commodore

comments and bulletins
concerning your
COMMODORE PET

The Transactor

BULLETIN #5

September 30, 1978

1. PET INSTRUCTION BOOK SET

All PETs are currently being shipped with the revised instruction book, plus a booklet titled "Pet Communications With The Outside World" (this information was issued with Transactor Bulletins 3 and 4), plus the booklet containing a description of the programs "Bigtime", "Squiggle", and "Machine Langugage Monitor" plus either the tape, or a printout of the programmes if the tapes are not available. Also available on request is a revised issue of the "Compendium of Notes." Enclosed is copy of all of the above - except "Pet Communications With The Outside World" which was issued in the last two bulletins, and "Bigtime/Squiggle" which were issued in Transactor Bulletin #2.

2. PERIPHERALS

C2N Second Cassette - available ex-stock $100 USA, $120 CDA.

Printer - Although many delivery dates have been quoted, the Commodore Printer is unlikely to be available before January, 1978. $695 USA, $995 CDA.

Floppy Disk - Still on target for the end of the year. Specification will certainly be worth waiting for. $1,000 USA approximately, $1,500 CDA. approximately. Dual drive.

Expansion Memory - End of the year. No price yet.

Modem - No price or delivery date yet.

Although only C2N cassette is available, yet various printers with parrallel and serial interfaces giving ASC11 characters will work with PET™ using adaption leads e.g. Serial from Connecticut Micro Computer, Parrallel from Computer Factory (see this bulletin). Also IEEE Floppy from H.P. or Tectronix, and Expansion Memory from Convenience Living Systems (see Bulletin 4). For modem, see Bulletin 2.

3. SOFTWARE

The only programmes available now are listed in this Bulletin.

4. Part 12 of the contents of Bulletin #4 was not available on August 31st, however this is listed below:-

12. Default Parameters

Table 12-1. Default values.

Parameter	Default Value	Default Operation
Device #	D=1	Cassette #1 selected
Secondary address	SA=Ø	On tape files open for read On IEEE-488 devices, no secondary address is sent.

Table 12-2. Example of default parameters.

Statement	Equivalent (Default) Parameter Values	Operation
OPEN#1	OPEN 1,1,Ø	Open logical file #1 for cassette #1 read no file name
OPEN#1,2	OPEN#1,2,Ø	Open logical file #1 for cassette #2 read no file name
OPEN#1,2,1	OPEN#1,2,1	Open logical file #1 for cassette #2 write no file name
OPEN#1,2,1, "DAT"	OPEN#1,2,1, "DAT"	Open logical file #1 for cassette #2 write file named "DAT"

5. Pages 9-35 are extract from the PET Users Club letter in Britain.

```
****************************************************
 1)  ROM112.MIC;1
 799     C589 8DFD01   M=A    STA 01FD
 800     C58C 68       (      PLA
 801     C58D 8DFE01   M>A    STA 01FE
 802     C590 A2FC     "<     LDX #FC
 803     C592 9A       Z      TXS
 804     C593 A900     )@     LDA #00
 805     C595 858D     EM     STA 8D
 806     C597 8561     E!     STA 61
 807     C599 60              RTS
 808     C59A 18       X      CLC
***************
 2)  ROM192.MIC;1
 799     C589 A8       (      TAY
 800     C58A 68       (      PLA
 801     C58B A2FE     ">     LDX #FE
 802     C58D 9A       Z      TXS
 803     C58E 48       h      PHA
 804     C58F 98       X      TYA
 805     C590 48       h      PHA
 806     C591 A900     )@     LDA #00
 807     C593 858D     EM     STA 8D
 808     C595 8561     E!     STA 61
 809     C597 60              RTS
 810     C598 5160     Q      EOR (60),Y
 811     C59A 18       X      CLC
****************************************************
```

DIFFERENCES FOUND BETWEEN ROM 011 AND ROM 019
TO CORRECT LOSS OF CURSOR.

NEW PRODUCTS FOR THE COMMODORE PET 2001

"Getting Started With Your PET" is a new workbook now available to PET users who are anxious to put their PET to work.

This beginner's workbook supplements the documentation provided by Commodore. It covers the fundamentals of PET BASIC and explains its characteristics, limitations, and useful features. The descriptive text is heavily laced with step-by-step, detailed exercises including the expected PET responses.

If you are already an expert on your PET, "Getting Started With Your PET" is an excellent guide for other members of your family who want to use the PET.

In addition to this beginning text, workbooks on advanced topics are available. Some of the advanced techniques covered in these workbooks include string handling, arrays and loopings, graphics, cursor control, PEEK and POKE memory, programmed cassette I/O, real time clock, linkage to assembly language subroutines, subroutine nesting.

TIS also provides software applications for the PET. These programs are available as source listings and cassettes with operating instructions, theory of operations description, and performance time and space limitations.

For more information contact Total Information Services, P.O. Box 921, Los Alamos, NM 87544.

1.6 MEGABYTE FLOPPY DISK SYSTEM

Datatronics has an 800K byte and 1.6M byte S-100 floppy disk storage system. Based on the PerSci Model 277 drive with voice coil head positioning, this system offers more storage in a standard size drive than most other currently available drives.

The 800K byte model is a single density drive system, while the others employ dual-density recording techniques. The S-100 controller is processor independent, and can be used with most 8080, 8085, Z-80 and 6502 based systems as well as with the Datatronics 6800 CPU (S-100 based). Several formats are allowed, including IBM 3740.

Software included with the systems is written for the 6800, but 8080 (8085 and Z-80) versions will be available soon. Termed SDOS, this Disk Operating System offers full dynamic file allocation and file maintenance. That means that data or program files may expand or shrink as needed with all necessary housekeeping being totally transparent to the user.

SWTPC (SS-50) and Digital Group Bus-Compatible systems are now available. Several 6800 based Business languages and complete 6800 Business packages are also available.

Complete systems start at $1999 (includes drive, case with fan and power supply, controller, cable and SDOS on disk). Availability is stock to 4 weeks. For more information, contact Datatronics, 208 W. Olive, Lamar, CO 81052 (303) 336-7956.

6502 ASSEMBLER FOR PET 2001

The 6502 Assembler in BASIC is designed to run on an 8K Commodore PET. It accepts all standard 6502 instruction mnemonics, pseudoops and addressing modes, and evaluates binary, octal, hex, decimal, and character constants, symbols and expressions. Source statements can be read from cassette or from DATA lists and machine code can be assembled anywhere in memory or directed to an external device through a user-supplied subroutine.

The package includes a text editor in BASIC, and an execution monitor with a disassembler. Price with documentation is $24.95 by cheque or Visa/MC from Personal Software, P.O. Box 136-17, Cambridge, MA 02138, (617)783-0694.

MACHINE LANGUAGE MONITOR

Below is a listing of the Machine Language Monitor in machine language (Hex), if listed into the PET this can be saved on cassette and run in the PET.

```
0400 00 0D 04 0A 00 9E 28 31
0408 30 33 39 29 00 00 00 A9
0410 27 8D 1B 02 A9 04 8D 1C
0418 02 A9 07 85 7D A9 6B 85
0420 7C A9 43 85 21 D0 12 A9
0428 42 85 21 D8 4A 68 85 1E
0430 68 85 1D 68 85 1C 68 85
0438 1B 68 69 FF 85 19 68 69
0440 FF 85 1A BA 86 1F 58 20
0448 F2 04 A6 21 A9 2A 20 22
0450 06 A9 52 85 0D D0 2B A9
0458 00 85 CA 85 0D 85 0A 20
0460 F2 04 A9 2E 20 D2 FF A6
0468 20 E0 02 F0 04 E0 03 D0
0470 06 20 3A 06 20 37 06 20
0478 90 06 C9 2E F0 F9 C9 20
0480 F0 F5 A2 07 DD 02 05 D0
0488 0F A5 20 85 0E 86 20 BD
0490 0A 05 48 BD 12 05 48 60
0498 CA 10 E9 A9 3F 20 D2 FF
04A0 4C 57 04 38 A5 13 E5 11
04A8 85 0B A5 14 E5 12 A8 05
04B0 0B 60 A5 11 85 19 A5 12
04B8 85 1A 60 85 21 A0 00 20
04C0 3A 06 B1 11 20 13 06 20
04C8 F7 04 C6 21 D0 F1 60 20
04D0 5E 06 90 0D A2 00 81 11
04D8 C1 11 F0 05 68 68 4C 9B
04E0 04 20 F7 04 C6 21 60 A9
04E8 1B 85 11 A9 00 85 12 A9
04F0 05 60 A9 0D 4C D2 FF E6
04F8 11 D0 06 E6 12 D0 02 E6
0500 0A 60 3A 3B 52 4D 47 58
0508 4C 53 05 05 05 05 05 05
0510 06 06 C1 B1 2C 5E D7 FD
0518 9E 9E 20 50 43 20 20 53
0520 52 20 41 43 20 58 52 20
0528 59 52 20 53 50 A5 0D D0
0530 06 20 F2 04 20 37 06 20
0538 37 06 A2 00 BD 1A 05 20
0540 D2 FF E8 E0 13 D0 F5 20
0548 F2 04 A2 2E A9 3B 20 22
0550 06 20 37 06 20 08 06 20
0558 E7 04 20 BB 04 F0 4D 20
0560 90 06 20 4F 06 90 48 20
0568 3F 06 20 90 06 20 4F 06
0570 90 3D 20 3F 06 A0 00 B9
0578 4A 07 30 06 20 D2 FF C8
0580 D0 F5 29 7F 20 D2 FF 20
0588 2A F3 F0 20 A6 0A D0 1C
0590 20 A3 04 90 17 20 F2 04
0598 A2 2E A9 3A 20 22 06 20
05A0 37 06 20 04 06 A9 08 20
```

MACHINE LANGUAGE MONITOR (Continued)

```
05A8 BB 04 F0 DB 4C 57 04 4C
05B0 9B 04 20 5E 06 20 4F 06
05B8 90 03 20 B2 04 20 E7 04
05C0 D0 0A 20 5E 06 20 4F 06
05C8 90 E5 A9 08 85 21 20 90
05D0 06 20 CF 04 D0 F8 F0 D4
05D8 20 CF FF C9 0D F0 0C C9
05E0 20 D0 CC 20 4F 06 90 03
05E8 20 B2 04 A6 1F 9A A5 1A
05F0 48 A5 19 48 A5 1B 48 A5
05F8 1C A6 1D A4 1E 40 A6 1F
0600 9A 4C 8B C3 A2 01 D0 02
0608 A2 09 B5 10 48 B5 11 20
0610 13 06 68 48 4A 4A 4A 4A
0618 20 2B 06 AA 68 29 0F 20
0620 2B 06 48 8A 20 D2 FF 68
0628 4C D2 FF 18 69 06 69 F0
0630 90 02 69 06 69 3A 60 20
0638 3A 06 A9 20 4C D2 FF A2
0640 02 B5 10 48 B5 12 95 10
0648 68 95 12 CA D0 F3 60 20
0650 5E 06 90 02 85 12 20 5E
0658 06 90 02 85 11 60 A9 00
0660 85 0F 20 90 06 C9 20 D0
0668 09 20 90 06 C9 20 D0 0E
0670 18 60 20 85 06 0A 0A 0A
0678 0A 85 0F 20 90 06 20 85
0680 06 05 0F 38 60 C9 3A 08
0688 29 0F 28 90 02 69 08 60
0690 20 CF FF C9 0D D0 F8 68
0698 68 4C 57 04 4C 9B 04 20
06A0 90 06 A9 00 85 EE 85 FA
06A8 A9 23 85 F9 20 5E 06 29
06B0 0F 85 F1 20 90 06 A2 00
06B8 20 CF FF C9 2C F0 55 C9
06C0 0D F0 0B E0 10 F0 F1 95
06C8 23 E6 EE E8 D0 EA A5 20
06D0 C9 06 D0 C8 A2 00 8E 0B
06D8 02 A5 F1 D0 03 4C 9B 04
06E0 C9 03 B0 F9 20 67 F6 20
06E8 3B F8 20 FF F3 A5 EE F0
06F0 08 20 95 F4 D0 08 4C 9B
06F8 04 20 AE F5 F0 F8 20 4D
0700 F6 20 22 F4 20 8A F8 20
0708 13 F9 AD 0C 02 29 10 D0
0710 E5 4C 57 04 20 4F 06 A5
0718 11 85 F7 A5 12 85 F8 20
0720 CF FF C9 20 F0 F9 C9 0D
0728 F0 A4 C9 2C F0 03 4C 9C
0730 06 20 4F 06 A5 11 85 E5
0738 A5 12 85 E6 A5 20 C9 06
0740 F0 92 A2 00 20 B1 F6 4C
0748 57 04 0D 20 20 20 20 20
```

```
 50 CLR:PRINT"♥";:POKE 245,6:PRINT
 100 PRINT"THIS  PROGRAM  TESTS  YOUR  REFLEXES  BY"
 200 PRINT"MEASURING YOUR REACTION  TIME.  WHENEVER"
 300 PRINT"THE SCREEN IS CLEARED HIT ANY CHARACTER-"
 400 PRINT"YOUR  REACTION TIME  IN  SECONDS WILL BE"
 500 PRINT"DISPLAYED--WHEN  THIS  DISSAPPEARS   HIT"
 600 PRINT"ANOTHER KEY (ANY KEY WILL DO) AND SO ON-"
 650 FOR I=1 TO 7500:NEXT:PRINT TAB(15)"▣GET READY"
 700 FOR I=1 TO  2500:NEXT:PRINT"♥":POKE 245,11
 800 PRINT:PRINT TAB(11)"(▣HIT ANY KEY NOW■)":GETA$,A$,A$,A$,A$,A$,A$:GOTO 1100
 1000 FOR I=1 TO RND(1)*2000+750:GET C$:NEXT:PRINT"♥";
 1100 T=TI:FOR I=1 TO 500:GET C$:IF C$<>"" THEN 1500
 1200 NEXT:PRINT"♥":POKE 245,10:PRINT
 1300 PRINT"▣ YOU SHOULD HAVE TYPED A CHARECTER WHEN ";
 1350 PRINT"▣                                        ";
 1400 PRINT"▣          THE SCREEN WAS CLEARED        ";
 1420 FOR I=1 TO 1000:NEXT
 1425 FOR I=1 TO 40:PRINT"▣ ";:NEXT
 1430 PRINT"▣    (STAND BY FOR MORE INSTRUCTIONS)    "
 1440 FOR I=1 TO 1000:GETC$:IF C$<>"" THEN 1500
 1470 NEXT:GOTO 50
 1475 GOTO 50
 1500 T1=TI-T:PRINT"♥";:POKE 245,11
 1530 FOR I=1 TO 2500:GETC$:IF C$<>"" THEN 1500
 1550 PRINT:POKE 226,17
 1600 PRINT INT((T1/60*1000)+.5)/1000:GOTO1000
READY.
```

commodore PET™

SOFTWARE PROGRAMS

The following programs on cassette are now available from Commodore and shortly its dealers. Each program uses 8K of RAM unless otherwise specified.

DIET PLANNER AND BIORHYTHM

Diet Planner (by Les Palanik) determines your ideal weight and computes the number of calories needed each day to maintain that weight or reduce to that weight. Biorhythm displays a chart of the intellectual, emotional and physical biorhythmic cycles.

Part 321002 — $14.95 USA / $15.95 CDA

TARGET PONG AND OFF THE WALL

Target Pong. Insert paddles in the path of a fast moving ball to deflect the ball into a target. The secret is to use the fewest number of paddles and the least time to hit the target just once. Off The Wall is exactly the opposite. Here the secret is to use as many paddles as you can without hitting the targets. And just to make the game more difficult, there are many targets in this game.

Part 321003 — $ 9.95 USA / $10.95 CDA

BASIC BASIC

Basic Basic (by Ralph James and Ron Lodewyck) is a tutorial program introducing you to the BASIC language. Thoroughly interactive; your PET will teach you how to operate your PET! Basic Basic is written by two very experienced college professors. The topics covered include line numbers, variables, strings, arrays, and the use of the various commands such as LIST, RUN, and SAVE. Also basic keywords will be explained and used such as PRINT, READ/DATA, INPUT, IF/THEN, GOTO, and FOR/NEXT. Fifteen chapters, six sample programs...and homework assignments. Uses just 4K of RAM memory.

Part 321005 — $14.95 USA / $15.95 CDA

GALAXY GAMES

Galaxy Games (by Peter Ruetz). Maneuver your space ship while firing at the enemy, and at the same time avoid hitting a star. In one game, you're firing at fixed targets. In the other game, you're firing at a spaceship that's being piloted by an obviously drunk astronaut!

Part 321006 — $ 9.95 USA / $10.95 CDA

MORTGAGE

Mortgage computes the payment amount, given the principal, interest rate and term of the loan. For any payment period it computes the amount that is principal and the amount that is interest (amortization schedule) and gives the interest, principal, and total amount paid to date.

Part 321007 — $14.95 USA / $15.95 CDA

SPACE FIGHT

Space Fight (by Leonard K. Sweatman). Fire missiles at each other in this two player game.

Part 321010 — $ 9.95 USA / $10.95 CDA

P19789

488 PARALLEL

IEEE 488 TO PARALLEL OUTPUT PORT

The 488/Parallel provides a flexible uni-directional interface from an IEEE 488 bus to any device with a TTL compatible, 8 bit parallel port with handshake. It is contained on a single printed circuit board designed to plug directly into the interface slot of any Centronics printer except Model 779. It is optionally available mounted in an attractive cabinet for use with the Centronics 779 or with other manufacturers' (e.g. Data Products, Integral Data, etc.) equipment.

Users of the Commodore Pet, Hewlett Packard computers, or intelligent instruments will find in this product an economical and effective method for connecting to their systems any of the many peripherals which have a parallel interface. Expensive optional adapters can be avoided while the handshake protocall allows data transfer to occur at the peripheral's maximum rate.

Convenient, on board dip switches tailor the signal polarities, device address (full 5 bit), and pulse durations to those required by your application. The IEEE capabilities supported are acceptor handshake, listener, and interface clear.

The unit comes fully documented and with a 90 day guarantee.

PRICE LIST

488/Parrallel Pet — Pet to Centronics printer interface complete, assembled and tested with necessary cabling

$225

488/Parallel Pet — Interface mounted in cabinet with connectors to Pet and 1 device (specify CEIA or microribbon connectors)

$255

OPTIONS

Power supply (not req'd when used with Centronics printers)

IEEE/488 connector instead of Pet connector

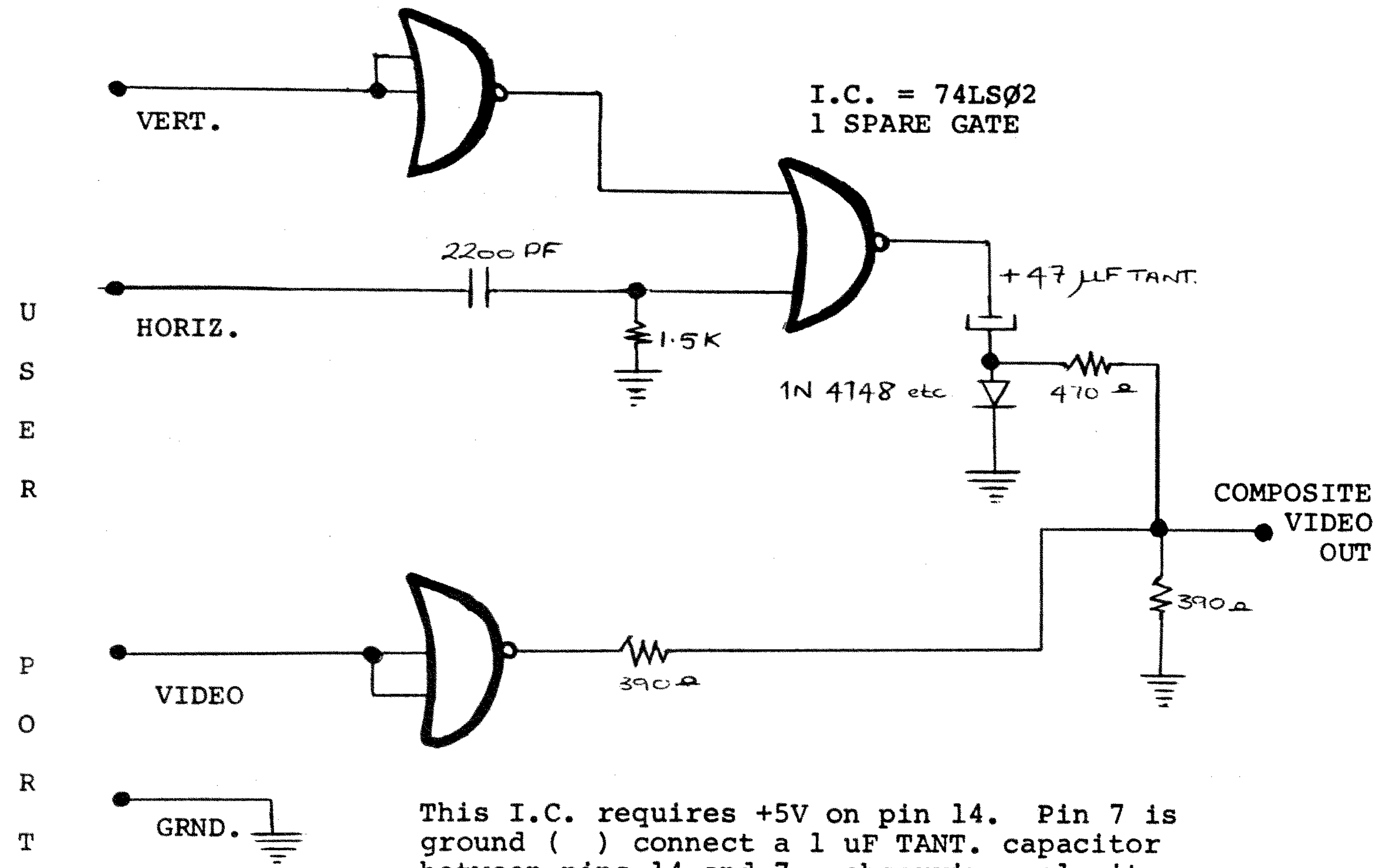
IEEE/488 connector in addition to Pet connector

AVAILABLE SOON

Bi-directional capability

DEALER INQUIRIES INVITED

ATTACHING A VIDEO MONITOR TO PET



This I.C. requires +5V on pin 14. Pin 7 is ground ( ) connect a 1 uF TANT. capacitor between pins 14 and 7. observing polarity

Above is a simple circuit which takes the horizontal drive, vertical drive and video waveforms from the PET User Port and converts them to composite video suitable for driving an RF modulator or a straightforward monitor. The circuit requires a 5 volt power supply and this may be obtained from a 2nd cassette socket which has a few milliamps available at 5 volts. There are no particular points to watch out for when constructing this circuit. Lay-out is not critical. In the unlikely event of the horizontal hold of your display device misbehaving, adjust the value of the 1.5K resistor. This will alter the horizontal sync. pulse width.

DELAYS

Quite a few people have asked how to put delays into programs. Here are two common methods:

10 FOR A = 1 to 1000 : NEXT this will cause a delay of approximately 1 second

10 FOR A = 1 to 2000 : NEXT this will cause a delay of approximately 2 seconds etc.

10 T=TI

20 IF TI - T < 60 THEN 20

Lines 10 and 20 cause a delay of approximately one second and work as follows:

Line 10 sets the variable T equal to the real time jiffy clock TI (a jiffy is 1/60 of a second)

Line 20 tests to see whether 60/60 of a second have elapsed, if not the program returns to the beginning of line 20 and checks again.

Here is a small program you might like to try which uses delays involving the real time clock in an interesting manner.

```
READY.

 5 PRINT"KEY IN A NUMBER>";
 10 T=0:A$=""
 20 GETK$:IFK$=""THEN20
 30 T=TI:GOTO60
 40 GETK$
 50 IFTI-T>60THEN70
 60 IFK$<>""THENPRINTK$;:A$=A$+K$:T=TI:GOTO40
 65 GOTO40
 70 IFA=0THENPRINT"+";:A=VAL(A$):GOTO10
 80 PRINT"="A+VAL(A$)
READY.
```

PLOTTING

It is possible, with very little effort, to address locations on the screen using simple XY co-ordinates. Below we have a program that uses a simple formula that enables one to do this.

```
READY.

 5 DATA12,15,22,5,12,25,33
 1Ø PRINT""
 2Ø PI=3.14159265
 3Ø FORA=ØTO4*PI STEP(4*PI)/39
 4Ø Y=INT(SIN(A)*12+12):X=X+1
 5Ø GOSUB8Ø
 6Ø NEXT
 7Ø FORA=33568TO33574:READZ:POKEA,Z:NEXT
 75 GOTO75
 8Ø POKE((24-Y)*4Ø+32768)+X,46:RETURN
READY.
```

The line that does the actual XY co-ordinate conversion is line 8Ø. For the sake of clarity line 8Ø has been made a sub-routine but the formula is so compact that in some cases, including this one, it is not necessary. Line 5 and 7Ø should be included when you test this program out but may be omitted subsequently. X has a range of Ø-39 and Y has a range of Ø-24.

INPUTTING

It is worth pointing out that commas and colons act as delimiters in input strings, eg.

10 INPUT A $:·? A $ If this program is run and you type HOWEVER, I THINK the machine will accept HOWEVER and print the error message EXTRA IGNORED. The same will happen if you use the : in similar circumstances. If you wish to include either of these characters in an input statement enclose your typed INPUT in quotes. Many people must have been annoyed by the way BASIC will abort if the return key is pressed when the machine is waiting on an INPUT statement and no data has been typed in.

It is possible to arrange an input statement so that it will never do this. The method is as follows:

(note; → means CURSOR RIGHT and ← means CURSOR LEFT)

```
10 INPUT "→ → * ←← ←  ";A
```

When this input statement is encountered the user must type a number in reply, anything other than a number, including no entry at all, will cause the machine to return to the input statement with the appropriate message. Symbols other than * can be used where required.

DATA FILE ERRORS

There is a bug in the file handling routine which causes data to be written on the tape prematurely, not allowing for cassette motor start up time.

This is temporarily curable by keeping the motor running whilst the tape buffer is being filled, or by starting the motor when the buffer is almost filled.

The method of turning on the motor is to change a bit in the appropriate PIA register. The location of the PIA register is 59411 and the correct byte to place in that register is 53. Therefore the syntax for turning on the cassette motor is POKE 59411,53. This should be done either every time PRINT # is used or just before the buffer is full. Using the latter method involves PEEKING location 625 which is the buffer pointer. When this pointer approaches 191 which is the size of the buffer, turn on the motor. The relevant locations of bytes for the second cassette port are 59456 and 223 for STOP and 2Ø7 for START.

DATA FILE ERRORS (cont.)

A problem with opening files to write on either built-in cassette #1, or external cassette #2, has been discovered. When a file is opened, garbage will be written out instead of a proper data tape file header. Without this header, it is impossible to open the tape file for reading.

You may not have encountered this problem previously, because it is disguised by having loaded a program on the cassette prior to writing a data file. In this mode, the start address of the buffer with the header information is initialized properly but cassette data file operation still could be random.

Fortunately, there is a software patch you can implement in your BASIC program to force the open for write on tape to work every time.

Before opening to write on #1 cassette:

```
POKE 243,122
POKE 244,2
```

and on #2 cassette:

```
POKE 243,58
POKE 244,3
```

Locations 243 and 244 contain the lo and hi order bytes respectively of the address of the currently active cassette buffer. The start address of buffer #2 is $33A which is 3,58 ($3=3,$3A=58) in double byte decimal. Similarly cassette #1 is $27A ($2=2,$7A=122).

TAPE HEAD CARE

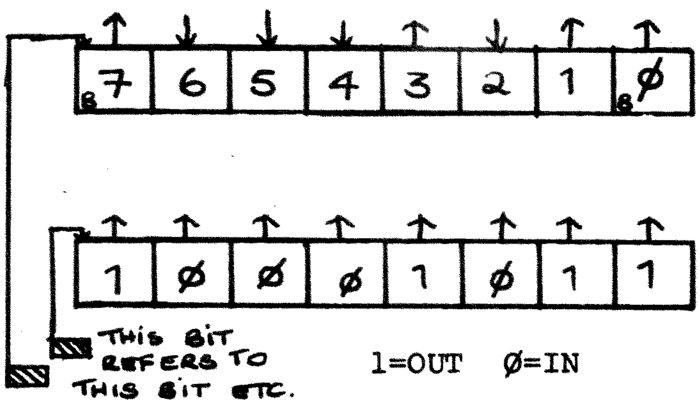
It has been noted that the READ/WRITE head in the PET cassette deck has the annoying habit of magnetising itself after a remarkably short period of operation. It is in fact possible to partially erase your tapes by up to 15% after only 15 or 2Ø passes over the head. The most convenient way to deal with this problem is to demagnestise the tape head very frequently, ie every couple of days with a demagnetising cassette. AMPEX market quite a good one for about £3.-

USEFUL ADDRESSES

On the following page you will find an extensive map of the PET memory. This list is "home" generated and not from CBM U.S. so may contain slight inaccuracies, but all the major buffers and ram areas are correct. Also here are some common PIA addresses and how to use them.

User Port - data register 59424

User Port Data Direction 59426



DATA REGISTER ADDRESS 59471

DATA DIRECTION REGISTER 59459

1=OUT Ø=IN

The major portion of the user port consists of 8 connections at the rear of the PET. Whether these connections are used for INPUT or OUTPUT is up to the programmer. These 8 wires may be used as either input or output. Before using this 8 bit port you must first configure these wires as inputs or outputs. This is done by writing a byte to the data direction register at address 59459. In the example above bits Ø, 1, 3 and 7 are configured as outputs. Bits 2,4,5 and 6 are configured as inputs. The bit that you see in the data direction register is generated by poke 59459, 139. In order to test a particular bit being used as an input in the data register

USEFUL ADDRESSES (CONT.)

(59471) one must peek 59471 and apply a "mask" in order to mask out unwanted bits. For instance to examine bit 2 we would use the expression PRINT PEEK (59471) AND 4. If the result of this expression is Ø then bit 2 of the data register (59471) has been held at Ø volts by the outside world.

PET MEMORY MAP

ØØØØ-ØØØ2	JUMP,USER ADDRESS
ØØØ5	CURSOR COLUMN
ØØØA-ØØ5A	BASIC INPUT BUFFER
ØØ5C	BASIC INPUT BUFFER POINTER
ØØ5E	CURRENT RESULT TYPE (FF) STRING (ØØ) NUMERIC
ØØ5F	" " " (80) INTEGER(ØØ)FLOATING POINT
ØØ7A-ØØ7B	START OF BASIC STATEMENTS
ØØ7C-OO7D	START OF VARIABLE TABLE
ØØ7E-ØØ7F	END OF VARIABLE TABLE
ØØ8Ø-ØØ81	START OF AVAILABLE SPACE
ØØ82-ØØ83	BOTTOM OF STRINGS (MOVING DOWN)
ØØ84-ØØ85	TOP OF STRINGS (MOVING DOWN)
ØØ86-ØØ87	TOP OF MEMORY ALLOCATED FOR BASIC WORKING AREA
ØØ88-ØØ89	CURRENT PROGRAM LINE NUMBER
ØØ8A-ØØ88	" " " " SAVED BY END
ØØ8C-ØØ8D	" " POINTER SAVED BY END
ØØ92-ØØ93	DATA STATEMENT POINTER
ØØ94-ØØ95	CURRENT VARIABLE SYMBOLS
ØØ96-ØØ97	CURRENT VARIABLE STARTING POINT
ØØAE-ØØAF	POINTER ASSOCIATED WITH BASIC BUFF TRANSFER
ØØ8Ø	EXPONENT + S8Ø)
ØØ81	MANTISSA MSB)
ØØ82	")--(FLOATING POINT ACCUMULATOR)
ØØ83	")
ØØ84	LSB)
ØØ85	SIGN OF MANTISSA (Ø IF ZERO)(+ IF POS.)(- IF NEG.)
ØØ88-ØØCØ	DYADIC HOLDING AREA
ØØØ2-	START OF ROUTINE FOR FETCHING NEXT BASIC CHARACTER
ØØC9-ØØCA	PROGRAM POINTER
-ØØD9	END OF CHARACTER FETCH
ØØEØ-	SCREEN POSITION ON LINE
ØØE1-ØØE2	POSITION OF LINE START
ØØE3-ØØE4	CURRENT TAPE BUFFER POINTER
ØØE5-ØØE6	END OF CURRENT PROGRAM
ØØEA	QUOTE MODE (ØØ IF NOT IN QUOTE)
ØØEE	NUMBER OF CHARACTERS IN FILE NAME

ØØEF	GPIB FILE#
ØØFØ	GPIB COMMAND
ØØF1	GPIB DEVICE#
ØØF3-ØØF4	START OF TAPE BUFFER
ØØF5	CURRENT SCREEN LINE#
ØØF6	RUNNING CHECKSUM OF BUFFER
ØØF7-ØØF8	POINTER TO PROGRAM DURING VERIFY,LOAD
ØØF9-ØØFA	FILENAME STARTING POINTER
ØØFC	SERIAL WORD
ØØFD	NUMBER OF BLOCKS REMIATNING TO WRITE
ØØFE	SERIAL WORD BUFFER
ØØFF-1FF	BASIC STACK ETC.
Ø2ØØ-Ø2Ø2	CLOCK H.M.S.
Ø2Ø3	MATRIX COORDINATE OF LAST KEY DOWN(255 IF NONE)
Ø2Ø4	SHIFT KEY STATUS (1 IF DOWN)
Ø2Ø5-Ø2Ø6	JIFFY CLOCK
Ø2Ø7	CASSETTE 1 ON SWITCH
Ø2Ø8	CASSETTE 2 ON SWITCH
Ø2Ø9	KEYSWITCH PIA
Ø2ØB	LOAD O, VERIFY 1
Ø2ØC	STATUS
Ø2ØD	NUMBER OF CHR IN KBD BUFFER
Ø2ØE-Ø216	KYBD INPUT BUFFER
Ø219-Ø21A	HARDWARE INTERRUPT VECTOR
Ø21B-Ø21C	BREAD INTERRUPT VECTOR
Ø223	KEY IMAGE
Ø225	CURSOR TIMING
Ø228	TAPE WRITE
Ø242-Ø24B	LOGICAL NUMBERS OF OPEN FILES
Ø24C-Ø255	DEVICE NUMBERS OF OPEN FILES
Ø256-Ø25F	R/W MODES OF OPEN FILES (COMMAND TABLE)
Ø262	GPIB TABLE LENGTH
Ø265	PARITY
Ø268	POINTER IN FILENAME TRANSFER
Ø26C	SERIAL BIT COUNT
Ø27Ø	TAPE WRITE COUNTDOWN
Ø273	LEADER COUNTER

Ø275	Ø IF FIRST HALF BYTE MARKER NOT WRITTEN
Ø276	Ø IF SECOND " " " " "
Ø279	CHECKSUM WORKING WORD
Ø27A-Ø339	BUFFER FOR CASSETTE #1
Ø33A-Ø3F9	" " " #2
Ø4ØØ	START OF BASIC STATEMENTS
-1FFF	END OF AVAILABLE RAM (8K VERSION)
-7FFF	END OF AVAILABLE RAM EXPANSION
8ØØØ-8FFF	VIDEO RAM
9ØØØ-BFFF	AVAILABLE ROM EXPANSION AREA
CØØØ-EØBØ	MICROSOFT "8K" BASIC
EØ85-E27D	SYSTEM SET UP
E294-E66A	VIDEO DRIVER
E66B-E684	INTERRUPT HANDLER
E685-E75B	CLOCK UPDATE,KYBD SCAN(6OHZ INT.)
E75C-E7D4	KYBD ENCODING TABLE
E8ØØ-EFFF	PIA'S
FØ86-F226	GPIB HANDLER
F346-F82C	FILE CONTROL
F82D-Fd15	TAPE CONTROL
FD38-FFB2	DIAGNOSTICS
FFCØ-FFED	JUMP VECTORS
FFFA-FFFF	65Ø2 INTERRUPT VECTORS (NMI NOT USED IN ORIG. VERSIONS)

MACHINE CODE ENVIRONMENT

If you wish to write machine code programs in your PET and do not wish to have BASIC trampling all over them here is a suggestion:

When the PET is first powered up a test pattern is written into and read back from the RAM in ascending address order. When this routine discovers a location which does not read back properly it presumes that it has run out of RAM and displays XXXX bytes free. At this point it makes a note of where it thinks the 'top of memory' is.. A quick glance at the memory map will show that BASIC program text is stored from location 1Ø25 upwards and strings are stored from the top of the memory downwards which means that in any normal circumstances there is nowhere in the PET main memory where you can hide your machine code routines.
If however, the first thing you do after powering up the PET is to alter the top of memory pointer to say 6ØØØ everything from 6ØØ1 upwards, as far as PET is concerned, does not exist. e.g. strings will be stored from 6ØØØ downwards etc. and machine code programs can be safely put in location 6ØØ1 upwards. This pointer is held in locations 134 and 135 constituting a 16 bit pointer with 134 being its lower 8 bits. This is a binary pointer which means that we must convert your 6ØØØ or whatever to binary before POKING locations 134 and 135 with the information. In the standard 8K PET 134 will be Ø and 135

will be 32 (32 x 256 = 8192) Remember that 1Ø25 bytes are used for house keeping by the PET (8192 - 1Ø25 = 7167) However to give the PET a ceiling of 6ØØØ we convert 6ØØØ into binary which gives us POKE 134, 112 and POKE 135, 23.

LIFE FOR YOUR PET

Here is a good example of what can be done in machine code in the PET. It is the game of "LIFE" by John H. Conway of Cambridge. If one attempts to write a Commodore PET screen size (1ØØØ cell) version of LIFE in BASIC it can take up to two or three minutes per generation. This program performs two generations per second. In order to use it type in a listing in the form of data statements and load in the machine code with a small BASIC routine being careful to fill in the gaps between 1928 (HEX) and 193Ø and also 1954 and 197Ø with no-ops. Below is a listing of the documentation provided by the author.

LIFE FOR YOUR PET

Since this is the first time I have attempted to set down a machine language program for the public eye, I will attempt to be as complete as practical without overdoing it.

The programs I will document here are concerned with the game of "LIFE", and are written in 6502 machine language specifically for the PET 2001 (8K version). The principles apply to any 6502 system with graphic display capability, and can be debugged (as I did) on non-graphic systems such as the KIM-1.

The first I heard of LIFE was in Martin Gardner's "Recreational Mathematics" section in Scientific American, Oct-Nov 1970; Feb. 1971. As I understand it, the game was invented by John H. Conway, an English mathematician. In brief, LIFE is a "cellular automation" scheme, where the arena is a rectangular grid (ideally of infinite size). Each square in the grid is either occupied or unoccupied with "seeds", the fate of which are governed by relatively simple rules, i.e. the "facts of LIFE". The rules are: 1. A seed survives to the next generation if and only if it has two or three neighbors (right, left, up, down, and the four diagonally adjacent cells) otherwise it dies of loneliness or overcrowding, as the case may be. 2. A seed is born in a vacant cell on the next generation if it has exactly 3 neighbors.

With these simple rules, a surprisingly rich game results. The original Scientific American article, and several subsequent articles reveal many curious and surprising initial patterns and results. I understand that there even has been formed a LIFE group, complete with newsletter, although I have not personally seen it.

The game can of course be played manually on a piece of graph paper, but it is slow and prone to mistakes, which have usually disasterous effects on the final results. It would seem to be the ideal thing to put to a microprocessor with bare-bones graphics, since the rules are so simple and there are essentially no arithmetic operations involved, except for keeping track of addresses and locating neighbors.

As you know, the PET-2001 has an excellent BASIC interpreter, but as yet very little documentation on machine language operation. My first stab was to write a BASIC program, using the entire PET display as the arena (more about boundaries later), and the filled circle graphic display character as the seed. This worked just fine, except for one thing - it took about 2-1/2 minutes for the interpreter to go through one generation! I suppose I shouldn't have been surprised since the program has to check eight neighboring cells to determine the fate of a particular cell, and do this 1000 times to complete the entire generation (40x25 characters for the PET display).

The program following is a 6502 version of LIFE written for the PET. It needs to be POKE'd into the PET memory, since I have yet to see or discover a machine language monitor for the PET. I did it with a simple BASIC program and many DATA statements (taking up much more of the program memory space than the actual machine language program!). A routine for assembling, and saving on tape machine language programs on the PET is sorely needed.

The program is accessed by the SYS command, and takes advantage of the display monitor (cursor control) for inserting seeds, and clearing the arena. Without a serious attempt at maximizing for speed, the program takes about 1/2 second to go through an entire generation, about 300 times faster than the BASIC equivalent! Enough said about the efficiency of machine language programming versus BASIC interpreters?

BASIC is great for number crunching, where you can quickly compose your program and have plenty of time to await the results.

The program may be broken down into manageable chunks by subroutining. There follows a brief description of the salient features of each section:

MAIN (hex 1900)

In a fit of overcaution (since this was the first time I attempted to write a PET machine language program) you will notice the series of pushes at the beginning and pulls at the end. I decided to save all the internal registers on the stack in page 1, and also included the CLD (clear decimal mode) just in case. Then follows a series of subroutine calls to do the LIFE generation and display transfers. The zero page location, TIMES, is a counter to permit several loops through LIFE before returning. As set up, TIMES is initialized to zero (hex location 1953) so that it will loop 256 times before jumping back. This of course can be changed either initially or while in BASIC via the POKE command. The return via the JMP BASIC (4C 8B C3) may not be strictly orthodox, but it seems to work all right.

INIT (hex 1930) and DATA (hex 193B)

This shorty reads in the constants needed, and stores them in page zero. SCR refers to the PET screen, TEMP is a temporary working area to hold the new generation as it is evolved, and RCS is essentially a copy of the PET screen data, which I found to be necessary to avoid "snow" on the screen during read/write operations directly on the screen locations. Up, down, etc. are the offsets to be added or subtracted from an address to get all the neighbor addresses. The observant reader will note the gap in the addresses between some of the routines.

TMPSCR (hex 1970)

This subroutine quickly transfers the contents of Temp and dumps it to the screen, using a dot (81 dec) symbol for a live cell (a 1 in TEMP) and a space (32 dec) for the absence of a live cell (a 0 in TEMP).

SCRTMP (hex 198A)

This is the inverse of TMPSCR, quickly transferring (and encoding) data from the screen into TEMP.

RSTORE (hex 19A6)

This subroutine fetches the initial addresses (high and low) for the SCR, TEMP, and RCS memory spaces.

NXTADR (hex 19BD)

Since we are dealing with 1000 bytes of data, we need a routine to increment to the next location, check for page crossing (adding 1 to the high address when it occurs), and checking for the end. The end is signaled by returning a 01 in the accumulator, otherwise a 00 is returned via the accumulator.

TMPRCS (hex 19E6)

The RCS address space is a copy of the screen, used as mentioned before to avoid constant "snow" on the screen if the screen were being continually accessed. This subroutine dumps data from TEMP, where the new generation has been computed, to RCS.

GENER (hex 1A00)

We finally arrive at a subroutine where LIFE is actually generated. After finding out the number of neighbors of the current RCS data byte from NBRS, GENER checks for births (CMPIM $03 at hex addr. 1A0E) if the cell was previously unoccupied. If a birth does not occur, there is an immediate branch to GENADR (the data byte remains 00). If the cell was occupied (CMPIM 81 dec at hex 1A08), OCC checks for survival (CMPIM $03 at hex 1A1A and CMPIM $02 at hex 1A1E), branching to GENADR when these two conditions are met, otherwise the cell dies (LDAIM $00 at hex 1A22). The results are stored in TEMP for the 1000 cells.

NBRS (hex 1A2F)

NBRS is the subroutine that really does most of the work and where most of the speed could be gained by more efficient programming. Its job, to find the total number of occupied neighbors of a given RCS data location, is complicated by page crossing and edge boundaries. In the present version, page crossing is taken care of, but edge boundaries (left, right, top, and bottom of the screen) are somewhat "strange". Above the top line and below the bottom line are considered as sort of forbidden regions where there should practically always be no "life" (data in those regions are not defined by the program, but I have found that there has never been a case where 81's have been present (all other data is considered as "unoccupied" characters). The right and left edges are different, however, and lead to a special type of "geometry". A cell at either edge is not considered as special by NBRS, and so to the right of a right-edge location is the next sequential address. On the screen this is really the left edge location, and one line lower. The inverse is true, of course for left addresses of left-edge locations. Topologically, this is equivalent to a "helix". No special effects of this are seen during a simple LIFE evolution since it just gives the impression of disappearing off one edge while appearing on the other edge. For an object like the "spaceship" (see Scientific American articles), then, the path eventually would cover the whole LIFE arena. The fun comes in when a configuration spreads out so much that it spills over both edges, and interacts with itself. This, of course cannot happen in an infinite universe, so that some of the more complex patterns will not have the same fate in the present version of LIFE. Most of the "blinkers", including the "glider gun" come out OK.

This 40x25 version of LIFE can undoubtedly be made more efficient, and other edge algorithms could be found, but I chose to leave it in its original form as a benchmark for my first successfully executed program in writing machine language on the PET. One confession, however - I used the KIM-1 to debug most of the subroutines. Almost all of them did not run on the first shot! Without a good understanding of PET memory allocation particularly in page zero, I was bound to crash many times over, with no recovery other than pulling the plug. The actual BASIC program consisted of a POKING loop with many DATA statements (always save on tape before running!).

A Brief Introduction to the Game of Life

One of the interesting properties of the game of LIFE is that such simple rules can lead to such complex activity. The simplicity comes from the fact that the rules apply to each individual cell. The complexity comes from the interactions between the individual cells. Each individual cell is affected by its eight adjacent neighbors, and nothing else.

The rules are:

1. A cell survives if it has two or three neighbors.

2. A cell dies from overcrowding if it has four or more neighbors. It dies from isolation if it has one or zero neighbors.

3. A cell is born when an empty space has exactly three neighbors.

With these few rules, many different types of activity can occur. Some patterns are STABLE, that is they do not change at all. Some are REPEATERS, patterns which undergo one or more changes and return to the original pattern. A REPEATER may repeat as fast as every other generation, or may have a longer period. A GLIDER is a pattern which moves as it repeats.

```
                                    REPEATERS
                              * *      *
       STABLE                                                 GLIDERS
                              * *        * *      * * *
                          * *        * *        * * *                        *
          *
                          * *            *                  *                  *
 * *    *   *   *   * *
                                                              *    *           *
 * *    *   *   * *   *
                                                           * * *     * * * * * *
          *
```

```
1900             LIFE   ORG    $1900

1900             BASIC  *      $C38B  RETURN TO BASIC ADDRESS
1900             OFFSET *      $002A  PAGE ZERO DATA AREA POINTER
1900             DOT    *      $0051  DOT SYMBOL = 81 DECIMAL
1900             BLANK  *      $0020  BLANK SYMBOL = 32 DECIMAL

1900             SCRL   *      $0020  PAGE ZERO LOCATIONS
1900             SCRH   *      $0021
1900             CHL    *      $0022
1900             CHH    *      $0023
1900             SCRLO  *      $0024
1900             SCRHO  *      $0025
1900             TEMPL  *      $0026
1900             TEMPH  *      $0027
1900             TEMPLO *      $0028
1900             TEMPHO *      $0029
1900             UP     *      $002A
1900             DOWN   *      $002B
1900             RIGHT  *      $002C
1900             LEFT   *      $002D
1900             UR     *      $002E
1900             UL     *      $002F
1900             LR     *      $0030
1900             LL     *      $0031
1900             N      *      $0032
1900             SCRLL  *      $0033
1900             SCRLH  *      $0034
1900             RCSLO  *      $0035
1900             RCSHO  *      $0036
1900             TMP    *      $0037
1900             TIMES  *      $0038
1900             RCSL   *      $0039
1900             RCSH   *      $003A

1900 08          MAIN   PHP           SAVE EVERYTHING
1901 48                 PHA           ON STACK
1902 8A                 TXA
1903 48                 PHA
1904 98                 TYA
1905 48                 PHA
1906 BA                 TSX
1907 8A                 TXA
1908 48                 PHA
1909 D8                 CLD           CLEAR DECIMAL MODE
190A 20 30 19           JSR    INIT
190D 20 8A 19           JSR    SCRTMP
1910 20 E6 19    GEN    JSR    TMPRCS
1913 20 00 1A           JSR    GENER
1916 20 70 19           JSR    TMPSCR
1919 E6 38              INCZ   TIMES  REPEAT 255 TIMES
191B D0 F3              BNE    GEN    BEFORE QUITTING
191D 68                 PLA           RESTORE EVERYTHING
191E AA                 TAX
191F 9A                 TXS
1920 68                 PLA
```

```
1921 A8             TAY
1922 68             PLA
1923 AA             TAX
1924 68             PLA
1925 28             PLP
1926 4C 8B C3       JMP   BASIC  RETURN TO BASIC

1930                ORG   $1930

              MOVE VALUES INTO PAGE ZERO

1930 A2 19    INIT  LDXIM $19    MOVE 25. VALUES
1932 BD 3A 19 LOAD  LDAX  DATA   -01
1935 95 1F          STAZX $1F    STORE IN PAGE ZERO
1937 CA             DEX
1938 D0 F8          BNE   LOAD
193A 60             RTS

193B 00       DATA  =     $00    SCRL
193C 80             =     $80    SCRH
193D 00             =     $00    CHL
193E 15             =     $15    CHH
193F 00             =     $00    SCRLO
1940 80             =     $80    SCRHO
1941 00             =     $00    TEMPL
1942 1B             =     $1B    TEMPH
1943 00             =     $00    TEMPLO
1944 1B             =     $1B    TEMPHO
1945 D7             =     $D7    UP
1946 28             =     $28    DOWN
1947 01             =     $01    RIGHT
1948 FE             =     $FE    LEFT
1949 D8             =     $D8    UR
194A D6             =     $D6    UL
194B 29             =     $29    LR
194C 27             =     $27    LL
194D 00             =     $00    N
194E E8             =     $E8    SCRLL
194F 83             =     $83    SCRLH
1950 00             =     $00    RCSLO
1951 15             =     $15    RCSHO
1952 00             =     $00    TMP
1953 00             =     $00    TIMES

1970                ORG   $1970

1970 20 A6 19 TMPSCR JSR  RSTORE GET INIT ADDRESSES
1973 B1 26    TSLOAD LDAIY TEMPL FETCH BYTE FROM TEMP
1975 D0 06          BNE   TSONE  BRANCH IF NOT ZERO
1977 A9 20          LDAIM BLANK  BLANK SYMBOL
1979 91 20          STAIY SCRL   DUMP IT TO SCREEN
197B D0 04          BNE   TSNEXT
197D A9 51    TSONE LDAIM DOT    DOT SYMBOL
197F 91 20          STAIY SCRL   DUMP IT TO SCREEN
1981 20 BD 19 TSNEXT JSR  NXTADR FETCH NEXT ADDRESS
1984 F0 ED          BEQ   TSLOAD
```

```
1986 20 A6 19        JSR   RSTORE RESTORE INIT ADDRESSES
1989 60              RTS

198A 20 A6 19 SCRTMP JSR   RSTORE GET INIT ADDRESSES
198D B1 20    STLOAD LDAIY SCRL   READ DATA FROM SCREEN
198F C9 51           CMPIM DOT    TEST FOR DOT
1991 F0 06           BEQ   STONE  BRANCH IF DOT
1993 A9 00           LDAIM $00    OTHERWISE ITS A BLANK
1995 91 26           STAIY TEMPL  STORE IT
1997 F0 04           BEQ   STNEXT UNCOND. BRANCH
1999 A9 01    STONE  LDAIM $01    A DOT WAS FOUND
199B 91 26           STAIY TEMPL  STORE IT
199D 20 BD 19 STNEXT JSR   NXTADR FETCH NEXT ADDRESS
19A0 F0 EB           BEQ   STLOAD
19A2 20 A6 19        JSR   RSTORE RESTORE INIT ADDRESSES
19A5 60              RTS

19A6 A9 00    RSTORE LDAIM $00    ZERO A, X, Y
19A8 AA              TAX
19A9 A8              TAY
19AA 85 20           STAZ  SCRL   INIT VALUES
19AC 85 26           STAZ  TEMPL
19AE 85 39           STAZ  RCSL
19B0 A5 25           LDAZ  SCRHO
19B2 85 21           STAZ  SCRH
19B4 A5 29           LDAZ  TEMPHO
19B6 85 27           STAZ  TEMPH
19B8 A5 36           LDAZ  RCSHO
19BA 85 3A           STAZ  RCSH
19BC 60              RTS

19BD E6 26    NXTADR INCZ  TEMPL  GET NEXT LOW ORDER
19BF E6 20           INCZ  SCRL   BYTE ADDRESS
19C1 E6 39           INCZ  RCSL
19C3 E8              INX
19C4 E4 33           CPXZ  SCRLL  IS IT THE LAST?
19C6 F0 0C           BEQ   PAGECH IS IT THE LAST PAGE?
19C8 E0 00           CPXIM $00    IS IT A PAGE BOUNDARY?
19CA D0 0E           BNE   NALOAD IF NOT, THEN NOT DONE
19CC E6 27           INCZ  TEMPH  OTHERWISE ADVANCE TO
19CE E6 21           INCZ  SCRH   NEXT PAGE
19D0 E6 3A           INCZ  RCSH
19D2 D0 06           BNE   NALOAD UNCONDITIONAL BRANCH
19D4 A5 34    PAGECH LDAZ  SCRLH  CHECK FOR LAST PAGE
19D6 C5 21           CMPZ  SCRH
19D8 F0 03           BEQ   NADONE IF YES, THEN DONE
19DA A9 00    NALOAD LDAIM $00    RETURN WITH A=0
19DC 60              RTS
19DD A9 01    NADONE LDAIM $01    RETURN WITH A=1
19DF 60              RTS

19E6                 ORG   $19E6

19E6 20 A6 19 TMPRCS JSR   RSTORE INIT ADDRESSES
19E9 B1 26    TRLOAD LDAIY TEMPL  FETCH DATA FROM TEMP
19EB D0 06           BNE   TRONE  IF NOT ZERO THEN ITS ALIVE
```

```
19ED A9 20          LDAIM BLANK  BLANK SYMBOL
19EF 91 39          STAIY RCSL   STORE IT IN SCREEN COPY
19F1 D0 04          BNE   NEWADR THEN ON TO A NEW ADDRESS
19F3 A9 51   TRONE  LDAIM DOT    THE DOT SYMBOL
19F5 91 39          STAIY RCSL   STORE IT IN SCREEN COPY
19F7 20 BD 19 NEWADR JSR  NXTADR FETCH NEXT ADDRESS
19FA F0 ED          BEQ   TRLOAD IF A=0, THEN NOT DONE
19FC 20 A6 19       JSR   RSTORE ELSE DONE. RESTORE
19FF 60             RTS

1A00 20 A6 19 GENER JSR   RSTORE INIT ADDRESSES
1A03 20 2F 1A AGAIN JSR   NBRS   FETCH NUMBER OF NEIGHBORS
1A06 B1 39          LDAIY RCSL   FETCH CURRENT DATA
1A08 C9 51          CMPIM DOT    IS IT A DOT?
1A0A F0 0C          BEQ   OCC    IF YES, THEN BRANCH
1A0C A5 32          LDAZ  N      OTHERWISE ITS BLANK
1A0E C9 03          CMPIM $03    SO WE CHECK FOR
1A10 D0 14          BNE   GENADR A BIRTH
1A12 A9 01   BIRTH  LDAIM $01    IT GIVES BIRTH
1A14 91 26          STAIY TEMPL  STORE IT IN TEMP
1A16 D0 0E          BNE   GENADR INCONDITIONAL BRANCH
1A18 A5 32   OCC    LDAZ  N      FETCH NUMBER OF NEIGHBORS
1A1A C9 03          CMPIM $03    IF IT HAS 3 OR 2
1A1C F0 08          BEQ   GENADR NEIGHBORS IT SURVIVES
1A1E C9 02          CMPIM $02
1A20 F0 04          BEQ   GENADR
1A22 A9 00   DEATH  LDAIM $00    IT DIED!
1A24 91 26          STAIY TEMPL  STORE IT IN TEMP
1A26 20 BD 19 GENADR JSR  NXTADR FETCH NEXT ADDRESS
1A29 F0 D8          BEQ   AGAIN  IF 0, THEN NOT DONE
1A2B 20 A6 19       JSR   RSTORE RESTORE INIT ADDRESSES
1A2E 60             RTS

1A2F 98      NBRS   TYA          SAVE Y AND X ON STACK
1A30 48             PHA
1A31 8A             TXA
1A32 48             PHA
1A33 A0 00          LDYIM $00    SET Y AND N = 0
1A35 84 32          STYZ  N
1A37 A2 08          LDXIM $08    CHECK 8 NEIGHBORS
1A39 B5 29   OFFS   LDAZX OFFSET -01
1A3B 10 15          BPL   ADD    ADD IF OFFSET IS POSITIVE
1A3D 49 FF          EORIM $FF    OTHERWISE GET SET TO
1A3F 85 37          STAZ  TMP    SUBTRACT
1A41 38             SEC          SET CARRY BIT FOR SUBTRACT
1A42 A5 39          LDAZ  RCSL
1A44 E5 37          SBCZ  TMP    SUBTRACT TO GET THE
1A46 85 22          STAZ  CHL    CORRECT NEIGHBOR ADDRESS
1A48 A5 3A          LDAZ  RCSH
1A4A 85 23          STAZ  CHH
1A4C B0 11          BCS   EXAM   OK, FIND OUT WHAT'S THERE
1A4E C6 23          DECZ  CHH    PAGE CROSS
1A50 D0 0D          BNE   EXAM   UNCOND. BRANCH
1A52 18      ADD    CLC          GET SET TO ADD
1A53 65 39          ADCZ  RCSL   ADD
1A55 85 22          STAZ  CHL    STORE THE LOW PART
```

```
1A57 A5 3A          LDAZ  RCSH   FETCH THE HIGH PART
1A59 85 23          STAZ  CHH
1A5B 90 02          BCC   EXAM   OK, WHAT'S THERE
1A5D E6 23          INCZ  CHH    PAGE CROSSING
1A5F B1 22    EXAM  LDAIY CHL    FETCH THE NEIGHBOR
1A61 C9 51          CMPIM DOT    DATA BYTE AND SEE IF ITS
1A63 D0 02          BNE   NEXT   OCCUPIED
1A65 E6 32          INCZ  N      ACCUMULATE NUMBER OF NEIGHBORS
1A67 CA       NEXT  DEX
1A68 D0 CF          BNE   OFFS   NOT DONE
1A6A 68             PLA          RESTORE X, Y FROM STACK
1A6B AA             TAX
1A6C 68             PLA
1A6D A8             TAY
1A6E 60             RTS
```

This program was prepared by:

Dr. F. H. Covitz,
Deer Hill Road,
Lebanon,
N.J. 08833,
USA.

Below we have a way of actually getting our HEX OP-CODES into the PET. Lines 1ØØ-2ØØ read the data statements convert them to decimal and POKE them sequentially into the memory. The first data item is expected to be the starting point of the loading in decimal and the last data item is expected to be an asterix. The beauty of this method is that you can use the screen edit facility on the PET for inserting and deleting codes. When you have inserted your own data statements from line 3ØØ upwards, save the entire performance prior to running as machine language routines rarely work first time around and the PET is quite likely to hang up and need turning off and on. The data statements in the example are for the game of LIFE. In the original version listed on the previous pages, 256 generations must occur before the control returns to BASIC. I have modified the program slightly in the beginning in order to allow the stop button to halt the binary routine. If you think you have loaded the following program correctly type RUN and press RETURN. This loads the binary program. When the machine prints READY, clear the screen. Type say eight shifted Qs in a row in the middle of the screen followed by SYS (64ØØ) (which is 19ØØH in decimal) and press return. GOOD LUCK!

```
100 READL
110 READ A$:C=LEN(A$):IFA$="*"THENEND
120 IFC<1ORC>2THEN200
130 A=ASC(A$)-48:B=ASC(RIGHT$(A$,1))-48
140 N=B+7*(B>9)-(C=2)*(16*(A+7*(A>9)))
150 IFN<0ORN>255THEN200
160 POKEL,N:L=L+1:GOTO110
200 PRINT"BYTE"L"=["A$"] ???":END
300 DATA6400
310 DATA 08,48,8A,48,98,48,BA,8A,48,D8,20,30,19,20,8A,19,20,E6,19,20,00,1A
320 DATA20,70,19,A9,FF,CD,12,E8,F0,F0,4C,8B,C3,AA,68,28,4C,8B,C3
330 DATA EA,EA,EA,EA,EA,EA,EA,A2,19,BD,3A,19,95,1F,CA,D0,F8,60,00,80,00,15,00
340 DATA80,00,1B,00,1B,D7,28,01,FE,D8,D6,29,27,00,E8,83,00,15,00,00
350 DATAEA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA
360 DATAEA,EA,EA,EA,EA,20,A6,19,B1,26,D0,06,A9,20,91,20,D0,04,A9,51,91,20,20
370 DATA BD,19,F0,ED,20,A6,19,60,20,A6,19,B1,20,C9,51,F0,06,A9,00,91,26,F0
380 DATA04,A9,01,91,26,20,BD,19,F0,EB,20,A6,19,60,A9,00,AA,A8,85,20,85,26,85
390 DATA39,A5,25,85,21,A5,29,85,27,A5,36,85,3A,60,E6,26,E6,20,E6,39,E8,E4
400 DATA33,F0,0C,E0,00,D0,0E,E6,27,E6,21,E6,3A,D0,06,A5,34,C5,21,F0,03,A9,00
410 DATA 60,A9,01,60,EA,EA,EA,EA,EA,EA,20,A6,19,B1,26,D0,06,A9,20,91,39,D0
420 DATA04,A9,51,91,39,20,BD,19,F0,ED,20,A6,19,60,20,A6,19,20,2F,1A,B1,39,C9
430 DATA51,F0,0C,A5,32,C9,03,D0,14,A9,01,91,26,D0,0E,A5,32,C9,03,F0,08,C9,02
440 DATAF0,04,A9,00,91,26,20,BD,19,F0,D8,20,A6,19,60,98,48,8A,48,A0,00,84,32
450 DATAA2,08,B5,29,10,15,49,FF,85,37,38,A5,39,E5,37,85,22,A5,3A,85,23,B0,11
460 DATAC6,23,D0,0D,18,65,39,85,22,A5,3A,85,23,90,02,E6,23,B1,22,C9,51,D0,02
470 DATAE6,32,CA,D0,CF,68,AA,68,A8,60,*
READY.
```

Mr. J. Smith of 38 Claremont Crescent, Croxley Green, Rickmansworth, Herts. WD3 3QR

wrote in: The error in the definition of arc cos X should, I feel, be corrected. A possible version is:- *

ACS X = ATN(SQR (1-X↑2)/X) + (1-SGN(X))*π/2

this correctly gives (<u>unless</u> <u>X=Ø</u>) arc cos (-Ø.5) as

Cont/...

$2\pi/3$ (12Ø°) ; your formula gives

arc cos (-Ø.5) as -6Ø° this would be incorrect in e.g. a "cosine rule" problem.

As you expect PET to be used in educational establishments for solving trig. problems, I think it important to put this right.

(*) Note that if X is negative

1-SGN(X) = 2

& if X is positive

1-SGN(X) = Ø

this ensures that a correct multiple of π is added to the arctangent. Also, would it not be better to suggest..

P = 18Ø/π (before FNS is used)

DEFFNS(V) = SIN (V/P) etc.

for the user defined functions?

HERE ARE SOME COMMENTS FROM MR. M.J. SMYTH who is the Senior Lecturer, Department of Astronomy, Royal Observatory, Edinburgh EH9 3HJ.

Using BASIC and the IEEE 488 bus, PET can input 4Ø numbers per second from a 3½ digit voltmeter (Hewlett Packard 3437A). Also using BASIC, the user port can generate an output trigger (e.g. to a measuring device) within about 1∅ ms of an input trigger. We have not yet tried using assembler. But the BASIC speeds make possible very interesting applications in equipment control and real-time data processing.