commodore

comments and bulletins
concerning your
COMMODORE PET tm

The Transactor

Vol. 2
BULLETIN # 1
May 31, '79

PETtm is a registered trademark of Commodore Business Machines Inc.

Watching a cassette load

Jim Butterfield, Toronto

It may not be too useful, but it's very satisfying to watch a program coming in from cassette tape. Much of what comes in will look like gibberish, since the program contains obscure things like pointers, flags, and tokens. But it's interesting to see, and here's how you can do it.

Step 1: load any Basic program on cassette 1. The program doesn't matter; the LOAD activity sets up certain internal things that will help us.

Step 2: set up the cassette with any Basic program ready to load. A short one would be good: that way you may catch the whole program on the screen. But any Basic program will do.

Step 3: set graphic mode with POKE 59468,14. This may help you spot a few recognizable pieces of your program.

Step 4: Give SYS 62894. PET will ask you to press PLAY. Do so, and in twenty seconds or so, PET will report FOUND... and stop.

Step 5: Clear the screen so you'll ~~b~~ get a better view of the program as it comes in. Now move the cursor down to a few lines from the bottom of the screen.

Step 6: enter POKE 636,128:POKE 638,132:SYS 62403.

Step 7: sit back and watch the program load to the screen. You won't be able to run it, of course, since it's in the wrong part of memory .. but isn't it fascinating to watch?

Decimal/Hex conversion: a few techniques

Jim Butterfield

If you stay clear of machine language, you'll never need to explore the mysteries of Hexadecimal numbering. If you do need to use this kind of numbering scheme, you'll often want to convert back and forth between decimal and hexadecimal. For example, a program contains a SYS(2345); and you want to use the Machine Language Monitor to see what's in that part of memory. But the MLM wants the address in Hex ... how to convert?

Most of these techniques can be readily done with a pocket calculator, or with a program. But when your calculator battery has gone dead, and your PET is already loaded with a different program that you don't want to disturb, it's handy to work it all out using immediate statements.

Converting Decimal to Hexadecimal.

My favorite quick method is this (let's convert 2345 as an example):

```
   X = 2345/4096 : FOR J=1 TO 4 : ? INT(X); : X=(X-INT(X))*16 : NEXT J
```

.. and out come the numbers 0 9 2 9, representing hex 0929. If you get numbers greater than 9, remember that 10 is written as A. 11 as B, and so on up to 15 as F.

Converting Hexadecimal to Decimal.

This is a simple matter of multiplying the previous total by 16 and adding the new digit. To convert hex 0929 back to decimal we type:

```
   ?  ((0*16 + 9)*16 + 2)*16 + 9
```

and we get our original value of 2345. If you don't like brackets, you could try the alternative:

```
   ? 0*4096 + 9*256 + 2*16 + 9
```

with the same result. In the example, the leading zero can be dropped from the calculation, of course.

Gilding the lily.

You really don't need to dress up immediate statements any more than is shown above. In programs, you'll probably want the value to print in a more classy manner - with the alphabetics already done.

The easiest way is a variant of IF X>9 THEN PRINT CHR$(X+55); but if you like to baffle your friends with obscure coding you can try either or both of these:

```
   X=2346/4096:FORJ=1TO4:Y=INT(X):?CHR$(Y+55+7*(Y<10));:X=(X-Y)*16:NEXTJ

Z=0: X$="092A":FORJ=1TO4:Y= ASC(MID$(X$.J)):Z=Z*16+Y-48+7*(Y>57):NEXTJ:?Z
```

The LIST Chain

One of the most often used commands to be executed directly from the keyboard is LIST: a most fundamental function as it allows us to observe the contents of our BASIC and proceed to implement the screen editor, a feature of the PET that most of us have taken for granted. This very powerful programming tool permits deletion, insertion and alteration of lines and characters. But as all this occurs, PET is busy doing some rather extensive internal housekeeping; checking available space, updating FRE(0), manipulating pointers in zero page and a number of other things.

Of these tasks PET performs for itself, it also creates the LIST chain, a function of equal importance to PET and User. As a line of BASIC is completed, PET inserts three extra bytes of information which it uses to keep track of where the line ends and also where the next line begins. The best way to observe this is to load 'View', one of the machine language programs which appeared in Transactor 10, Volume 1. Proceed as follows:

1. LOAD and RUN View. This will set up the machine language for View in the second cassette buffer.

2. SYS64824. This will clear out the BASIC memory space but will not affect the second cassette buffer.

3. POKE 849,4 This will cause View to display page 4 of memory which is the first block or 'page' of BASIC memory space. (BASIC begins at Hex Ø4ØØ or decimal 1Ø24 which is 4x256.)

4. SYS826 The View program should now be operating and displaying page 4 at the top of the screen. The display should consist of mostly '$' signs representing empty space.

Preceding the '$' signs you should see three '@' signs. The '@' sign represents a zero (try POKE 33400,0). The first '@' or zero in location 1024 is a dummy end-of-line character. The next two zeroes represent the first pointer to the next line of BASIC but since they are zeroes this indicates to PET that nothing exists beyond this point. The three '@'s are automatically placed at the end of the last line of BASIC. The first '@' or zero is automatically placed at the end of every line of BASIC to indicate, of course, 'end-of-line'.

The three zeroes will not stay zeroes for long as we are about to enter into PET the following:

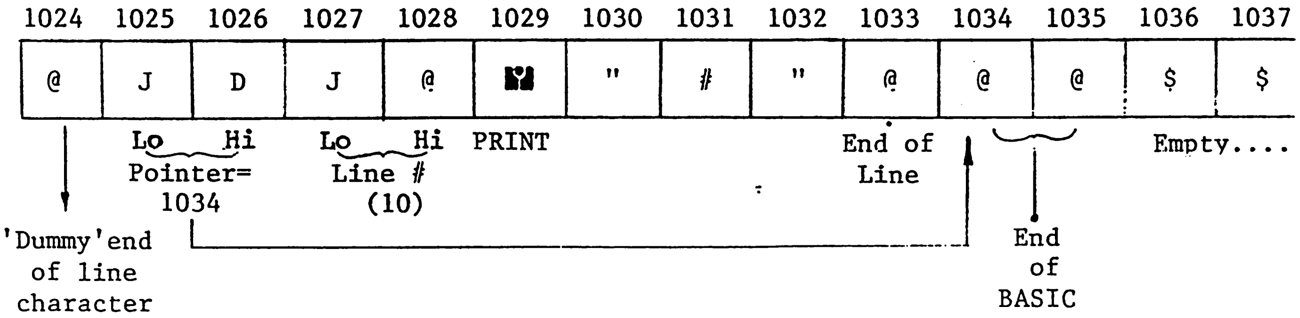
```
10?"#"     (without spaces)
```

Upon hitting 'RETURN' you should notice the top line of the screen change. The first character will still be an '@' representing our dummy end-of-line. (As a rule, location 1024 will always be a zero unless POKEd by the User.) The next two characters should be, in order, a 'J' and a 'D' where J=10 and

D=4. These represent the low order and high order bytes (respectively) of the pointer to next line of BASIC. But just exactly what do they point at? Since these numbers are in hexadecimal, the high order byte is used as a multiplier of 256 and the low order byte is multiplied by 1 and added to give us the decimal byte address. In this case the result will be:

$$\begin{aligned} & P = (D \times 256) + (J \times 1) = (D \times 256) + J \\ \text{OR}\quad & P = (4 \times 256) + (10 \times 1) = 1024 + 10 \\ & = 1034 \end{aligned}$$

If we start counting at 1024 (the 'HOME' position) across 10 character spaces to 1034, we find ourselves at the byte which our pointer points at (Figure 1.0). Since the only existing program consists of line 10, this byte will be a zero as is the following byte.



1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037
@	J	D	J	@	Y	"	#	"	@	@	@	$	$

Figure 1.0

These indicate end-of-valid-BASIC. Notice also the preceding byte which indicates end-of-line.

Getting back to our pointer, immediately following should be another 'J' and an'@' (Figure 1.0). These represent the low and high order bytes, respectively, of the line number. These are also in Hex such that:

$$\begin{aligned} & L.N. = (@ \times 256) + (J \times 1) = (@ \times 256) + J \\ \text{OR}\quad & L.N. = (0 \times 256) + (10 \times 1) = 0 + 10 \\ & = 10 \end{aligned}$$

...which is of course the line number of the only existing BASIC so far.

The next character on the top line is a reverse field 'Y' or the token for 'PRINT' which is 153. (see table following) The remaining characters are self explanatory.

Now let's get a little deeper and enter the following extra code:

```
400?"!"
```

Before hitting 'RETURN' watch closely the two last '@' characters (locations

1034 and 1035). Now hit 'RETURN' and they will change to an 'S' and a 'D' or a 19 and a 4 which equals 1043 (4 x 256 + 19). If we count across to 1043 we find ourselves once again at the second last '@' character indicating end-of-BASIC (Figure 1.1). Recall that our first pointer in locations 1025 and 1026 pointed at 1034. Therefore the first line pointer points at the next line pointer which in turn points at the next line pointer and so on to the end of your BASIC program. Sounds simple doesn't it? Well it is! This is the LIST Chain and PET employs these pointers to execute commands such as:

1. LIST
2. LIST-500
3. LIST500-5000
4. RUN20
5. GOTO500
6. GOSUB1000

When implementing these commands, either directly or under program control, PET immediately jumps to the pointer at 1025 and 1026 and stores it. PET then examines the following two bytes (1027 and 1028) which make up the line number and compares them against the given argument. If none is given the comparison is of course unnecessary. In the case of LIST 500-5000, PET will first compare the line number bytes with 500. If the test yields a "less than", PET recalls the pointer bytes and uses their values to jump to the next pointer. This new pointer is stored in place of the first and the above procedure is repeated until an "equal" or "greater than" test result is obtained. PET then begins LISTing by 'PRINTing' out the converted-to-decimal line number followed by a space followed by the text belonging to that line number. Text continues 'PRINTing' until the zero end-of-line character is detected. PET halts here, recalls the pointer last stored and makes the jump to the next pointer. This pointer is again stored and the line # bytes are examined...but this time compared to the second argument; 5000. If a greater than result occurs the LIST procedure terminates. Otherwise PET continues to:

a) display text while testing for Ø
b) recall pointer in storage
c) jump to new pointer and store
d) examine line # if so instructed
e) continue or terminate

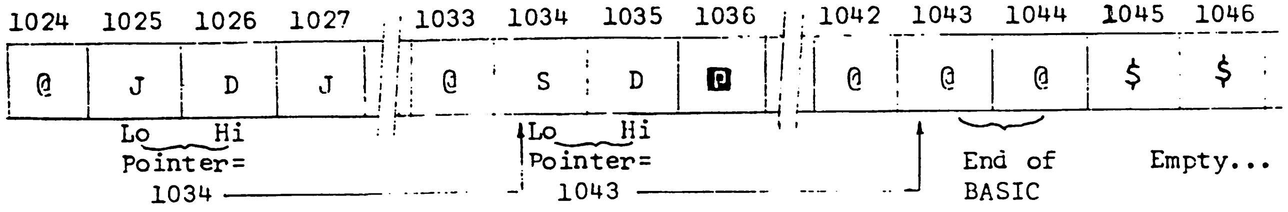


Figure 1.1

Some of you may now be asking....

> "Why does PET do all that back-tracking to recall a pointer value which is simply the address of the byte immediately following the zero end-of-line character? Why doesn't PET just test for zero and have the line # bytes i- the following two locations. This would free up those extra two bytes used by PET every time it creares a line pointer...: "

The answer is speed. The difference would not be noticed so much with LIST or even LIST with parameters. The real decrease in speed would occur upon execution of GOTO or GOSUB instructions. PET would have to test every byte for a zero (starting at 1024) and then, of course, look at the line #. This testing for zero would take some time, especially if the average number of bytes per line were up around 30 or 35. Coupled with the number of GOTOs and GOSUBs in your program, BASIC execution speed would be considerably slower. Using the present method, PET skips across the pointers like a frog across the lily pads (only it won't eat up your bugs at the same time).

Insertion and Deletion

When we program a line of text that is to fall between two existing lines, PET must know where to put it. We won't discuss how PET splits the existing code; that's another subject. PET jumps along looking at line numbers until an "in-between" condition is satisfied. Existing text is moved up from the right point exactly far enough for the code to be inserted. The pointers (line pointers and pointers in zero page) are updated and control is returned to the keyboard.

For deletion of lines, PET simply finds the line and "squeezes" it out. Pointers are updated....operation complete.

Try experimenting with the View program to watch how PET handles its editing.

Assuming that View is still running, type in NEW but before hitting 'RETURN', record the second and third characters of page 4. They are reset by a 'NEW' but notice how the rest of memory still exists. If these locations are POKEd back to what they were, the program can now be LISTed once again. However, zero page pointers were all reset by NEW also. Editing or assigning variables to values will cause a crash (and a rather interesting one at that) so do not try a RUN. About the only way to 'SAVE' it is to use the UNLIST routine.

What You Can Do

Some interesting results can be obtained by manipulating these line pointers; particularly locations 1025 and 1026. If we POKE1025,0 we've essentially aimed the pointer at location 1024 (4 x 256 + 0). If a LIST is executed, PET will pick up the first pointer, display the text and jump to 1024. Since 1024 is a zero and is now followed by a zero, PET is fooled into thinking end-of-BASIC....try it!

Similarly, if we POKE1025,1 we have aimed the first pointer at itself! Now try a LIST. By the same token you can point that first pointer (or any pointer for that matter) at any other pointer in BASIC; but only at pointers ....anything else and PET will crash. By doing this manipulation of pointers, you can have LIST-inhibit on any lines you wish without affecting RUN (so long as you do not use RUN with arguments that lie within the LIST-inhibited lines). Be careful though....mistakes can be hazardous!

Now let's have some fun with View. Type the following on a clear screen near the bottom (View, of course, will not clear):

```
FOR T = 32 TO 135: POKE849,T: FOR R = 1 TO 250: NEXT R,T
```

Hit 'RETURN' and next month we'll discuss how PET stores variables.

16/32 K PET Notes - Collected by Jim Russo

The Operating System of the 16/32K PET is about 90% the same as the 8K PET, but has been re-assembled so that almost everything is in a slightly different place in memory than it used to be. Most bugs have been fixed and some limitations removed.

Any pure BASIC program (no PEEK, POKE, SYS, or WAIT) that works on an 8K PET should also work on a 16/32K PET. POKing and PEEKing screen memory (32768 to 33767) is still safe but POKing the operating system (below 1024 decimal) or using an operating system PEEK value to make a decision could be hazardous. Other programs can be made to work properly if references to RAM and ROM locations are changed. Commodore's 16/32K PET manual contains a memory map for pages 0. 1 and 2. A list of new ROM addresses follows. These two lists should contain the information needed in most cases.

Some Hardware Differences:

- The character generator ROM has been revised so that when lower case mode is selected, upper and lower case are interchanged. That is, the ' SHIFT' key must be used to obtain an upper case character. Also, 8K programs using lower case that are run on a 16/32K PET will display all lower case as upper case and vice versa.

- The signal which blanks the video on the 8K is not connected on the 16/32, so POKE 59409, 52 no longer works. The ROM routines still reference this address but the required hardware seems to have been omitted.

Summary of Differences:

- The bug in TI has been fixed. Now every 623rd. interrupt doesn't increment TI. Also, TI is allowed to count 1/60 sec. too far: 24ØØØØ precedes ØØØØØØ.

- Execution (of at least some code) is faster due to more efficient coding and better use of zero page. PRINT (to screen) is faster because extra code to maintain separate POS pointer has been eliminated. Also, screen snow and 'scroll - up flash' has been eliminated thanks to dynamic screen RAMs.

- Standard typewriter operation i.e., shift for upper case.

- RND (0) returns a number derived from interval timers.

- OPENing more than 10 files no longer crashes system.

- OPEN statement correctly sets "current tape buffer pointer".

Review: Basic for Home Computers John Wiley & Sons, Inc.
by Bob Albrecht, LeRoy Finkel, and Jerald R. Brown.

A good teaching book that deals with MICROSOFT™ Basic fundamentals. This is the type of Basic that PET has, and readers will find it a suitable introduction to PET programming.

The book is a self-teaching guide, which means that on almost every paragraph you are asked to "fill in the blanks". The idea behind this type of programmed instruction is that you do more than read - you participate. This makes for a good teaching text; but the book becomes less useful for reference or quick scanning.

The absolute beginner with a PET will encounter a few stumbling blocks at the start of the book. This is due to slight differences in the Basic being described. PET says READY instead of OK; it says ?SYNTAX ERROR instead of SN ERROR; it uses the Delete key instead of the back arrow to correct entry errors. All very minor problems, but they might shake the confidence of a neophyte. Once he gets over these initial rough spots, however, it's all clear sailing. By chapter three, the authors get into the meat of Basic programming, and the reader should have no further trouble.

Each chapter is well organized. First, you're told what you may expect to learn in this chapter. Then the text material, broken into neatly numbered sections. Liberal use is made of illustrations, diagrams, and sample programs; and the programs are usually aimed towards real world examples, not just abstract mathematics. Finally, each chapter ends with a self-test, complete with answers, which allows you to review and make sure you've got it all right.

The order of the chapters is well planned, proceeding from the more basic programming commands towards more advanced concepts. The authors don't suggest it, but by Chapter 5 the reader is equipped to skip ahead to subjects in which he may have a special interest. So if he's anxious to learn about string variables or about subroutines, he can leap ahead to chapter 9 or 10. It's nice to see a book that's so well organized that you can do this.

The book has a light touch, especially in the illustrations and choice of programs. It helps to relieve the hard slugging that is often needed to learn a programming language.

The book is a pretty good approach to learning Basic. It won't take the reader into advanced concepts, but it will give him a good start.

Review by Jim Butterfield

PET RESET SWITCH

The PET Reset Switch allows you to regain control of your PET without turning the power OFF and back ON. If the blinking cursor should disappear, and none of the keyboard keys work, simply press the button on the Reset Switch, and the message "*** COMMODORE BASIC ***, 7167 BYTES FREE, READY" (8K PETS) followed by the cursor will be displayed on the screen.

You lose whatever Basic program that was in the computer, but it does not clear memory contents below $0400. Since the PET is not reset by the ON/OFF power switch, you avoid the danger of electrical failure in the power supply, CRT display and the logic components.

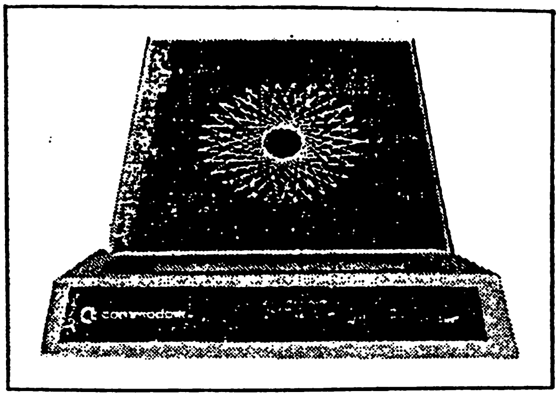
The PET Reset Switch may be installed on all PET models. Installation is simple and full instructions are supplied. The switch contains a protective circuit to avoid damage to the computer in the event it is installed incorrectly.

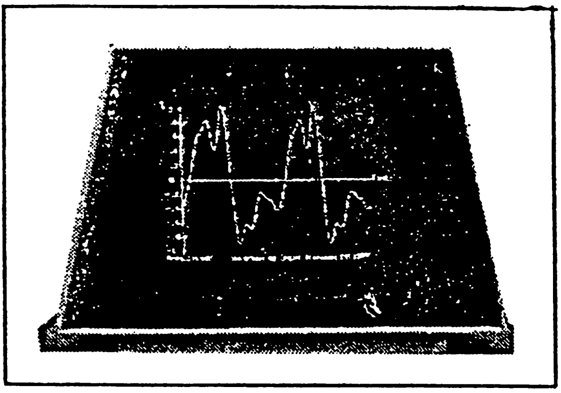
To obtain a PET Reset Switch send $10.00 (cheque or money order) to :

G.P. Reithmeier
411 Duplex Ave., Apt. 611
Toronto, Ontario
M4R 1V2

PET is the registered trademark of Commodore International Inc.

REAL GRAPHICS FOR PET





How would you like to display some REAL graphics on your PET computer? Real graphics such as finely detailed mathematical curves or perspective drawings or computer art or proportionally spaced text or whatever directly on the PET screen. Graphics in which each individual dot in a huge 64,000 dot array 320 wide and 200 high is INDIVIDUALLY controllable. As a bonus, how would you like an extra 8K of memory when super graphics were not being used?

At last the famous MTU VISIBLE MEMORY available to KIM-1 users for well over a year is now available with an adaptor for the PET. The K-1007 PET-to-MTU adaptor plugs into the PET expansion connector and includes a cable and edge socket assembly which will accept our K-1008-PET Visible Memory board. Most all of the assembly can be hidden inside the PET if only the Visible Memory is used or the unit can be plugged into our K-1005 motherboard/card file for further expansion with other advanced design MTU boards. Use of the K-1007 does not tie up the PET's expansion port however.

Besides the bus adaptor, the K-1007 contains video processing circuitry to translate standard Visible Memory composite video into the form required by the PET monitor although composite VM video is still available for an external CCTV monitor. An on-board latch allows software switching between standard PET video and VM video. All necessary operating power is taken from the PET transformer but as most of our customers know, MTU boards require very little power for operation.

K-1008-PET $243.00 K-1007 $79.00

ALSO REAL 4 VOICE MUSIC

By now you have probably heard of the "DAC music system" for the KIM-1 that was pioneered by Hal Chamberlin in association with MTU. Up to 4 independent, simultaneous musical voices, each with an independent waveform, can be synthesized for true 4-part harmony. While software is the key to such performance, our K-1002-3C software package allows most of the potential of the technique to be realized on a standard 8K PET.

We have now redesigned the original DAC board specifically for the PET. The K-1002-2A operates from a single 5 volt power supply and includes an accurate 8 bit digital-to-analog converter, sharp 6 pole low-pass filter, and audio power amplifier usable with any kind of speaker. The board plugs directly onto the PET user port and second cassette port for power yet all signals feed-through to a second set of board edge fingers.

K-1002-2 $50.00 K-1002-3C $20.00

WANT MORE EXPANSION YET?

K-1005 5 Slot Card File $75.00 K-1016 16K Low Power Memory $340.00

K-1012 24K PROM, PROGRAMMER, I/O, COMM $237.00

K-1020 Prototyping Board $42.00

MICRO TECHNOLOGY UNLIMITED
841 Galaxy Way, P.O. Box 4596 • Manchester, NH 03108
603-627-1464