069F- F0 03                        BEQ DONELINE
                  ;
06A1- EE 26 06                     INC LINETOT  ;GOT A WORD
                  ;
```

```
06A4- 4C 92 06   DONELINE   JMP FINISH
                 ;
06A7- AE 29 06   MORE       LDX LINFLG     ;STARTED LINE YET ?
06AA- D0 21                 BNE CONTLIN
                 ;
06AC- AE 27 06   LEADBLK    LDX CHARTOT    ;LEAD BLANK IMPORTANT ?
06AF- CA                    DEX
06B0- D0 13                 BNE NOTLEAD    ;NOT FIRST CHAR.
                 ;
06B2- C0 20                 CPY #BLANK     ;LAST OF PREV. LINE ?
06B4- F0 0F                 BEQ NOTLEAD
                 ;
06B6- C0 6F                 CPY #FORSPC
06B8- F0 0B                 BEQ NOTLEAD
                 ;
06BA- C9 20                 CMP #BLANK     ;CURRENT CHARACTER ?
06BC- F0 04                 BEQ COUNT
                 ;
06BE- C9 6F                 CMP #FORSPC
06C0- D0 03                 BNE NOTLEAD
                 ;
06C2- EE 26 06   COUNT      INC LINETOT    ;LEAD BLANK MEANS A WORD
                 ;
06C5- C9 20      NOTLEAD    CMP #BLANK     ;IGNORE LEAD BLANKS
06C7- F0 AC                 BEQ CHKLINE    ;CONTINUE THE LINE
                 ;
06C9- C9 6F                 CMP #FORSPC
06CB- F0 A8                 BEQ CHKLINE    ;CONTINUE THE LINE
                 ;
06CD- A2 FF      CONTLIN    LDX #$FF       ;START THE LINE
06CF- 8E 29 06              STX LINFLG
                 ;
06D2- C9 20                 CMP #BLANK
06D4- F0 16                 BEQ WORDCOUNT
                 ;
06D6- C9 6F                 CMP #FORSPC
06D8- F0 12                 BEQ WORDCOUNT
                 ;
06DA- C9 2D                 CMP #DASH      ; '--' IS ALSO A WORD END
06DC- D0 04                 BNE NOTDASH
                 ;
06DE- C0 2D                 CPY #DASH
06E0- F0 0F                 BEQ DASHCOUNT
                 ;
06E2- 8E 2B 06   NOTDASH    STX WORDFLG
06E5- E8                    INX
06E6- 8E 2A 06              STX BLNKFLG
06E9- 4C 75 06              JMP CHKLINE    ;CONTINUE THE LINE
                 ;
06EC- AE 2A 06   WORDCOUNT  LDX BLNKFLG    ;FOUND END OF WORD ?
06EF- D0 84                 BNE CHKLINE    ;CONTINUE THE LINE
                 ;
06F1- AE 2B 06   DASHCOUNT  LDX WORDFLG    ;WERE WE ON A WORD ?
06F4- F0 08                 BEQ NOTYET
                 ;
06F6- EE 26 06              INC LINETOT    ;COUNT A WORD
06F9- A2 00                 LDX #0
06FB- 8E 2B 06              STX WORDFLG
```

```
                  ;
06FE- A2 FF       NOTYET      LDX #$FF      ;MARK THE BLANK
0700- 8E 2A 06                STX BLNKFLG
0703- 4C 75 06                JMP CHKLINE   ;CONTINUE THE LINE
                  ;
0706- AE 2C 06    GET         LDX CURCHAR   ;GET A CHARACTER
0709- 8E 2D 06                STX LASTCHAR
070C- 20 E4 FF                JSR GETCHR
070F- D0 02                   BNE NONZERO
                  ;
0711- 09 40                   ORA #%01000000
0713- 8D 2C 06    NONZERO     STA CURCHAR
                  ;
0716- A5 96                   LDA *ST       ;END OF TEXT ?
0718- F0 06                   BEQ OK
                  ;
071A- 20 CC FF                JSR CLRCHN    ;RESTORE NORMAL I/O DEVS.
071D- A2 F8                   LDX #$F8      ;RESTORE STACK AND
071F- 9A                      TXS  ;GO BACK TO BASIC
                  ;
0720- 60          OK          RTS
                  ;
0721- AE 28 06    GETREST     LDX LINLEN    ;IGNORE REST OF LINE
0724- CA                      DEX
0725- 86 B4       LOOP2       STX *SAVX     ;KEEP R(X)
0727- 20 06 07                JSR GET
072A- A6 B4                   LDX *SAVX
072C- CA                      DEX
072D- EC 27 06                CPX CHARTOT
0730- B0 F3                   BCS LOOP2
                  ;
0732- 60                      RTS
                  ;
```

```
0207            .EJ
0208            .EN
```

LABEL FILE: [/ = EXTERNAL]

```
/CHANNEL=0002         /ENDLIN=001F          /BLANK=0020
/LENGTH=0028          /DASH=002D            /SHFSPC=0060
/FORSPC=006F          /FORCMD=007A          /ST=0096
/SAVX=00B4            /SCREEN=8000          /IMAGES=8400
/SETINP=FFC6          /CLRCHN=FFCC          /WRT=FFD2
/GETCHR=FFE4          WORDTOT=0624          LINETOT=0626
CHARTOT=0627          LINLEN=0628           LINFLG=0629
BLNKFLG=062A          WORDFLG=062B          CURCHAR=062C
LASTCHAR=062D         START=062E            LOOP=0632
SETLEN=0638           EIGHTY=0646           FORTY=0647
SETCHN=064A           LOADADR=064F          LINESTRT=0655
ADDLINE=0660          CHKLINE=0675          FORMAT=068F
FINISH=0692           NOTFORMAT=0698        DONELINE=06A4
MORE=06A7             LEADBLK=06AC          COUNT=06C2
NOTLEAD=06C5          CONTLIN=06CD          NOTDASH=06E2
WORDCOUNT=06EC        DASHCOUNT=06F1        NOTYET=06FE
GET=0706              NONZERO=0713          OK=0720
GETREST=0721          LOOP2=0725
//0000,0733,0733
```

The battle between BASIC and PASCAL may soon be over. Enter **COMAL**, a new programming language that combines the best of both. **COMAL** gives you the ease of BASIC, along with the power and structures of PASCAL.

Commodore has generously placed CBM COMAL in the public domain. Feel free to make disk copies for your friends or include it in a User Group Program Library.

If you are interested in **COMAL**, keep us in mind. We are the **COMAL USERS GROUP**. We keep you in mind with a Program Exchange, Newsletter, COMAL Manual, COMAL Handbook, and COMAL Reference Card. If you would like to be a COMAL PIONEER, we have a complete deluxe COMAL STARTER KIT that provides you with everything you need now, and includes a subscription to our newsletter, the COMAL CATALYST, as well as one year of free updates to the Manual and Handbook. Here is what you get:

* CBM COMAL Interpretter
  FULL version (only 5K free in a 32K PET/CBM)
  SPLIT version (INPUT and EXECUTE modules, each 16K long)
* USER GROUP DISK #1 (Introduction to COMAL)
* HELP DISK (includes 90 sample programs, one for almost every COMAL Keyword - with an automatic loading MENU)
* CBM COMAL Manual
* One year free updates to the manual
* COMAL Handbook
* One year free updates to the handbook
* COMAL Reference Card
* Subscription to the COMAL CATALYST Newsletter
* Plastic diskette sleeve that holds two diskettes

This comes neatly packaged in our custom padded 3 ring binder, with a notepad included for your notes as you use COMAL. We ship the kit in a box we had specially made to provide safe shipping.

The complete COMAL STARTER KIT costs only $47.50 (plus $2.00 shipping within the US, $4.50 shipping to Canada & Mexico, $7.00 Air Mail shipping elsewhere). Or we provide an 18 page COMAL INFORMATION PACKET for FREE if you send us a large business size envelope, self addressed with 40 cents postage.

COMAL USERS GROUP

5501 Groveland Terrace, Madison, WI 53716 U.S.A.

SX-100 IEEE Modem Software: A Review

Don White
Nepean, Ontario

The SX-100 IEEE Modem Software was written by Eugene Fisher, designer of the Livermore STAR Modem and, as marketed under another cover, the CBM 8010 Modem. Gene is also co-author of 'PET and the IEEE-488 Bus (GPIB)'. The package is marketed by ECX Company, 2678 North Main Street #6, Walnut Creek, CA 94596.

According to the advertising, the SX-100 software offers the following features:

1. Menu driven
2. Communications mode
3. Save communications to disk
4. PET to ASCII conversions
5. Save communications to printer
6. Business keyboard conversions
7. 40 or 80 character PET/CBM compatible
8. Full/half duplex operation
9. Receive files to disk(prg/seq)
10. Line verification before transmission, protocol
11. Disk directory handler
12. Automatic file creation for text storage
13. File playback for off-line viewing
14. Automatic talker/listener syncronization
15. WordPro III or IV compatible
16. Control operation: formfeed, linefeed, tab, backspace, delete, escape, break, bell, etc.

The program requires a 16K PET/CBM. When run, it lowers the top-of-memory and pokes a machine language routine into memory and then requests you to input the date. Following this you are presented with a menu of seven options.

C Start Communication
D Directory Listing
K Key Function Tables
L Look At Disk File
Q Quit
R Receive To Disk
T Transmit To Disk

The 'Start Communication' option allows you to use the PET as a terminal to communicate with another system. In this mode it will be possible to use the printer to retain a hardcopy of the session if data is not being input too quickly from the other computer, i.e. if the data being transmitted is being typed into the transmitting computer. While in the communications mode you can activate the disk log. You will be prompted for the drive number and then a SEQ file will be opened under a name created using the date (ex: MODEM81-11-11.A). This file can be closed at any time or it will be closed automatically on returning to the menu. Subsequent files can be opened during the same session. The new file name will have the last letter incremented to differentiate it from previous files.

The 'Directory Listing' option allows you to view the directories of either drives and is useful in preparation for disk-to-disk communications.

The 'Key Function Tables' is simply a help mode that informs you of the keys to push to transmit the control functions of the ASCII code to timesharing systems and bulleting boards requiring them. Special IEEE and other functions are controlled by typing a shifted number (or shift-return, number on the business keyboard).

The 'Look At Disk Files' option allows you to view the contents of program or sequential disk files. A copy of the file can be sent to the printer and the file dump can be stopped by simply typing 'end'. The dump of a PRG file is only useful in giving you and indication of what the program is about. It does not provide a program listing.

The 'Quit' option resets the top-of-memory pointers and ends the program.

The 'Receive To Disk' option allows you to receive program and sequential disk files. The program will automatically generate a file name utilizing the date or you can supply a file name. The routine uses a handshake routine which is only available from another SX-100 program. If you indicate that the transmitting program is the SX-100 then nothing will be sent to the disk until the proper link has been established. Otherwise, everything received is sent to the disk.

Finally, the 'Transmit To Disk' routine is the companion option to 'Receive To Disk'. If connection with the other computer cannot be established, you can exit this routine by pressing any key on the keyboard. If you indicate that the receiving computer is operating under an SX-100 program, nothing will be sent until the proper handshake has taken place. If the receiving program is not the SX-100 then transmission will begin immediately. Exit from the routine is automatic once transmission is completed.

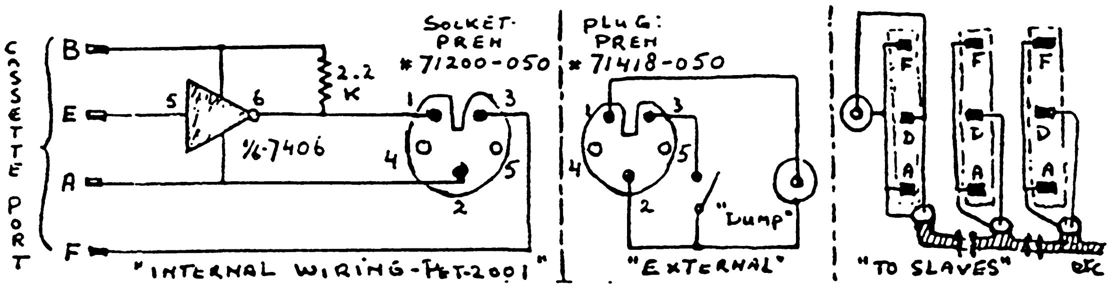
I have used this program for a number of weeks now and it seems to operate as described. It also appears to be 'bug-free'. I only have two complaints. Firstly, no cursor is displayed when in the communications mode and secondly, there is no routine included to handle parity and this has prevented me from communicating with some time-sharing systems. However, for anyone requiring the capability to easily transmit and receive files between PETs I would recommend this program.

The major drawback is the price, $79.95(US) - NOT $49.95 as advertised in COMPUTE!. There is also a 5% service charge if you use VISA and a $1.50 shipping charge. By the time I received the VISA statement I was committed to $102.75 (Canadian). The choice is yours.

DUMP-MATE Ted Evers, Toronto

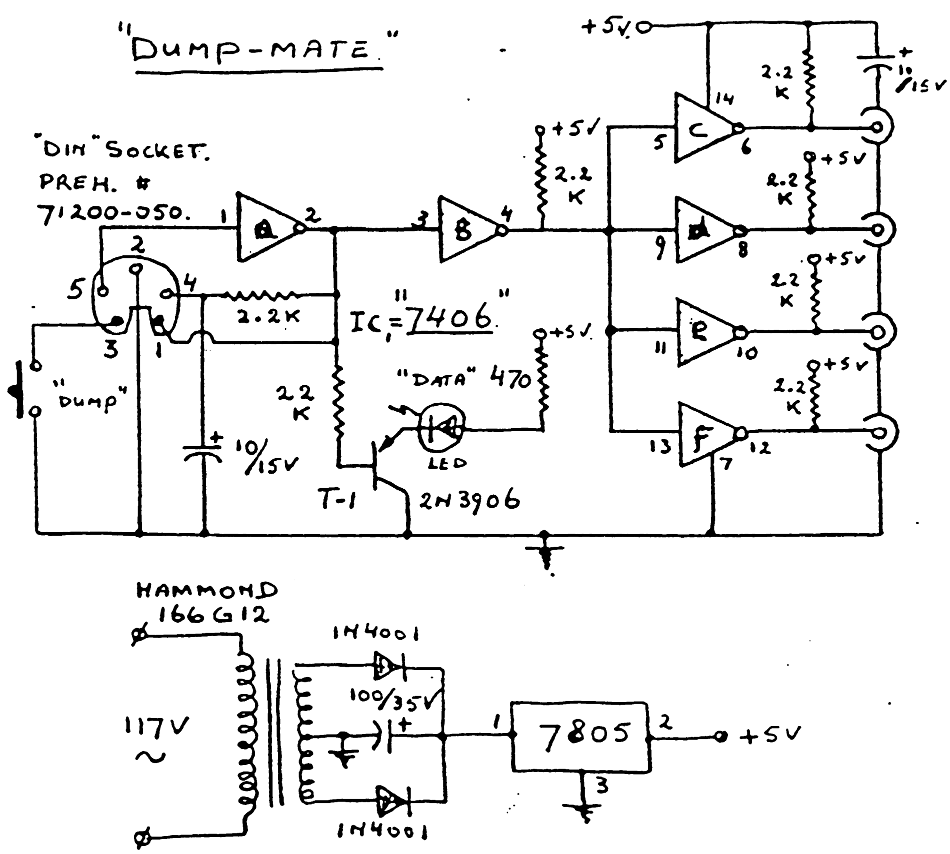
A multi-load system for use with Commodore PET/CBMs.

As mentioned in a previous article, the original multi-load system was part of our AV-8101 video-audio interface for the Commodore 2000 series computers, as shown below.

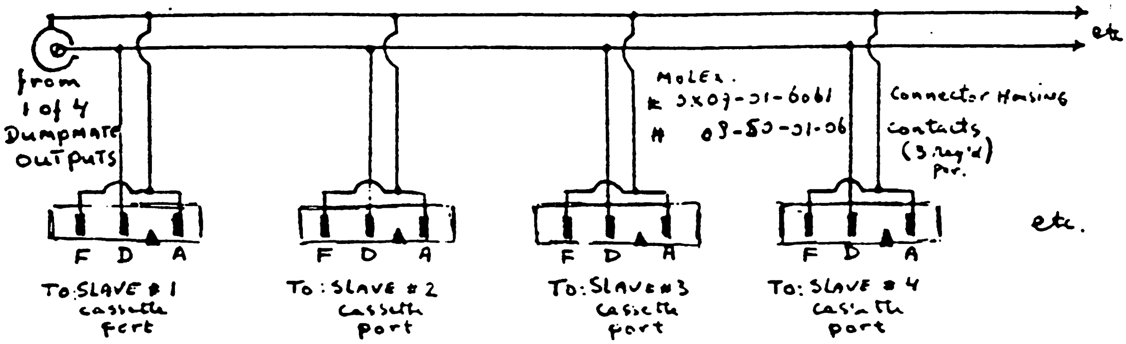


By means of the spare inverter-driver on this board, programs could be dumped from the master computer to about twenty slave units. In order to increase its capability to load programs to up to sixty slaves, when so required, the first "Dump-Mate", a multi-output driver, was built.

However, with the introduction of the Commodore 8032 and 4032 (12" screen), the multi-load system used in the 2001 was no longer possible, as all six inverters of the 7406 I.C. were now required for the video interface. This problem was overcome by the redesign of the "Dump-Mate" into a self-contained, external type multi-loader.

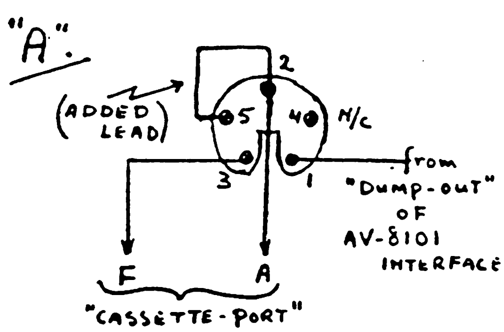


Each of the four outputs can be connected to up to twenty "slave" computers by means of the cassette-ports interface assembly shown below.



Connection between the input of the Dump-Mate and the output of the master computer is made by a short length of five-conductor cable with "DIN" plugs (PREH #71418-50) on both ends.

The output socket at the computer end is wired as per diagram below:



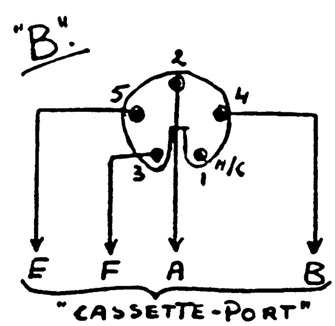
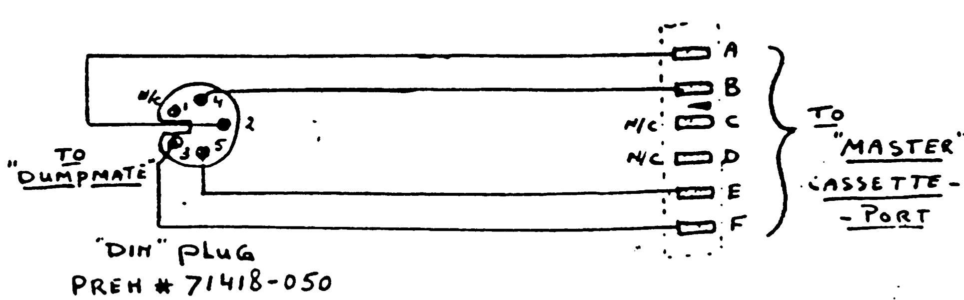


Figure "A" is used for PET 2001 series with the AV-8101 interface and dump circuit, while figure "B" is the wiring required for use with the regular 2000, 4000 and 8000 series computers.

Another way of connecting the Dump-Mate to the computer is shown below:



In this manner, any PET computer can be utilized as the master unit, however, the cassette port will not be available for program loading.

The following is a short "how to" guide:

1. Be sure that the power to all equipment is OFF before connecting or disconnecting cables.

2. When everything is in place, switch on all units, including the Dump-Mate.

3. LOAD a program into the master computer.

4. The slave computers requiring this program should now type:

   NEW <return>

   LOAD <return>

5. The monitors of these units should now show:

   SEARCHING

6. On the master unit, type:

   SAVE "name" <return>

7. Push the "dump" switch.

8. After about seven seconds, the "data" light will go off and the slave monitors will show:

   FOUND "name"
   LOADING

9. Push the "dump" switch again.

10. The "data" light will stay on until the program is loaded, at which time READY. and flashing cursor should appear on all monitors.

11. Typing RUN <return> will execute the program.

Construction

Although the cicuit is simple enough to use direct point-to-point wiring, for convenience sake. However, our unit was built on two 2 1/2" x 1 3/4" printed circuit boards, mounted back-to-back on a "U"-bracket.

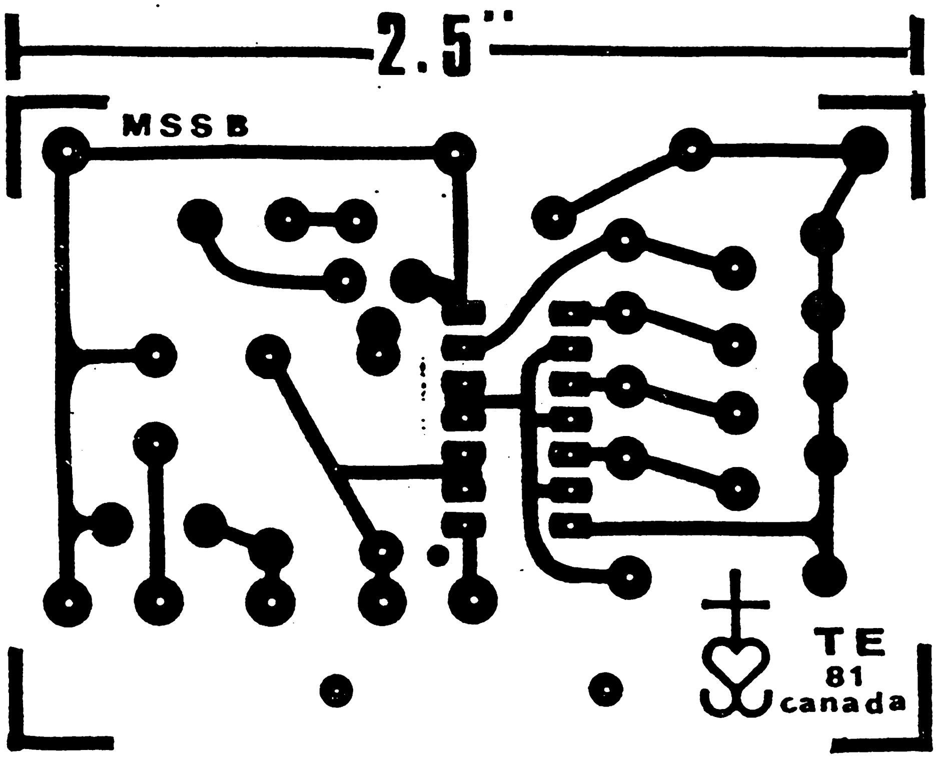
Etching and drilling guides, with a components placement diagram has been included.

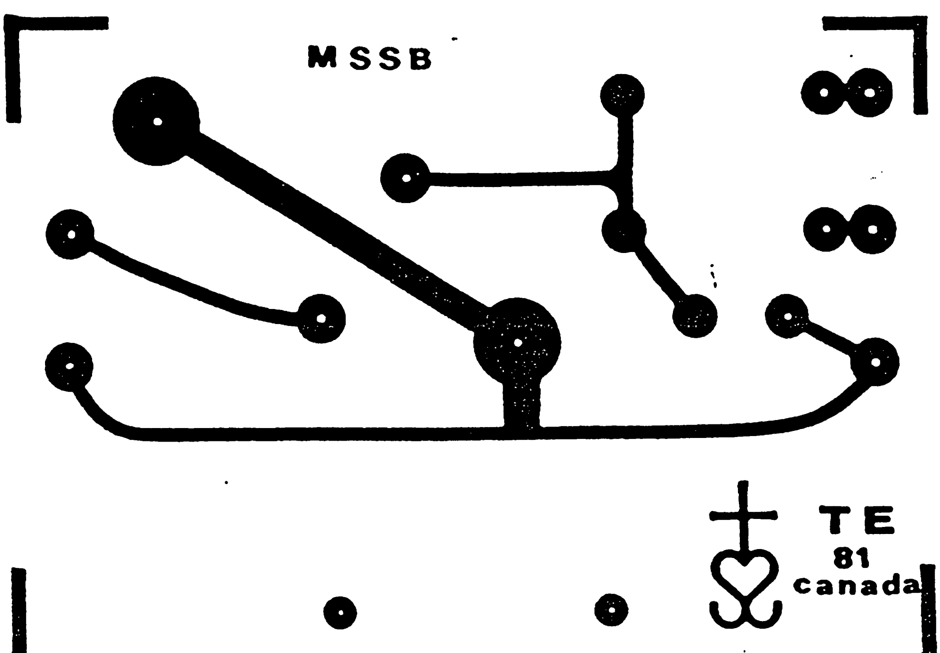
DUMP-MATE Suggested Parts List

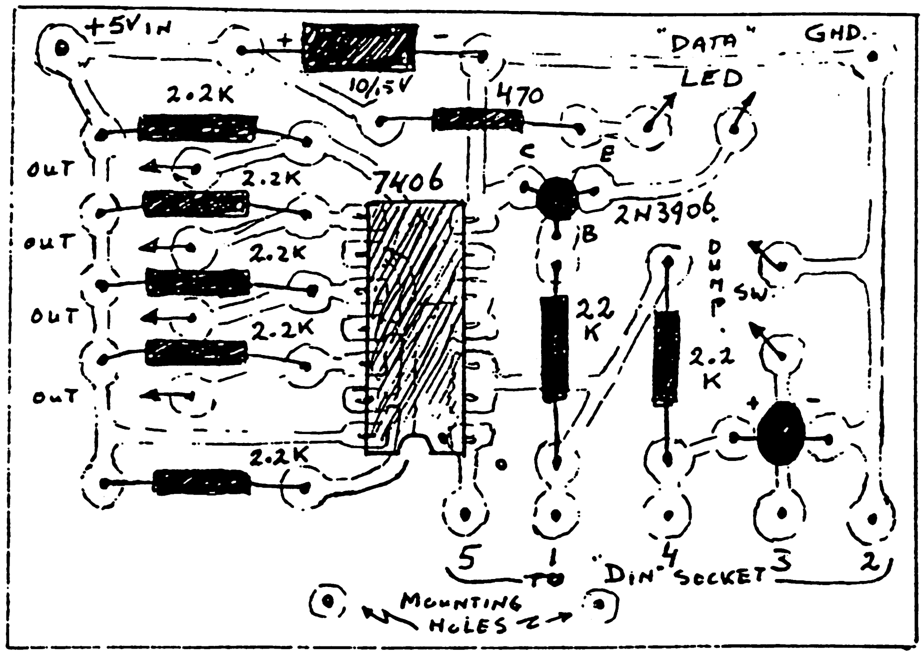
1 - Hammond #1454G Case
1 - Hammond #166G12 Transformer
1 - Preh #71200-050 Socket
4 - Switchcraft #3501-FP Connectors
1 - N/O pushbutton - Grayhill
1 - L.E.D. Mount
1 - 3-wire AC Cord Assy.
1 - AC Cord Retainer - Heyco
2 - Marrette Connectors
1 - "U" Bracket
1 - 7406 IC
1 - 7805 Regulator (TO-3 pkg)
1 - 2N3906
1 - L.E.D.
2 - 1N4001 diode
1 - 10 microfd 15v Tant. Cap.
1 - 10 microfd 15v Elco
1 - 100 microfd 35v Elco
1 - 470 ohm resistor
6 - 2200 ohm resistor
1 - 22 K-ohm resistor
- Miscellaneous Mounting Hardware

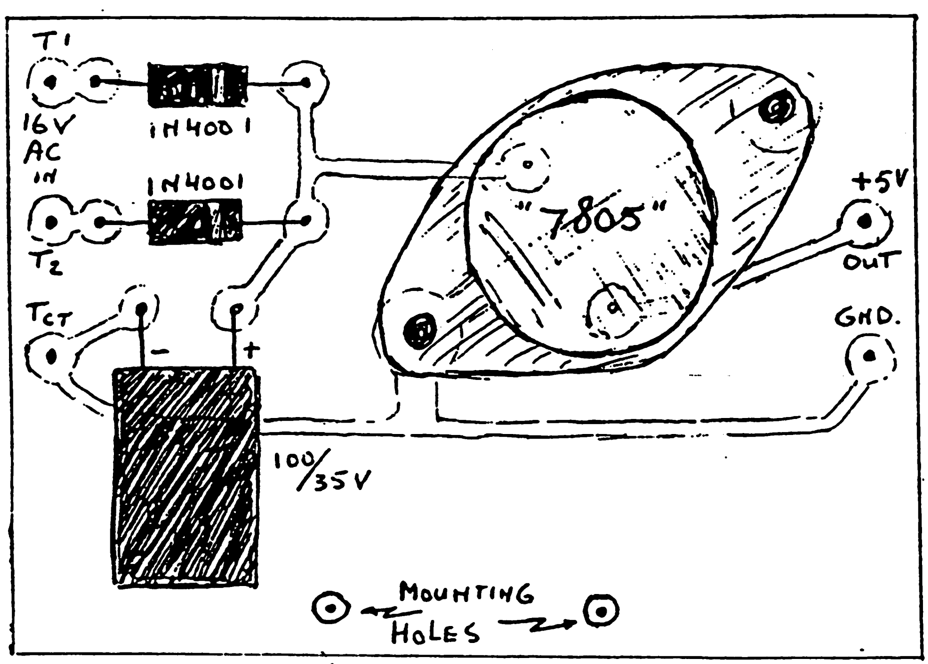
Editor's Note

Dump-Mate was built originally for PET/CBMs, but it will no doubt work with the VIC-20 since the cassette interface is identical to the PETs.


2.5"
MSSB
TE
81
canada


MSSB
TE
81
canada


+5V IN
10/15V
"DATA" LED
GND.
2.2K
470
OUT
7406
C
E
2N3906
B
22 K
DUMP SW.
2.2 K
5
1
4
3
2
TO "DIN" SOCKET
MOUNTING HOLES


T1
16V AC IN
1N4001
T2
TCT
"7805"
+5V OUT
GND.
100/35V
MOUNTING HOLES

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
0	00		@	end-line	BRK	0
1	01		A		ORA(I,X)	1
2	02		B			2
3	03		C			3
4	04		D			4
5	05	white	E		ORA Z	5
6	06		F		ASL Z	6
7	07	bell	G			7
8	08	lock	H		PHP	8
9	09	unlock	I		ORA #	9
10	0A		J		ASL A	10
11	0B		K			11
12	0C		L			12
13	0D	car ret	M		ORA	13
14	0E	text	N		ASL	14
15	0F	top-left	O			15
16	10		P		BPL	16
17	11	cur down	Q		ORA(I),Y	17
18	12	reverse	R			18
19	13	cur home	S			19
20	14	delete	T			20
21	15	del.line	U		ORA Z,X	21
22	16	er.begin	V		ASL Z,X	22
23	17		W			23
24	18		X		CLC	24
25	19	scr.up	Y		ORA Y	25
26	1A		Z			26
27	1B		[			27
28	1C	red	\			28
29	1D	cur right	]		ORA X	29
30	1E	green	↑		ASL X	30
31	1F	blue	←			31
32	20	space	space	space	JSR	32
33	21	!	!	!	AND(I,X)	33
34	22	"	"	"		34
35	23	#	#	#		35
36	24	$	$	$	BIT Z	36
37	25	%	%	%	AND Z	37
38	26	&	&	&	ROL Z	38
39	27	'	'	'		39
40	28	(	(	(	PLP	40
41	29	)	)	)	AND #	41
42	2A	*	*	*	ROL A	42
43	2B	+	+	+		43
44	2C	,	,	,	BIT	44
45	2D	-	-	-	AND	45
46	2E	.	.	.	ROL	46
47	2F	/	/	/		47
48	30	0	0	0	BMI	48
49	31	1	1	1	AND(I),Y	49
50	32	2	2	2		50
51	33	3	3	3		51
52	34	4	4	4		52
53	35	5	5	5	AND Z,X	53
54	36	6	6	6	ROL Z,X	54
55	37	7	7	7		55
56	38	8	8	8	SEC	56
57	39	9	9	9	AND Y	57
58	3A	:	:	:		58
59	3B	;	;	;		59
60	3C	<	<	<		60
61	3D	=	=	=	AND X	61
62	3E	>	>	>	ROL X	62
63	3F	?	?	?		63
64	40	@		@	RTI	64
65	41	A		A	EOR(I,X)	65
66	42	B		B		66
67	43	C		C		67
68	44	D		D		68
69	45	E		E	EOR Z	69
70	46	F		F	LSR Z	70
71	47	G		G		71
72	48	H		H	PHA	72
73	49	I		I	EOR #	73
74	4A	J		J	LSR A	74
75	4B	K		K		75
76	4C	L		L	JMP	76
77	4D	M		M	EOR	77
78	4E	N		N	LSR	78
79	4F	O		O		79
80	50	P		P	BVC	80
81	51	Q		Q	EOR(I),Y	81
82	52	R		R		82
83	53	S		S		83
84	54	T		T		84
85	55	U		U	EOR Z,X	85
86	56	V		V	LSR Z,X	86
87	57	W		W		87
88	58	X		X	CLI	88
89	59	Y		Y	EOR Y	89
90	5A	Z		Z		90
91	5B	[		[		91
92	5C	\		\		92
93	5D	]		]	EOR X	93
94	5E	↑		↑	LSR X	94
95	5F	←		←		95
96	60				RTS	96
97	61				ADC(I,X)	97
98	62					98
99	63					99
100	64					100
101	65				ADC Z	101
102	66				ROR Z	102
103	67					103
104	68				PLA	104
105	69				ADC #	105
106	6A				ROR A	106
107	6B					107
108	6C				JMP(I)	108
109	6D				ADC	109
110	6E				ROR	110
111	6F					111
112	70				BVS	112
113	71				ADC(I),Y	113
114	72					114
115	73					115
116	74					116
117	75				ADC Z,X	117
118	76				ROR Z,X	118
119	77					119
120	78				SEI	120
121	79				ADC Y	121
122	7A					122
123	7B					123
124	7C					124
125	7D				ADC X	125
126	7E				ROR X	126
127	7F					127
128	80		r-@	END		128
129	81		r-A	FOR	STA(I,X)	129
130	82		r-B	NEXT		130
131	83		r-C	DATA		131
132	84		r-D	INPUT#	STY Z	132
133	85	f1	r-E	INPUT	STA Z	133
134	86	f3	r-F	DIM	STX Z	134
135	87	f5	r-G	READ		135
136	88	f7	r-H	LET	DEY	136
137	89	f2	r-I	GOTO		137
138	8A	f4	r-J	RUN	TXA	138
139	8B	f6	r-K	IF		139
140	8C	f8	r-L	RESTORE	STY	140
141	8D	car ret	r-M	GOSUB	STA	141
142	8E	graphic	r-N	RETURN	STX	142
143	8F	btm.rt	r-O	REM		143
144	90	black	r-P	STOP	BCC	144
145	91	cur up	r-Q	ON	STA(I),Y	145
146	92	rvs off	r-R	WAIT		146
147	93	clear	r-S	LOAD		147
148	94	insert	r-T	SAVE	STY Z,X	148
149	95	ins.line	r-U	VERIFY	STA Z,X	149
150	96	er.end	r-V	DEF	STX Z,Y	150
151	97		r-W	POKE		151
152	98		r-X	PRINT#	TYA	152
153	99	scr.dn	r-Y	PRINT	STA Y	153
154	9A		r-Z	CONT	TXS	154
155	9B		r-[	LIST		155
156	9C	magenta	r-\	CLR		156
157	9D	cur left	r-]	CMD	STA X	157
158	9E	yellow	r-↑	SYS		158
159	9F	cyan	r-←	OPEN		159
160	A0			CLOSE	LDY #	160
161	A1		r-!	GET	LDA(I,X)	161
162	A2		r-"	NEW	LDX #	162
163	A3		r-#	TAB(		163
164	A4		r-$	TO	LDY Z	164
165	A5		r-%	FN	LDA Z	165
166	A6		r-&	SPC(	LDX Z	166
167	A7		r-'	THEN		167
168	A8		r-(	NOT	TAY	168
169	A9		r-)	STEP	LDA #	169
170	AA		r-*	+	TAX	170
171	AB		r-+	-		171
172	AC		r-,	*	LDY	172
173	AD		r--	/	LDA	173
174	AE		r-.	↑	LDX	174
175	AF		r-/	AND		175
176	B0		r-0	OR	BCS	176
177	B1		r-1	>	LDA(I),Y	177
178	B2		r-2	=		178
179	B3		r-3	<		179
180	B4		r-4	SGN	LDY Z,X	180
181	B5		r-5	INT	LDA Z,X	181
182	B6		r-6	ABS	LDX Z,Y	182
183	B7		r-7	USR		183
184	B8		r-8	FRE	CLV	184
185	B9		r-9	POS	LDA Y	185
186	BA		r-:	SQR	TSX	186
187	BB		r-;	RND		187
188	BC		r-<	LOG	LDY X	188
189	BD		r-=	EXP	LDA X	189
190	BE		r->	COS	LDX Y	190
191	BF		r-?	SIN		191
192	C0		Reverse of ASCII	TAN	CPY #	192
193	C1	,a		ATN	CMP(I,X)	193
194	C2	,b		PEEK		194
195	C3	,c		LEN		195
196	C4	,d		STR$	CPY Z	196
197	C5	,e		VAL	CMP Z	197
198	C6	,f		ASC	DEC Z	198
199	C7	,g		CHR$		199
200	C8	,h		LEFT$	INY	200
201	C9	,i		RIGHT$	CMP #	201
202	CA	,j		MID$	DEX	202
203	CB	,k		GO		203
204	CC	,l		CONCAT	CPY	204
205	CD	,m		DOPEN	CMP	205
206	CE	,n		DCLOSE	DEC	206
207	CF	,o		RECORD		207
208	D0	,p		HEADER	BNE	208
209	D1	,q		COLLECT	CMP(I),Y	209
210	D2	,r		BACKUP		210
211	D3	,s		COPY		211
212	D4	,t		APPEND		212
213	D5	,u		DSAVE	CMP Z,X	213
214	D6	,v		DLOAD	DEC Z,X	214
215	D7	,w		CATALOG		215
216	D8	,x		RENAME	CLD	216
217	D9	,y		SCRATCH	CMP Y	217
218	DA	,z		DIRECTORY		218
219	DB					219
220	DC					220
221	DD				CMP X	221
222	DE				DEC X	222
223	DF					223
224	E0				CPX #	224
225	E1				SBC(I,X)	225
226	E2					226
227	E3					227
228	E4				CPX Z	228
229	E5				SBC Z	229
230	E6				INC Z	230
231	E7					231
232	E8				INX	232
233	E9				SBC #	233
234	EA				NOP	234
235	EB					235
236	EC				CPX	236
237	ED				SBC	237
238	EE				INC	238
239	EF					239
240	F0				BEQ	240
241	F1				SBC(I),Y	241
242	F2					242
243	F3					243
244	F4					244
245	F5				SBC Z,X	245
246	F6				INC Z,X	246
247	F7					247
248	F8				SED	248
249	F9				SBC Y	249
250	FA					250
251	FB					251
252	FC					252
253	FD				SBC X	253
254	FE				INC X	254
255	FF					255

Jim Butterfield

PET User Port Cookbook

Gregory Yob
Box 354
Palo Alto CA 94302

The PET personal computer has several expansion capabilities, including one known as the *user port*. This is a set of eight bidirectional lines and two handshake lines intended as a parallel port for the hobbyist to use in his experimental projects. Commodore has not released much information regarding the user port, and the object of this article is to explain the user port and its use.

Fig. 1 shows the location of the user port on the back of the PET and the pin-out of the PC edge. If you do not have a 12-position, 24-contact edge connector, use a larger one and cut it off to the 12-position size. If you do this, be sure to insert a polarization key in your connector; I found that it was easy to misalign a sawed-off connector with the PC edge, causing various mysterious glitches. Also, be sure that the top and bottom connections are really separate—the upper edge has a variety of signals that will interfere with the correct operation of the user port.

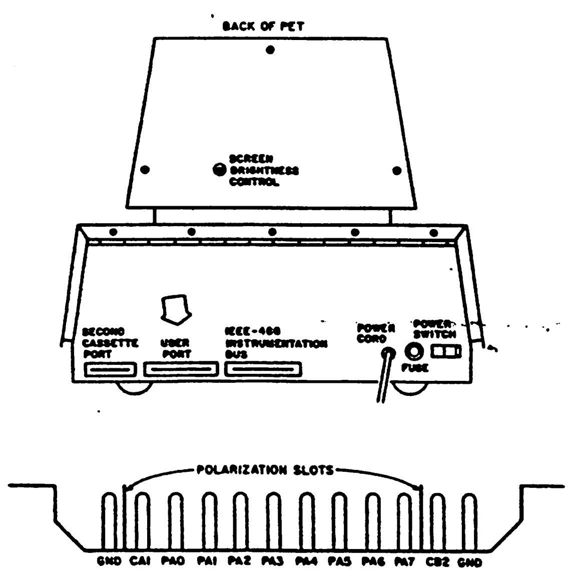


Fig. 1. The user port—location and pin-out. The user port pin-out as seen from the top. The user port pins are on the bottom of the PC card edge. The pins on top carry a variety of signals that are not related to the user port. Electrically, the lines correspond to one TTL source or load, depending on whether the line is in output or input mode. Use buffering or short cables if high data rates are required. The CB2 line does not have a pull-up resistor, so you may have to provide one if you are using CB2 in input mode.

The pin designations correspond to those on a MOS 6522 VIA (Versatile Interface Adapter), which is a complex LSI I/O chip produced by MOS Technology. (Write MOS Technology, 950 Rittenhouse Road, Norristown PA 19401, for the specification sheet.) The user port is connected directly to the VIA within the PET, and the lines are capable of sourcing or sinking *one* TTL load. If your application calls for a high data rate, note that your cables should be short or some buffering will be required.

As with all of the 650X microcomputer systems, the input and output appear to the microprocessor as a group of memory locations. PET's BASIC does not have any PRINT or INPUT statements for the user port, which requires you to use the PEEK and POKE statements. This also places another limitation, that is, BASIC's speed, which limits I/O through the user port to around 50 characters per second. If you want to use a more rapid rate, you must use machine language.

Since this article is concerned with the mechanics of using the user port, most of the examples will be in BASIC. Table 1 shows the memory locations for the 6522 in the PET.

At this point I must warn you: all of the other VIA lines are used within the PET for internal uses. If you fail to restore the VIA to its original state when you are finished, you will find that the PET behaves strangely, especially when dealing with the tape drives.

When I wrote the program for display of the VIA registers (which you will see later on), I didn't save it until I had it debugged. The PET wouldn't verify or even find the copy I had tried to save, and after handwriting the program, I realized the next morning that the VIA registers were not in their original states. Fortunately I had left the PET on overnight, and when I restored the registers, I was able to save the program.

The Blinkin' Lights Machine

For experimentation with the user port it is convenient to build a miniature "front panel" to indicate the state of each line and to control the lines via manual switches. A breadboard and some $20 worth of parts (bought at the local costly retail outlet) provided a handy "Blinkin' Lights Machine" that hooked to the user port and used the +5 volt supply from the second cassette drive.

```
10  REM SIMPLE OUTPUT EXAMPLE
20  REM SET DATA DIRECTION REGISTER TO OUTPUT
30  POKE 59459,255
40  REM COUNT FROM 0 TO 255
50  FOR J=0 TO 255
60  REM POKE TO OUTPUT REGISTER
70  POKE 59471,J
80  NEXT J
90  REM DO IT AGAIN
100 GOTO 50
```

Example 1. Simple output example for user port.

Name	Address(hex)	Address(decimal)	Function
ORB	E840	59456	## (internal to PET)
ORA	E841	59457	Data with Handshake
PDRB	E842	59458	##
DDRA	E843	59459	Data Direction
T1L-W	E844	59460	##
T1C-H	E845	59461	##
T1L-L	E846	59462	##
T1L-H	E847	59463	##
T2L-W	E848	59464	##
T2C-H	E849	59465	##
SR	E84A	59466	Shift Register
ACR	E84B	59467	Auxiliary Control
PCR	E84C	59468	Peripheral Control
IFR	E84D	59469	Interrupt Flags
IER	E84E	59470	Interrupt Enable
ORA	E84F	59471	Data (no handshake)

Table 1. PET VIA register addresses. The named registers may be used to work with the user port. Some of the settings used may disable other PET functions, such as tape I/O, so you should restore the original settings when you are done. The registers with "##" in the Function column are used internally by the PET. If you are bold, there are two other I/O chips in the PET. These are MOS 6520s, with one starting at $E810 (59408) for internal uses and one at $E820 (59425) for the IEEE-844 bus.

Note that the circuit draws 200 mA, which is close to the maximum you can steal from the PET. If you have other PET extensions that use the PET supply, power the Blinkin' Lights externally.

Fig. 2 shows the circuit for the Blinkin' Lights Machine. The extra inverter and capacitor on the CB2 line are for an audio output to attach to your hi-fi set for some simple music making. One of the best ways to build this device is on a Vector breadboard, which has the fingers for an edge connector. This permits putting the Blinkin' Lights in series with a device under test to help with debugging the interface software and hardware.

Most of the examples shown below make use of the Blinkin' Lights Machine, so building one might be handy.

```
10 POKE 59459,255
20 K=1
30 POKE 59471,K
40 FOR J=1 TO 200 : NEXT
50 K=K*2
60 IF K=256 THEN 20
70 GOTO 30
```

Example 2. Another simple output example.

Simple Output

The simplest thing to do is output bytes to the user port. To do this, you must first set the Data Direction register to 255 (all bits set) and then set the Output register to the byte(s) that are to be output. Example 1 is a short program that counts from 0 to 255 and outputs the count to the user port.

The Data Direction register controls the PA0 through PA7 lines' data direction. If the bit is set for a given line (i.e., bit 0 is for line PA0), the line will be an output. If the bit is zero, the line will be an input.

When the PET is turned on with the Blinkin' Lights attached, all the LEDs will be lit. The PA0-PA7 lines are initially set for input, and the Blinkin' Lights will see lines in the high-impedance state as "high" (pulled up by the 7404s), turning on the LED for the line.

When the program (Example 1) is RUN, the data lines show that a binary count appears, which cycles through about once every three seconds. To slow the rate down so that the least significant bits (PA0 and PA1) will change state, add:

```
65 FOR K=1 TO 50 : NEXT
```

This will slow the counting loop down to around 10 Hz.

To see the effect of changing the Data Direction register, change line 30 to:

```
30 POKE 59459,15
```

Now the lines PA0-PA3 will count, and lines PA4-PA7 will remain lit (recall that an unconnected line will float to high with the Blinkin' Lights).

Example 2 shows another short program. Try it and see what it does! Note that in PET BASIC the NEXT statement may omit the loop counter if the innermost loop is being terminated. Another diversion is to change the program in Example 2.

```
20 K=1 : L=128
30 POKE 59471, K OR L
50 K=K*2 : L=L/2
```

(Just change these lines and let the others remain the same.)

Simple Input

To see simple input, POKE the Data Direction register to input mode and connect the switches to the PA0-7 lines. Note that the Blinkin' Lights has some DIP switches to isolate the manual switches from the data lines. This is because if they were always tied in, the switch setting would force the line to the switch's state.

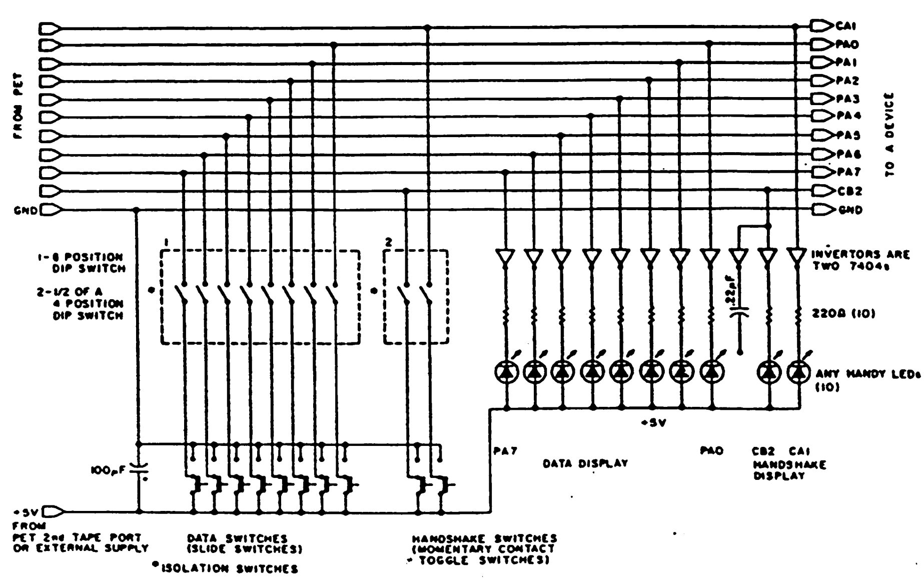


Fig. 2. Blinkin' Lights—PET user port switch register and indicator.

Shown	What It Represents
b	SPACE character (when not clear)
[□]	A lowercase character in a square box represents the corresponding graphics character. For example, [a] is the spade graphics character, or SHIFT-A:
Ⓒ	Clear Screen
Ⓗ	Home Cursor
Ⓤ	Cursor Up
Ⓓ	Cursor Down
Ⓡ	Cursor Right
Ⓛ	Cursor Left
Ⓘ	INST key
Ⓓ	DEL key

Table 2. PET program listing special characters.

Data Register	DATA	59471
Data Register, Handshake	HDATA	59457
Peripheral Control Register	PCR	59468
Auxiliary Control Register	ACR	59467
Interrupt Flag Register	IFR	59469

Table 3.

Then, PEEK the Data register and display the result on the PET display screen in a loop. As you change the switches, the number displayed will change. Example 3 is a program that does this. (Note: Table 2 shows how this article represents PET listings.) Line 70 homes the cursor and prints the value of the Data register. It then prints a CURSOR LEFT and three blanks. The reason for the CURSOR LEFT is that the PET has an oddity when it prints numbers onto the screen. When a number is printed, the format is: (SPACE or +)(Digits of Number)(CURSOR RIGHT).

When a short number is printed over a longer one, the printing stops after the CURSOR RIGHT. It is necessary to erase the old numbers with some blanks, so the cursor is moved left once and three blanks are printed. This prevents spurious numbers, such as "328," appearing on the display. (Try it, you won't like it!)

RUN this program and try the manual switches one at a time. You should see the sequence 0, 1, 2, 4, 8 . . . 128 appear on the PET screen.

If you set all the manual switches to zero and disconnect one of them with the DIP switch, the line will go high and the PET will see the bit as set. Be careful of this when you are using the Blinkin' Lights for debugging.

Joysticks

A simple and enjoyable way to use the user port is to attach a switch-operated pair of joysticks to your PET. Each joystick has four switches—one for each direction—that are closed when the stick is pointed that way. Fig. 3 shows a joystick circuit.

The program in Example 4 sets up the screen with a solid and hollow ball. Each joystick controls one of the balls, and both balls may be in motion at the same time. The switches and bit settings are the same as in Fig. 3.

Lines 170 and 180 clear the screen and print the character for the right and left joysticks. The PEEK sets the cursors (C1 and C2) to the value needed for use by POKE later. The value 32768 is the first address in memory in the display, which occupies memory locations 32768 to 33767.

Line 260 fetches the data from the user port. Since the joysticks ground the lines to indicate switch closures, the byie is complemented. It is then ANDed with 255 to return to eight bits, as the integer operations of the PET are 2's complement for 16 bits.

In Line 2010, the value for Z must be shifted right by four bits. This is done by dividing by 16 and truncating.

Lines 3020 and 3140 place a blank and the cursor, respectively, on the screen. The multiplication by 40 for Y is because the PET screen is 40 characters wide. If you delete line 3020, the motions of the joysticks will leave trails and let you draw pictures.

Transferring Data with Handshakes

The CA1 and CB2 lines permit data transfer with full handshaking for input and output. The 6522 VIA has a variety of options, and these are controlled by the registers in Table 3. In the 6522, the Peripheral Control register and the Auxiliary Control register select the various options for the operational modes for the VIA. Some of these bits affect the CA1 and CB2 lines and will be described in detail later.

The Interrupt Flag register has bits for the detection of several conditions that may be used for interrupts. In the PET, the use of the interrupts is a hazardous affair, as the PET has a 60 Hz internal interrupt, which handles various housekeeping tasks such as scanning the keyboard and maintaining the internal clock. Since these functions can only be handled in machine language, this article will not discuss how to handle the Interrupt Enable register.

To detect a condition, such as the transition of the CA1 line, PEEK the Interrupt Flag register and AND for the desired bit. The bit in the Flag register will remain set until other actions are taken, usually the reading or writing of data through the Data Handshake register.

If the above sounds confusing, that is because it *is* confusing, and with this in mind, you should attempt the examples in the following sections when you try to use the PET user port.

Using CA1

The CA1 line is an input-only line usually used to detect the handshakes for data transfers. For example, if a device is send-

```
10  REM SIMPLE INPUT EXAMPLE
20  REM SET DATA DIRECTION TO INPUT
30  POKE 59459,0
40  REM CLEAR SCREEN
50  PRINT" Ⓒ ";
60  REM PEEK DATA REGISTER & SHOW IT
70  PRINT" Ⓗ "PEEK(59471)" Ⓛ bbb";
80  REM DO IT AGAIN
90  GOTO 70
```

Example 3. Simple input example for user port.

```
5 REM BY GREGORY YOB, MAY 1978
10 REM DUAL CURSORS FOR JOY-STICKS
20 REM ATTACHED TO USER PORT WITH
30 REM BITS AS FOLLOWS:
40 REM LINE GROUNDED MEANS SWITCH IS
50 REM CLOSED AND TO MOVE CURSOR
60 REM  BIT 7 = LEFT STICK  UP
70 REM   "  6 =             DOWN
80 REM   "  5 =             RIGHT
90 REM   "  4 =             LEFT
100 REM  "  3 = RIGHT STICK UP
110 REM  "  2 =             DOWN
120 REM  "  1 =             RIGHT
130 REM  "  0 =             LEFT
140 REM DISPLAY IS WRAPAROUND
150 REM
160 REM PUT YOUR OWN CURSORS HERE
170 PRINT"Ⓒ [q]";:C1=PEEK(32768)
180 PRINT"Ⓒ [w]";:C2=PEEK(32768)
190 REM INITIALIZE SCREEN & POSITIONS
200 PRINT"Ⓒ";
210 X1=4:Y1=12:X2=35:Y2=12
220 POKE 33252,C1:POKE 33283,C2
230 REM SET UP DATA DIRECTION REG
240 POKE 59459,0
250 REM LOOK AT PORT
260 P=NOT(PEEK(59471))AND 255
270 REM CHECK RIGHT & LEFT
280 IF P AND 15 THEN GOSUB 1000
290 IF P AND 240 THEN GOSUB 2000
300 GOTO 260
500 REM ROUTINES 1000 & 2000 SET UP
510 REM X,Y = POSITION
520 REM Z = SWITCH SETTINGS
530 REM C = CURSOR CHARACTER
540 REM FOR ROUTINE 3000 WHICH
550 REM DOES MOVING & WRAPAROUND
560 REM
1000 REM RIGHT STICK
1010 X=X1:Y=Y1:Z=P AND 15:C=C1
1020 GOSUB 3000
1030 X1=X:Y1=Y:RETURN
2000 REM LEFT STICK
2010 X=X2:Y=Y2:Z=INT((P AND 240)/16)
2020 C=C2:GOSUB 3000
2030 X2=X:Y2=Y:RETURN
2500 REM
3000 REM MOVE CURSOR
3010 REM ERASE OLD ONE
3020 POKE 32768+40*Y+X,32
3030 REM FIND NEW POSITION
3040 IF Z AND 8 THEN Y=Y-1
3050 IF Z AND 4 THEN Y=Y+1
3060 IF Z AND 2 THEN X=X+1
3070 IF Z AND 1 THEN X=X-1
3080 REM WRAPAROUND CHECK
3090 IF X>39 THEN X=0
3100 IF X<0  THEN X=39
3110 IF Y>24 THEN Y=0
3120 IF Y<0  THEN Y=24
3130 REM POKE IN NEW CURSOR
3140 POKE 32768+40*Y+X,C
3150 RETURN
```

Example 4. Program to move two cursors with the joysticks in Fig. 3.

ing data to the PET, the CA1 line will be used to say that the data is now valid. If the PET is sending data, the CA1 line is used by the device to signal that it is ready for the data.

Using the CA1 line involves these steps:

1. Select the options you want and POKE the Peripheral Control register (PCR) and Auxiliary Control register (ACR) accordingly.

2. In a loop, check the CA1 Flag bit in the Interrupt Flag register (IFR) until it is set.

3. PEEK or POKE the HDATA (Data with Handshake) register with the data. This will reset the CA1 bit in the IFR.

Your options are as follows:

1. *Positive or negative transition.* CA1 will set its flag bit when the line goes high or low, depending on bit 1 in the PCR.

For a *negative* transition, use:

```
POKE (59468), PEEK(59468)AND 254
```

This is the value the PET initializes to when it is powered up. The reason it uses a PEEK instead of just POKEing to a 1 is that the other bits in the PCR should not be changed because they control other things.

For a *positive* transition, use:

```
POKE (59468), PEEK(59468)OR 1
```

2. *Latching of the input data.* If the input data is latched, the values present on the data lines will be latched when the CA1 line makes the correct transition. If the data is not latched, the values in the HDATA register will change as the data lines change. It is safest to use the latched mode when handshaking your data.

To enable latching, use this statement:

```
POKE (59467), PEEK(59467)OR 1
```

To disable latching, use:

```
POKE (59467), PEEK(59467) AND 254
```

To detect the Flag bit in the IFR, use a statement of the form:

```
IF PEEK(59469)AND 2 THEN ———
```

or

```
WAIT 59469,2
```

If you use the WAIT statement, note that the STOP key will be ignored by the PET, which means you must be *sure* that the CA1 line will make a transition—otherwise your PET will be hung up. For debugging, use the IF-THEN form. For reading or writing the HDATA register use:

```
PEEK (59457)
```

or

```
POKE 59457, ———
```

At last it is time for some examples. First, let's try counting from 0 to 255, with a wait for the CA1 line to be toggled before the next value is sent to the user port. Enter the program in Example 5, recalling Example 1.

When this program is run, the data lights will go out and will stay out until the CA1 switch is toggled. (If it doesn't, be sure that your DIP swich has been closed for CA1.) The first light (PA0) will then light, and as you toggle the CA1 swich, the Blinkin' Lights will count in binary.

Two things should be noted. First, the bounce of the CA1 switch will guarantee that *both* transitions occur, so the setting of the transition bit doesn't matter. Also, the speed of BASIC is slow enough that the bounce of CA1 doesn't cause double or more rapid counts. (If you try the equivalent program in machine language, your CA1 will count 10 to 25 times each time you flick the switch unless you have debounced it.)

Second, you can shorten your program by using the inverse condition in line 110, eliminating line 120:

```
110 IF(PEEK(59469)AND 2)=0THEN 110
```

Beware of the precedence of operators. If you tried:

```
110 IF PEEK(59469) AND 2=0THEN 110
```

your lights would have counted up ignoring the CA1 line. The reason for this is that the operator = is evaluated *before* AND is. So, the sub-expression 2=0 is evaluated, giving a −1, which is ANDed with the IFR with the result that *any* bit will make the relation true. In this case, no other bits are set; the program then thinks that the CA1 line had toggled; and it drops through the loop.

Try it out—this error is quite common, and that's the reason

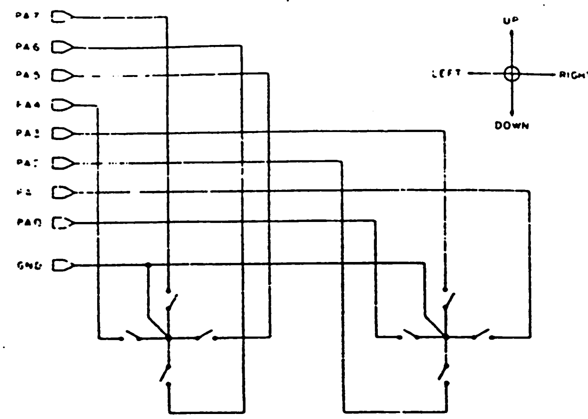


Fig. 3a. Joysticks for the PET. The switch arrangement for my PET joysticks is shown here. The switches are normally open.

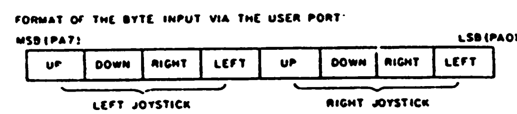


Fig. 3b. The byte input from the user port is shown here. This design exploits the fact that the PET lines PA0 to PA7 will float to high when they are disconnected. When a line goes low, the corresponding switch is closed.

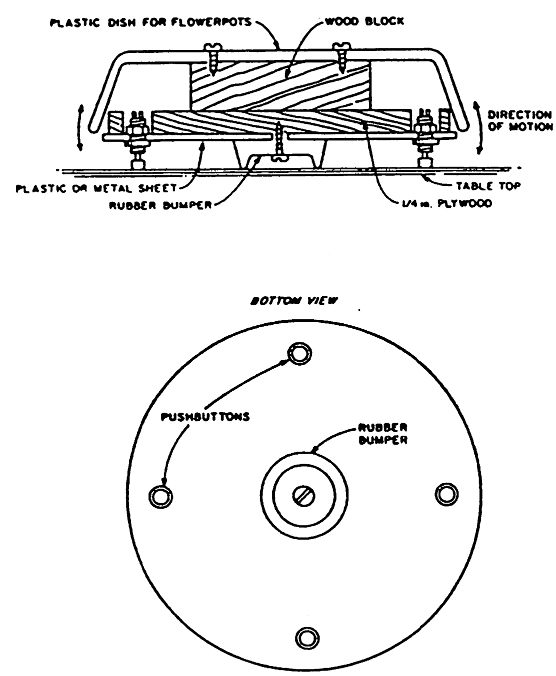


Fig. 3c. The Wobbilator—a low-cost alternative to joysticks that is easier to use as well. Eight low-cost miniature push buttons are used to build two of these units. Either normally open or normally closed push buttons may be used. (If normally closed, change lines 260 in Example 4 accordingly.) The push buttons should not be "snap action" or "detent" or go "click" when depressed, and should only move about 1/8 inch for closure. Use a bit of ribbon cable to attach the connector for the user port to the Wobbilators. Mark each Wobbilator with a dot for "Up" and "Right" and "Left." Choose a dish that fits your hand comfortably.

```
10  REM SIMPLE OUTPUT WITH HANDSHAKE
20  REM SET DDR TO OUTPUT
30  POKE 59459,255
40  REM SET POSITIVE TRANSITION FOR CA1
50  POKE 59468,PEEK(59468)OR 1
60  REM COUNT 0 TO 255
70  FOR J = 0 TO 255
80  REM OUTPUT TO PORT
90  POKE 59457,J
100 REM WAIT FOR FLAG BIT
110 IF PEEK(59469)AND 2 THEN 130
120 GOTO 110
130 NEXT J
140 REM DO IT AGAIN
150 GOTO 70
```

Example 5. Simple output with handshake for PET user port. This program waits for a strobe on CA1 before sending the data from the PET.

```
10  REM SIMPLE INPUT VIA HANDSHAKE
20  REM DDR TO INPUT
30  POKE 59459,0
40  REM NEGATIVE CA1 TRANSITION
50  POKE 59468,PEEK(59468)AND 254
60  REM CLEAR SCREEN
70  PRINT" Ⓒ ";
80  REM WAIT FOR CA1
90  IF (PEEK(59469)AND 2)=0 THEN 90
100 REM FETCH DATA & DISPLAY
110 C=C+1
120 A=PEEK(59457)
130 PRINT" Ⓗ bbbbbbbbbbbbbbbbbbbbb Ⓗ ";
140 PRINT"COUNT"C"DATA"A
150 GOTO 90
```

Example 6. Simple input with handshake for PET user port. This program waits for a low on CA1 before accepting the data and then displays the decimal value of the data on the PET screen.

```
10   REM INPUT ASCII FROM KEYBOARD
20   REM CONVERT & DISPLAY ON SCREEN
30   GOSUB 1000: REM INITIALIZE
40   GOSUB 2000: REM GET CHAR AS A$
50   PRINT A$;
60   GOTO 40
1000 REM INITIALIZE PORT & TABLE
1010 POKE 59468,PEEK(59468)OR 1
1020 POKE 59467,PEEK(59467)OR 1
1030 DIM TB(31)
1040 FOR J=0 TO 31
1050 READ TB(J): NEXT J
1060 MD=0 : RETURN
1100 DATA 0,0,0,0,19,145,29,0,0,18,0,0
1110 DATA 0,13,0,146,0,147,0,157,0,20
1120 DATA 0,0,17,148,0,0,0,0,0,0
1130 REM
2000 REM FETCH CHAR & CONVERT
2010 IF(PEEK(59469)AND 2)=0 THEN 2010
2020 CH=PEEK(59457)AND 127
2030 REM TEST IF CTRL CHAR
2040 IF CH>31 THEN 2130
2050 REM MODE FLAG TESTS
2060 IF CH=10 THEN MD=0
2070 IF CH=27 THEN MD=128
2080 REM CONVERT VIA TABLE
2090 CH=TB(CH)
2100 IF CH=0 THEN 2010
2110 GOTO 2160
2120 REM CASE CONVERT
2130 IF CH>95 THEN CH=CH-32
2140 REM MODE CONVERT
2150 CH=CH OR MD
2160 A$=CHR$(CH): RETURN
```

Example 7. Input ASCII from keyboard, convert for all PET keys and display on PET screen. This program will accept the ASCII codes from the user port and follow the convention in Table 5 and in the text.

for this lengthy explanation. Be sure your expression is doing what you want it to, and if you aren't sure, use parentheses or try trial variations and print the results on your screen.

The next thing to try is entering a value on the data switches with the Blinkin' Lights and have the PET accept the data when the CA1 line is toggled. The program in Example 6 shows how.

When the program is run, you may set the switches to a value (be sure your DIP switches are closed or you will just get 255s), and when you toggle the CA1 switch, the count and value will appear at the top of the PET display. The count is used so you can tell when you reenter the same data value. Though the desired transition for CA1 is not important in this example, line 50 shows the opposite direction from the preceding output example. In line 140, the delimiter ";" is ignored because PET BASIC will permit this.

A Keyboard Via the User Port

As an example of a useful project for the user port, I interfaced an ASCII-encoded keyboard to the PET. Since I am a fair typist, the PET keyboard is frustrating for program entry and debugging. The following example is specific to my keyboard, but almost any full ASCII keyboard and most "Dumb Teletype" keyboards can be interfaced in a similar way.

The pin-out for the keyboard was determined and wired to the PET user port as shown in Table 4. Since the keyboard drew 500 mA, it was connected to a separate 5 volt supply.

At this point, the card edge on the Blinkin' Lights was very handy. The keyboard was connected to the Blinkin' Lights and the Blinkin' Lights *not* connected to the PET. Some investigation revealed that the keyboard did encode the parity bit and that it had a 2-key rollover.

The CA1 LED would turn on when a key was depressed and when a second key was depressed, it would flicker when the first key was released. This indicated that the strobe was a positive transition and that there was a 2-key rollover.

The keyboard was then attached to the PET, and the Simple Input via Handshake program (Example 6) was tried with line 50 changed to a positive CA1 transition. After a short warm-up, each keypress showed a value, and the rollover worked just fine.

Now that the keyboard was working electrically, a dilemma appeared: How can you emulate all the PET keyboard functions? A careful study of the PET keyboard, character set and cursor control functions reveals that there are 138 functions and that the ASCII code has only 128 characters in it.

The solution I chose (feel free to choose one of your own) was to let the control character represent the various nonprinting keys (cursor movements, RVS and so on) and to convert all other characters from the keyboard to uppercase. Since the high bit for a given PET character is set if the character is a graphics character, I decided to have a Mode flag—if you pressed ESCAPE, all further alphanumeric keys would show their graphics character, and when you pressed LINEFEED, the mode would be "normal," and the character would appear.

It should be noted that the PET character set is *not* ASCII but is similar to ASCII. This resulted in some further translation steps, and the entire conversion routine used these steps:

1. Get the character from the user port and remove the Parity bit.
2. If it was a control character (0 to 31), do the following:
   (a) Find a value in a 32-element translation array for the correct PET character.
   (b) If the table value is zero, ignore and go to step 1.
   (c) Print the character on the screen and go to step 1.
3. If the character is in the range 96 to 127, subtract 32. (Converts lowercase to uppercase.)
4. If the Mode flag is set (for graphics), OR with 128 to set the highest bit.

5. Print the character on the PET.

6. Go to step 1.

Note: in step 2, if the character was an ESCAPE or a LINEFEED, the Mode flag would be set or reset, respectively, and the table entry for these characters would be a zero.

The next thing to do was to choose the meanings for the control characters. Some control characters, such as CTRL-M and CTRL-J, were already used for RETURN, LINEFEED, etc. Keys were chosen for their convenience on the keyboard in Table 5.

The appropriate PET character values were then placed in a 32-value table for lookup by the translating routine. A BASIC program was written to test this scheme out (see Example 7). Note that RETURN is the same value, 13, as the value fetching it (i.e., CH is 13 also). In line 2020, the masking is done to remove parity when the character is read from the user port. The Mode flag is set to 0 or 128, which permits the use of OR in line 2150.

Though this program is suitable for entering data into a BASIC program, the keyboard cannot be used in direct mode, that is, entering BASIC statements or LIST, etc. Example 8 shows a BASIC program which, when run, will load a machine-language program into the second cassette buffer. When this machine-language program is executed (by SYS(826)), the keyboard attached to the user port will operate "in parallel" with the PET keyboard. If you follow the cautions indicated in Example 8, you will be able to use the auxiliary keyboard for other programs, etc.

The first program, A BASIC Machine-Language Loader, will load any machine-language code in this format: AAAA HH HH HH HH HH HH HH HH. AAAA is the starting address for the first hexadecimal value, HH. Eight hexadecimal values are permitted per DATA string. Each string must begin with the address, and a space must separate the values.

If the characters in an HH field are "XX," the program will not load a value into the corresponding byte (skipping it). The characters "**" in an HH field, or "*****" in an AAAA field, will end the load.

This data format (except "XX" and "**"," ***") is identical to the one used by the PET TIM monitor, so at a later time you can easily use the PET monitor to directly load this code from the DATA statements.

The DATA statements in this program contain the object code for the second command keyboard program described in the text. To start the machine program, enter SYS(826) and press RETURN. The PET tape I/O will not work while the machine code is running! Use SYS(863) to stop the machine code and make the tape I/O workable.

Input from the second keyboard follows the rules in Table 5 and as described in the text.

It is beyond the scope of this article to describe the details of the machine-language program. A source listing is provided in Example 8 for those who wish to puzzle it out.

A User Port Monitor Program

When you are attempting to interface to the user port, it is often necessary to write several small programs to set and display the VIA registers. The program in Example 9 performs these functions and will often

```
10 REM **** PET MACHINE CODE LOADER ****
20 REM BY GREGORY YOB, 1978
30 REM READS DATA STRINGS IN FORMAT
40 REM IDENTICAL TO PET MONITOR AND
50 REM LOADS INTO INDICATED MEMORY
60 REM LOCATIONS. FIRST NUMBER IS
70 REM START ADDRESS, NEXT 8 VALUES
80 REM ARE BYTES TO LOAD.
90 REM   IF A BYTE IS 'XX' IT WILL NOT
100 REM BE LOADED, AND MEMORY CELL WILL
110 REM BE UNCHANGED, AND NEXT BYTE
120 REM LOADED INTO NEXT CELL.
130 REM IF A BYTE IS '**' OR AN ADDRESS
140 REM IS '*****', THE LOAD WILL STOP.
150 REM LINE 20000 GUARANTEES END IF
160 REM '**' OR '*****' IS NOT FOUND.
170 REM
180 REM NOTE: THIS PGM MORE USEFUL IF
190 REM EXTENDED TO DATA TAPE FILES.
200 REM
300 PRINT"Ⓒ bPET LOADER PROGRAM"
310 READ A$: IF A$="END" THEN 950
315 PRINT" Ⓓ Ⓓ "A$"Ⓓ"
320 GOSUB 400 : REM GET ADDR
330 IF ADDR <0 THEN 950
340 FOR B = 1 TO 8
350 GOSUB 500 : REM GET BYTE
355 IF BYTE = -2 THEN 380
360 IF BYTE < 0 THEN 950
370 POKE ADDR,BYTE : REM DO THE DEED
375 PRINT ADDR;TAB(10);BYTE
380 ADDR=ADDR+1 : NEXT B
390 GOTO 310
400 REM ** PARSE ADDRESS **
410 B$=MID$(A$,1,4)
420 IF B$="*****" THEN ADDR=-1 :RETURN
430 GOSUB 600 : REM HEX CONVERTER
440 ADDR=HEX
450 RETURN
500 REM ** PARSE BYTES **
510 B$=MID$(A$,B*3+3,2)
520 IF B$="XX" THEN BYTE=-2 :RETURN
530 IF B$="**" THEN BYTE=-1 :RETURN
540 GOSUB 600 : REM HEX CONVERTER
550 BYTE =HEX
560 RETURN
600 REM HEX CONVERTER
610 HEX=0
620 FOR H=1 TO LEN(B$)
630 H$=MID$(B$,H,1)
640 IF H$ <"0" THEN 900        ("0" is zero)
650 IF H$ >"F" THEN 900
660 IF H$ <":" THEN 700
670 IF H$ <"A" THEN 900
680 D=ASC(H$)-55 : GOTO 710
700 D=ASC(H$)-48
710 HEX=HEX*16 + D
720 NEXT H
730 RETURN
900 PRINT"Ⓓ Ⓓ //// BAD VALUE IN DATA ////"
910 PRINT"Ⓓ LOAD ABORTED":END
950 PRINT"Ⓓ Ⓓ LOAD FINISHED":END
1000 DATA"0338 XX XX 78 A9 75 8D 19 02"      (Note: all 0
1010 DATA"0340 A9 03 8D 1A 02 A9 00 8D"       are zeroes)
1020 DATA"0348 43 E8 8D C7 03 AD 4C E8"
1030 DATA"0350 09 01 8D 4C E8 AD 4B E8"
1040 DATA"0358 09 01 8D 4B E8 58 60 78"
1050 DATA"0360 A9 85 8D 19 02 A9 E6 8D"
1060 DATA"0368 1A 02 58 60 A9 00 48 48"
1070 DATA"0370 48 48 4C 85 E6 AD 4D E8"
1080 DATA"0378 29 02 D0 07 20 6C 03 EA"
1090 DATA"0380 4C 7E E6 AD 41 E8 29 7F"
1100 DATA"0388 C9 1F 10 30 C9 0A D0 07"
1110 DATA"0390 A9 00 8D C7 03 F0 E5 C9"
1120 DATA"0398 1B D0 07 A9 80 8D C7 03"
1130 DATA"03A0 D0 DA AA BD C8 03 F0 D4"
1140 DATA"03A8 EA AE 00 02 90 0F 02 E8"
1150 DATA"03B0 E0 0A D0 02 A2 00 8E 0D"
1160 DATA"03B8 02 4C 7C 03 C9 60 30 02"
1170 DATA"03C0 E9 20 00 C7 03 D0 E2 00"
1180 DATA"03C8 00 00 00 00 13 91 1D 00"
1190 DATA"03D0 00 12 00 00 00 0D 00 92"
1200 DATA"03D8 00 93 00 9D 00 14 00 00"
1210 DATA"03E0 11 94 00 00 00 00 00 00"
1220 DATA"03E8 ** ** ** ** ** ** ** **"
20000 DATA"END"
```

Machine-Language Source

```
       ! FOOL THE PET INTO READING THE USER PORT AS THE
       ! COMMAND KEYBOARD IN PARALLEL WITH THE NORMAL
       ! KEYBOARD BY READING THE USER PORT WHEN THE 60 HZ
       ! INTERRUPT IS SERVICED.  IF A CHARACTER IS
       ! PRESENT, TRANSLATES ACCORDING TO SCHEME DESCRIBED
       ! IN USER PORT ARTICLE AND PUTS CHARACTER INTO
       ! THE PET INPUT BUFFER.
       !         THIS CODE TAKEN FROM AN IDEA BY RICHARD
       ! TOBEY. IMPLEMENTED BY GREGORY YOB.
       !
       ! *** INITIALIZATION CODE ***
       !         TURN OFF INTERRUPTS, AND SET THE PET
       ! "INTERRUPT VECTOR" TO POINT TO THE ACTIVE CODE.
       ! SET UP THE USER PORT TO READ THE KEYBOARD, AND
       ! SET THE MODE VARIABLE TO "CHARACTER MODE" (0)
       !
       ! NOTE*** THIS CODE RESIDES IN THE SECOND CASSETTE
       ! BUFFER ( 033A TO 03FF )
       !
033A  78        XON   SEI        ! DISABLE INTERRUPTS
033B  A9 75           LDA #$75   ! SET UP NEW
033D  8D 19 02        STA $0219  !    "INTERRUPT
```

Keyboard Pin		PET User Port
1	INT Key	—
2	RPT Key	—
3	No connection	CB2
4	No connection	—
5	GND	GND
6	+5 Volts (separate supply)	—
7	Strobe	CA1
8	Parity	PA7
9	Bit 4	PA3
10	Bit 3	PA2
11	Bit 1	PA0
12	Bit 7	PA6
13	Bit 2	PA1
14	Bit 6	PA5
15	Bit 5	PA4

Table 4. ASCII keyboard to PET user port wiring list. Your keyboard will, no doubt, have a different pin-out—just notice the data and handshake lines. If your keyboard requires an acknowledge, connect your ACK to CB2.

```
0340  A9 03          LDA #$03   !          VECTOR"
0342  8D 1A 02       STA $021A
                !
0345  A9 00          LDA #$00   ! SET UP USER PORT & MODE
0347  8D 43 E8       STA $E843  ! DATA DIRECTION REGISTER
034A  8D C7 03       STA MODE   ! MODE CELL
034D  AD 4C E8       LDA $E84C  ! PERIPHERAL CONTROL REGISTER
0350  09 01          ORA #$01
0352  8D 4C E8       STA $E84C  ! PCR
0355  AD 4B E8       LDA $E84B  ! AUXILIARY CONTROL REGISTER
0358  09 01          ORA #$01
035A  8D 4B E8       STA $E84B  ! ACR
035D  58             CLI        ! ENABLE INTERRUPTS
035E  60             RTS        !  AND RETURN TO CALLER
                !
                ! *** RESTORATION CODE ***
                !       RESTORE THE "INTERRUPT VECTOR" SO THAT TAPE
                ! I/O CAN WORK PROPERLY.
                !
035F  78        XOFF SEI        ! DISABLE INTERRUPTS
0360  A9 85          LDA #$85   ! SET UP OLD
0362  8D 19 02       STA $0219  !     "INTERRUPT
0365  A9 E6          LDA #$E6   !        VECTOR"
0367  8D 1A 02       STA $021A
036A  58             CLI        ! ENABLE INTERRUPTS
036B  60             RTS        !  AND RETURN TO CALLER
                ! *** STACK ADJUSTMENT ROUTINE ***
                !
036C  A9 00     STAX LDA #$00   ! DUMMY PUSHES TO PET STACK FOR
036E  48             PHA        ! CORRECT OPERATION OF THE
036F  48             PHA        ! RESTORATION CODE
0370  48             PHA
0371  48             PHA
0372  4C 85 E6       JMP $E685  ! JUMP TO PET INTERRUPT HANDLER
                                ! TO CONTINUE PROCESSING
                !
                ! *** ACTIVE CODE ***
                !       CHECKS USER PORT IFR FOR CHARACTER. IF NOT
                ! PRESENT, RETURNS TO PET INTERRUPT PROCESSOR.
                !       IF PRESENT, TRANSLATES ACCORDING TO SCHEME
                ! AND PUTS INTO THE INPUT BUFFER.
                !
0375  AD 4D E8  PCODE LDA $E84D ! INTERRUPT FLAGS REGISTER
0378  29 02          AND #$02
037A  D0 07          BNE KEYS   ! DETECTED CHARACTER
037C  20 6C 03  FINISH JSR STAX ! SET UP TO CALL THE
037F  EA             NOP        ! PET RESTORATION CODE
0380  4C 7E E6       JMP $E67E  ! WHICH IS FROM HERE
                !
                !CHARACTER PROCESSING
                !
0383  AD 41 E8  KEYS LDA $E841  ! ORA HANDSHAKE DATA REGISTER
0386  29 7F          AND #$7F   ! MASK OFF PARITY
0388  C9 1F          CMP #$1F
038A  10 30          BPL NCTR   ! IF POSITIVE, ISN'T A CONTROL CHAR
038C  C9 0A          CMP #$0A
038E  D0 07          BNE NLFD   ! CHAR ISN'T A LINEFEED
0390  A9 00          LDA #$00
0392  8D C7 03       STA MODE   ! SET MODE TO CHARACTERS
0395  F0 E5          BEQ FINISH ! BEQ SAVES A BYTE
                !
0397  C9 1B     NLFD CMP #$1B   ! ESCAPE?
0399  D0 07          BNE CTRL   ! OTHER CTRL CHARS
039B  A9 80          LDA #$80   ! SET MODE TO
039D  8D C7 03       STA MODE   !    GRAPHICS
03A0  D0 DA          BNE FINISH ! SAVE ANOTHER BYTE
                !
                ! PROCESS CONTROL CHARS BY TABLE LOOKUP
                !
03A2  AA        CTRL TAX
03A3  BD C8 03  LDA TABL, X
03A6  F0 D4     BEQ FINISH ! IGNORE IF TABLE RETURNS ZERO
03A8  EA        NOP
                !
                ! *** STASH CHARACTER INTO INPUT BUFFER ***
                !       NOTE THAT BUFFER POINTER MUST BE CHECKED &
                ! CORRECTLY ADJUSTED.
                !
03A9  AE 00 02  STASH LDX $0200 ! PET INPUT BUFFER POINTER
03AC  9D 0F 02       STA $020F,X ! BASE OF INDEX IS START OF BUFFER
03AF  E8             INX
03B0  E0 0A          CPX #$0A   ! CHECK IF FULL
03B2  D0 02          BNE *+4    ! SHORT JUMP (SKIP ONE INSTR)
03B4  A2 00          LDX #$00
03B6  8E 00 02       STX $0200  ! SAVE NEW POINTER
03B9  4C 7C 03       JMP FINISH ! I KNOW, I COULD HAVE SAVED A BYTE.
                !
03BC  C9 60     NCTRL CMP #$60  ! CONVERT TO UPPER CASE
03BE  30 02          BMI NCASE
03C0  E9 20          SBC #$20
03C2  0D C7 03       ORA MODE   ! CONVERT TO GRAPHIC IF
03C5  D0 E2          BNE STASH  !  MODE > 0
                !
                ! *** DATA STORAGE AREA ***
                !
03C7  00        MODE     ! MODE BYTE = 0 IF CHARACTERS
                         !         128 IF GRAPHICS
03C8            TABL     ! CONTROL CHARACTERS CONVERSION TABLE
03C8                     .BYTE.  00,00,00,00,13,91,1D,00
03D0                     .BYTE.  00,12,00,00,00,00,00,92
03D8                     .BYTE.  00,93,00,9D,00,14,00,00
03E0                     .BYTE.  11,94,00,00,00,00,00,00
                !
03E8            ! *** END OF CODE ***
```

Example 8. PET machine code program for a second command keyboard.

save some time and trouble.

Some comments concerning the code are in order:

Lines 70 to 90 hold the register names, which are similar to, and in the same order as, those in Fig. 2.

Line 210 puts a colon and some blanks at the end of each register name for display purposes.

Line 250 sets the Flags array to display the most commonly used registers when the program starts.

Notice the *three* blanks between the 4 and the 3 in line 310.

Line 320 moves the menu to a position that will not be overwritten when the program is displaying all 16 registers.

Cursor movements are used extensively to control the display. Be sure to count them carefully.

Lines 1000 to 1050 display a number in binary by moving a mask bit (variable Z1) to the right and printing the sign of the result (line 1030).

Subroutine 2000 is required to permit you to choose the time to access the Handshake Data register. The reason is that each access to this register will reset the Interrupt Flag bit. The D (DATA) command will read this register.

Subroutine 3000 lets you change the registers you want to see displayed. If you forget the names (I often do), enter a meaningless name, such as "XXX," and all the names will be shown.

Since the display is in binary, so is the input (see subroutine 4500).

Subroutine 4990 provides a "False Cursor," which is handy in many programs.

When the CB2 line is toggled, the original values of the PCR and ACR are saved, and after toggling, restored. CB2 is forced both high and low to guarantee a handshake pulse.

Using the User Port Monitor Program

After you have tried out the various commands and are familiar with them, attach the Blinkin' Lights to the user port and run the Monitor program. Close all of the Data Isolation switches and set the Data switches to low. If you are starting from a reset PET (you haven't changed any of the user port registers), the PET display will look like this:

```
       7 6 5 4 3 2 1 0
DDRA : 0 0 0 0 0 0 0 0
ACR  : 0 0 0 0 0 0 0 0
PCR  : 0 0 0 0 1 1 0 0
IFR  : 0 1 1 0 0 0 0 0
DATA : 0 0 0 0 0 0 0 0




D=DATA P=POKE S=SHOW
H=HELP Q=QUIT T=TOGGLE
```

The "1" bits are aspects of the registers used internally by the PET for its housekeeping functions. If you set the low

Character + CTRL	PET Function
Q	Clear Screen
D	Home Cursor
E	Cursor Up
S	Cursor Left
F	Cursor Right
X	Cursor Down
Y	INST
U	DEL
I	RVS on
O	RVS off

Table 5. Control characters for PET special keys.

```
10  REM CB2 BLINKER
20  POKE 59467, PEEK(59467) AND 227
30  POKE 59468,(PEEK(59468) AND 31) OR 192
40  FOR J = 1 TO 300: NEXT
50  POKE 59468, PEEK(59468) OR 244
60  FOR J = 1 TO 300: NEXT
70  GOTO 30
```

Example 10. CB2 Blinker program. The CB2 LED in the Blinkin' Lights will blink at about 1Hz.

four bits on the Blinkin' Lights Data switches to high, the DATA: line will become 0 0 0 0 1 1 1 1. As you change the switch settings, you will notice that there is a lag of about one second before the display responds.

This illustrates how the Monitor program can show the data you input to the user port. Now disconnect the Data switches by opening the Data Isolation switches—the DATA: will now become all ones.

With the P command, change the DDRA to 1 1 1 1 1 1 1 1. The DATA: is now 0 0 0 0 0 0 0 0. This is the initial value stored in the PET. Using P again, change the DATA register to some other value and watch it appear on the LEDs on the Blinkin' Lights. This illustrates data output.

If you close the Data Isolation switches and change these registers with the P command, you can demonstrate input via handshake with the CA1 line:

```
DDRA  set to  0000 0000
PCR   set to  0000 1100  (Negative transition)
ACR   set to  0000 0001  (Enable latching)
```

When you return to the display, the IFR may look like: 0 1 1 0 0 0 1 0. If it does, press D and then press any key. The IFR will now return to: 0 1 1 0 0 0 0 0, indicating that the Flag bit was reset when the Data with Handshake was read.

Set the Blinkin' Lights Data switches to some value and watch the DATA: on the display. The value will follow the switch settings. Now, flick the CA1 toggle switch (be sure the isolation switch is closed), and the IFR will show bit 1 as set. If you now change the Data switches, the DATA: value will not change. It will remain latched until you do the D command. This illustrates input with latching and handshaking.

Feel free to experiment with other settings for the user port with the Monitor program.

The CB2 Line

The CB2 line is the most complex of the user port lines. It can be operated in a variety of modes, including the provision of an output handshake and the serial transfer of data. As most of the CB2 modes can only be controlled from machine language, this article will cover only the two modes that are usable from BASIC.

CB2 as an Output or Handshake

The CB2 line may be turned off or on directly to provide either a handshake line or a 9th output bit for the user port. In either case, the shift register modes must be disabled by setting the Auxiliary Control register (ACR) as follows:

```
POKE 59467, PEEK(59467) AND 227
```

(In most cases the ACR is already zero, so this may be ignored. However, safety first!)

```
10 REM 6522 VIA DISPLAY AND MONITOR
20 REM PROGRAM
30 REM BY: GREGORY YOB, 1978
40 REM SET UP R$= REGISTER NAMES,
50 REM A()=REGISTER ADDRESSES,
60 REM F()=SHOW REGISTER IF >0
70 DATA "ORB","ORA","DDRB","DDRA"
80 DATA "T1LC-L","T1C-H","T1L-L","T1L-H"
90 DATA "T2LC-L","T2C-H","SR","ACR"
100 DATA"PCR","IFR","IER","DATA"
110 REM 'DATA' IS ORA WITHOUT HANDSHAKE
120 DIM R$(16),A(16),F(16)
200 A=59456: FOR J=1 TO 16
210 READ A$:R$(J)=LEFT$(A$+"bbbbbbbb",6)+";"
220 A(J)=A:A=A+1
230 NEXT J
240 REM SET FLAGS FOR INITIAL DISPLAY
250 F(4)=1:F(12)=1:F(13)=1:F(14)=1:F(16)=1
300 REM SET UP DISPLAY
310 PRINT"Ⓒ bbbbbbbb7bb6bb5bb4bbb3bb2bb1bb0"
320 PRINT"ⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹⒹ";
330 PRINT"D=DATA  P=POKE  S=SHOW"
340 PRINT"H=HELP  Q=QUIT  T=TOGGLE"
400 REM DISPLAY LOOP
410 PRINT"Ⓗ Ⓓ Ⓓ ";
420 FOR J=1 TO 16
430 IF F(J)=0 THEN 450
440 Z=PEEK(A(J)):PRINTR$(J);:GOSUB1000
450 NEXT J
460 REM IF NO INPUT DO LOOP AGAIN
470 GETA$:IFA$=""THEN 410              ( "" is a null string)
500 REM DO COMMANDS
510 IF A$="D" THEN GOSUB 2000
520 IF A$="P" THEN GOSUB 2500
530 IF A$="S" THEN GOSUB 3500
540 IF A$="H" THEN GOSUB 3000
550 IF A$="T" THEN GOSUB 5500
560 IF A$="Q" THEN END
700 GOTO 310
1000 REM DISPLAY IN BINARY
1010 Z1=128
1020 FOR Z2=1 TO 8
1030 PRINT SGN(Z AND Z1);
1040 IF Z2=4 THEN PRINT "b";
1050 Z1=Z1/2 : NEXT Z2: PRINT : RETURN
2000 REM DISPLAY HANDSHAKE REGISTER
2010 Z = PEEK(59457) :PRINT"Ⓓ"R$(2); GOSUB 1000
2020 PRINT"Ⓓ"; : GOSUB 4990:RETURN
2500 PRINT"Ⓒ POKE REGISTERⒽⒹⒹ"
2510 GOSUB 4000
2520 GOSUB 4500
2530 POKE A(Z),B
2540 RETURN
3000 PRINT"Ⓒ bb 6522 REGISTER DISPLAY AND CHANGE ⒹⒹ
3010 PRINT"THIS SHOWS THE VALUES FOR THE PET'S
3020 PRINT"VIA REGISTERS. YOU CAN LOOK AT ALL OF
3030 PRINT"THEM. THOSE USED FOR THE USER
3040 PRINT"PORT ARE SHOWN WHEN THE PROGRAM
3050 PRINT"STARTS. ⒹⓁⓁⓁTHE DISPLAY IS REFRESHED ABOUT ONCE
3060 PRINT"PER SECOND. PRESS A KEY TO DO A COMMAND
3070 PRINT"Ⓓ bbbD=DATA  READS ORA WITH HANDSHAKE
3080 PRINT"   P=POKE  LETS YOU POKE A REGISTER
3090 PRINT"   S=SHOW  SELECTS REGISTERS TO DISPLAY
3100 PRINT"   Q=QUIT  STOPS PROGRAM
3110 PRINT"   T=TOGGLE  TURNS CB2 ON, THEN OFF TO
3120 PRINT"             FORCE HANDSHAKE & THEN
3130 PRINT"             RESTORES TO PRIOR STATE
3300 PRINT"ⒹⒹ";:GOSUB4990:RETURN
3500 REM CHANGE DISPLAYED REGISTERS
3510 PRINT"Ⓒ SHOW REGISTERSⒹⒹⒹ
3520 GOSUB 4000
3530 PRINT"S=SHOW,E=ERASE,X=FINISHED";:GOSUB 5000
3540 IF A$="S" THEN F(Z)=1
3550 IF A$="E" THEN F(Z)=0
3560 IF A$="X" THEN RETURN
3570 PRINT"ⒽⒹⒹⒹⒹ";
3580 GOTO 3520
4000 REM GET REGISTER NAME, RETURN Z=INDEX
4010 PRINT"ⒹⒹ REGISTER NAME:bbbbbbbbbbbbⓁⓁⓁⓁⓁⓁⓁⓁⓁⓁⓁⓁ";
     :INPUT A$
4020 RESTORE:FORZ=1TO16:READB$
4030 IFB$=A$THEN RETURN
4040 NEXTZ:PRINT"ⒹⒹⒹ THE REGISTERS ARE CALLED:
4050 FOR J=1TO16:PRINT LEFT$(R$(J),6)"bbbb";:NEXT J
4060 PRINT"ⓊⓊⓊⓊⓊⓊⓊⓊⓊⓊⓊⓊ";: GOTO 4010
4500 REM - GET BINARY NUMBER
4510 PRINT"BINARY VALUE: ";:INPUT A$:Z1=128:B=0
4520 IF LEN(A$) < 8 THEN PRINT "Ⓤ";: GOTO 4510
4530 FOR J=1TO8
4540 IF MID$(A$,J,1)="1"THEN B=B OR Z1
4550 Z1=Z1/2:NEXT J
4560 RETURN
4990 PRINT"PRESS A KEY";
5000 GET A$: PRINT"[&]Ⓛ";:FOR K=1 TO 20: NEXT K
5010 PRINT"bⓁ";:FOR K = 1 TO 20: NEXT K
5020 IF A$=""THEN 5000                  ( "" is a null string)
5030 RETURN
5500 REM TOGGLE CB2
5510 A=PEEK(59467):B=PEEK(59468)
5520 C=B AND 131 OR 192
5530 D= B OR 224
5540 POKE 59468,C
5550 POKE 59468,D
5560 POKE 59468,B
5570 POKE 59467,A
5580 RETURN
```

Example 9. PET user port display and monitor program.

Then, the CB2 line is set high by:

POKE 59468, PEEK(59468) OR 224

and it is set low by:

POKE 59468,(PEEK(59468)AND 31) OR 192

The parentheses are required to ensure that the operations AND and OR are done correctly. Example 10 is a short "CB2 Blinker" that blinks CB2 at about 1 Hz.

Interfacing the Writehander

The Writehander is a one-handed input keyboard manufactured by the NewO Company, 246 Walter Hays Drive, Palo Alto CA 94303 (see *Kilobaud* No. 23, p. 9, for a description of the Writehander).

The Writehander is a gray plastic ball about six inches across with switches placed so that the fingers and thumb may touch them. By altering the finger arrangements, you can send any of the 128 ASCII codes to the computer. When the byte is ready, the Writehander provides a strobe and then requires an acknowledge signal before it sends the next byte.

The wiring to the PET user port is shown in Table 6. The grounds were connected together for the power supply, the PET and the Writehander. The Writehander has several jumper options that were wired as:

1) Strobe goes active low +to−˥_
2) Acknowledge active low +to−˥_
3) Parity (Bit 8) set low Gnd

This means that the following steps are required to talk with the Writehander.

1. Poke the DDR to all inputs
2. Set CA1 to detect the Hi to Low transition
3. Disable the CB2 Shift Register mode
4. Enable latching with CA1
5. Turn CB2 on (high)
6. Wait for the Interrupt flag in the IFR
7. Read the Data with Handshake
8. Mask off the parity bit and display the data (or whatever)
9. Turn CB2 off (low)
10. Go to step 5

These steps were incorporated into a program, Example 11, which was only intended to accept characters from the Writehander and display their values on the PET screen. See the program in Example 7 for a more complete processing of the characters. (If you are a real diehard, modify the assembly program in Example 8 to provide the required CB2 logic.)

Lines 30 and 40 can be combined, but this program keeps them separate to show the different things being done. If you want to show the character rather than the value, use:

```
90 PRINT CHR$( X AND 127 );
```

I encountered several frustrating experiences during the development of the above (simple!) program:

1. The Writehander would work perfectly when attached to the Blinkin' Lights by itself, and the program would work perfectly when it was attached to the Blinkin' Lights . . . and (guess), when the Writehander was attached to the PET, it *wouldn't* work! After much fiddling, I discovered that the Writehander required that the ACK (CB2) be *high* before it would bring the Strobe (CA1) low. Thus CB2 had to be set high before trying to look for a character.

2. The parenthesis around the PEEK in line 80 is required for the CB2 to be set low due to the precedence relations of AND and OR.

3. PET ASCII isn't ASCII, so the "wrong" character would be displayed (see A Keyboard Via the User Port section for a detailed discussion).

```
5   PRINT" © ";
10  POKE 59459,0
20  POKE 59468, PEEK(59468) AND 254
30  POKE 59467, PEEK(59467) AND 227
40  POKE 59467, PEEK(59467) OR 1
50  POKE 59468, PEEK(59468) OR 224
60  IF (PEEK(59469) AND 2) = 0 THEN 60
70  X = PEEK(59457)
80  POKE 59468, (PEEK(59468)AND 31) OR 192
90  PRINT X AND 127;
100 GOTO 50
```

Example 11. Writehander input program.

CB2 as a Shift Register

The CB2 line may be made to act as a shift register by setting a combination of bits 2, 3 and 4 in the Auxiliary Control register (ACR). Only one of these modes is usable from BASIC. The others require the use of machine language to be controlled properly (see the 6522 VIA specification for details).

One nice way to experiment with this is to use the PET to make "square wave music." Fig. 4 shows two ways to attach an audio extension to the PET. Each of these simply uses the CB2 line for the audio signal.

Checking It Out

Once you have your audio extension together, one way to check it out is to toggle CB2 in Handshake mode as fast as BASIC will go:

```
10 POKE 59467,PEEK(59467)AND 227
20 A=59468: X=PEEK(A)AND 131 OR 192
30 Y=PEEK(A) OR 224
```

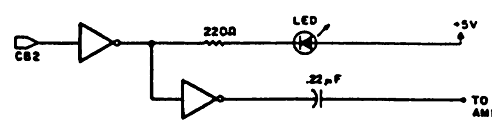


Fig. 4a. Add the inverter and capacitor to the output of the CB2 inverter in the Blinkin' Lights. Fig. 2 has this addition indicated.

Line	Color	Function	PET
1	Brown	Bit 1	PA0
2	Red	+7 to +23 V power (unused)	
3	Orange	Bit 2	PA1
4	Yellow	Ground	GND
5	Green	Bit 3	PA2
6	Blue	+5 V (separate power supply)	
7	Violet	Bit 4	PA3
8	Gray	—	
9	White	Bit 5	PA4
10	Black	—	
11	Brown	Bit 6	PA5
12	Red	—	
13	Orange	Bit 7	PA6
14	Yellow	Strobe	CA1
15	Green	Bit 8	PA7
16	Blue	Acknowledge(ACK)	CB2

Table 6. Writehander wiring list.

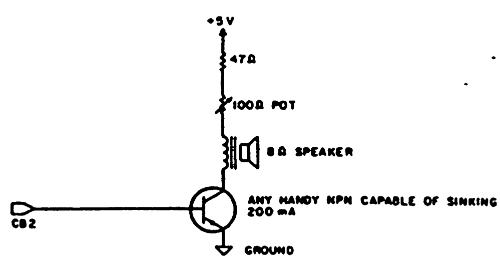


Fig. 4b. This circuit lets you add sound effects, etc., for you PET without any additional equipment. Take the +5 volts from the second tape port. (That's the top or bottom pin, second in from the side of the PET. Check your first tape recorder to find whether it is on top or bottom—Commodore makes both kinds!) Find a 2 or 3 inch speaker and any handy NPN transistor capable of 200 mA current. The 47 Ohm resistor should be 1/2 Watt or larger and should not be omitted. My unit was put on a 3 × 5 inch perfboard with connectors glued to one edge, which makes it easy to hook to my PET.

Data Directions Register	
POKE 59459, 255	Set user port to 8 bits output.
POKE 59459, 0	Set user port to 8 bits input.
Simple Input and Output (no handshakes)	
(value) = PEEK(59471)	Input (value) from user port.
POKE 59471, (value)	Output (value) to user port.
Input and Output with Handshaking	
POKE 59468, PEEK(59468) AND 254	CA1 will trigger on falling edge.
POKE 59468, PEEK(59468) OR 1	CA1 will trigger on rising edge.
POKE 59467, PEEK(59467) OR 1	Data is latched when CA1 triggers.
POKE 59467, PEEK(59467) AND 254	Data is not latched.
IF PEEK(59469) AND 2 THEN —	Three ways of detecting the CA1 Flag Bit.
WAIT 59469, 2	Be careful with using WAIT.
nnn IF(PEEK(59469) AND 2) = 0 THEN nnn	
(value) = PEEK(59457)	Reads from user port, resets CA1 flag bit.
POKE 59457, (value)	Writes to user port, resets CA1 flag bit.
POKE 59468, PEEK(59468) OR 224	Set CB2 line high.
POKE 59468, (PEEK(59468) AND 31) OR 192	Set CB2 line low.
Shift Registery	
POKE 59467, PEEK(59467) AND 227 OR 16	Sets shift register to free running mode.
POKE 59467, PEEK(59467) AND 227	Disables shift register modes.
POKE 59466, (value)	Puts (value) into shift register.
POKE 59464, (value)	Sets timer 2 to (value)
Miscellany	
(value) = PEEK(515)	Reads matrix value of key pressed. 255 = no keys pressed.
(value) = PEEK(516)	Reads shift keys. 1 if pressed, 0 otherwise.

Table 7. Summary of BASIC statements used to control the PET user port.

```
40 POKE A,X:POKEA,Y: GOTO 40
```

Line 10 disables the Shift Register mode, and line 40 turns CB2 on and off. The reason that variables are used in line 40 for the addresses is that BASIC runs much faster when variables are substituted for constants.

RUN the program, and a buzz will emerge from your speaker.

Try changing line 40 to:

```
40 POKE59468,X:POKE59468,Y:GOTO 40
```

and you will notice that the pitch of the buzz is much lower. (Note: You will also hear a variation in the pitch of the buzz. This is caused by the PET's interrupt routines "beating" with the execution of the BASIC program.)

A last variation before going on to the shift register is to change the above program as follows:

```
40 Z=515
50 POKE A,X:FOR J = 1 TO PEEK(Z):
   NEXT: POKE A,Y: GOTO 50
```

Pressing different keys will vary the rate of clicking. (Note: Location 515 indicates which key is depressed on the PET keyboard. This is not in PET ASCII but represents the matrix position of the key.)

Shift Register Mode

When the ACR bits 4, 3 and 2 are "100" the shift register is in "free running mode." Two addresses are now of interest:

SR	Shift Register	59466
T2L-W	Timer-2	59464

At a rate determined by the contents of Timer-2, the contents of the shift register are placed on the CB2 line. When eight bits have been shifted out, the shift register is again shifted out. This creates a continuous stream of bits that repeats every eight Timer-2 cycles.

Timer-2 accepts a number from 0 to 225 and counts it down to zero at the PET clock rate. When it reaches zero, the shift register is shifted and the least significant bit (bit 0) is placed on the CB2 line.

By placing an appropriate number into Timer-2 for the pitch and a 15 into the shift register, square waves at audio frequency will emerge from CB2. Here is the world's clumsiest musical instrument (see Example 12). Try it and you will know why. Line 50 inputs a waveform to be put into the shift register when a key is pressed. Line 60 guarantees that the waveform will result in a sound (a 0 or a 255 will come out as a dc voltage).

Line 90 detects the state of the PET keyboard matrix. When no key is depressed, the value in this address is 255. Line 100 puts a zero into the shift register, turning the sound "off." Then the keyboard is checked again.

If a key is depressed, the "pitch," or the matrix value of the key, is put into the timer and the timbre is put into the shift register. Now a sound is heard (for most of the keys; some will

```
10  REM CLUMSY MUSIC MACHINE
20  REM SET S.R. MODE IN ACR
30  POKE 59467, PEEK(59467) AND 227 OR 16
40  PRINT"TIMBRE :";
50  INPUT TC
60  IF TC<1 OR TC>254 THEN 40
70  REM CHECK FOR KEYPRESSES
80  PRINT"PRESS KEYS FOR TONES"
90  K = PEEK(515)
100 IF K = 256 THEN POKE 59466,0: GOTO 90
110 POKE 59464,K: POKE 59466,TC
120 K = PEEK(515): IF K = 256 THEN 100
130 GOTO 120
```

Example 12. A clumsy music machine.

```
10  POKE 59467,PEEK(59467)AND 227 OR 16
20  POKE 59466,15
30  FOR J = 0 TO 255; POKE 59464,J: NEXT
100 GET A$: IF A$ = "" THEN 30
110 POKE 59466,0
```

Example 13. Program for effect 1.

```
30 FOR J = 10 TO 256 STEP 10: POKE 59464,J: NEXT
40 FOR J = 255 TO 10 STEP - 10: POKE 59464,J: NEXT
```

Example 14. Changes in Example 13 for effect 2.

```
30 FOR J = 1 TO 100: POKE 59464, 240*RND(1) + 10: NEXT
```

Example 15. Change in Example 13 for effect 3.

```
30 FOR J = 1 TO 30: POKE 59464,100: POKE 59464,200: NEXT
40 FOR J = 1 TO 30: POKE 59464,150: POKE 59464,250: NEXT
```

Example 16. Changes in Example 13 for effect 4.

make inaudibly high notes). Line 120 waits until the key is released before starting over at line 100.

Some time spent with a calculator or scope will yield about two octaves of pitches that are reasonably close to the musical scale(s). Feel free to write your own musical programs.

```
10   REM BETTER WOLF
20   REM GREGORY YOB
30   REM CB2 ON USER PORT & AMP
100  POKE 59467,16 :POKE 59466,15
110  FOR L = 180 TO 50 STEP  -3:POKE 59464,L:NEXT
111  FOR J = 1 TO 6:NEXT
112  POKE 59466,0
115  FOR J = 1 TO 150: NEXT
117  POKE 59466,15
120  FOR L = 150 TO 80 STEP  -2: POKE 59464,L:NEXT
130  FOR L =90 TO 190: POKE 59464,L
132  FOR J = 1 TO L/70: NEXT
134  NEXT
135  POKE 59467,0
140  PRINT"PRESS KEY TO DO IT AGAIN"
150  GET A$: IF A$ ="" THEN 150
160  GOTO 100
```

Example 17.

Since the CB2 line, once in Shift Register mode, will run independently of the PET's other activities, other computations may be done while a tone is sounded. Another aspect is the making of sound effects for games. See Examples 13-17 and try them out to find out what they do.

Lines 100 and 110 in Example 13 provide a way of turning the sound off. If you don't do this, the PET will squeak at you after you press the STOP key—and only a direct version of line 110 will turn the squeak off! Examples 14-16 show changes to Example 13.

Summing Up

The PET user port is a versatile way with which to communicate between the PET and the rest of the world. This article has shown you the "nuts and bolts" required to interface many devices, including joysticks, keyboards and music makers, that add to the capabilities or your PET.

For your convenience, Table 7 summarizes the various BASIC statements used to control the user port. Now let me see . . . robots, turtles, printers, my lawn sprinklers. . . . ■

Get Your Pet on The IEEE 488 Bus

This 3-part odyssey takes you along route 488. The first stop is here . . . tickets, please.

Gregory Yob
Box 354
Palo Alto, CA 94301

Perhaps the most obscure Commodore PET feature is its IEEE 488 (or HPIB or GPIB) interface. This three-part article describes the rudiments of the 488 bus and how to use your PET to communicate with instruments having the 488 interface. Several working examples with Hewlett-Packard equipment are shown. (HP lent me several 488-compatible instruments to prepare this article.)

If you just want your PET to talk to that costly instrument on your bench, skip this month's installment and start next time with part 2. The first two parts of this article will sketch the prerequisites and give you enough information to track down bugs on your own.

What's a 488 Bus?

In 1972, engineers—some with Hewlett-Packard—proposed a method of joining many instruments in a standardized way to help automate lab and test measurements. This resulted in the IEEE Standard 488-1975, which describes how to connect as many as 15 instruments on the same cable.

HP and several other laboratory-instrument manufacturers then offered the IEEE 488 scheme as an option. Presently, several hundred instruments have the 488 capability; Commodore used to provide a 5-page list of these. The PET was later designed with the instrumentation and control market in mind, so the IEEE 488 interface was put into the PET.

Before the introduction of the PET, instruments capable of controlling the 488 bus cost several thousand dollars. Now the PET often costs less than the instruments it controls. Some 488 manufacturers have trouble adjusting to this—their customers balk at the idea of purchasing an $800 microcomputer to control a $30,000 instrument!

Now one connector joins the PET to many peripherals. You don't need a separate interface and connector for each new gadget. Commodore's printer and disk are designed to use the PET's 488 interface.

Physical Aspects

A PET and a 488-compatible device have different connectors. Your first project is to wire a cable to tie the two machines together.

Fig. 1 shows the location of the IEEE 488 connector on the back of the PET, and Fig. 2 describes the pins and connectors used for the PET and the IEEE 488. I used a 20-conductor ribbon cable and tied the

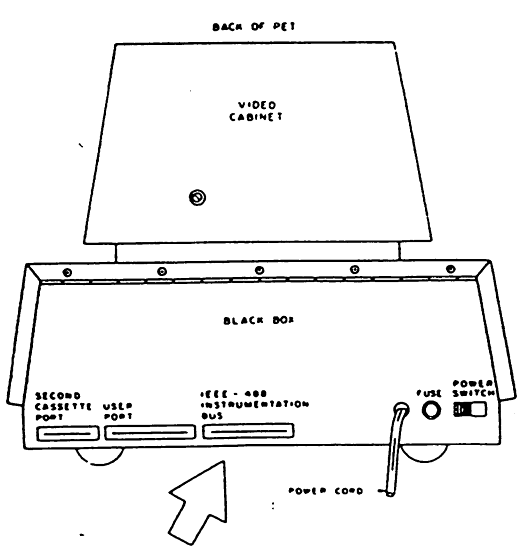


Fig. 1. Location of PET IEEE 488 port on the back of the PET next to the power switch and fuse.

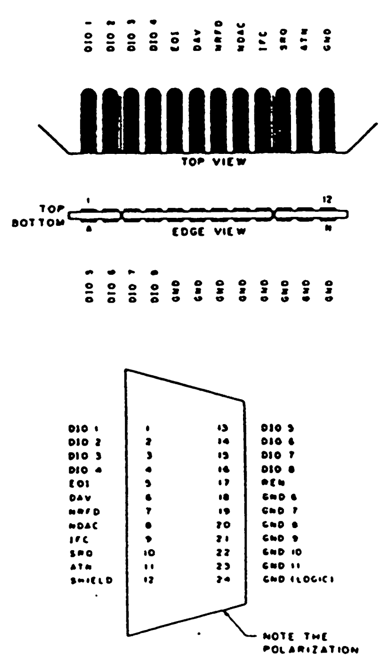


Fig. 2. Pin-outs and connectors for the IEEE 488.

grounds together into the four lines left over after I connected the signal wires.

When making the cable, bear in mind that there are strict limits to cable lengths:

1. The maximum distance between two devices is 5 meters.
2. The longest distance from one end of your setup to the other is 20 meters.
3. A maximum of 15 devices, including the PET, can be hooked together.

It is also wise to avoid electrically noisy areas; don't drape your IEEE 488 cable over your TV set.

If more than one device is connected to the 488, you must use extension cables. HP has cables for about $50. If you want to make your own, consult the two configurations in Fig. 3. The 488 instruments always have a female connector, so have an excess of male connectors on your cables.

Electrically, the 488 bus works on an active-low principle. Fig. 4 shows a circuit similar to a 488 bus line. When all the switches are open, the voltmeter will show 5 volts, which is the false state (or 0) for the line. If any of the switches are closed, the line is grounded, and the voltmeter shows zero volts, or the true state.

This peculiar arrangement permits several devices to be connected to the same line. If any one of them has a switch closed, the line is true. Devices frequently operate at different speeds, and when each device is ready, it opens its switch. However, the line remains true (low) until the slowest device opens its switch.

IEEE Blinkin Lites Display

It is always convenient to have a display and switches to perform a front panel function when you debug interfaces. I built a box, which I call the 488 Blinkin Lites, to display the states of each of the IEEE 488 lines and some switches to force lines low if needed. Fig. 5 shows the circuit, and Fig. 6 is a sketch of my box.

Each line is pulled up to +5 volts with a 10k resistor—the high value was chosen to minimize the load on the 488 bus. The switches can override any line when they are closed to ground. Though the PET doesn't use all the IEEE 488 lines, future machines will—so I put them all in my box.

If you build this box, don't use the PET's +5 volts from the tape port—the LEDs draw 170 mA, which is too much for the PET. Provide a connector to the PET's IEEE port and a male and female IEEE connector. This lets you interpose the IEEE Blinkin Lites between the PET and an instrument.

I mounted a 5×7 inch perfboard with 0.10 inch holes into a standard breadboard box and placed a label near each switch/LED combination to identify the IEEE lines. The three ICs are the 7404s used to drive the LEDs. The cable leads to a homemade junction with a PET connector and IEEE male and female connectors. A mini phono jack connects to a separate +5 volt supply (see Fig. 6).

When you plug in the IEEE Blinkin Lites, the LEDs will show the state of the lines—an LED that is off indicates a low line, which is true; an on LED indicates high, which is false.

The IEEE 488 Lines

The IEEE 488 is composed of 16 lines. Eight are for transfer of data, five are for bus management and three are for handshaking. The eight data lines are labeled DIO1 through DIO8, with the most significant bit (MSB) on DIO8. The 488 bus can transfer one byte at a time and is sometimes called byte-parallel.

The five bus-management lines in various combinations and sequences provide many bus facilities, most of which are rarely used:

EOI—End of Message. When a group of bytes is sent via the DIO lines, EOI is made true on the last byte to indicate that the message is completed. This is optional, and many instruments send the ASCII characters CR and LF as data instead. Check your instrument's manual.

IFC—Interface Clear. When this line is true, all instruments disconnect to a defined state. (This usually is unaddressed and untalked.) When you turn on the PET, IFC is true for about 100 ms. If the PET is reset, IFC will again be true.

SRQ—Service Request. This permits an instrument to signal that it needs attention . . . and the device in charge of the bus must find out what it needs. The PET has this line as an input, but it takes some programming effort to use SRQ; most instruments don't use SRQ.

REN—Remote Enable. Most IEEE instruments have front panels that permit stand-alone operation—that is, they work as ordinary instruments when the 488 bus isn't connected. REN lets the instrument disconnect from the bus and be controlled from its front panel.

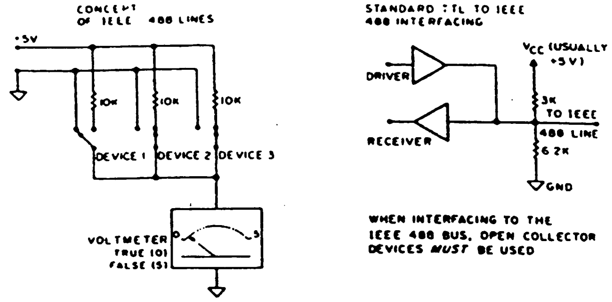


Fig. 4. IEEE 488 equivalent circuits. The lower circuit is the standard method of connecting TTL logic to the 488 bus. The driver must be an open collector and able to sink at least 48 mA at .4 volts and source 5.2 mA at 2.4 or more volts. The PET uses MC 3446P bidirectional line interface ICs for this function.

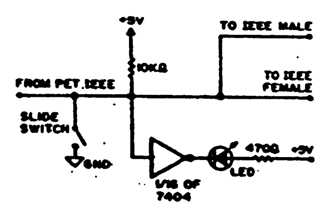


Fig. 5. IEEE "Blinkin Lites" circuit. Each IEEE line uses one copy of this circuit.

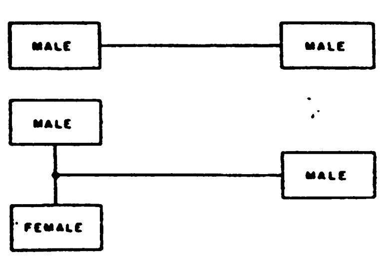


Fig. 3. Convenient cable configurations for the IEEE 488 bus.

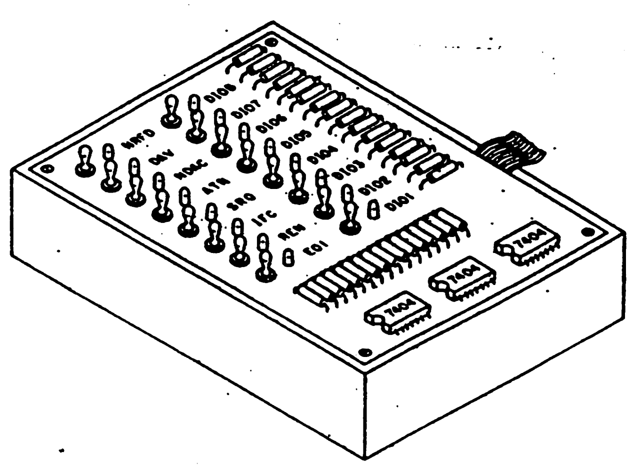


Fig. 6. Sketch of the "Blinkin Lites."

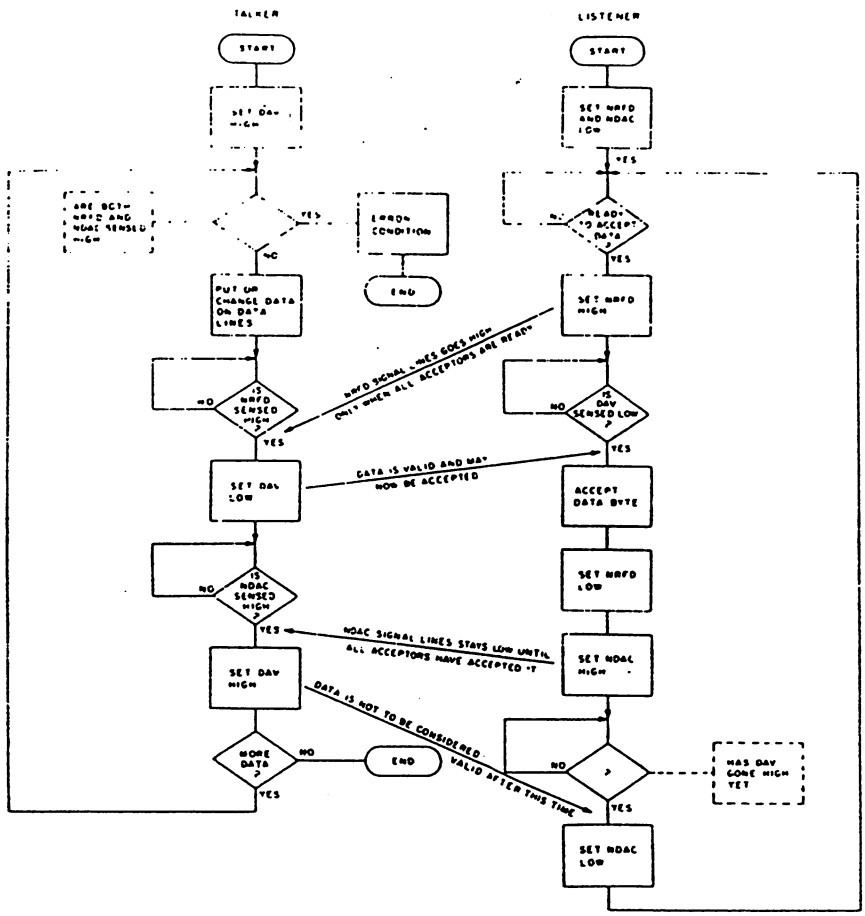


Fig. 7. The IEEE 488 handshake reproduced from Electronics, *Nov. 14, 1974, p. 98, as reproduced in HP part #5952-0058.*

The PET's REN line is always grounded.

ATN—Attention. This is the most relevant line for this article. It tells the device whether to regard the data on the DIO lines as a command or as data. When ATN is true, the byte on the DIO lines is a command. When ATN is false, DIO is seen as data.

The three handshake lines are used to pass bytes on the DIO lines. When a byte is transferred, the slow devices will keep one or more of the handshake lines true until they are finished. This ensures that data is passed at the speed of the slowest device and isn't lost. The handshake lines are:

DAV (Data Valid)—When this line is true, the data on the DIO lines is correct and the receiving instruments can pick up the byte.

NRFD (Not Ready For Data)—When a receiving device is busy or is still processing prior data, it will make NRFD true, which stops data transfers.

NDAC (Not Data Accepted)—When the data is on the DIO lines, the receiving devices keep NDAC true until all of them have read the data byte. Note that the handshake lines don't care whether the data is a command or not; every byte of data or command has to undergo the handshake sequence.

The Handshake

For data transfer, one device is the "talker," which provides the data or commands for transfer. The recipients, or "listeners," pick up the data, and more than one device may listen at the same time. The handshake specifies exactly how the data transfer is accomplished.

Fig. 7 shows a flowchart of the handshake sequence. When the first event, NRFD, goes false, this tells the talker that all of the listeners are now ready to receive a new data byte. The slowest listener is the last one to release NRFD, which will go high.

Next, the talker puts the data byte on the DIO lines and waits briefly to let the signals settle (usually about 10 µs). Once the data is on the DIO lines, NRFD is checked by the talker; if it is false, the talker sets DAV to true. The listeners now know that the new data is ready for pickup. (If NRFD is true, the talker waits until it goes false.)

The first listener that detects DAV true now sets NRFD true, and all of the listeners pick up the data byte from the DIO lines. Up to now, NDAC has been true, and as each listener gets its byte, it releases NDAC. NDAC goes false when all the listeners have the data. The talker waits for NDAC to go false, and when it does, the talker sets DAV to false. The listeners then make NDAC true, and the entire handshake sequence begins again.

Since a device is either a listener, talker or not addressed, Fig. 7 is broken into two flowcharts: one for the talker and one for the listener. A listener will start the handshake with NRFD and NDAC true, while the talker checks these. If both are false—the listener isn't there—an error condition exists.

Commands and Messages

When ATN is true, any data on DIO is seen as a command. Fig. 8 shows the entire ASCII set of 128 characters devoted to IEEE 488 commands.

The ASCII codes 32 through 62 (all numbers in decimal) designate the listen address for a device. Most IEEE-488-compatible devices have a five-position DIP switch next to the 488 connector set to the device's address, a number from 0 to 31. (Note: For the PET, use 4-15.) When the listen address is sent with ATN true and this address matches the device's address, the device will now be addressed to listen and will accept any data sent with ATN false.

If the device is supposed to send data, the talk address—from ASCII codes 64 through 94—will be used instead. The device (if with matching address) will now send data bytes to the bus.

If the device's address (by the switches) is number 7, the listen address value will be 32 + 7, or 39 (apostrophe). The talk address will be 64 + 7, or 71 (letter G). Notice that bits 5-7 designate talk or listen, and bits 0-5 designate the address. Address 31 is reserved for two special commands. Although you can set the switches on a device to 31, it won't operate with this setting.

One instrument must provide these talk and listen addresses. This device is the controller, and the PET is always the controller. The controller can talk and listen too, but only the controller can set ATN true.

Two of the ASCII codes, 63 and 95, serve as "universal" commands. The 63 code is known as "unlisten" and tells all addressed devices to stop listening to the bus. This is faster than trying to tell the devices one at a time to stop listening. The 95 code, "untalk," stops all data transmitters (talkers).

When a message—or a group of data bytes—is sent on the 488 bus, the controller sets ATN true and sends a listen address; the controller sets ATN true and sends a talk address; the talker puts data on the bus, and the listener picks it up. When the talker is finished, it may set EOI true on the last byte or send CR LF as the last bytes. The controller now sets ATN true and sends untalk (UNT) and unlisten (UNL), which reset the two devices.

In many cases, the controller—in this case, the PET—does the talking or listening. The controller can make everything stop by either setting IFC true or setting ATN true and putting UNT on the bus. Since UNT has its five lowest significant bits true, the active low operation of the IEEE lines overrides whatever data is present. (In normal operation of the bus, the controller doesn't have to take these drastic measures.)

In some cases, a device will have a secondary address, which permits more than 31 effective addresses on the bus. For example, the Commodore printer might be set as device 4. To control internal functions, secondary addresses select the function in use. (See Commodore's "PET Communication with the Outside World," p. 19.) If a secondary address is in use, it is sent immediately after the talk or listen address, known as the primary address, with ATN true.

Several of the bus-management lines, such as SRQ, EOI, REN and IFC, serve special functions. Many instruments do respond to these, and often the response depends upon the instrument.

When ATN is low, about half the ASCII code is devoted to special commands, which come in defined sequences whose definition takes about two-thirds of the formal IEEE 488 specification. Most instruments use only a few of these.

Table 1. All PET I/O lines.

IEEE 488 PIA (6520) — ADDRESS: $ E820 59424

PA0	IEEE Data In	1	PB0	IEEE Data Out	1
PA1	" "	2	PB1	" "	2
PA2	" "	3	PB2	" "	3
PA3	" "	4	PB3	" "	4
PA4	" "	5	PB4	" "	5
PA5	" "	6	PB5	" "	6
PA6	" "	7	PB6	" "	7
PA7	" "	8	PB7	" "	8
CA1	ATN In		CB1	SRQ In	
CA2	NDAC Out		CB2	DAV Out	

MULTILINE INTERFACE MESSAGES ISO-7 BIT CODE REPRESENTATION
(SENT AND RECEIVED WITH ATN=1)

b4	b3	b2	b1	ROW \ COL	0 (b7 b6 b5 = 000)	MSG ①	1 (001)	MSG	2 (010)	MSG	3 (011)	MSG	4 (100)	MSG	5 (101)	MSG	6 (110)	MSG	7 (111)	MSG
0	0	0	0	0	NUL		DLE		SP	MLA ASSIGNED TO DEVICE	0	MLA ASSIGNED TO DEVICE	@	MTA ASSIGNED TO DEVICE	P	MTA ASSIGNED TO DEVICE		MEANING DEFINED BY PCG CODE	p	MEANING DEFINED BY PCG CODE
0	0	0	1	1	SOH	GTL	DC1	LLO	!		1		A		Q		a		q	
0	0	1	0	2	STX		DC2		"		2		B		R		b		r	
0	0	1	1	3	ETX		DC3		#		3		C		S		c		s	
0	1	0	0	4	EOT	SDC	DC4	DCL	$		4		D		T		d		t	
0	1	0	1	5	ENQ	PPC③	NAK	PPU	%		5		E		U		e		u	
0	1	1	0	6	ACK		SYN		&		6		F		V		f		v	
0	1	1	1	7	BEL		ETB				7		G		W		g		w	
1	0	0	0	8	BS	GET	CAN	SPE	(		8		H		X		h		x	
1	0	0	1	9	HT	TCT	EM	SPD	)		9		I		Y		i		y	
1	0	1	0	10	LF		SUB		*		:		J		Z		j		z	
1	0	1	1	11	VT		ESC		+		;		K		[		k		{	
1	1	0	0	12	FF		FS		,		<		L		\		l		|	
1	1	0	1	13	CR		GS		-		=		M		]		m		}	
1	1	1	0	14	SO		RS		.		>		N		^		n		~	
1	1	1	1	15	SI		US		/		?	UNL	O		_	UNT	o		DEL	

Columns 0 MSG: ADDRESSED COMMAND GROUP (ACG); column 1 MSG: UNIVERSAL COMMAND GROUP (UCG); columns 2–3: LISTEN ADDRESS GROUP (LAG) ④; columns 4–5: TALK ADDRESS GROUP (TAG); columns 6–7: SECONDARY COMMAND GROUP (SCG). ACG, UCG, LAG and TAG: PRIMARY COMMAND GROUP (PCG). BITS ②.

NOTES: ① MSG=INTERFACE MESSAGE
② b_1 = DIO1 .. b_7 = DIO7
③ REQUIRES SECONDARY COMMAND
④ DENSE SUBSET (COLUMN 2 THROUGH 5)

Fig. 8. IEEE 488 command set reproduced from the IEEE Standard 488-1975/ANSI MC 1.1-1975, *p. 77.*

Flipping Bits

The PET ultimately communicates to the rest of the world by the screen and some interface chips—two 6520s and one 6522. (For the specs on these chips, contact MOS Technology.) The 6520 and 6522 chips can only drive one TTL load, so the PET's IEEE lines are connected to some buffer chips to provide the currents needed in the IEEE 488 bus.

Table 1 indicates all of the PET's I/O line assignments as a reference. The PET utilizes all 60 I/O lines as shown here. Most of the IEEE lines are buffered with MC 3446P bidirectional line driver chips to provide the IEEE current requirements. SRQ is an input only and connects directly to the 6520 chip. IFC is buffered with a NAND and some resistors to the IEEE specification.

Table 1 reveals some interesting irregularities concerning the IEEE 488 bus: If EOI is true, the PET's display is turned off. (Programs that PEEK and POKE the display area in memory can use this to avoid snow.) Later-model PETs don't have this problem. REN isn't listed; the PET's REN line is wired to ground (true). IFC is not shown. The PET's IFC is connected to the power-on one-shot, which sets IFC true for about 100 ms when the PET is turned on. If you reset the PET by grounding the RES line, IFC may not go true. A better approach is to trigger the power-on one-shot by inserting a switch between power and the 555's power pin. The SRQ line is an input only. The PET's firmware does not use SRQ, so you have to program it directly.

In a 650x-based system, all I/O is seen as a set of memory addresses. This means that BASIC's PEEK and POKE can be used to control the IEEE 488 lines. Table 2 indicates the addresses and bits involved for the PET's IEEE lines. In most cases, a direct PEEK or POKE will do. Two lines, ATN in and SRQ in, require a more complex sequence. These are connected to CA1 and CB1 of a 6520, which set flag bits in the Interrupt Flag register. Resetting these bits requires a memory access to the DIO data register.

Table 3 lists the specific PEEKs and POKEs to individually sense or modify the IEEE lines. In many cases the PEEK or POKE values can be ANDed or ORed together to do several operations at once. If you have built the IEEE Blinkin Lites, try a

```
KEYBOARD PIA (6520)                        ADDRESS: $ E810     59408

PA0  Keyboard Row Select, LSB             PB0  Keyboard Column A
PA1     "       "     "                   PB1     "        "   B
PA2     "       "     "                   PB2     "        "   C
PA3     "       "     "     , MSB         PB3     "        "   D
PA4  Switch, Cassette #1                  PB4     "            E
PA5     "         "   #2                  PB5     "        "   F
PA6  EOI In                               PB6     "        "   G
PA7  Diagnostic Jumper                    PB7     "        "   H

CA1  Read, Cassette #1                    CB1  Video Horiz Sync In
CA2  Screen Blank & EOI Out               CB2  Motor, Cassette #1

The Diagnostic LED will light if PA0-High, PA1-High, PA2-Low, PA3-High.

USER PORT VIA (6522)                       ADDRESS: $ E840     59456

PA0  User Port LSB                        PB0  NDAC In
PA1    "    "                             PB1  NRFD Out
PA2    "    "                             PB2  ATN Out
PA3    "    "                             PB3  Write, Both Cassettes
PA4    "    "                             PB4  Motor, Cassette # 2
PA5    "    "                             PB5  Video Horiz Sync In
PA6    "    "                             PB6  NRFD In
PA7    "    "   MSB                       PB7  DAV In

CA1  User Port Handshake                  CB1  Read, Cassette #2
CA2  Characters ROM Select                CB2  User Port Handshake

CA2 selects the MSB of the characters ROM, selecting the PET's
graphics or lower case characters for the display.
```

few of these PEEKs and POKEs to see how they work.

When I was flipping bits with PEEK and POKE for the IEEE lines, I was confused each time I had to figure out the decimal numbers for each changed bit. Perhaps it would be easier to display a byte of memory on the PET's screen in a "front panel" format with simulated LEDs for each bit and some simple keyboard commands to change bits and addresses. Memory Monitor (see Listing 1) does this.

When Memory Monitor is loaded and run, and the first page of instructions is read, the display in Fig. 9 is shown. A box with four parts appears in the middle of the screen with the title Memory Monitor placed above the box. Left of the box is a marker, >>, which indicates the part of the box accessible by the keyboard.

The top of the box shows the address of a memory location in decimal. If you press SPACE, the address will be erased, and a new number can be entered. Pressing number keys enters a new address, and a reverse-field cursor appears.

When a cursor isn't on the screen, pressing RETURN will move the marker to the next part of the box. (The second part in the box indicates the bit numbers and is skipped by the marker.)

The third part of the box displays a front panel made of sclid or hollow "balls" (or "LEDs"). This shows the eight bits of the byte under investigation. The numbers above the "LEDs" indicate the bit numbers, 7 the MSB and 0 the LSB. To change the byte, enter 0 or 1 (or Shift-Q and Shift-W), and the cursor will appear. Pressing RETURN enters the value.

The fourth part of the box is the value of the byte in decimal and is entered in the same way

Listing 1. Memory Monitor.

```
10 PRINT"clr sp sp sp sp sp sp sp --> MEMORY MONITOR <--
20 PRINT"dn sp sp THIS PGM DISPLAYS A LOCATION IN THE
30 PRINT"PET'S MEMORY IN BOTH DECIMAL AND IN A
40 PRINT"'FRONT PANEL' FORMAT.
50 PRINT"dn sp sp YOU CAN CHANGE THE ADDRESS OR VALUE
60 PRINT"BY ENTERING A NEW VALUE WHEN THE '>>'
70 PRINT"MARKER IS NEXT TO THE ITEM YOU ARE
80 PRINT"CHANGING.
90 PRINT"dn sp sp PRESS 'RETURN' TO ENTER THE CHANGE
100 PRINT"OR TO MOVE THE MARKER.
110 PRINT"dn sp sp THE PGM CONSTANTLY PEEKS THE LOCATION
120 PRINT"WHEN YOU AREN'T CHANGING A VALUE. IF
130 PRINT"YOU CHANGE THE ADDRESS, THE PGM
140 PRINT"WILL SHOW THE NEW VALUE. IF YOU CHANGE
150 PRINT"A VALUE, IT IS POKED INTO MEMORY.
160 PRINT"dn sp sp 'H' WILL GIVE YOU SOME HELP FOR EACH
170 PRINT"ITEM.
190 PRINT"dn PRESS ANY KEY TO START
195 GETA$:IFA$=""THEN195

200 REM DRAW DISPLAY FORMAT
210 PRINT"clr dn dn dn dn dn dn";
220 D1$="@ @ @ @ @ @ @ @ @ @ @ @ @ @ @"        (15 shift-@)
230 D3$="rt rt rt rt rt rt rt rt rt rt"        (10 rt's)
240 PRINT"rt rt rt rt rt rt rt --> sp MEMORY MONITOR sp<--"
250 PRINTD3$"0"D1$"."
255 PRINTD3$"] ADDRESS:sp sp sp sp sp sp sp sp ]"   (] is a vertical line)
260 PRINTD3$"+"D1$"3"
265 PRINTD3$"] 7 sp 6 sp 5 sp 4 sp 3 sp 2 sp 1 sp 0 ]"
270 PRINTD3$"+ @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 3"
280 PRINTD3$"] sp ] sp ] sp ] sp ] sp ] sp ] sp ] sp ]"
290 PRINTD3$"+ @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 3"
300 PRINTD3$"] DECIMAL: sp sp sp sp sp sp sp ]"
310 PRINTD3$"-"D1$"="
320 PRINT"dn dn dn"
```

NOTE: For Lines 200–320 see Fig. 9.

```
400 REM IDLING PROGRAM
410 AD=59471:PT=1
500 REM DISPLAY ADDRESS
510 GOSUB 1000
520 REM DISP PANEL LITES
525 DT=PEEK(AD)
530 GOSUB 1200
540 REM DISP DECIMAL
550 GOSUB 1400
560 REM DISP PTR
570 GOSUB 1600
580 REM GET CHAR
590 GET A$
600 IF A$="" THEN 500
610 IF A$=CHR$(13) THEN 700
620 FG=0:GOSUB 2500
630 IF FG=0 THEN 510
640 GOTO 210
700 REM BUMP PTR
710 GOSUB 1800
720 GOTO 510

1000 REM DISP ADDR
1010 PRINT"hm dn dn dn dn dn dn dn"
1020 V$=STR$(AD)+"sp sp sp sp sp sp sp"
1030 V$=MID$(V$,2,6)
1040 PRINT TAB(20);V$
1050 RETURN

1200 REM DISP PANEL
1210 PRINT"dn dn dn"TAB(11);
1220 VT=DT:DV=128
1230 FOR J=1 TO 8
1240 IF VT/DV< 1 THEN 1260
1250 PRINT" Q rt";:VT=VT-DV:GOTO 1300
1260 PRINT" W rt";
1300 DV=DV/2
1310 NEXT J

1400 REM DISP DECIMAL
1410 PRINT"dn"
1420 V$=STR$(DT)+" sp sp sp sp sp sp sp sp "
1430 V$=MID$(V4,2,6)
1440 PRINT TAB(20);V$
1450 RETURN

1600 REM DISP PTR
1610 PRINT"hm dn dn dn dn dn dn dn"
1620 IF PT >1 THEN 1640
1630 PRINT TAB(8)">>";:RETURN
1640 PRINT"dn dn dn"
1650 IF PT >2 THEN 1670
1660 GOTO 1630
1670 PRINT"dn":GOTO 1630

1800 REM BUMP PTR
1810 GOSUB 1600
1820 PRINT"rt rt sp sp"
1830 PT=PT+1: IF PT >3 THEN PT=1
1840 GOSUB 1600
1850 RETURN

2000 RETURN                    (This line probably isn't needed)

2500 REM CHANGE ITEM
2510 ON PT GOSUB 3000,3500,4000
2520 RETURN

3000 REM CHANGE ADDR
3010 IF A$="H" THEN GOSUB 4500:RETURN
```

VALUES FOR INPUTS: IEEE LINE	ADDRESS (HEX)	ADDRESS (DECIMAL)	BIT	VALUES FOR OUTPUTS: IEEE LINE	ADDRESS (HEX)	ADDRESS (DECIMAL)	BIT
DIO 1	E820	59424	0	DIO 1	E822	59426	0
DIO 2	E820	59424	1	DIO 2	E822	59426	1
DIO 3	E820	59424	2	DIO 3	E822	59426	2
DIO 4	E820	59424	3	DIO 4	E822	59426	3
DIO 5	E820	59424	4	DIO 5	E822	59426	4
DIO 6	E820	59424	5	DIO 6	E822	59426	5
DIO 7	E820	59424	6	DIO 7	E822	59426	6
DIO 8	E820	59424	7	DIO 8	E822	59426	7
EOI	E810	59408	6	EOI	E811	59409	3
IFC	----	-----	-	IFC	----	-----	-
SRQ	E823	59427	7	SRQ	----	-----	-
REN	----	-----	-	REN	----	-----	-
ATN	E821	59425	7	ATN	E840	59456	2
DAV	E840	59456	7	DAV	E823	59427	3
NRFD	E840	59456	6	NRFD	E840	59456	1
NDAC	E840	59456	0	NDAC	E821	59425	3

Table. 2 Addresses and bits for the IEEE 488 lines.

as the address.

If you press RETURN several times, the marker rotates through the three accessible parts of the box. To recall how to enter a value, press the letter H, which clears the screen and provides instructions.

The Memory Monitor eased the tedium and frustration of checking the PEEKs and POKEs used in the IEEE 488 memory locations. I have made Memory Monitor simple to use, and I consider it a good example of user-oriented programming.

Doing It the Hard Way

With direct access to the PET's IEEE 488 lines, you can use PEEK and POKE to operate an IEEE instrument "by hand." This is probably more difficult than using the IEEE Blinkin .ites box to communicate switch by switch because it takes more keystrokes to change a bit with POKE.

The next step is to write a BASIC program that performs the required IEEE 488 operations directly. Though the PET has these "built in," there are a few advantages to doing the whole thing in BASIC.

Everything goes slowly. As events happen, there is a chance of seeing them as they go by.

BASIC is accessible. If the PET or your instrument decides that the sky's the limit, pressing the STOP key can illuminate where the difficulties lie. The PET's built-in IEEE 488 services

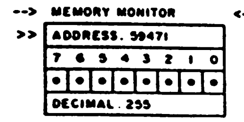


Fig. 9. Listing 1's initial display.

```
3020 V1=AD
3030 GOSUB 5000
3040 IF V2< 0 THEN RETURN
3050 IF V2>65535 THEN RETURN
3060 AD=V2:RETURN

3500 REM CHANGE BINARY VALUE
3510 IF A$="H"THEN GOSUB 4600: RETURN
3520 V1=DT
3530 GOSUB 5500
3540 IF V2< 0 THEN RETURN
3550 IF V2>255 THEN RETURN
3560 DT=V2:POKE AD,DT:RETURN

4000 REM CHANGE VALUE
4010 IF A$="H" THEN GOSUB 4500: RETURN
4020 V1=DT
4030 GOSUB 5000
4040 IF V2< 0 THEN RETURN
4050 IF V2> 255 THEN RETURN
4060 DT=V2:POKE AD,DT: RETURN

4500 PRINT"clr sp sp TYPE IN THE NEW NUMBER AND PRESS
4505 FG=1
4510 PRINT"RETURN. PRESS 'x' TO ABORT & NOT MAKE
4515 PRINT"THE CHANGE.
4520 PRINT" sp sp PRESS SPACE TO ERASE REST OF NUMBER.
4530 PRINT"dn sp sp PRESS ANY KEY
4540 GETA$: IFA$=""THEN 4540
4550 RETURN

4600 PRINT"clr sp sp ENTER '1' OR ' Q ' TO SET A BIT, AND
4610 PRINT"'0' OR ' W ' TO RESET A BIT. PRESS
4620 PRINT"RETURN WHEN DOEN.
4625 PRINT" sp sp PRESS SPACE TO SKIP A BIT.
4630 PRINT"dn sp sp PRESS ANY KEY
4640 GETA$:IF A$="" THEN 4540
4650 RETURN

5000 REM NUMERIC ENTRY
5010 REM POS CURSOR
5020 PRINT TAB(20);
5030 REM MAKE DISP STR
5040 D$=MID$(STR$(V1),2)+"sp sp sp sp sp sp sp"
5050 D$=LEFT$(D$,6)
5060 REM SET RVS PTR & RETURN VALUE
5070 PC=1:V2=-1
5080 REM SEE INPUT & ACT
5090 IF A$="X" THEN RETURN
5100 IF A$=CHR$(13) THEN V2=VAL(D$):RETURN
5110 IF A$<>" sp " THEN 5120
5112 IF PC=1 THEN D$="sp sp sp sp sp sp":GOTO 5210
5114 D$=LEFT$(D4,PC-1)+"sp sp sp sp sp sp":D$=LEFT$(D$,6)
5118 GOTO 5210
5120 IF A$< "0" OR A$ >"9" THEN 5210
5125 REM REMAKE STRING
5130 DX$=D$:D$=""
5140 FOR J=1 TO 6
5150 IF PC=J THEN D$=D$+A$:GOTO 5170
5160 D$=D$+MID$(DX$,J,1)
5170 NEXT J
5180 PC=PC+1: IF PC >7 THEN PC=1

5200 REM DISPLAY RESULT & RESTORE CURSOR
5210 FOR J=1 TO 6
5220 IF J=PC THEN PRINT "rvs";
5230 PRINT MID$(D$,J,1);
5240 IF J=PC THEN PRINT "off";
5250 NEXT J:PRINT"lft lft lft lft lft lft";
5260 GET A$: IF A$="" THEN 5260
5270 GOTO 5090

5500 REM BINARY ENTRY
5510 PRINT TAB(11);
5520 FOR J= 1 TO 8
5525 V1=V1/2
5530 IF V1=INT(V1) THEN D$=" W "+D$: GOTO 5540
5535 D$=" Q "+D$
5540 V1 = INT(V1)
5550 NEXT J
5570 REM SET RVS PTR
5580 PC=1:V2=-1
5590 REM LOOK AT INPUT
5600 IF A$="X" THEN RETURN
5605 IF A$=CHR$(13)THEN 5780
5610 IF A$="sp" THEN 5715
5620 IF A$="1" OR A$=" Q " THEN A$=" Q ":GOTO 5660
5630 IF A$="0" OR A$=" W " THEN A$+" W ":GOTO 5660
5640 GETA$:IFA$=""THEN 5640
5650 GOTO 5600
5660 REM REMAKE STRING
5670 DX$=D$:D$=""
5680 FOR J= 1 TO 8
5690 IF PC=J THEN D$=D$+A$: GOTO 5710
5700 D$=D$+MID$(DX$,J,1)
5710 NEXT J
5715 PC=PC+1:IF PC>8 THEN PC=1
5720 REM DISP & FIX CURSOR
5730 FOR J= 1 TO 8
5735 IF J = PC THEN PRINT "rvs";
5740 PRINT MID$(D$,J,1)"rt";
5745 IF J=PC THEN PRINT "off";
5750 NEXT J:PRINT" lft lft lft lft lft lft lft lft
                 lft lft lft lft lft lft lft lft";  (16 lft's)
5760 GOTO 5640
5770 REM MAKE VALUE
5780 V2=0:FORJ=1 TO 8
5785 V2=V2*2
5790 IF MID$(D$,J,1)=" W " THEN 5810
5800 V2=V2+1
5810 NEXT J
5820 RETURN
```

are mostly invisible, and there's often no way to find out why something went wrong.

Everything is under control. It is simple enough to display every step with suitable messages to the screen. If necessary, you can insert a GET loop to make the PET wait until a key is pressed before proceeding.

Changes are easy.

It's an educational experience—those who must learn the "nuts and bolts" of the IEEE bus will find a BASIC emulator useful.

I constructed the BASIC 488 program (see Listing 2) to provide the following essential services: put the PEEK and POKE values into variable form for reasonably fast execution and to simplify debugging with direct commands; do most of the PEEKs and POKEs for line control as short subroutines; provide the listen and talk handshake sequences for one byte and display their progress; provide a way to send and receive strings to a device on the bus; set the program up as a skeleton onto which you can add specific programs to suit changing needs.

Table 4 indicates the subroutines and variables used in the BASIC 488 program. Load these subroutines and then add the code you need for your devices. Some devices, such as those by Commodore, may not follow the IEEE time standard, and the BASIC 488 program will not be fast enough to prevent time-outs.

I built the program from the bottom up, starting with subroutines 1500 and the series starting at 9000. Subroutine 1500 sets up the essential variables. A1-7 are the addresses of the PEEK/POKE locations; M0-M7 and N0-N8 are AND and OR masks to extract bits 0-7 from a location (or to set the desired bits); 01-07 are the original values for addresses A1-A7. (POKE A1,01, for example, will restore location A1 to the PET's power-on value, which helps you to recover from disasters.)

The variables H1 to H6 are the sense values for the IEEE lines. For example, if H1 is 1, the DAV line is true. If H1 is zero, DAV is false.

When you enter BASIC 488, enter lines 1000-1620 and lines 9000-9640 first. Use the IEEE Blinkin Lites to check that the subroutines in the 9000 series function correctly. First, GOSUB 1000 in direct mode to set things up. Then, GOSUB to the section under test and look at the Blinkin Lites to see what happened. A PRINT H1 will inform you of the sensing subroutines' results. Be sure to thoroughly test the 9000 series first!

```
All DIO Lines *
        IN          POKE 59426,255 V = PEEK(59424) V = (NOT(V))AND255
        OUT         V = (NOT(V))AND255. POKE 59426,V

EOI
        IN          V = 1 IF PEEK(59408)AND 64 THEN V = 0
        TRUE OUT    POKE 59409, PEEK(59409) AND 247
        FALSE OUT:  POKE 59409, PEEK(59409) OR 8

REN & IFC — Not Applicable

SRQ**
        IN.         V = 0 IF PEEK(59427) AND 128 THEN V = 1
                    Z = PEEK(59426)
        LO-HI       POKE 59427 PEEK(59427) OR 2
        HI-LO       POKE 59427. PEEK(59427) AND 253

ATN**
        IN:         V = 0 IF PEEK(59425) AND 64 THEN V = 1
                    Z = PEEK(59424)
        LO-HI:      POKE 59409. PEEK(59409) OR 2
        HI-LO:      POKE 59409 PEEK(59409) AND 253

        TRUE OUT    POKE 59456. PEEK(59456) AND 251
        FALSE OUT   POKE 59456. PEEK(59456) OR 4

DAV
        IN          V = 1: IF PEEK(59456) AND 128 THEN V = 0
        TRUE OUT:   POKE 59427. PEEK(59427) AND 247
        FALSE OUT   POKE 59427. PEEK(59427) OR 8

NRFD
        IN:         V = 1: IF PEEK(59456) AND 64 THEN V = 0
        TRUE OUT.   POKE 59456. PEEK(59456) AND 253
        FALSE OUT:  POKE 59456. PEEK(59456) OR 2

NDAC
        IN          V = 1. IF PEEK(59456) AND 1 THEN V = 0
        TRUE OUT:   POKE 59425. PEEK(59425) AND 247
        FALSE OUT.  POKE 59425. PEEK(59425) OR 8
```

*The extra parenthesis in the complementation of V is required, for the PET evaluates AND before NOT

**The HI-LO or LO-HI determines which transition the CA/CB1 inputs will respond to. Set the HI-LO or LO-HI before doing the IN. line. The Z = PEEK resets the flag bit. Be sure to reset the flag bit before checking the first time

SRQ OUT is not available on the PET

Table 3. PEEKs and POKEs for the IEEE 488 lines.

Listing 2. BASIC 488 program.

```
1000 REM **** IEEE 488 ****
1005 REM   GREGORY YOB, JAN 1979
1010 REM   BOX 354, PALO ALTO CA 94301
1015 REM
1020 REM THESE ROUTINES PERMIT DIRECT
1025 REM MANIPULATION OF THE PET IEEE
1030 REM 488 BUSS LINES AND (SLOW)
1035 REM IEEE 488 COMMAND AND DATA
1040 REM TRANSFERS
1045 REM
1500 REM -- INITIALIZATION --
1510 RESTOR:READ A1,A2,A3,A4,A5,A6,A7
1520 DATA 59424,59426,59425,59427,59408,59456,59409
1530 READ M0,M1,M2,M3,M4,M5,M6,M7
1540 DATA 1,2,4,8,16,32,64,128
1550 READ N0,N1,N2,N3,N4,N5,N6,N7
1560 DATA 254,253,251,247,239,223,191,127
1570 READ N8
1580 DATA 255
1590 READ O1,O2,O3,O4,O5,O6,O7     (Each of these is Letter O)
1600 DATA 255,266,60,60,249,255,60
1610 DEF FNF(X)=(NOT(X))AND255
1620 RETURN

7000 PRINT"clr GET MESSAGE"
7010 PRINT"dn PRESS KEY TO START"
7020 GETA$:IFA$=""THEN RETURN      ("" is an empty string)
7030 D2=FNF(DV+64):GOSUB9450:GOSUB8500:GOSUB9470
7040 B$=""
7050 GOSUB 8000:IF FNF(D1)=13THEN7070
7060 B$=B$+CHR$(FNF(D1)):GOTO7050
7070 GOSUB8000:REM LF BUCKET
7080 PRINT"dn dn MESSAGE IS: sp"B$
7090 RETURN

7500 PRINT"clr SEND MESSAGE"
7510 INPUT"dn dn MESSAGE:";C$
7520 D2=FNF(DV+32):GOSUB9450:GOSUB8500:GOSUB9470
7530 FOR J=1 TO LEN(C$)
7540 D2=FNF(ASC(MID$(C$,J)))
7550 GOSUB8500:NEXTJ
7560 PRINT"dn dn MESSAGE SENT: sp"C$
7570 RETURN

8000 PRINT"clr LISTEN HANDSHAKE dn"
8010 GOSUB9350:GOSUB9250:GOSUB9370
8020 PRINT" sp NRFD TRUE dn":PRINT" sp NDAC TRUE"
     :PRINT" sp NRFD FALSE"
8030 PRINT"WAITING FOR DAV TRUE"
8040 GETA$:IFA$<>""THENPRINT"--FORCED":GOTO8060
8050 GOSUB9100:IFH1=0THEN8040
8060 GOSUB9000:PRINT"dn spDATA:"FNF(D1)CHR$(FNF(D1))
8070 GOSUB9350:GOSUB9270
8080 PRINT"dn sp NRFD TRUE":PRINT" sp NDAC FALSE"
8090 PRINT"WAITING FOR DAV FALSE"
8100 GETA$:IFA$<> ""THENPRINT"--FORCED":GOTO8120
8110 GOSUB9100:IFH1=1THEN8100
8120 GOSUB9250
8130 PRINT"dn sp NDAC TRUE"
8140 RETURN

8500 PRINT"clr TALK HANDSHAKE"
8510 GOSUB9170
8520 PRINT"dn sp DAV FALSE"
8530 GOSUB9200:GOSUB9300
8540 IF H1+H2>0 THEN 8570
8550 PRINT"dn>> sp ERROR STATE-PRESS KEY TO FORCE"
8555 PRINT"NOTE: MAKE NRFD, NDAC TRUE"
8560 GETA$;IFA$=""THEN8560
8570 GOSUB9050
8580 PRINT"dn DATA ON LINE:"FNF(D2)CHR$(FNF(D2))
```

Nothing else will work if these don't!

If all else fails, refer to Tables 1, 2 and 3 and try a few direct PEEKs and POKEs to ensure that the IEEE lines are functional.

Add lines 8000–8140 and lines 8500–8690, which you can check by attaching the 488 Blinkin Lites and carefully tracing through the handshake flowchart in Fig. 7. Again, it is essential to be sure these routines work correctly. An additional benefit is that you will learn the handshake sequence in detail.

Note that the data transferred, D1 or D2, must be complemented with the FNF function as it enters or leaves the IEEE bus. In some of the waiting loops, such as lines 8030–8050, a GET A$ check is inserted. If the instrument hangs up, pressing a key will force the handshake to proceed, and a suitable message will appear on the screen. As the handshakes proceed, their progress is reported to the screen for your reference.

Next, add lines 7000–7570. These routines require a device address, DV, to function correctly. Subroutine 7000 will fetch a message from a device, and subroutine 7500 will send a message. The strings B$ and C$ are used to store the messages.

Most devices will send an EOI along with the last character of their messages. This will turn off the screen. In some cases, you will have to provide an EOI, which will again turn off the screen. To recover, enter:

```
GOSUB 9570 (and RETURN)
```

Another approach is to move the cursor down until the screen scrolls. A scroll turns the screen off, and then on. If you have a 16K PET, the screen will not blink.

Testing the last part via the IEEE Blinkin Lites is tedious. If you have an instrument available, try talking to it! Be sure you know *exactly* what your instrument expects and its responses!

Talking to the HP Clock via BASIC 488

Now that you have checked out BASIC 488 by hand, try it with a real live instrument! I connected the HP clock, loaded BASIC 488 and gave it a try (see Example 1). The clock's front panel shows the reset worked.

These commands can be compressed to one line (see Example 2).

Next, try to read the clock. Address the clock to talk, then read the 14-character message shown in Example 3. If you look at the line DATA: on the display for the Listen Handshake, you can barely see the clock's message. A different version (see Example 4) will pick up the message and leave it later. Below the Listen Handshake display appears the clock's message:
0101000520

The BASIC 488 program has two routines for sending and reading entire strings via the IEEE 488. Subroutine 7000 addresses device DV to talk and read a string. Subroutine 7500 addresses device DV to listen and sends a string. (Note: Routine 7000 reads a string until a carriage return is seen, and then reads one more character. This is because the HP clock ends messages with CR and LF. You might have to change this for your device.)

To reset the clock:
DV = 7:GOSUB 7500
The screen clears and asks for

```
8590 print"dn WAITING FOR NRFD FALSE"
8600 GETA$:IFA$<>""THENPRINT"--FORCED":GOTO8620
8610 GOSUB9300:IFH3=1THEN8600
8620 GOSUB9150.
8630 PRINT"dn sp DAV TRUE"
8640 PRINT"WAITING FOR NDAC FALSE"
8650 GETA$:IFA$<>""THEN8670
8670 GOSUB9170
8680 PRINT"dn sp DAV FALSE"
8690 RETURN

9000 POKEA2,N8:D1=PEEK(A1):RETURN
9050 POKEA2,D2:RETURN
9100 H1=1:IFPEEK(A6)ANDM7THENH1=Ø
9110 RETURN
9150 POKEA4,PEEK(A4)ANDN3:RETURN
9170 POKEA4,PEEK(A4)ORM3:RETURN
9200 H2=1:IFPEEK(A6)ANDMØTHENH2=Ø
9210 RETURN
9250 POKEA3,PEEK(A3)ANDN3:RETURN
9270 POKEA3,PEEK(A3)ORM3:RETURN
9300 H3=1:IFPEEK(A6)ANDM6THENH3=Ø
9310 RETURN
9350 POKEA6,PEEK(A6)ANDN1:RETURN
9370 POKEA6,PEEK(A6)ORM1:RETURN
9400 PRINT"NO ATN LEVEL":STOP
9430 H4=Ø:IFPEEK(A3)ANDM7THENH4=1
9440 ZZ=PEEK(A1):RETURN
9450 POKEA6,PEEK(A6)ANDN2:RETURN
9470 POKEA6,PEEK(A6)ORM2:RETURN
9500 H5=1:IFPEEK(A5)ANDM6THENH5=Ø
9510 RETURN
9550 POKEA7,PEEK(A7)ANDN3:RETURN
9570 POKEA7,PEEK(A7)ORM3:RETURN
9600 REM SRQ NOT OUTPUT
9630 H6=Ø:IFPEEK(A4)ANDM7THENH6=1
9640 ZZ=PEEK(A2):RETURN
```

Entry Points:

SUBROUTINE 1500	Initialization (Must be done first)
SUBROUTINE 7000	Get Message as B$, Requires DV
SUBROUTINE 7500	Put Message C$, Requires DV
SUBROUTINE 8000	Listen Handshake
SUBROUTINE 8500	Talk Handshake
SUBROUTINES 9000ff 9600	IEEE Lines Primitives
9000	Read DIO as D1
9050	Write DIO as D2
9100	Read DAV as H1
9150	Set DAV TRUE
9170	Set DAV FALSE
9200	Read NDAC as H2
9250	Set NDAC TRUE
9270	Set NDAC FALSE
9300	Read NRFD as H3
9350	Set NRFD TRUE
9370	Set NRFD FALSE
9400	Trap for ATN
9430	Check ATN as H4 (if changed)
9450	Set ATN TRUE
9470	Set ATN FALSE
9500	Read EOI as H5
9550	Set EOI TRUE (Screen will blank)
9570	Set EOI FALSE (Screen returns)
9630	Check SRQ as H6 (if changed)

Variables:

PEEK/POKE ADDRESSES		ORIGINAL VALUES	
A1	59424	01	255
A2	59426	02	255
A3	59425	03	60
A4	59427	04	60
A5	59408	05	249
A6	59456	06	255
A7	59409	07	60

Masks:

M0	0000 0001	1	N0	1111 1110	254
M1	0000 0010	2	N1	1111 1101	253
M2	0000 0100	4	N2	1111 1011	251
M3	0000 1000	8	N3	1111 0111	247
M4	0001 0000	16	N4	1110 1111	239
M5	0010 0000	32	N5	1101 1111	223
M6	0100 0000	64	N6	1011 1111	191
M7	1000 0000	128	N7	0111 1111	127
			N8	1111 1111	255

Miscellaneous:

DV	Device Address
A$	Keyboard dummy entry
B$	Message from Device
C$	Message to Device

Functions:

FNF(X)	Returns complement of argument

Table 4. BASIC 488 program notes.

```
RUN
GOSUB [illegible] GOSUB 1900          Get everything ready
PRINT FNF(32 + 7)
   216                                This is the value for D2 as a listen address
GOSUB 9450                            Make ATN true
D2 = 216 GOSUB 8500                   Send listen address via handshake
TALK HANDSHAKE                        The PET responds with the step-by-step
   DAV FALSE                          output handshake and goes successfully
DATA ON LINE 39                       through the entire process
WAITING FOR NRFD FALSE
   DAV TRUE
WAITING FOR NDAC FALSE                The HP Clock's 'addressed' light turns on!
   DAV FALSE
READY
GOSUB 9470                            Make ATN false
PRINT FNF(ASC("R"))                   R resets the clock
   173
D2 = 173:GOSUB 8500                   Send 'R' as data
( . . . . . )                         And this handshakes through OK too
```

Example 1. My dialogue with the HP clock via BASIC 488.

```
D2 = 216 GOSUB9450 GOSUB8500 GOSUB9470 D2 = 173.GOSUB8500
```

Example 2. A one-line command for Example 1.

```
PRINT FNF(64 + 7)                          Find out D2 for talk address
 184
D2 = 184 GOSUB9450 GOSUB8500 GOSUB9470
(                )                         The handshake goes through
FOR J = 1 TO 14:GOSUB 8000:NEXT
(        for 14 times .      )
```

Example 3. The dialogue for reading the clock.

the message (see Example 5).

The Talk Handshake flashes on the screen twice, and the message sent is displayed below:

```
( . . . . .      .
MESSAGE SENT: R
```

The program uses routine 7000 to read the time. Since DV is already set, we don't have to reassign DV = 7 again. See Example 5. Note that there are three spaces between the colon and the first zero. Two of these are from the HP clock, which starts all messages with two blanks.

The BASIC 488 program, though slow to operate, never times-out and lets you control the IEEE 488 bus. This is helpful when you debug a new IEEE device with your PET.

If you are an experienced 6502 programmer, it is simple to translate the BASIC 488 program into a set of machine-language routines. If you do so, I'd like a copy (tape and source). Listing 3 shows a copy of the IEEE handshakes in machine language. (From the *PET User Notes*, PO Box 371, Montgomeryville, PA 18936, Vol. 1, Issue 7, (Nov.–Dec. '78), p. 8. This is a reprint from the Commodore PET Users Club of England.)

The PET handles the IEEE 488 as a file. Part 2 will cover this. ■

```
IEEE Bus Handshake Routine
- Main Program

1800 A200   LDX #00      prepare index register
1802 A9FB   LDA #FB      set ATN low
1804 2D40E8 AND E840
1807 8D40E8 STA E840
180A A928   LDA #28      MLA (28 for this device)
180C 8501   STA 01
180E 208018 JSR 1880     handshake into bus
1811 A908   LDA #08      GET
1813 8501   STA 01
1815 208018 JSR 1880     handshake
1818 A948   LDA #48      MTA
181A 8501   STA 01
181C 208018 JSR 1880     handshake
181F A9FD   LDA #FD      set NRFD low
1821 2D40E8 AND E840        (ready to receive data)
1824 8D40E8 STA E840
1827 A9F7   LDA #F7      and NDAC low also
1829 2D21E8 AND E821
182C 8D21E8 STA E821
182F A904   LDA #04      set ATN high
1831 0D40E8 ORA E840
1834 8D40E8 STA E840
1837 A008   LDY #08      ready to count 8 bytes
1839 20B018 JSR 18B0     handshake data from bus
183C A502   LDA 02       result to A
183E 9D0119 STA 1901,X   store in 1901+X
1841 E8     INX
1842 88     DEY
1843 D0F4   BNE 1839     jump if Y not zero
1845 A9FB   LDA #FB      set ATN low
1847 2D40E8 AND E840
184A 8D40E8 STA E840
184D A902   LDA #02      set NRFD high
184F 0D40E8 ORA E840
1852 8D40E8 STA E840
1855 A908   LDA #08      set NDAC high
1857 0D21E8 ORA E821
185A 8D21E8 STA E821
185D A95F   LDA #5F      UNT
185F 8501   STA 01
1861 208018 JSR 1880     handshake to bus
1864 A904   LDA #04      set ATN high
1866 0D40E8 ORA E840
1869 8D40E8 STA E840
186C CE0019 DEC 1900     decrease counter
186F D091   BNE 1802     jump if not zero
1871 60     RTS          return to BASIC program

Subroutine to Handle
Handshake Into Bus

1880 AD40E8 LDA E840     NRFD ?
1883 2940   AND #40
1885 F0F9   BEQ 1880     jump back if not ready
1887 A501   LDA 01       ready:  get data byte
1889 49FF   EOR #FF      complement it
188B 8D22E8 STA E822     send to bus
188E A9F7   LDA #F7      set DAV low
1890 2D23E8 AND E823
1893 8D23E8 STA E823
1896 AD40E8 LDA E840     NDAC ?
1899 2901   AND #01
189B F0F9   BEQ 1896     jump back if not accepted
189D A908   LDA #08      accepted;  set DAV high
189F 0D23E8 ORA E823
18A2 8D23E8 STA E823
18A5 A9FF   LDA #FF      255 (base 10) into bus
18A7 8D22E8 STA E822
18AA 60     RTS          return to main

Subroutine to Handle
Handshake From Bus

18B0 A902   LDA #02      set NRFD high
18B2 0D40E8 ORA E840
18B5 8D40E8 STA E840
18B8 AD40E8 LDA E840     DAV ?
18BB 2980   AND #80
18BD D0F9   BNE 18B8     jump back if not valid
18BF AD20E8 LDA E820     get data byte from bus
18C2 49FF   EOR #FF      complement
18C4 8502   STA 02       store in $ 0002
18C6 A9FD   LDA #FD      set NRFD low
18C8 2D40E8 AND E840
18CB 8D40E8 STA E840
18CE A908   LDA #08      set NDAC high
18D0 0D21E8 ORA E821
18D3 8D21E8 STA E821
18D6 AD40E8 LDA E840     DAV high ?
18D9 2980   AND #80
18DB F0F9   BEQ 18D6     jump back if not
18DD A9F7   LDA #F7      set NDAC low
18DF 2D21E8 AND E821
18E2 8D21E8 STA E821
18E5 A9FF   LDA #FF      255 (base 10) into bus
18E7 8D22E8 STA E822
18EA 60     RTS          return to main

IEEE Bus Handshake Routine
Object Listing

1800 A2 00 A9 FB 2D 40 E8 8D
1808 40 E8 A9 28 85 01 20 80
1810 18 A9 08 85 01 20 80 18
1818 A9 48 85 01 20 80 18 A9
1820 FD 2D 40 E8 8D 40 E8 A9
1828 F7 2D 21 E8 8D 21 E8 A9
1830 04 0D 40 E8 8D 40 E8 A0
1838 08 20 B0 18 A5 02 9D 01
1840 19 E8 88 D0 F4 A9 FB 2D
1848 40 E8 8D 40 E8 A9 02 0D
1850 40 E8 8D 40 E8 A9 08 0D
1858 21 E8 8D 21 E8 A9 5F 85
1860 01 20 80 18 A9 04 0D 40
1868 E8 8D 40 E8 CE 00 19 D0
1870 91 60 EA EA EA EA EA EA
1878 EA EA EA EA EA EA EA EA
1880 AD 40 E8 29 40 F0 F9 A5
1888 01 49 FF 8D 22 E8 A9 F7
1890 2D 23 E8 8D 23 E8 AD 40
1898 E8 29 01 F0 F9 A9 08 0D
18A0 23 E8 8D 23 E8 A9 FF 8D
18A8 22 E8 60 EA EA EA EA EA
18B0 A9 02 0D 40 E8 8D 40 E8
18B8 AD 40 E8 29 80 D0 F9 AD
18C0 20 E8 49 FF 85 02 A9 FD
18C8 2D 40 E8 8D 40 E8 A9 08
18D0 0D 21 E8 8D 21 E8 AD 40
18D8 E8 29 80 F0 F9 A9 F7 2D
18E0 21 E8 8D 21 E8 A9 FF 8D
18E8 22 E8 60

0001 data to go into bus
0002 data from bus

1900 counter for number of data transfers

1901 start of results area
```

Listing 3. IEEE bus handshake routine in machine language. MLA is My Listen Address; MTA is My Talk Address; UNT is Untalk Command.

Gregory Yob
Box 354
Palo Alto, CA 94301

Get Your PET on the IEEE 488 Bus

Part 2 of this "opus computerus" examines the file characteristics of the IEEE 488 bus.

Your PET has a "built-in" way of communicating through the IEEE 488 bus. In BASIC, the IEEE 488 looks like a file—just as the cassettes are files. The OPEN statement is used to specify a physical device number of 4 to 30, and the open logical file now talks via the IEEE 488 bus.

A complete understanding of PET tape files is a prerequisite for working with the IEEE 488 as a BASIC file. An article in the January 1979 *Kilobaud Microcomputing* ("PET Techniques Explained") covers many "innocent" errors that will result in mysterious malfunctions.

IEEE 488 Information Transfers

Talking to a Device.

1. OPEN a BASIC file to the device's address. For example, OPEN 1,4 will open the IEEE bus to device 4. Your BASIC program will see this as file #1.
2. PRINT# to your OPENed file. PRINT#1,"HELLO, DEVICE" will address the device to listen, send the string HELLO, DEVICE, add a carriage return with EOI true and then issue the UNT (Untalk) command.
3. Repeat step 2 as needed. Note that after each PRINT#, the IEEE bus is free, since the UNT has been sent.

PRINT# will send the same characters, including the skip character after numbers, as PRINT does to the screen. If you want to send several items, be sure that any needed delimiters, such as ",", are included.

Listening to a Device.

1. OPEN a BASIC file to the device's address.
2. Use INPUT# or GET# to fetch a line or a character from the IEEE bus.
3. Check the status word, ST, for an error, such as time-out. If the device is slow, the PET will complete the INPUT# or GET# and put a nonzero value into ST, which must be checked immediately after the I/O operation. If ST indicates a time-out, jump back to step 2.
4. Convert the data from the INPUT# or GET# as needed, and if more is needed, go to step 2.

Note that after each INPUT# or GET#, the UNT command is sent to the IEEE bus. This will truncate long messages from the device, especially with GET#. Also note that INPUT# (string) and GET# (string) work the best. The BASIC string functions (MID$, RIGHT$, LEFT$ and VAL) will help you get the data into a usable form.

Talking to More than One Device.

1. OPEN a file for each device.
2. Using CMD, send a dummy message to each device. For example, CMD 1:CMD 2:CMD 3 will set up each device (as specified in the OPENs for files 1, 2 and 3) by sending carriage returns to the devices and leaving them as listeners on the bus.
3. PRINT# to the IEEE bus. Any of the OPENed files may be used.
4. Repeat steps 2 and 3 as needed. Since PRINT# ends with the UNT, step 2 must be repeated after each PRINT#.

Transfer from One Device to Another.

1. OPEN a file for each device.
2. CMD to the device that is to be the listener.
3. INPUT# from the device that is to be the talker.
4. Repeat step 3 as needed.

INPUT# does not send a UNL, so the device that was CMDed remains on the bus as a listener. All information sent by the talker to the PET is also received by the listener. To turn off the listener, use a PRINT# to the listener's file. If the talker is slow, check ST and repeat step 3 as required.

LISTing a BASIC Program to a Device

1. OPEN a file to the device.
2. CMD to the device.
3. Enter the LIST command.
4. When the LIST is finished, do a CLR.

The PET's graphics and cursor characters will not print correctly on a standard ASCII printer. (I have a BASIC listing program available.)

The best way to learn the PET files and IEEE 488 is by specific examples. After a detour through CMD, we will continue with two examples. These should provide you with enough information to get started. If you have no success, refer to the section on Common Errors (found later in this installment).

CMD

CMD is an unusual PET command. Consider its functions:

1. Anything that BASIC wants to say is now routed to the device that CMD's file number refers to. If this isn't the screen, nothing that BASIC says will appear on the screen.
2. If a list of variables and literals is provided after the CMD, they will be sent to the device in the same way as PRINT# will.
3. However, if the device is on the IEEE bus, no UNL will be sent, so the device will remain in the listening state and receive any following data sent on the IEEE bus.

To see how CMD operates, get two scratch tapes and enter the program in Example 1. Now SAVE and VERIFY this program on one of your tapes. Put the other tape in the tape unit and execute the following:

```
OPEN 1,1,1
PRESS PLAY & RECORD ON TAPE#1
```

Perform this and wait until the tape stops.

```
OK
READY.
```

Now enter CMD 1. Note that READY. didn't appear; it was provided by BASIC and is now residing in the tape buffer. The cursor is blinking below the C in CMD. Continue with:

```
10  REM CMD EXAMPLE
20  PRINT"••••"
30  OPEN 1
40  GET#1, A$
50  PRINT A$;
60  IF A$=CHR$(90)THEN PRINT"••••":END
70  GOTO 40
80  REM Z
```

Example 1.

```
.*S*
CLOSE 1
CLR
READY
```

Note that the CLOSE 1 didn't get the READY. back. It took the CLR to return BASIC's messages to the screen. If you enter LIST, the program will appear on the screen. Rewind the tape and RUN. Three asterisks now appear after the RUN. These were printed by the program. This is one reason I don't trust my PET after a CMD. The text between the OK and the ending READY was found as a data file.

When the PET was under the influence of CMD, the letters you typed in were put onto the screen. This echoing is done by the PET's operating system, so CMD won't put these out to the device.

Though CMD looks like a good way to LIST program to tapes as data files, there is a snag. My example is shorter than 191 characters, and a LIST via CMD isn't smart enough to "jiffy" the data tape (this has been fixed on the new PETs). You run the risk of losing tape records when you try to read an "unjiffied" tape.

Try to verify that CMD 1,"HELLO OUT THERE" will print HELLO OUT THERE onto the tape. Remember that if you CMD a device on the IEEE 488 bus, any PRINT# to the bus will require a repetition of the CMD if you want the device to remain in the listening state.

Talking to the Clock Again

(For a description of the HP clock see part 1 of this article.)

First, you must check the device address on the DIP switch (which will be near the 488 female connector) and make sure the address is in the range 4 to 15. The enter a short program (Example 2) into the PET. This program consists of three subroutines to facilitate communicating with the clock. Remember that the PET will not accept an INPUT statement as a direct command.

```
10  OPEN 1,7
20  RETURN
100 INPUT"SAY TO CLOCK:";S$
110 PRINT#1,S$
120 RETURN
200 INPUT#1,C$
210 PRINT"CLOCK SAYS: ";C$
220 RETURN
```

Example 2.

First, enter GOSUB 10 as a direct command. This opens file 1 to device 7, which is our clock on the IEEE bus. OPEN merely sets things up; nothing is sent to the bus yet.

To read the time, enter GOSUB 200:

```
GOSUB 200
CLOCK SAYS: 0103020204 (Jan 3, 2:02:04 AM)
```

Your PET might give ? SYNTAX ERROR after this operation. This is a harmless feature of the PET.

To set the clock, using Jan. 29, 9:17 PM, as our example, enter:

```
GOSUB 100
SAY TO CLOCK? RDDDDDDDDDDDDDDD
DDDDDDDDDDDDD(28 Ds)
```

The clock starts at day 1. To set to day n, use n – 1 Ds. To set the hour, enter the following.

```
GOSUB 100
SAY TO CLOCK? HHHHHHHHHHHHHHH
HHHHHH(21Hs)
```

Minutes and seconds are set similarly.

```
GOSUB 100
SAY TO CLOCK? MMMMMMMMMMMMM
MMMMSSS (17Ms, 3Ss)
```

We are now set to 9:17:03. When I did this by hand, the clock moved forward about a minute, so the number of M's used should be changed to accommodate for this.

Talking to the HP 8165A Programmable Signal Source

(For a description of the HP 8165A, see part 1 of this article.)

The 8165A is a fine instrument with many switches, knobs, buttons and options and a correspondingly wide array of IEEE 488 commands (see Fig. 12, part 1).

The precise contents of each example concern the 8165A, which is an instrument you will probably never meet! My intention is to show you how direct mode commands—that is, BASIC statements without line numbers—can be used to control an instrument and help in debugging.

First, I hooked the 8165 to the 488 cable, and the PET turned on. The 8165 was addressed to 8. When the PET came on, IFC was true for about one second. This put the 8165 in local mode, where the front panel works as usual. Many instruments will ignore their front panels when the 488 bus addresses them. Once the PET addresses the 8165, you cannot control it from the front panel anymore. (An LED indicates this on the 8165.)

The following short program takes care of input from the instrument:

```
10  INPUT#1, A$
20  PRINT A$
```

This substitutes for the illegal direct command (INPUT#1,A$: PRINTA$), which I would like to use, but the PET forbids (try it and see!).

Since I wanted the 8165 to output a 1 kHz sine wave at an amplitude of 1.5 volts, I used the following IEEE commands:
F1—Set to sine wave
FRQ 1 kHz—Set frequency
AMP 1.5 V—Set amplitude
I1—Set to normal operation (continuous signal output)

First, open the IEEE file:

```
OPEN 1,8
READY.
```

Then send the settings:

```
PRINT #1,"F1"   (At this point, the
"Remote" LED went on, and I can no longer
work the front panel.)
PRINT #1,"FRQ1KHZ"
PRINT #1,"AMP1.5V"
PRINT #1,"I1"
```

Nothing happened! My scope showed only a flat trace! Upon reviewing my steps, I noticed that I overlooked the Disable Output (OD) and Enable Output (OE) commands. I entered PRINT #1,"OE", and a sine wave appeared on the scope.

You could also send this setting as one string. For example, PRINT #1,"F2FRQ1.2KHZAMP 1.2VI1OE" sets up a 1.2 kHz triangle wave at 1.2 V amplitude.

The 8165 can also report some of its switch settings. Now we can use the tiny program in the PET:

```
GOTO 10
F1 D2 I2 FM0 AM0
```

Since the PET has difficulty with GOSUB in direct mode and the IEEE bus, we must make a program change:

```
10  INPUT#1, A$
20  PRINT A$
30  RETURN
```

We will quickly be reminded that any time we change a program, all the variables, including opened files, will be lost:

```
GOSUB 10
?FILE NOT OPEN ERROR IN 10
```

So we try again:

```
OPEN 1,8
GOSUB 10
F1 D2 I2 FM0 AM0
?SYNTAX ERROR IN 22066
```

The PET will provide the ?SYNTAX ERROR about 90 percent of the time when the IEEE is accessed via the INPUT# statement and the PET is executing a directly called subroutine. However, this doesn't appear to affect anything. I avoided this by not making the little program a subroutine the first time.

So, if you are in a pinch, remember that the PET's direct command capability can rescue you with IEEE 488 devices and provides an inexpensive way to explore a new instrument.

Talking to More than One Device

Now that each of the instruments has been in the bus individually, the next step is to try the 488 with both of them on at the same time. I connected the HP clock and the 8165 to the 488 bus and gave the clock address #7, and the 8165 address #8. Then I entered the short program for INPUTs:

```
10  INPUT #1, A$
20  PRINT A$
30  END
100 INPUT #2, B$
110 PRINT B$
120 END
```

First, OPEN the files:

```
OPEN 1,7
OPEN 2,8
```

If you get a ?FILE OPEN ERROR, just enter CLR and start over.

Taking a peek at the clock resulted in:

```
GOTO 10
0130051957     (30 Jan., 5:19:57)
```

And peeking at the 8165 gets me:

```
GOTO 100
F1 D2 I2 FM0 AM0
```

which is the usual mystery message that the 8165 says to me. There isn't any point in explaining this message, for your instrument will say something different and meaningful only to you.

PRINT #1 and PRINT #2 will

```
X$ = "":FORJ = 1TO14:GOSUB8000:X$ = X$ + CHR$(FNF(D1)) NEXT:PRINTX$
0101000520
```

Example 4. Putting the clock's message into X$, and the contents of X$.

```
DV = 7:GOSUB7500
SEND MESSAGE

MESSAGE:? R                    R for reset

GET MESSAGE

PRESS KEY TO START

(.....A lot of Listen Handshakes....)

MESSAGE IS: 0101000158
```

Example 5. Resetting the clock.

Program Listing Conventions

The PET's graphics and cursor control characters aren't easily duplicated for program listings, so the conventions described here will be used instead.

If a letter or numeral (or any character) is underlined, it means the corresponding graphics character is to be used. (A is the spade symbol on the PET.)

Lowercase letters indicate PET special functions:

clr	Clear Screen	hm	Home Cursor
rt	Cursor Right	lft	Cursor Left
up	Cursor Up	dn	Cursor Down
rvs	RVS field on	off	RVS field off
cr	RETURN key	sp	SPACE key

Sp in a line indicates leading or more-than-one blank. For example, dn/sp/sp/HELLO THERE means Cursor Down space HELLO space THERE.

Two IEEE 488 Instruments

The two instruments described here are typical in the way they are controlled via the IEEE 488 bus. Most instruments are controlled by sending and receiving ASCII characters, which are mnemonics of the function being controlled. For example, the HP clock uses the letter D to increment its days' counter. Numbers are usually sent as ASCII strings—in the same way that PRINT provides an ASCII string of digits to a terminal. CR and LF usually indicate a message's end.

Some instruments will use more difficult formats. Two popular forms are BCD, in which two digits per byte are sent, and pure binary, where the value 0-255 is sent. Be sure you know the *exact* formats used by your instruments! Most instruments are unforgiving of bad data; and the responses range from ignoring meaningless characters to the instrument's unaddressing and leaving the bus. Check your instrument's manual!

The HP 59309A Digital Clock

The HP clock is almost the simplest instrument that uses the IEEE 488 bus. Your options are to either set the time or read the time.

When the clock is addressed to talk, it will provide a string of characters with the time in the following format:

(sp or ?) sp NNDDHHMMSS cr lf

The first character is a space or a question mark. If the clock hasn't been set since the last power-off, the question mark will indicate this. The next two digits indicate the month, from 01 to 12. Then comes the day of the month, 01 to 31. (The clock keeps track of the days in each month correctly and has a leap-year switch). Then the hours (00 to 23), minutes and seconds are sent. The carriage return and line feed indicate the end of the message.

Inside the clock are switches that provide variations of the format—colons or commas can either separate the fields, i.e., NN:DD:HH:MM:SS, or simply send the 24-hour time.

When the clock is addressed to listen, eight ASCII characters are used for control:

P—Stop the clock
T—Start the clock
R—Reset the 01:01:00:00:00
S—Each S will increment the Seconds counter
M—Increment Minutes counter
H—Increment Hours counter
D—Increment Days counter
C—Note time, send it when addressed to talk.

For example, the following string will reset the clock to Jan 5, 8:07:12 AM.

PRDDDDHHHHHHHHMMMMMMMSSSSSSSSSSSST

The T at the end restarts the clock.

The HP 8165A Programmable Signal Source

This is a "cadillac" 488 instrument—the front panel of this machine has 41 buttons for selection of modes and a 12-button number pad for entering times, and frequencies. This works out to 35 different command formats for setting up parameters and switch settings and nine commands for telling the controller the machine's setting or starting a sequence of actions. Some of the formats include:

F1—Select Sine Wave
F2—Select Triangle Wave
F3—Select Square Wave
FRQ f MZ—Select frequency in MHz. f is a number from 1 to 9999.
FRQ f MZ—Same for Hz
FRQ f KHZ—Same for kHz
SET:—Report all parameters currently operating when addressed to talk.
SET: n—Report setting in memory # n (0-9)

The 8165 can store up to ten complete settings in its memories, so the SET commands permit the controller to find out what's in the 8165.

An instrument of this complexity is usually programmed with a set of special-purpose programs as needed. Writing a general-purpose BASIC program would be both tedious and wasteful. My experience is that the hardest part is to get the PET and the instrument to communicate. Once that is accomplished, the rest is easy.

work just fine, and so two instruments and the PET can live in harmony together.

A Gotcha

I decided to turn off the 8165 with the PET set up for two instruments as described above. Sure enough, strange things happened.

The clock worked fine:

```
GOTO 10
0130052525
```

And just for fun, look what happens with the 8165 (which isn't on):

```
GOTO 100
F1 D2 I2 FM0 AM0
```

The 8165 has some internal batteries to store and memorize settings until it is turned on again. It also will respond to the IEEE 488 bus.

Now to try things in reverse—the clock doesn't have any batteries. (Clock is off; 8165 is on.)

```
GOTO 100
F1 D2 I2 FM0 AM0      The 8165 is fine
GOTO 10
F1 D2 I2 FM0 AM0      What's this?
```

The 8165 will reply to any address if it is the only device on the bus. The clock acts in the same way. (I don't know if this is a PET fault or an HP design decision. Check your device.)

If your program is intended for more than one device, this can be a disaster. Make sure all required devices are operating when using multiple devices on the bus.

I ran into another gotcha: the 8165 wouldn't accept every frequency change. I tracked this problem down to the presence of the HP clock on the bus. When I turned the clock off, everything worked fine. When debugging, remember to have only one device on your bus.

Common Errors

In theory, if you have understood everything to this point, you can now get an IEEE 488 instrument and make it play with your PET. In practice, this won't happen.

Finding errors is the hardest part of programming, and when you work with the IEEE bus, you can make many mistakes that don't look like errors. When you are able to see errors easily and immediately, you won't need this article.

Here is an incomplete list of the common errors in wait for the unwary IEEE/PET programmer.

The misplaced address. The PET's IEEE addresses are from 4 through 30. The addresses 0 to 3 are reserved for the PET's other I/O devices:

0—Keyboard
1—Tape unit #1
2—Tape unit #2
3—Video screen

If you OPEN a file to the reserved addresses, you won't be speaking to the IEEE bus!

If a device isn't running when the PET wants to talk to it, you will usually get a ?DEVICE NOT PRESENT ERROR. However, if some other device is operating on the bus, you might get the other device's response instead. This happened to me with the HP clock and the 8165. If one was turned off, the other would respond, even though the OPEN statement was referring to the inactive device. This can badly confuse your program.

Time-outs. The PET will only wait for 64 milliseconds before giving up on a device that is slow to respond to the IEEE 488 handshake. Though the IEEE 488 is supposed to work at *any* speed, you may wonder what to do if a device on the bus has failed. If the PET were to wait for a response, there would be no way to return to the user. The 64 ms interval was chosen from the timers available on the 6522 VIA chip, which can count up to 65535 at the 1 MHz clock rate of the PET.

Most instruments will respond within the 64 ms interval, and the PET will read and write the data correctly. This was true of the HP instruments at my disposal. To exercise the PET time-outs, I attached both the clock and the 8165 to the bus, and then OPENed a file to a nonexistent address:

```
NEW

10  INPUT#3,A$
20  IF ST THEN PRINT"ST IS" ST
30  PRINT A$
40  A$ = ""

OPEN 1,7     (Open the clock to file 1)

OPEN 2,8     (Open the 8165 to file 2)

OPEN 3,10    (The nonexistent device)
```

The little program attempts to input from the nonexisting device. The ST value is a reserved BASIC variable used by the PET for indicating I/O conditions. If ST isn't zero, something went awry.

Now to talk a bit to the devices to wake them up:

```
PRINT #1,"R"     (And the clock resets)
PRINT #2,"E0"    (And the 8165 puts out
                  a signal)
```

If a look at ST is made, all's well:

```
PRINT ST
 0
```

This may take a few tries to work right.

Now to try that nonexistent device:

```
PRINT #3, "HELLO"
```

Looks OK, right? Well, let's see . . .

```
PRINT ST
-128
```

This is the PET's ST code for "device not present."

Now to try the little program:

```
GOTO 10
ST IS 2

READY.
```

The ST code is 2, which is the time-out for reading data; the nonexistent device didn't say anything. Recall that line 30 said to print A$. The PET *did* print A$, which was an empty string.

The solution to this dilemma is to keep on trying! Write a loop that redoes the INPUT# or PRINT#. In most cases, a slow device will send its characters rapidly enough—once it has its message ready.

Consider these two sample loops:

```
100  PRINT #5," some message or other"
110  IF ST = 1 THEN 100
200  INPUT #6,B$
210  IF ST = 2 THEN 200
```

If you want to mask for certain bits, you can use the AND operator, but parentheses are needed. The above examples would read:

```
110  IF (ST) AND 1 THEN 100     and
210  IF (ST) AND 2 THEN 200
```

The removal of the parentheses makes the PET see the expression as:

```
IF ST AND 1     looks like     IF S TAN D 1
```

which will result in a ?SYNTAX ERROR. Use parentheses or rearrange the order of operations in these cases.

The literal principle. PET outputs to a file the same characters that it sends to the screen. This is also true for the IEEE 488. The PET's format for PRINTing a number is:

(space or - sign) (digits) (optional exponent) (cursor right)

This can raise havoc with an IEEE device that is expecting a character after the number.

Consider the following example:

```
10  PRINT "clr";      (clear screen)
20  FOR J = 1 TO 10
30  PRINT "*********************"
40  NEXT J
50  PRINT "hm";       (home cursor)
60  FOR J = 1 TO 10
70  PRINT J"IS A NUMBER"
80  NEXT J

RUN
 1*IS A NUMBER*****
 2*IS A NUMBER*****
 3*IS A NUMBER*****

 etc
```

The asterisk after the number comes from the cursor right character that was sent to the screen. The cursor right follows any numbers sent to the IEEE 488 bus.

The following program sets the frequency of the 8165.

```
10  OPEN 1,8    (The 8165 is at address 8)
20  FOR J = 1000 TO 2000 STEP 10
30  PRINT #1,"FRQ"J"HZ"
40  FOR K = 1 TO 1000
50  NEXT K      (This is a 3 second delay
                 loop)
60  NEXT J
```

When this is RUN, the 8165 gives all signs of distress. The frequency appears on the front panel, but the LED that indicates correct entry stays blinking (not completed). Also, the scope shows no change. The PET screen blinks at intervals, indicating that EOI is made true now and then. (I suspect the instrument is making this happen.)

The following modification will fix this:

```
30  PRINT#1,"FRQ"STR$(J)"HZ"
```

The STR$ function converts a number to the string that would be PRINTed, without the cursor right at the end! The general fix for numbers is simple: convert *all* numbers to strings before putting on the IEEE 488 bus.

Fractions. Now that the frequency example is working right, how about trying some other STEP sizes. Here is a simple change:

```
20  FOR J = 1 TO 2 STEP .01
30  PRINT #1,"FRO"STR$(J)"KHZ"
```

The J loop was changed to do the same thing, but in kilohertz. Line 30 was changed to reflect this. When RUN, it all works fine until about 1.25 kHz—the 8165 now shows 1.259 kHz instead of 1.260. A look at J gives us the clue we need:

```
BREAK IN 40      (Press STOP key)
PRINT J
 1.25999999
```

The PET slips up when computing with fractions . . . and this eventually shows up. The fraction .01 becomes a repeating binary decimal, and after repeated addition, the round-off appears as a slight reduction of the number being added to. In this case, 1.260 turns into 1.25999999.

Catching this is easy . . . if J were put onto the screen first!

```
35  PRINT STR$(J)
```

If you do this, the first "blow up" comes at 1.22999999. Now you are faced with a programming problem: how to get around nasty numbers. One way is to take the INT function, such as:

```
STR$(INT(J*100+.5)/100)
```

which rounds the number in the hundredths place. More complex tricks will be needed if the PET insists on scientific notation, such as

```
2.35E-03
```

PRINT your IEEE output onto the screen while debugging.

Next month, we will wrap up our three-part series with a further look at the programming style with the IEEE 488. ■

The PET IEEE 488 File I/O Statements

The PET sees the IEEE 488 bus as a file, and the file I/O statements apply to IEEE 488 transfers. Be sure you know the cassette file I/O before tackling the IEEE 488 bus.

The PET file I/O statements are:

● OPEN (file number), (device number), (secondary address), (filename)

OPEN instructs the PET to associate the file number with the desired I/O device. BASIC uses the file number in its PRINT#, INPUT# and GET# statements to determine where the I/O is to take place. The file number may be from 1 to 255.

The device numbers are assigned as follows:

0—Keyboard
1—Cassette unit #1
2—Cassette unit #2
3—Screen
4—30 IEEE 488 bus

This implies that your IEEE device must be addressed in the range of 4 to 30. Most IEEE devices have a switch or jumpers that permit the changing of their addresses.

The secondary address and filename are optional. However, if you want to use the filename, the secondary address must also be included. The secondary address has the range of 0 to 31.

If the filename is not specified, the OPEN statement sends nothing to the IEEE 488 bus. When BASIC sees the PRINT#, INPUT# and GET# statements, the device number (and secondary address, if specified) are put on the IEEE bus as part of the usual transfer sequences.

If a filename is specified, (i.e., A$ or "SOME NAME"), the OPEN statement activates the IEEE bus making ATN true and sends:

LISTEN (to the appropriate device)
SECONDARY ADDRESS (ORed with 11110000)
FILENAME (all characters)

This permits suitably complex command sequences that require ATN to be true to be sent. If the command sequence has to be repeated later, CLOSE the file and OPEN it again. I haven't been able to check if the above assertions about the filename are true. If you have a bus analyzer, check this out!

● PRINT# (file number), (values to be sent)

First, don't use the abbreviation ?# ; it won't work (when executed, you will see ?SYNTAX ERROR) and will list as PRINT#. Spell out PRINT completely!

The PRINT# sets ATN true and sends the device number as a LISTEN address. If a secondary address as specified, it will be sent also. The device number and secondary address are taken from the appropriate OPEN statement.

ATN is then made false, and the values to be sent are transmitted as ASCII characters in exactly the same way as they would be sent to the screen. For example, if a number is sent, a cursor right character follows the last digit. If you use "," to separate columns, lots of cursor rights are sent. If the PET feels a number should be in scientific format (i.e., 1.53E-07), that's what is sent! EOI is made true with the last character of data sent.

After the values are sent, an UNLISTEN is sent (with ATN true), and all listening devices are set free.

● INPUT# (file number), (values to be input)

INPUT# sets ATN true and sends the device number as a TALK address. If a secondary address was specified, it will be sent too. The pertinent OPEN statement is used for these values.

ATN is then made false, and the PET accepts characters from the device to the PET's input buffer. If the talker activates EOI, a carriage return is added to the end of the buffer.

After the characters are accepted and carriage return or EOI is recognized, the PET sets ATN true and sends an UNTALK, which releases the device.

BASIC then scans the input buffer in the same way that an ordinary INPUT statement looks at what is typed in. This means that commas and quotes will have the same effects as with normal INPUT. It is best to use an INPUT (string) form and hope your device doesn't send any commas!

As with cassette INPUT#, an 80-character buffer is used. If more than 79 characters arrive without a carriage return, the PET will go into "limbo," and all is lost. (New PETs have this fixed. Over 80 characters are ignored (or worse, the buffer is initialized, and the first 80 characters are lost!). If you have a new PET, try it with cassettes and find out what happens.

INPUT# is susceptible to "time out," and ST should be checked for a time out. Repeat the INPUT# if a time out is detected.

● GET# (file number), (value for entry)

GET# sets ATN true and sends the device number as a TALK address and the secondary address, if specified. ATN is made false, and a single character is accepted.

Then, the UNTALK with ATN true is sent, and the character given to BASIC. For the reasons that make GET X unusable, be sure to only use the GET# (string) form.

The assertion of the UNTALK after GET# makes transmission of multicharacter messages from devices impractical, as most devices will try to repeat their message on repeated application of GET#.

As with INPUT# ST should be checked for a time out, and if timed out, the GET# should be repeated.

● CLOSE (file number)

CLOSE releases the I/O assignments. The PET will allow a maximum of ten files OPEN at one time, and CLOSE will let you reuse an I/O assignment. If you OPEN more than ten files, old PETs will go into limbo and all will be lost. New PETs presumably have this fixed.

If the corresponding OPEN statement had a filename specified, CLOSE sets ATN true and sends the device number and secondary address (ORed 11100000). This feature is intended for PET peripherals.

● CMD (file number), (values to be sent)

CMD initiates the same sequence as PRINT# and sends the values, if any, in the same way that PRINT# does. When finished, CMD does not send the UNLISTEN, so any devices addresses with CMD will listen to further CMDs or PRINT# to the IEEE bus.

All of BASIC's output will be routed to the device defined in the OPEN statement for the file number. If the PET is in command mode, this includes the READY., error messages and LIST. If in run mode, any BASIC printouts, from PRINT to the screen, will go to the IEEE bus instead. A PRINT# will recover from the effects of CMD.

If you are using CMD in command mode, the cursor may not echo the RETURNs you press. The PET will "echo" your keystrokes, but any outputs from BASIC will vanish to the IEEE device. The PRINT# to your IEEE device is the safest recovery from CMD. Remember that any editing of a BASIC program will destroy all variables. This includes open files and CMDs.

● ST (status word)

After each I/O operation, the PET sets the value of a special variable named ST, which will hold its value until the next I/O operation. So the best policy is to check it immediately! The values of ST for the IEEE bus are:

1	Timeout on write	
2	Timeout on read	(This one should *always* be checked)
64	EOI true	
-128	Device not present	

The PET waits for 64 milliseconds to see if a device will respond to the IEEE handshake. If the device doesn't, the I/O operation is quietly aborted, and ST is set. If you are INPUT#ing, you will get "nothing" or zeroes back. If you are PRINT#ing, everything seems to be all right. If your device is slow to respond, checking ST is mandatory.

PRINT#, INPUT# and GET# will return the ?DEVICE NOT PRESENT error if the bus is in an illegal state (which is true if the bus has no devices or the LISTEN or TALK isn't responded to). ST will also be set.

● LOAD, SAVE and VERIFY

The old PETs have a severe error in their IEEE software which prevents the functioning of LOAD, SAVE or VERIFY. The ATN line was left true during the data part of the transfer. This is why owners of old PETs who purchase the PET disk get the new ROMS; the disk won't function with the old ROMs.

The format is the same as with tapes:

LOAD (filename), (device number)
SAVE " "
VERIFY " "

Once the IEEE bus is set to listen or talk, the first four bytes must contain the beginning and ending address + 1 of the block to be transferred. The transfer is then done as pure binary until finished. The bus is then released with an UNT or UNL as needed.

VERIFY will say ?VERIFY ERROR and set ST to 16 if any mismatches were found between the incoming data and the core image in the PET's memory. Since my PET is an old model with the original ROMs, I haven't been able to check LOAD, SAVE and VERIFY for the IEEE 488 bus.

Get Your PET On the IEEE 488 Bus

The final stop on this three-part tour.

Gregory Yob
Box 354
Palo Alto, CA 94302

Commodore's printer and disk use the secondary addresses to control special functions within each device. The secondary address extends the range of allowable addresses on the IEEE 488 bus and is included after the LISTEN or TALK address with ATN made true. Most IEEE devices do not use secondary addresses.

The secondary address permits the device to distinguish between data transfers (for example, file I/O via the disk) and command sequences (for example, to initialize a new disk). The following is a brief summary of the secondary addresses used by Commodore's devices.

PET Printer.

0 – Normal printing. The printer accepts characters and prints them as received.

1 – Formatted printing. The characters are accepted and rearranged according to an internally stored format specification.

2 – Format specification. The characters specifying the format to be used are accepted by the printer.

3 – Pagination control. Accepts a number indicating the number of lines per page.

4 – Control of diagnostic messages. If desired, diagnostic messages will be printed when errors are found. For example, if a number overflows its format, a message indicating this will be printed. This secondary address controls the options to use this feature.

5 – Load programmable character. The printer accepts bytes that specify the dot matrix for one programmable character.

PET Disk.

2 to 14 – Disk "channels" data transfers. The PET disk can have from zero to five files open at once. Each file is defined with an OPEN statement of the form:

OPEN (Log Addr), (Device Addr), (Channel Number), (Command String)

The channel number is a secondary address in the range of 2 to 14. The command string specifies the file type and drive. For example, "0,FILEONE, SEQ, WRITE" means open the file named FILEONE on drive 0 as a sequential file for write only access.

15 – Disk command channel. A variety of commands to the disk is sent via PRINT# to a file opened to the secondary address of 15. The disk can also send error and diagnostic messages to the PET through this channel.

Though it is possible to control complex devices in this manner, these methods can become awkward and clumsy if many data transfers are needed, as is the case for disks and printers. Commodore chose this method to avoid having to modify or extend the PET's BASIC.

Ironically, Commodore now offers a machine-language program, WEDGE, which functions as an extension to BASIC for control of the PET Disk.

Two Examples

In most applications of IEEE instruments, your task will extend beyond communicating with the device. Once communications with the device are established, there remains the conversion of the data to a form usable by people or some other instrument that uses a different form of data. Also, care should be taken to make human communications as pleasant as possible. If your application is in a production (that is, for daily use, and not as an occasional experiment), clarity and reliability are important.

Two BASIC programs, which illustrate how the HP Clock and the HP Signal Source might be used in real-life situations, follow. They are presented here as examples of programming style with the IEEE 488.

Example 1: The HP Clock

Part 1 (*Microcomputing*, July 1980) describes the codes used for the HP Clock with the IEEE 488 bus. Listing 1 interacts with the HP clock in a "human-workable" form. Let's first take a look at how the program is seen from the outside (often called "human engineering" or "the user interface").

When the program is RUN, the following message appears on the screen:

```
HP CLOCK PROGRAM
PRESS ANY KEY WHEN YOU HAVE THE
CLOCK CONNECTED VIA THE IEEE 488
AND THE POWER ON.
```

This reminds the user to connect the clock on the bus and turn on the clock's power. If the PET tries to address a device that isn't connected or turned on, the ?DEVICE NOT PRESENT error message will appear and stop the program. Unfortunately, there is no graceful way to prevent this and keep the program running (some versions of BASIC have error traps: i.e., ON ERROR 5 GOTO . . .).

After you press a key, the request appears:

Listing 1. HP Clock program.

```
10 REM NICE HP CLOCK PROGRAM
20 PRINT"clr HP CLOCK PROGRAM"
30 PRINT"dn dn PRESS ANY KEY WHEN YOU HAVE THE
40 PRINT"CLOCK CONNECTED VIA THE IEEE 488
50 PRINT"AND THE POWER ON.
60 GET A$:IFA$="" THEN 60
70 REM INITIALIZE
80 DIM M$(12),M(12)
90 FOR J=1 TO 12:READ M$(J),M(J):NEXT
100 DATA JAN,31,FEB,28,MAR,31
110 DATA APR,30,MAY,31,JUN,30
120 DATA JUL,31,AUG,31,SEP,30
130 DATA OCT,31,NOV,20,DEC,31
140 INPUT"dn dn CLOCK'S DEVICE ADDRESS:",AD
150 IF AD<3 AND AD>16 THEN 170
160 PRINT"SORRY, LEGAL ADDRESSES ARE 4 - 15":GOTO 140
170 OPEN 1,AD
180 INPUT"dn IS THIS A LEAPYEAR";L$
190 IF LEFT$(L$,1)="Y" THEN M(2)=29:PRINT"BE SURE TO SET
    THE CLOCK TO 366 DAYS"
200 REM TIME SETTING REQUEST
210 INPUT"dn dn SET THE TIME";L$
220 IF LEFT$(L$,1)="Y" THEN GOSUB 1000
230 REM DISPLAY TIME
240 GOSUB 2000
250 GOTO 210

1000 REM TIME SETTING ROUTINE
1010 PRINT"clr sp SET THE DATE"
1020 PRINT"dn dn ENTER MONTH AND DAY IN THE FORM:
1030 PRINT"dn sp sp sp sp sp MONTH (SPACE) DAY
1040 PRINT"dn FOR EXAMPLE: sp sp MARCH 25
1050 INPUT"dn";MD$
1100 REM PARSE OUT MONTH & DAY `
1110 MI$=LEFT$(MD$,3)
1120 FOR MN=1 TO 12
1130 IF MI$=M$(MN) THEN 1200
1140 NEXT MN: PRINT"dn dn I DON'T RECOGNIZE THE MONTH.
1150 PRINT"PLEASE SPELL THE MONTH COMPLETELY.
1160 PRINT"dn dn PRESS ANY KEY TO TRY AGAIN
1170 GETA$:IFA$="" THEN 1170
1180 GOTO 1010
1200 FOR J=1 TO LEN(MD$)
1210 IF MID$(MD$,J,1)=" sp " THEN 1300
1220 NEXT J
1230 PRINT"dn dn YOU FORGOT THE DAY
1240 GOTO 1160
1300 DY=VAL(MID$(MD$,J))
1310 IF DY>Ø AND DY<M(MN)+1 THEN 1400
1320 PRINT"dn dn YOUR DAY IS INCORRECT. IT MUST BE
1330 PRINT"FROM 1 TO"M(MN)"."
1340 GOTO 1160
1400 REM COMPUTE NUMBER OF DAY TICKS
1410 DT=Ø: IF MN:1 THEN 1430
1420 FOR J=1 TO MN-1: DT=DT+M(J):NEXT J
1430 DT=DT+DY-1
1450 REM DT IS # OF DAYS TO ADVANCE

1500 PRINT"clr sp SET THE TIME"
1505 PRINT"dn dn ENTER THE TIME IN THE FORM:
1510 PRINT"dn sp HOUR : MINUTE : SECOND : AM OR PM
1520 PRINT"dn FOR EXAMPLE: sp sp 2:25:36:PM"
1530 PRINT"dn";:GOSUB 4000
1600 REM PARSE OUT HOURS,MINS,SECS,AM?PM
1610 T$=T$+"xx":TH=VAL(T$)
1620 GOSUB 3000: IF PT>Ø THEN 1700
1630 PRINT"dn YOU DIDN'T INCLUDE EVERYTHING
1640 PRINT"PLEASE ENTER ALL FOUR ITEMS WITH
1650 PRINT"COLONS BETWEEN EACH OF THEM
1660 PRINT"dn PRESS ANY KEY TO TRY AGAIN
1670 GETA$:IFA$="" THEN 1670
1680 GOTO 1500 .
1700 T$=MID$(T$,PT+1)
1710 TM=VAL(T$)
1720 GOSUB 3000: IF PT=Ø THEN 1630
1730 T$=MID$(T$,PT+1)
1740 TS=VAL(T$)
1750 GOSUB 3000: IF PT = Ø THEN 1630
1760 T$=MID$(T$,PT+1,2)
1800 REM ERROR MESSAGES
1810 IF TH<1 OR TH>12 THEN PRINT"dn dn YOUR HOURS MUST
     BE FROM 1 TO 12":GOTO 1660
1820 IF TM<Ø OR TM>59 THEN PRINT"dn dn YOUR MINUTES
     MUST BE FROM Ø TO 59":GOTO 1660
1830 IF TS<Ø OR TS>59 THEN PRINT"dn dn YOUR SECONDS
     MUST BE FROM Ø TO 59":GOTO 1660
1840 IF T$="AM" OR T$="PM" THEN 1860
1850 PRINT"dn dn PLEASE USE AM OR PM ONLY":GOTO 1660
1860 REM AM/PM LOGIC
1870 IF T$="AM" AND TH=12 THEN TH=Ø
1880 IF T$="PM" AND TH<12 THEN TH=TH+12
1900 REM SET CLOCK AT LAST
1910 PRINT#1,"RP";
1920 IF DT>Ø THEN FOR J=1 TO DT:PRINT#1,"D";:NEXT
1930 IF TH>Ø THEN FOR J=1 TO TH:PRINT#1,"H";:NEXT
1940 IF TM>Ø THEN FOR J=1 TO TM:PRINT#1,"M";:NEXT
1950 IF TS>Ø THEN FOR J=1 TO TS:PRINT#1,"S";:NEXT
1960 PRINT#1,"T"
1970 RETURN

2000 REM DISPLAY TIME
2010 PRINT"clr sp sp sp sp sp THE CURRENT TIME IS
2020 PRINT"dn dn sp sp DATE:"
2030 PRINT"dn dn sp sp TIME:"
2040 PRINT"dn dn dn dn dn dn dn dn dn dn dn dn dn dn      (14 dn's)
     sp sp PRESS ANY KEY TO SET TIME
2050 GETA$:IFA$<>"" THEN RETURN
2060 REM FETCH TIME
2070 INPUT #1,T$
2080 IF LEFT$(T$,1)="?" THEN GOSUB 5000
2090 REM PARSE OUT PARTS
2100 T1=VAL(MID$(T$,1,2))
2110 T2=VAL(MID$(T$,3,2))
2120 T3=VAL(MID$(T$,5,2))
2130 T3$=MID$(T$,5,2)
2140 T4$=MID$(T$,7,2)
2150 T5$=MID$(T$,9,2)
2160 PRINT"hm dn dn dn rt rt rt rt rt rt rt rt rt"M$(T1);T2
2170 REM AM/PM CALCS
2180 T6$="AM"
2190 IF T3>11 THEN T6$="PM"
2200 IF T3>12 THEN T3=T3-12
2210 IF T3=Ø THEN T3=12
2220 T3$=RIGHT$(STR$(T3),2)
2250 PRINT"dn dn rt rt rt rt rt rt rt rt rt"
     T3$":"T4$":"T5$":"T6$
2500 GOTO 2050

3000 REM SCAN T$ FOR COLONS
3010 FOR PT=1 TO LEN(T$)
3020 IF MID$(T$,PT,1)=":" THEN RETURN
3030 NEXT PT
3040 PT=Ø: RETURN

4000 REM FETCH STRING VIA GET DUE TO
4010 REM FLAKEY PET INPUT STATEMENT
4015 T$=""
4020 GET A$: IF A$<>"" THEN 4100
4030 PRINT"rvs sp lft";:FOR J=1 TO 300: NEXT
4040 GET A$: IF A$<>"" THEN 4100
4050 PRINT"off sp lft";:FOR J=1 TO 300: NEXT
4060 GOTO 4020
4100 PRINT"off sp lft";
4110 IF A$=CHR$(13) THEN PRINT: RETURN
4120 PRINT A$;: T$=T$+A$: GOTO 4020

5000 PRINT"hm dn dn dn dn dn dn dn dn dn dn dn dn
     >>>>>>> sp TIME NEEDS TO BE SET sp<<<<<<<"
5010 PRINT"dn>>>>>>> sp DUE TO POWER FAILURE sp <<<<<<"
5020 T$=MID$(T$,3):RETURN
```

CLOCK'S DEVICE ADDRESS:?

Now enter the address on the DIP switches for the device. If an unacceptable value, such as 16, is entered, the PET will respond with:

SORRY, LEGAL ADDRESSES ARE 4-15

and ask again. The best way to avoid problems is to forbid illegal values for inputs, tell the user that he has goofed and mention the correct range of values.

Once the device address is in, the PET asks:

IS THIS A LEAP YEAR?

If "YES" is entered, a reminder appears to set the clock accordingly.

BE SURE TO SET THE CLOCK TO 366 DAYS

The last request asks:

SET THE TIME?

If the user doesn't want to set the time, the screen clears and the date and time are shown:

THE CURRENT TIME IS
DATE: JAN 29
TIME: 7:02:54 PM

PRESS ANY KEY TO SET TIME

The time ticks away with the seconds changing the most rapidly. A different set of values will appear on the clock:

01 29 19 02 54

The program has translated from 24-hour time to normal AM/PM time and changed the month from a number to the month's name.

The HP clock will send a ? as the first time character if the clock has not been set since a loss of power. If you pull the plug on the clock and plug it in again, the program will stop with a ?DEVICE NOT PRESENT ERROR. When the program is RUN, the time will be displayed with the following in the space between the time and the PRESS ANY KEY line

[illegible]
[illegible]

Now if you press a key, the SET THE TIME? request will reappear:

SET THE TIME? YES

The screen clears and will display:

```
SET THE DATE
ENTER MONTH AND DAY IN THE FORM.
     MONTH (SPACE) DAY
FOR EXAMPLE. MARCH 25
? JANUARY 29
```

If the first three letters in the month are incorrect, the program will make you start over:

```
I DON'T RECOGNIZE THE MONTH
PLEASE SPELL THE MONTH COMPLETELY.
PRESS ANY KEY TO TRY AGAIN
```

If you missed the date, the PET says:

```
YOU FORGOT THE DAY
PRESS ANY KEY TO TRY AGAIN
```

If you enter an inappropriate date, such as JAN 45, the PET, will say:

```
YOUR DAY IS INCORRECT. IT MUST BE
FROM 1 TO 31.
```

The program has the number of days for each month stored inside. If the month were February, the range 1 to 28 would have been shown instead.

Now that the date is entered correctly, the screen clears to let the time be entered.

```
SET THE TIME
ENTER TIME IN THE FORM
HOUR : MINUTE : SECOND : AM OR PM
FOR EXAMPLE: 2:25:36:PM
7:19:25:PM (you enter this line)
```

The screen will flicker a bit, and then the time display will appear.

The PET won't correctly input a string with colons in it, so the entry here is "faked" to look like a normal INPUT line. Unfortunately, if you must INST or DEL to correct your line, the correction won't really be entered. This can be programmed around, but I didn't feel like doing it with an instrument on loan to me for a week. The subject of faking INPUT is an article in itself.

Again, there are some error messages to help and assist the user:

```
YOU DIDN'T INCLUDE EVERYTHING
PLEASE ENTER ALL FOUR ITEMS WITH
COLONS BETWEEN EACH OF THEM
PRESS ANY KEY TO TRY AGAIN
YOUR HOURS MUST BE FROM 1 TO 12
YOUR MINUTES MUST BE FROM 0 TO 59
YOUR SECONDS MUST BE FROM 0 TO 59
PLEASE USE AM OR PM ONLY
```

Here, a bad entry only forces you to reenter the time. The date is OK, so why redo it?

Perhaps this example is extreme. In many situations it isn't worth the programming time to make a program completely convenient to use. As an idealist, I wrote it up to show what can be done if ease of use is required.

HP Clock BASIC Program Review (Listing 1)

Lines 10 to 60 announce the program and force the user to wait until he has made sure the HP Clock is attached to the PET's IEEE 488 and the power is turned on. DATA in lines 100 to 130 are placed in the months' names' array M$ and the months' lengths' array M.

Lines 140 to 170 request the HP Clock's address and check to see if the address is legal. Line 160 tells the user to try again and mentions the legal range as a hint. Lines 180 and 190 take care of the leap-year problem by changing the month length for February to 29 days and reminds the user to check the leap-year switch on the HP Clock.

In lines 200–220, the user is asked if the time is to be set (which must be done when the clock is first used), and a loop is entered in lines 240 and 250. Subroutine 1000 sets the time, and subroutine 2000 displays the time. The program will not leave subroutine 2000 until a key is pressed. Line 250 jumps to the time-change request as needed.

Setting the time in subroutine 1000 is a complicated job, requiring correctly entering the data. First, you must enter the month and day as explained in lines 1010 to 1040, which give an example of the expected format.

Line 1050 picks up the user's entry, and lines 1000 to 1180 take a look at the first three characters to see if they fit a month's name. Lines 1140 to 1180 take care of any mistake in the entry of a month's name.

Lines 1200 to 1220 scan the input string, MD$, until a space is found. This removes the remnants of the month's name and brings us up to the date digits. Failure to find a space means the day was forgotten, and the user is told to start all over.

Lines 1300 to 1340 check the day number with the number of days in the month M(MN). If everything is OK, lines 1400 to 1450 will figure out the value DT, which is used to send the correct number of Ds to the clock for date setting.

Now that we have the number of days from Jan. 1 (in the number DT), lines 1500 to 1530 will tell the user to enter the time in a familiar format—HH:MM:SS:AM or PM. Subroutine 4000 is used to enter the string T$ via the GET statement. In lines 1620 to 1850, the string T$ is snipped apart at the colons, and each part is examined for the correct range of values; subroutine 3000 looks for the colons, and lines 1680 to 1760 do the scissor-work. We eventually end up with the values TH, TM, TS and T$, for hours, minutes, seconds and AM/PM values.

Lines 1860 to 1880 adjust the hours, TH, according to the AM or PM value. Lines 1900 to 1970 set the HP Clock—first the clock is reset via "RP," and then the correct numbers of "D," "H," "M" and "S" are sent to set the time. Then "T" is sent to start the clock.

Subroutine 2000 sets up the screen in lines 2010 to 2060. Note that the GET in line 2050 only checks if a character was entered. If not, it will continue to line 2070. The HP Clock is accessed in line 2070, and line 2080 checks for "?." The "?" means the clock saw a power failure, and subroutine 5000 will warn of this event.

Lines 2100 to 2150 get the various parts of the HP Clock's message. T1 is the month number; T2 is the day number. Line 2160 displays the month and day values.

Lines 2170 to 2220 adjust the hours value, T3$, to reflect whether an AM or PM time is being shown. Then line 2250 prints the hours, minutes, seconds and AM/PM marker.

In subroutine 3000, PT is the position of the first colon found in the string T$.

Subroutine 4000 simulates a cursor and constructs T$ from the characters entered through GET A$. No editing is provided, so if you make an error, the entry must be repeated. A little more code could catch A$ = 20 (code for DEL) and give some limited editing (equivalent to back space or rubout on a terminal).

Subroutine 5000 puts the power failure message on the screen and strips the "?" from T$. This permits the display of time code to work correctly.

The astute programmer will note that no provision is made for bad messages from the HP clock (which might make the program fail in some cases). You should check the values T1, T2, T3, T3$, T4$ and T5$ for their legal values and make another attempt to read the time made in case of an error. In the event of several consecutive errors, the program should mention this to the user.

There are limits to how "failsafe" a program must be made. In many cases, malfunctions will not be critical, and it isn't worth the effort required to make the program survive the errors. I do not recommend the PET for any real-time control applications that may result in injury or loss of property in the event of the PET's failure!

Example 2: The HP 8165A Signal Source

Part 1 introduced the 8165A. Naturally, your interest will be with the devices that you have available, and the example shown here is a "laboratory application"; that is, a program similar to one you might want to build for your instrument.

Let's pretend that the response of a stereo amplifier needs to be tested in a production line. The frequencies and voltages to be tested are:

10 Hz.	Sine Wave.	1.000 volts
10 Hz.	Square Wave.	1.000 volts
20 Hz.	.....	
20 Hz.	.....	
50 Hz.		

Test sine wave and square wave responses at 1.000 volts for 10, 20, 50, 100 . . . up to 20 kHz.

The plan for a program is as follows:

1) Initialize. For example, open

```
10 PRINT"clr STEREO TEST PROGRAM
20 PRINT"dn dn BE SURE THE 8165 IS ON AND THAT
30 PRINT" THE IEEE 488 IS CONNECTED.
40 PRINT"dn REMEMBER THE ADDRESS FOR THE 8165
50 PRINT"MUST BE 8. PLEASE CHECK THIS.
60 GOSUB 1000
70 OPEN 1,8
80 REM SET UP 8165
90 PRINT#1,"FRQ10HZAMP1.000VF1D2OD"
100 REM HOOK UP STEREO
110 PRINT"clr STEREO AMPLIFIER TEST"
120 PRINT"dn ATTACH THE NEW UNIT TO THE
130 PRINT"TEST STATION."
140 GOSUB 1000
200 REM PERFORM TEST
210 PRINT"clr >>>> TEST IN PROGRESS<<<<  "
220 FOR L1=1 TO 4
230 FA=10↑L1
240 FOR L2 = 1 TO 3
250 IF L2 = 1 THEN FR=FA/1000
260 IF L2 = 2 THEN FR=FA*2/1000
270 IF L2 = 3 THEN FR=FA*5/1000
275 IF FR >25 THEN 430
280 FOR W = 1 TO 2
290 IF W=1 THEN W$ = "SINE"
300 IF W=2 THEN W$ = "SQUARE"
310 REM SET 8165 UP
320 PRINT#1,"FRQ"STR$(FR)"KHZ"
330 IF W=1 THEN PRINT#1,"F1OE"            (letter F, numeral 1,
340 IF W=2 THEN PRINT#1,"F3OE"             letters OE)
350 REM SET TIMER & REPORT
360 T1 = TI                                (tee one = tee eye)
370 PRINT"hm dn dn dn TEST AT:";
380 PRINT"sp sp FREQ:"FR*1000"sp sp"W$"sp sp sp"
390 IF TI - T1<600 THEN 390
400 REM TURN 8165 OFF
410 PRINT#1,"OD"                           (letters OD)
420 NEXT W
430 NEXT L2
440 NEXT L1
450 REM TEST COMPLETE
460 PRINT"clr ****** TEST COMPLETED *******"
470 PRINT"dn dn REMOVE AMPLIFIER FROM TEST STATION"
480 GOSUB 1000
490 GOTO 110

1000 PRINT"dn dn PRESS ANY KEY WHEN READY"
1010 GETA$:IF A$="" THEN 1010
1020 RETURN
```

Listing 2. Stereo Test program.

the IEEE 488 file.

2) Tell the operator to hook up an amplifier

3) Start the test

4) Loop through the frequencies for each frequency

5) Loop through sine and square

6) Wait for 10 seconds before continuing

7) Report where the test is on the screen

8) End of both loops

9) Tell the operator the test is finished

10) Go to step 2

Listing 2 shows these steps in a BASIC program. From the user's point of view, when the program is RUN, the message below appears:

STEREO TEST PROGRAM

BE SURE THE 8165 IS ON AND THAT THE IEEE 488 IS CONNECTED.

REMEMBER THE ADDRESS FOR THE 8165 MUST BE 8. PLEASE CHECK THIS.

PRESS ANY KEY WHEN READY

This reminder ensures that the 8165 is properly connected, powered and addressed. The PET program won't work if these conditions aren't met.

Now it is time to test a unit. The screen clears (after a key is pressed) and displays:

STEREO AMPLIFIER TEST

ATTACH THE NEW UNIT TO THE TEST STATION.

PRESS ANY KEY WHEN READY

Now the test commences, with a report on the current frequency and waveform being used:

>>>>>TEST IN PROGRESS<<<<<

TEST AT: FREQ: 200 SQUARE (current freq & waveform)

After about two minutes (each frequency and waveform takes ten seconds), the screen clears and tells the user:

•••••TEST COMPLETED•••••

REMOVE AMPLIFIER FROM TEST STATION

PRESS ANY KEY WHEN READY

Now we are ready to perform another test. Look at the scope and notice that the output of the 8165 is turned off between tests and between mounting the new amplifiers. Though unimportant in this example, more serious equipment should always be set to a "safe" state when the operator has to handle the equipment.

Lines 10 to 60 in the BASIC code state the program's name and remind the user to check the address setting on the HP 8165. Subroutine 1000 waits for you to press a key.

Three nested loops are used to scan through the frequencies and waveforms. The L1 loop sets the frequency decade from the range 10–99 Hz to 10000–99999 Hz. The L2 loop is used to select between 1, 2 and 5 times the frequency selected by L1. W chooses between sine and square waves.

Lines 200 to 300 compute the frequency FR in two steps (FA is set to 10^{L1}, and FR is set to 1,2 or 5 times FA), and W$ is set to report sine or square. In line 275 the top value to be tested is 20000 Hz, so to terminate the loops requires a test of the frequency larger than 20000 Hz.

Instead of using 20000 for the test, I am using 25000. (If you look at the code, FA is in kilohertz, so the test is for 25.) Due to the PET's way of computing numbers, when L1 is 3 and L2 is 2, FA turns out to be a tiny amount over 20, which terminates the test too soon.

When testing for equality or differences, make sure the number in the PET is what you think it is. Most floating point numbers will be slightly (and unprintably) different than the value you want, so fudge accordingly.

Line 320 sends the correct command to the 8165 for frequency. Note that FR is sent as the string STR$(FR). This avoids the Cursor Right after the number, which could totally confuse the 8165. Lines 330 and 340 specify the waveshape by directly sending the correct set of characters to the 8165. "OE" turns the 8165 on.

Lines 350 to 390 print the test values and wait for 600 jiffies, or ten seconds. When they are finished, line 410 turns the 8165 off (this is a "safe" state; e.g., during hook-up, the test leads could be shorted).

Lines 450 to 490 announce the end of the test and tell the user to remove the stereo amplifier. Note that the 8165 is in the "off" state.

I will leave it to you to figure out how to use the HP clock to control the timing of the stereo test program (Listing 2, part 2) instead of the PET's internal clock. Another variation is to put up the time each test is run for logging purposes.

More "Gotchas"

Program bugs. When I was debugging the HP Clock program (see Listing 1), the days' count wouldn't come out right. Some months tended to have two or three too many days, while others ran short. For example, May 5 put May 11 on the clock, and February 10 showed February 7.

I first thought that the IEEE 488 device was miscounting characters. I checked by printing the number sent onto the screen. The error wasn't here.

The eventual source of the problem was that the routine that counted the total days in

Function	Old Pet (hex)	Old Pet (dec)	New PET (hex)	New PET (dec)
Send TALK (MTA)	F0B6	61622	F0B6	61622
Send LISTEN (MLA)	F0BA	61626	F0BA	61626
Send UNTALK	F17A	61818	F17F	61823
Send UNLISTEN	F17E	61822	F183	61827
Set ATN true and send character in accumulator	F0BC	61628	F0BC	61628
Send data character in accumulator••	F0F1	61681	F0EE	61678
Get data character in accumulator	F187	61831	F18C	61836
Flag byte	0222	545	00A5	165

••Set flag byte to FF (255) before calling this routine.

Table 1. PET IEEE ROM and RAM locations.

the previous months just added the same number each time. For May, it added 31 four times, and for February, it added 28 once!

Another bug came from the "hidden bits" in PET numbers. In the Stereo Test program (Listing 2), there was the following line:

```
IF FR>20 THEN....
```

The testing program stopped at 10 kHz instead of 20 kHz. When I printed FR, I got 20. FR was formed from the two computations:

```
FA = 10↑L1
FR = FA*2/1000
```

The PET's exponentiation operator isn't totally exact, so a few bits slipped through. The division didn't help, and FR ended up a slight amount over 20, which is enough to make the condition true. The cure was to test for more than 25 instead.

These errors are subtle. If you aren't a total expert on your PET, these are nearly impossible to find.

```
10 REM PET SERIAL OUTPUT
20 REM GREGORY YOB
30 PT = 826
40 READ BT: IF BT  0 THEN 60
50 POKE PT,BT: PT=PT+1: GOTO 40
60 DIM BD(6),RT(6)
70 FOR J=1 TO 6
80 READ BD(J),RT(J)
90 NEXT J
100 PRINT"clr SERIAL OUTPUT"
110 PRINT"dn PARITY"
120 PRINT"0=EVEN, 1=ODD, 2=MARK"
130 INPUT P
140 IF P=0 THEN 180
150 IF P=1 THEN 180
160 IF P=2 THEN P=255: GOTO 180
170 GOTO 110
180 POKE 994,P
190 PRINT"dn BAUD RATE"
200 INPUT BT
210 FOR J=1 TO 6
220 IF BT=BD(J) THEN 380
230 NEXT J
240 PRINT"RATES ARE :"
250 FOR J=1 TO 6. PRINT BD(J): NEXT
260 GOTO 190
380 POKE 995, RT(J)
390 PRINT"# TIMES TO REPEAT CHAR"
400 INPUT N
410 N=INT(N): IF N<0 OR N>255 THEN 390
420 PRINT"PRESS ANY KEY TO SEND CHAR"
430 GET A$: IF A$="" THEN 430
440 PRINT A$
450 POKE 997,N: POKE 992, ASC(A$)
460 SYS(826)
470 GOTO 420

1000 DATA 173,4,2,234,234,240,1
1010 DATA 96,173,64,232,41,64,240
1020 DATA 241,120,21,192,3,144,2
1030 DATA 88,96,32,98,3,32,153
1040 DATA 3,88,76,58,3,234,24
1050 DATA 173,224,3,96,234,169,0
1060 DATA 141,225,3,173,224,3,162
1070 DATA 1,160,0,24,74,144,5
1080 DATA 160,225,238,225,3,72,152
1090 DATA 157,240,3,104,232,224,8
1100 DATA 208,234,273,226,3,48,12
1110 DATA 240,3,238,225,3,173,225
1120 DATA 3,41,1,240,2,169,255
1130 DATA 157,240,3,96,162,255,232
1140 DATA 189,240,3,141,34,232,172
1150 DATA 227,3,173,0,64,173,0
1160 DATA 64,173,0,64,136,208,244
1170 DATA 234,236,228,3,208,228,96
1180 DATA 96,0,0,0,0,0,0
1190 DATA 0,24,173,229,3,208,2
1200 DATA 56,96,173,224,3,206,229
1210 DATA 3,96,0,0,0,0,0
1220 DATA 0,0,0,0,0,0,0
1230 DATA 0,0,0,0,0,65,2
1240 DATA 0,195,11,0,0,0,0
1250 DATA 0,0,0,0,0,0,0
1260 DATA 0,255,0,0,0,0,0
1270 DATA 255,0,255,255,0,0,0
1280 DATA 0,0
1300 DATA -1
1999 REM PARAMETERS FOR BAUD RATES
2000 DATA 9600,5,4800,11,2400,23
2010 DATA 1200,46,600,97,300,195
```

Listing 3. Serial output via the IEEE 488 bus port.

Using the PET ROM

Since the PET knows the IEEE bus, there have to be routines in the PET ROM that know how to work the bus. A year ago, some of my clients' requirements forced me to access the PET's ROM for the IEEE. (One had a machine that didn't like the PET's state between IEEE messages; the other wanted to know the PET's maximum IEEE transfer rate.)

Table 1 indicates the pertinent RAM and ROM locations for the PET IEEE routines. Use caution when working with these, as I have only been able to check the ones mentioned below. In one case, a routine sent a character at an apparent rate of 5000 characters/second! (The listener didn't see anything at all.) The routine in question took a look at the bus, decided the bus wasn't in a legal state and returned, instead of sending the character! If you have an accurate PET disassembly, here is a good place to use it.

Input from the IEEE Bus. This can be approached either from machine language or as a mix of machine language and BASIC. In all cases, the first step is to open a file to the bus via BASIC. (This must be done; make sure that only one file is open.)

The next step is to send a TALK to the device. From BASIC, this is a SYS(61622), and in machine language it is a JSR F0B6 (or 20 B6 F0).

To handshake a character in requires calling the machine language in ROM. Here's a catch: the character arrives in the A register. From BASIC, you must SYS to a short routine that performs JSR F187 and an STA (somewhere) (and RTS to get back). Then PEEK (somewhere) gets your character. The machine code in hexadecimal is 20 87 F1 8D xx xx 60. The xx xx is your "somewhere." The value from the IEEE bus is the complement of your character; that is, the 1's and 0's are exchanged.

Send to the IEEE Bus. Again, the first step is to open a file to the bus and be sure that only one file is open. Then, send the ATN LISTEN via SYS(61626). (In machine language, JSR F0BA, or 20 BA F0.) Now, put the complemented value into location $0222 with a POKE 546, CHARACTER.

The last step is to SYS (61681), which sends the character. In some cases, you will have to set a flag first by setting location $021D to $FF by POKE 541,255. I have used both methods with success.

The machine-language sequence is A9 FF 8D 1D 02 20 xx xx 8D 22 02 20 F1 F0 60. The 20 xx xx is a JSR to your routine at xx xx, which gets a character in the A register.

Both the input and the output will leave the device active on the bus. Make ATN true and send the UNL and UNT value to release the device.

The IEEE lines in the PET don't have to be used for the IEEE 488 bus. There are 12 easily used bits of parallel I/O that can be controlled with suitable PEEKs and POKEs, and two PET Hard Copy Easy," *Kilobaud Microcomputing*, September 1979, p. 100.

Printing Hazards

The difference between the PET's display and character codes and the ASCII character set causes some difficulties when you use the IEEE 488 bus for printed output.

1. ASCII printers use the most significant bit (MSB) as a parity bit. If the PET is sending a graphics character (or lowercase, as provided by the POKE 59468,14 for old PETs), the printer will either ignore this and print the corresponding ASCII for the seven less significant bits or print a "parity error" character. If you get a parity error character, set your printer to the "no parity," or "mark" parity, option.

2. The PET cursor control characters result in the ASCII values in the range 0 to 31, which are control characters in ASCII. If you are lucky, these will be ignored; if you aren't, some of these may result in setting your printer to unwanted modes. (The Comprint printer is

Listing 4. Serial output, machine-language assembly listing.

```
     This code was hand assembled and then patched - so the flow
isn't continuious and there are occasional NOPs that aren't needed.

                              ! Check SHIFT key
033A AD 04 02  SENSE          LDA SHIFT (0203)     read shift key location
     EA EA                    NOP, NOP             (tis a patch)
     F0 01                    BEQ GO (0342)
     60                       RTS                  back to BASIC if SHIFT
                                                    pressed
                              ! See if device is ready
0342 AD 40 E8  GO             LDA $E840            Get all PB2 lines from VIA
     29 40                    AND #40              Mask NRFD bit
     F0 F1                    BEQ SENSE (033A)     Wait if not ready

                              ! Set up PET for transmission of characters
                              !  Turn off interrupts
                              !  Get character
                              !   Set carry if no more characters
                              !  Set up Xmission table
                              !  Send character

0349 78                       SEI                  Interrupts off
034A 20 C0 03                 JSR FETCH (03C0)     Fetch Character
                                                   (Set up as a subroutine
                                                    to let you "roll your own"
                                                    routine)
     90 02                    BCC GO1 (0351)
     58                       CLI                  Interrupts on. If Carry is
     60                       RTS                  set, no more chars to send.
                                                   If you make your own FETCH,
                                                   use this convention.

0351 20 62 03  GO1            JSR SETUP            Set up Xmit table for char in A
     20 49 03                 JSR XMIT             Send char
     58                       CLI                  restore interrupts
     4C 3A 03                 JMP SENSE            Look at SHIFT key again
     EA                       NOP                  (patch)

035C 18        FFETCH         CLC                  Dummy version of FETCH
     AD E0 03                 LDA CHAR (03E0)      Test Char location
     60                       RTS
     EA                       NOP                  (guess)

                              ! Set up Xmission Table
0362 A9 00     SETUP          LDA #00
     8D E1 03                 STA PARITY (03E1)    Initialize parity counter
     AD E0 03                 LDA CHAR (03E0)      Get char
     A2 01                    LDX #01              X reg counts/7 bits of char.
                              ! Shift char & if carry set, load FF into
                              ! Xmit table. If carry not set, load 00
                              ! (NOTE: Start & Stop bits are assumed preset
                              ! in Xmit table. Be sure this is so in your
                              ! program too.)

036C A0 00     SLOAD          LDY #00              Y holds 00 or FF for bit
     18                       CLC                   in char.
     4A                       LSR A                Shift LSB into Carry
     90 05                    BCC HOPPITY          Bit is zero
     EE E1 03                 INC PARITY (03E1)    '1' bit adds to parity count
     48                       PHA                  Stuff A on stack
     98                       TYA                  Y to A
     9D F0 03                 STA START,X          Put into Xmit table. I just
                                                   love non-symmetrical
                                                   instruction sets! (6502
                                                   has no Y indexed addressing)
     68                       PLA                  Restore A from stack
     E8                       INX                  On to next bit
     E0 08                    CPX #08              7 bits yet?
     D0 EA                    BNE SLOAD (036C)     no, repeat

                              ! According to  option, set the parity
                              ! bit in the Xmit table

0382 AD E2 03                 LDA POPTION (03E2)   Get option value
     30 0C                    BMI MARK             MSB means MARK parity
     F0 03                    BEQ EVEN             zero is EVEN
     EE E1 03                 INC PARITY           Add 1 for odd parity
     AD E1 03  EVEN           LDA PARITY
     29 01                    AND #01              LSB has parity in it
     F0 02                    BEQ ZILCH            Save LDA #00 if A is 00
     A9 FF     MARK           LDA #FF
     9D F0 03  ZILCH          STA START,X          Put in Xmit table. X happens
     60                       RTS                  to be right value!

                              ! Send Character
0399 A2 FF     XMIT           LDX #FF              The next instruction
     E8        CONT           INX                    makes X zero.
     BD F0 03                 LDA START,X          Get byte to send
     8D 22 E8                 STA $E822            Put on IEEE DIO Lines (out)

                              ! Delay loop for baud rate
03A2 AC E3 03                 LDY RATE (03E3)      Get countdown value
03A5 AD 00 40  AGAIN          LDA $0400            4 cycles of delay
     AD 00 40                 LDA $0400              ditto
03AB AD 00 40                 LDA $0400              ditto
```

a "lucky" one.)

3. As a result of these first two steps, if you use CMD and LIST, the listings you get will have missing or misleading characters. I have a program (drop me a card) that will list a BASIC program in a legible form.

4. The PET does not transmit a line feed. You must provide CHR$(10) after every carriage return.

5. If your printer needs a carriage return delay, either print the required number of CHR$(0) or insert a small waiting loop; i.e., FORJ = 1TO20:NEXT.

6. Most printers have no formatting capabilities. If you keep careful count of the number of characters sent, formatting is clumsy, but possible. Pitfalls include:

- A printed number has a CHR$(29) sent after the last digit, which is not a space and is usually ignored by printers.
- TAB and SPC provide CHR$(29), and not spaces. (New PETs have this fixed.)
- LEN(STR$(number)) will not count a CHR$(29), since STR$ produces a string without a blank or skip after the last digit.
- If the number is small or large, beware of scientific format; i.e., 1.23E+23.

7. If you are attempting a word-processing program, the PET's codes for the lowercase characters and the ASCII codes are different. The PET thinks the lowercase letters lie in the range 192 to 223, and ASCII likes the range 96 to 127.

A further complication is that the ASCII character set and the PET character sets don't match. Backarrow on the PET is ASCII underline; the curly brackets, vertical bar and tilde in ASCII don't exist on the PET. The ASCII accent mark (looks like a reverse apostrophe) is seen by the PET as a space. Your printer might have other character options to puzzle you.

Wrapping It Up

Working with the IEEE 488 bus is nearly an entire engineering discipline in itself. I hope my efforts enable you to get

```
        88                    DEY                reduce countdown
        D0 E4                 BNE AGAIN (03A5)   keep going till count is zero
        EA                    NOP                Successful branch takes 3
                                                 so this compensates to
                                                 make a 17 cycle per loop
                                                 delay
        EC E4 03              CPX BITCOUNT       Check number of bits to
                                                 be sent.
        D0 E4                 BNE CONT           Do next bit
        60                    RTS

.......(some room here) .......

                              ! Fetch Character for real. Feel free to
                              ! make your own routine. Set carry bit when
                              ! out of characters.
03C0 18          FETCH        CLC                Be sure to do this!
     AD E5 03                 LDA CHCOUNT (03E5) # chars to send
     D0 02                    BNE OK
     38                       SEC                Set carry, out of chars
     60                       RTS

     AD E0 03    OK           LDA CHAR           Get char - you might use
                                                 TAX & LDA CHAR,X here.
     CE E5 03                 DEC CHCOUNT        decmt chars counter
     60                       RTS

....... (some room here) .......

                              ! Data Area

03E0 00          CHAR         ! Character to send. (Move elsewhere if you
                                   want to send more than one)
03E1 00          PARITY       ! Parity Counter
03E2 00          POPTION      ! Parity Option. 0-even,1-odd,FF-mark
03E3 00          RATE         ! Initial countdown for baud rate. POKEd
                                 by the BASIC program.
03E4 00          BITCOUNT     ! Number of bits to send (10 or 11 decimal)
03E5 00          CHCOUNT      ! Number of chars to send

....... (a gap again) ......

03F0 00          START        ! Start of Xmit table
03F1 00 00 00 00 00 00 00         ! Character, lsb first
03F8 00                       ! Parity bit
03F9 FF FF                    ! Stop bit(s)
```

aboard the IEEE 488 bus of your PET and turn it to some profitable use. ■

References

1. "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std 488-1975, ANSI MC 1.1-1975.
2. Hewlett-Packard, 1502 Page Mill Road, Palo Alto, CA or PO Box 301, Loveland, CO 80537. Several publications are available on request.
3. "PET 2001-8 User's Manual" and "PET Communication with the Outside World," Commodore Business Machines.
4. "Getting Aboard the 488-1975 Bus," Motorola.
5. "PET User Notes," PO Box 371, Montgomeryville, PA 18936.
6. MOS Technology, Inc., 950 Rittenhouse Road, Norristown, PA 19401.

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
0	00		@	end-line	BRK	0
1	01		A		ORA(I,X)	1
2	02		B			2
3	03		C			3
4	04		D			4
5	05	white	E		ORA Z	5
6	06		F		ASL Z	6
7	07	bell	G			7
8	08	lock	H		PHP	8
9	09	unlock	I		ORA #	9
10	0A		J		ASL A	10
11	0B		K			11
12	0C		L			12
13	0D	car ret	M		ORA	13
14	0E	text	N		ASL	14
15	0F	top-left	O			15
16	10		P		BPL	16
17	11	cur down	Q		ORA(I),Y	17
18	12	reverse	R			18
19	13	cur home	S			19
20	14	delete	T			20
21	15	del.line	U		ORA Z,X	21
22	16	er.begin	V		ASL Z,X	22
23	17		W			23
24	18		X		CLC	24
25	19	scr.up	Y		ORA Y	25
26	1A		Z			26
27	1B		[			27
28	1C	red	\			28
29	1D	cur right	]		ORA X	29
30	1E	green	↑		ASL X	30
31	1F	blue	←			31
32	20	space	space	space	JSR	32
33	21	!	!	!	AND(I,X)	33
34	22	"	"	"		34
35	23	#	#	#		35
36	24	$	$	$	BIT Z	36
37	25	%	%	%	AND Z	37
38	26	&	&	&	ROL Z	38
39	27	'	'	'		39
40	28	(	(	(	PLP	40
41	29	)	)	)	AND #	41
42	2A	*	*	*	ROL A	42
43	2B	+	+	+		43
44	2C	,	,	,	BIT	44
45	2D	-	-	-	AND	45
46	2E	.	.	.	ROL	46
47	2F	/	/	/		47
48	30	0	0	0	BMI	48
49	31	1	1	1	AND(I),Y	49
50	32	2	2	2		50
51	33	3	3	3		51
52	34	4	4	4		52
53	35	5	5	5	AND Z,X	53
54	36	6	6	6	ROL Z,X	54
55	37	7	7	7		55
56	38	8	8	8	SEC	56
57	39	9	9	9	AND Y	57
58	3A	:	:	:		58
59	3B	;	;	;		59
60	3C	<	<	<		60
61	3D	=	=	=	AND X	61
62	3E	>	>	>	ROL X	62
63	3F	?	?	?		63

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
64	40	@	▯	@	RTI	64
65	41	A	▯,a	A	EOR(I,X)	65
66	42	B	▯,b	B		66
67	43	C	▯,c	C		67
68	44	D	▯,d	D		68
69	45	E	▯,e	E	EOR Z	69
70	46	F	▯,f	F	LSR Z	70
71	47	G	▯,g	G		71
72	48	H	▯,h	H	PHA	72
73	49	I	▯,i	I	EOR #	73
74	4A	J	▯,j	J	LSR A	74
75	4B	K	▯,k	K		75
76	4C	L	▯,l	L	JMP	76
77	4D	M	▯,m	M	EOR	77
78	4E	N	▯,n	N	LSR	78
79	4F	O	▯,o	O		79
80	50	P	▯,p	P	BVC	80
81	51	Q	▯,q	Q	EOR(I),Y	81
82	52	R	▯,r	R		82
83	53	S	▯,s	S		83
84	54	T	▯,t	T		84
85	55	U	▯,u	U	EOR Z,X	85
86	56	V	▯,v	V	LSR Z,X	86
87	57	W	▯,w	W		87
88	58	X	▯,x	X	CLI	88
89	59	Y	▯,y	Y	EOR Y	89
90	5A	Z	▯,z	Z		90
91	5B	[	▯	[		91
92	5C	\	▯	\		92
93	5D	]	▯	]	EOR X	93
94	5E	↑	▯	↑	LSR X	94
95	5F	←	▯	←		95
96	60		▯		RTS	96
97	61		▯		ADC(I,X)	97
98	62		▯			98
99	63		▯			99
100	64		▯			100
101	65		▯		ADC Z	101
102	66		▯		ROR Z	102
103	67		▯			103
104	68		▯		PLA	104
105	69		▯		ADC #	105
106	6A		▯		ROR A	106
107	6B		▯			107
108	6C		▯		JMP(I)	108
109	6D		▯		ADC	109
110	6E		▯		ROR	110
111	6F		▯			111
112	70		▯		BVS	112
113	71		▯		ADC(I),Y	113
114	72		▯			114
115	73		▯			115
116	74		▯			116
117	75		▯		ADC Z,X	117
118	76		▯		ROR Z,X	118
119	77		▯			119
120	78		▯		SEI	120
121	79		▯		ADC Y	121
122	7A		▯			122
123	7B		▯			123
124	7C		▯			124
125	7D		▯		ADC X	125
126	7E		▯		ROR X	126
127	7F		▯			127

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
128	80		r-@	END		128
129	81		r-A	FOR	STA(I,X)	129
130	82		r-B	NEXT		130
131	83		r-C	DATA		131
132	84		r-D	INPUT#	STY Z	132
133	85	f1	r-E	INPUT	STA Z	133
134	86	f3	r-F	DIM	STX Z	134
135	87	f5	r-G	READ		135
136	88	f7	r-H	LET	DEY	136
137	89	f2	r-I	GOTO		137
138	8A	f4	r-J	RUN	TXA	138
139	8B	f6	r-K	IF		139
140	8C	f8	r-L	RESTORE	STY	140
141	8D	car ret	r-M	GOSUB	STA	141
142	8E	graphic	r-N	RETURN	STX	142
143	8F	btm.rt	r-O	REM		143
144	90	black	r-P	STOP	BCC	144
145	91	cur up	r-Q	ON	STA(I),Y	145
146	92	rvs off	r-R	WAIT		146
147	93	clear	r-S	LOAD		147
148	94	insert	r-T	SAVE	STY Z,X	148
149	95	ins.line	r-U	VERIFY	STA Z,X	149
150	96	er.end	r-V	DEF	STX Z,Y	150
151	97		r-W	POKE		151
152	98		r-X	PRINT#	TYA	152
153	99	scr.dn	r-Y	PRINT	STA Y	153
154	9A		r-Z	CONT	TXS	154
155	9B		r-[	LIST		155
156	9C	magenta	r-\	CLR		156
157	9D	cur left	r-]	CMD	STA X	157
158	9E	yellow	r-↑	SYS		158
159	9F	cyan	r-←	OPEN		159
160	A0		■	CLOSE	LDY #	160
161	A1	▯	r-!	GET	LDA(I,X)	161
162	A2	▯	r-"	NEW	LDX #	162
163	A3	▯	r-#	TAB(		163
164	A4	▯	r-$	TO	LDY Z	164
165	A5	▯	r-%	FN	LDA Z	165
166	A6	▯	r-&	SPC(	LDX Z	166
167	A7	▯	r-'	THEN		167
168	A8	▯	r-(	NOT	TAY	168
169	A9	▯	r-)	STEP	LDA #	169
170	AA	▯	r-*	*	TAX	170
171	AB	▯	r-+	+		171
172	AC	▯	r-,	-	LDY	172
173	AD	▯	r--	/	LDA	173
174	AE	▯	r-.	↑	LDX	174
175	AF	▯	r-/	AND		175
176	B0	▯	r-0	OR	BCS	176
177	B1	▯	r-1	>	LDA(I),Y	177
178	B2	▯	r-2	=		178
179	B3	▯	r-3	<		179
180	B4	▯	r-4	SGN	LDY Z,X	180
181	B5	▯	r-5	INT	LDA Z,X	181
182	B6	▯	r-6	ABS	LDX Z,Y	182
183	B7	▯	r-7	USR		183
184	B8	▯	r-8	FRE	CLV	184
185	B9	▯	r-9	POS	LDA Y	185
186	BA	▯	r-:	SQR	TSX	186
187	BB	▯	r-;	RND		187
188	BC	▯	r-<	LOG	LDY X	188
189	BD	▯	r-=	EXP	LDA X	189
190	BE	▯	r->	COS	LDX Y	190
191	BF	▯	r-?	SIN		191

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
192	C0	▯	↓ Reverse of ASCII (192–223)	TAN	CPY #	192
193	C1	▯,a		ATN	CMP(I),X	193
194	C2	▯,b		PEEK		194
195	C3	▯,c		LEN		195
196	C4	▯,d		STR$	CPY Z	196
197	C5	▯,e		VAL	CMP Z	197
198	C6	▯,f		ASC	DEC Z	198
199	C7	▯,g		CHR$		199
200	C8	▯,h		LEFT$	INY	200
201	C9	▯,i		RIGHT$	CMP #	201
202	CA	▯,j		MID$	DEX	202
203	CB	▯,k		GO		203
204	CC	▯,l		CONCAT	CPY	204
205	CD	▯,m		DOPEN	CMP	205
206	CE	▯,n		DCLOSE	DEC	206
207	CF	▯,o		RECORD		207
208	D0	▯,p		HEADER	BNE	208
209	D1	▯,q		COLLECT	CMP(I),Y	209
210	D2	▯,r		BACKUP		210
211	D3	▯,s		COPY		211
212	D4	▯,t		APPEND		212
213	D5	▯,u		DSAVE	CMP Z,X	213
214	D6	▯,v		DLOAD	DEC Z,X	214
215	D7	▯,w		CATALOG		215
216	D8	▯,x		RENAME	CLD	216
217	D9	▯,y		SCRATCH	CMP Y	217
218	DA	▯,z		DIRECTORY		218
219	DB	▯				219
220	DC	▯				220
221	DD	▯			CMP X	221
222	DE	▯.▯			DEC X	222
223	DF	▯.▯				223
224	E0		■		CPX #	224
225	E1		▯		SBC(I),X	225
226	E2		▯			226
227	E3		▯			227
228	E4		▯		CPX Z	228
229	E5		▯		SBC Z	229
230	E6		▯		INC Z	230
231	E7		▯			231
232	E8		▯		INX	232
233	E9		▯ ▯		SBC #	233
234	EA		▯		NOP	234
235	EB		▯			235
236	EC		▯		CPX	236
237	ED		▯		SBC	237
238	EE		▯		INC	238
239	EF		▯			239
240	F0		▯		BEQ	240
241	F1		▯		SBC(I),Y	241
242	F2		▯			242
243	F3		▯			243
244	F4		▯			244
245	F5		▯		SBC Z,X	245
246	F6		▯		INC Z,X	246
247	F7		▯			247
248	F8		▯		SED	248
249	F9		▯		SBC Y	249
250	FA		▯·▯			250
251	FB		▯			251
252	FC		▯			252
253	FD		▯		SBC X	253
254	FE		▯		INC X	254
255	FF		▯			255

Jim Butterfield