



presents

**The Best of
The Transactor
Volume 1**

COMMODORE BUSINESS MACHINES LIMITED
3370 PHARMACY AVENUE, AGINCOURT, ONTARIO M1W 2K4
TELEPHONE (416) 499-4292 - CABLE ADDRESS: COMTYPE
TELEX NUMBER 06-525400

Preface

The Best of The Transactor, Volume 1, is a collection of the finer articles published in the first 11 issues of The Transactor. These articles are printed in their entirety and follow the original format. The Best of The Transactor was not intended to introduce new material, but rather to unite material in co-existence.

I hope you will find this book a useful teaching tool and an interesting reference manual. The Transactor is put together, at least in part, with your input and I would like to say a special "thank you" to all who have contributed. If you have an interesting discovery, article or program listing that you would like to share, or if there is a subject you would like covered, I would be pleased to hear from you.

I have enjoyed my work with The Transactor over the past year and hope, with your continued support, that The Transactor Volume 2 will be even better.

Karl J. Hildon

Karl J. Hildon,
Editor.



BITS AND PIECES

ARE YOU READY?

There have been reported mysterious occurrences of the out of data error when editing and fiddling about in general.

This is not a bug, but is due to pressing RETURN whilst the cursor is over the READY prompt. The machine interprets this as READ Y and as there is usually no corresponding data statement around we get the error.

REDO

It must be remembered that when RETURN is pressed, the machine consumes everything on the same line as the cursor, so even if you have correct information at the beginning of a given line, a single character of an incorrect type far over on the right hand side of the screen on the same line is likely to cause problems. A rather problematical example of this situation occurs if you try and put up a graphic form or set of boxes on the screen and then under programme control ask for data with an input statement, e.g.

NUMBER ?

When the number is typed and RETURN is pressed, the graphics character making up the right-hand side of the box will be entered as part of the inputting data. In the case of input to a numeric variable, the graphics character is of course non-numeric and not allowed and will give the error ? Redo from start, so you must always leave such boxes open ended.

INVERSE TRIGNOMETRIC FUNCTIONS

Here are a couple of handy methods of obtaining sine and arc cosine (remember, the result will be in radians).

ASNX = ATN (X/SQR(1-X²))

ACSX = ATN (SQR(1-X²)/X)

For those of you who are used to working in degrees, here are some handy user defined functions:

DEFFNS(V) = SIN(V/(180/ π))

DEFFNC(V) = COS(V/(180/ π))

DEFFNT(V) = TAN(V/(180/ π))

These are three user defined functions which when called with arguments and degrees will give the appropriate results. In these examples V can be any variable but if all three are defined in the same programme, you must use three different dummy variables.

EXAMPLE: PRINT FNS(30)

Result of this will be .5. Notice that the argument for FNS, or FN anything for that matter, can be either a variable or numeric constant. Also, after a programme containing these definitions has been run, these functions may be called using FN in the direct mode, that is, from the keyboard directly without being in a programme.

INTERRUPT STRUCTURE

Interrupts (including Break or Software Interrupts) are handled by software polling.

When the processor recognizes an interrupt it vectors through FFFE, FFFF in ROM to a routine that first inspects the processor hardware (IRQ line low).

If it was caused by a Break instruction, a Jump Indirect is executed through locations 021B, C. If by a hardware interrupt then a Jump Indirect is taken through locations 0219, A.

These locations being in RAM may be user-modified to point to extra user code ahead of normal interrupt processing.

Note, however that the IRQ pointer is used by the cassette routines and should be restored to standard values before the cassette Save or Load functions are called.

Various sections of the I/O chips can be set up to cause interrupts through the IRQ line.

Example: POKE 59470,2 enables a negative edge on the user port CAL line to cause an interrupt.

However, have your code set up to handle it when it happens!

Also note that each pass through the regular interrupt code increments the time register.

EDITING

There is an interesting property of the screen edit routine which gives rise to the following effects:-

If you insert using the INS key, more spaces than you type in characters, the DEL key must be pressed twice the number of times there are spare spaces. E.g. If you insert six spaces in a middle of a line and only type in four new characters, the first two presses of the DEL key will produce inverse characters which will disappear on the next two presses. Remember, the INS key will move all characters including the one under the cursor to the right, whilst the DEL key will delete the character on its immediate left.

PET Matrix-Decoded Keyboard

See 515 & 516 in table below

| | | | | | | | | | | | |
|----|----|----|----|-----|----|-----|----|----|-----|-----|----|
| | 8 | 7 | 6 | 5 | 4 | (3) | 2 | 1 | | | |
| 64 | | " | # | \$ | % | ' | & | \ | () | + - | |
| 48 | Q | W | E | R | T | Y | U | I | O | P | ↑ |
| 32 | A | S | D | F | G | H | J | K | L | : | |
| 16 | Z | X | C | V | B | N | M | . | : | ? | re |
| 0 | sh | rv | @ | [] | sp | < | > | sh | sh | | |
| | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | | | |

Interesting Locations Accessible from BASIC

| Location (decimal) | Contents |
|--|---|
| $\begin{matrix} H & L \\ 225, & 224 \\ 226 \end{matrix}$ | Byte address of screen line with Cursor Character position of Cursor (0 to 79) |
| 515 | Matrix-coordinate (row+column) of last key down 255 if no key down |
| 516 | 1 if shift down, 0 if shift up |
| $\begin{matrix} 525 \\ 526-534 \end{matrix}$ | No. of characters in Keyboard Buffer Keyboard Buffer |
| $\begin{matrix} 578 \text{ to } 587 \\ 588 \text{ to } 597 \\ 598 \text{ to } 607 \\ 610 \end{matrix}$ | Logical numbers of open files Device numbers of open files Read/write modes of open files How many open files |
| $\begin{matrix} 512, 513, 514 \\ 518, 517 \\ 59465, 59464 \end{matrix}$ | Clock that increments 60 times a second Clock that increments 30 times a second? Clock that decrements every microsecond |
| 59456 | WAIT 59456,32,32 waits for vertical retrace of display |
| 64824 | SYS(64824) simulates power-on reset |
| 59469 | Interrupt Flag Register; e.g., to input user port CA1: I=PEEK(59469) AND 2: POKE 59469,I: IF I=0 THEN CA1 low |
| 59411 | IEEE PIA B Control, e.g., to run cassette#1 motor N jiffies: 100 POKE 59411,53: T=TI 200 IF TI-T<N GOTO 200 300 POKE 59411,61 ADVICE: Run motor at least 3 jiffies per 191 output chars |

TIMING TABLES

BASIC STATEMENTS AND I/O

CONSTRUCT

| | |
|-------------------------------------|--------------------|
| FRE | 1 to 10 |
| PEEK, POKE | 1 |
| TI\$ | 3 to 4 |
| TI | 1 |
| GET | 1 to infinity |
| POS | 1 |
| PRINT X or PRINT | 15 to 19 |
| PRINT X\$; | 14 + LEN (X\$) / 2 |
| READ X and DATA 3 | 9 |
| REM | 0.2 to 2 |
| RESTORE | 0.3 |
| TAB | 2 |
| SPC (N) | 1 + 0.6*N |
| FOR I = ... NEXT I | 4.0 + (1.6 each) |
| STEP | 1.3 |
| IF | 0.4 |
| GOTO or GOSUB | 1.1 |
| ON A GOTO or GOSUB | 0.5 + (0.3*A) |
| L ₁ L _M | + (0.2*M) |
| RETURN | 0.9 |

Using colon, :, saves 0.6 over new line.
 SAVE or LOAD
 15 sec + (2 sec per 100 char)
 i.e. 500 baud.

STRING FUNCTIONS

FUNCTION

| | |
|-----------------|----------------------|
| + | 0.5 + (0.2 per char) |
| ASC | 1 |
| CHR\$ | 1.2 |
| LEFT\$, RIGHT\$ | 3 + (0.025 per char) |
| LEN | 0 to 8 |
| MID\$ | 4 + (0.025 per char) |
| STR\$ | 7 to 10 |

STRING FUNCTIONS (Cont'd)

FUNCTION

VAL
 =, <, >, < =, >, > =

ARITHMETIC FUNCTIONS

FUNCTION

ABS
 ATN
 COS
 EXP
 INT
 LOG
 RND RND (-1)
 RND RND (0)
 RND RND (1)
 SGN
 SIN
 TAN
 user FN

ARITHMETIC OPERATORS

SYMBOL

0↑B, 1↑B
 2↑B
 else
 / O/B, A/1
 else
 * O*B, A*A
 else
 +
 -
 =, <, >, < =, >, > =
 AND, OR
 NOT

| | |
|--|-------------------------|
| | APPROX. TIME (MILLISEC) |
| | 1.3 |
| | 3 to 4 |

| | |
|--|-------------------------|
| | APPROX. TIME (MILLISEC) |
| | 0.6 |
| | 42 |
| | 27 |
| | 27 |
| | 1.2 |
| | 23 |
| | 1.0 |
| | 0.9 |
| | 4.1 |
| | 1.1 |
| | 25 |
| | 50 |
| | 2.4 |

| | |
|--|-------------------------|
| | APPROX. TIME (MILLISEC) |
| | 0.3 |
| | 32 |
| | 50 to 100 |
| | 0.5 |
| | 2 to 5 |
| | 0.4 |
| | 1.5 to 3 |
| | 0.3 to 1 |
| | 0.3 to 1 |
| | 0.7 |
| | 1.7 |
| | 1.4 |

VARIABLES AND CONSTANTS

ITEM

A, A\$, A =, A\$ =
 AA, AA\$, AA =, AA\$ =
 A%
 A%=
 999
 .999
 E16
 E-16
 "ABCDE"
 M (I, J, ...)

APPROX. TIME (MILLISEC)

0.7 to $(0.7 + \pi v * 0.1)$
 πv = no. of variables
 in program
 0.2 more than above
 0.3 more than A
 0.6 more than A =
 1 per digit
 0.7 + (4.2 per digit)
 0.2 + (0.4* exponent)
 0.2 + (3.0* exponent)
 (0.6 to 0.7) + (0.02
 per char)
 (1 to 1.5)*
 (no. of subscripts)

TIMING PROGRAM

```

100 N = 300
200 T1 = T1
300 FOR I = 1 TO N
400 REM PUT TEST CONSTRUCT HERE
500 NEXT I
600 T2 = T1
700 FOR I = 1 TO N
800 NEXT I
900 T3 = T1
1000 PRINT 1000* (2*T2-T1-T3) / (60*N)
1100 END
  
```

MEMORY USAGE (IN BYTES)

BASIC 1028 (I/O buffers, tables etc)
 each statement
 4 for line number and following space,
 regardless for the line number
 1 for each BASIC keyword
 1 for each other character, including
 RETURN
 each variable with a value assigned, regard
 less of spelling or value takes 7 bytes;
 for string variables, add the length of
 the string
 each array (N.B., size includes 0th element)
 take $f * (size + 1) + (2 \text{ per dimension})$ where
 $f=5$ for floating point arrays, $f=2$ for
 integer arrays, and $f=3$ for string arrays.
 The system slows down noticeably when memory is
 nearly full.

| | | | | | | | | | | | |
|----|----------|----|-------|-----|----|-----|--------|--|----|-----|-----|
| 0 | | | @ | 64 | 0 | 128 | | | s | 192 | 64 |
| 1 | | | A | 65 | 1 | 129 | | | ,a | 193 | 65 |
| 2 | | | B | 66 | 2 | 130 | | | ,b | 194 | 66 |
| 3 | | | C | 67 | 3 | 131 | | | ,c | 195 | 67 |
| 4 | | | D | 68 | 4 | 132 | | | ,d | 196 | 68 |
| 5 | | | E | 69 | 5 | 133 | | | ,e | 197 | 69 |
| 6 | | | F | 70 | 6 | 134 | | | ,f | 198 | 70 |
| 7 | | | G | 71 | 7 | 135 | | | ,g | 199 | 71 |
| 8 | | | H | 72 | 8 | 136 | | | ,h | 200 | 72 |
| 9 | | | I | 73 | 9 | 137 | | | ,i | 201 | 73 |
| 10 | | | J | 74 | 10 | 138 | | | ,j | 202 | 74 |
| 11 | | | K | 75 | 11 | 139 | | | ,k | 203 | 75 |
| 12 | | | L | 76 | 12 | 140 | | | ,l | 204 | 76 |
| 13 | RETURN | | M | 77 | 13 | 141 | | | ,m | 205 | 77 |
| 14 | | | N | 78 | 14 | 142 | | | ,n | 206 | 78 |
| 15 | | | O | 79 | 15 | 143 | | | ,o | 207 | 79 |
| 16 | | | P | 80 | 16 | 144 | | | ,p | 208 | 80 |
| 17 | | | Q | 81 | 17 | 145 | | | ,q | 209 | 81 |
| 18 | ↓ RVS | | R | 82 | 18 | 146 | ↑ | | ,r | 210 | 82 |
| 19 | HOME | | S | 83 | 19 | 147 | RVSoff | | ,s | 211 | 83 |
| 20 | DEL | | T | 84 | 20 | 148 | CLEAR | | ,t | 212 | 84 |
| 21 | | | U | 85 | 21 | 149 | INST | | ,u | 213 | 85 |
| 22 | | | V | 86 | 22 | 150 | | | ,v | 214 | 86 |
| 23 | | | W | 87 | 23 | 151 | | | ,w | 215 | 87 |
| 24 | | | X | 88 | 24 | 152 | | | ,x | 216 | 88 |
| 25 | | | Y | 89 | 25 | 153 | | | ,y | 217 | 89 |
| 26 | | | Z | 90 | 26 | 154 | | | ,z | 218 | 90 |
| 27 | | | [| 91 | 27 | 155 | | | | 219 | 91 |
| 28 | | | \ | 92 | 28 | 156 | | | | 220 | 92 |
| 29 | ⇒ | |] | 93 | 29 | 157 | ⇐ | | | 221 | 93 |
| 30 | | | ↑ | 94 | 30 | 158 | | | | 222 | 94 |
| 31 | | | → | 95 | 31 | 159 | | | | 223 | 95 |
| 32 | space | 32 | space | 96 | 32 | 160 | | | | 224 | 96 |
| 33 | ! | 33 | ! | 97 | 33 | 161 | 97 | | | 225 | 97 |
| 34 | " | 34 | " | 98 | 34 | 162 | 98 | | | 226 | 98 |
| 35 | # | 35 | # | 99 | 35 | 163 | 99 | | | 227 | 99 |
| 36 | \$ | 36 | \$ | 100 | 36 | 164 | 100 | | | 228 | 100 |
| 37 | % | 37 | % | 101 | 37 | 165 | 101 | | | 229 | 101 |
| 38 | & | 38 | & | 102 | 38 | 166 | 102 | | | 230 | 102 |
| 39 | ' | 39 | ' | 103 | 39 | 167 | 103 | | | 231 | 103 |
| 40 | (| 40 | (| 104 | 40 | 168 | 104 | | | 232 | 104 |
| 41 |) | 41 |) | 105 | 41 | 169 | 105 | | | 233 | 105 |
| 42 | * | 42 | * | 106 | 42 | 170 | 106 | | | 234 | 106 |
| 43 | + | 43 | + | 107 | 43 | 171 | 107 | | | 235 | 107 |
| 44 | , | 44 | , | 108 | 44 | 172 | 108 | | | 236 | 108 |
| 45 | - | 45 | - | 109 | 45 | 173 | 109 | | | 237 | 109 |
| 46 | . | 46 | . | 110 | 46 | 174 | 110 | | | 238 | 110 |
| 47 | / | 47 | / | 111 | 47 | 175 | 111 | | | 239 | 111 |
| 48 | 0 | 48 | 0 | 112 | 48 | 176 | 112 | | | 240 | 112 |
| 49 | 1 | 49 | 1 | 113 | 49 | 177 | 113 | | | 241 | 113 |
| 50 | 2 | 50 | 2 | 114 | 50 | 178 | 114 | | | 242 | 114 |
| 51 | 3 | 51 | 3 | 115 | 51 | 179 | 115 | | | 243 | 115 |
| 52 | 4 | 52 | 4 | 116 | 52 | 180 | 116 | | | 244 | 116 |
| 53 | 5 | 53 | 5 | 117 | 53 | 181 | 117 | | | 245 | 117 |
| 54 | 6 | 54 | 6 | 118 | 54 | 182 | 118 | | | 246 | 118 |
| 55 | 7 | 55 | 7 | 119 | 55 | 183 | 119 | | | 247 | 119 |
| 56 | 8 | 56 | 8 | 120 | 56 | 184 | 120 | | | 248 | 120 |
| 57 | 9 | 57 | 9 | 121 | 57 | 185 | 121 | | | 249 | 121 |
| 58 | : | 58 | : | 122 | 58 | 186 | 122 | | ☑ | 250 | 122 |
| 59 | ; | 59 | ; | 123 | 59 | 187 | 123 | | | 251 | 123 |
| 60 | < | 60 | < | 124 | 60 | 188 | 124 | | | 252 | 124 |
| 61 | = | 61 | = | 125 | 61 | 189 | 125 | | | 253 | 125 |
| 62 | > | 62 | > | 126 | 62 | 190 | 126 | | | 254 | 126 |
| 63 | ? | 63 | ? | 127 | 63 | 191 | 127 | | | 255 | 127 |

Hardware available:

Convenience Living Systems, 648 Sheraton Drive, Sunnyvale, CA 94087 EXPANDAPET for \$435 assembled with 16K RAM, sockets for 4K EPROM, 2 parallel I/O ports with handshake, slots for 3 option cards, and all cables and brackets.

Forethought Products, Box 8066, Coburg, OR 97401. The PETS I PET to S-100 interface/motherboard. \$105 kit or \$160 assembled. Includes 4 slots, dynamic memory controller, and sockets for 8K 2716 EPROM.

The Net Works, 5014 Narragansett #6, San Diego, CA 92107 has an IEEE to RS-232 board (with dual ports) for \$160 assembled and tested including on board power supplies. They also announced their TNW-488 Low Speed Modem Module to interface IEEE-488 (PET connector version) to the telephone network using the BELL 103A standard. Doug Gage, one of TNW's proprietors sent preliminary announcement specs and some documentation. He also said they had a prototype running for some time, and are now producing the first units at \$225 assembled.

The 8 bit user port is actually part of an MOS Technology MCS 6522 Versatile Interface Adapter (VIA). You can get a copy of the VIA data sheet from Commodore Business Machines, 3370 Pharmacy Avenue, Agincourt, Ont. (416)499-4292. Most of the VIA's features apparently are used for the PET itself, leaving only an 8 bit port and two handshake lines, which are really quite simple to use.

The new PET user manual briefly describes the 8 bit port edge connector, pins A and N are grounded, pin B is CA1, the input handshake line, pin M is CB2, the output handshake line, and pins C through L are the 8 data lines, with C being the high order (leftmost) and L the low order bit. When the PET is turned on, the 8 data bits are programmed to act as inputs and CA1 is programmed to recognize a negative transition (from 1 to 0).

To generate an audio signal from the PET a programmable square wave generator is included in the MCS6522 which interfaces the PET Parallel User Port. When the tape drive is not in operation, the generator can be used to produce one of 514 different frequencies between 243HZ and 125KHZ on CB2 (User Port pin M). The 6522 makes this possible by recirculating shift register intended for serial data input and output. With a square wave pattern loaded into the shift register and the control set for free running output under timer controlled rate, a continuous square wave is produced on CB2.

The BASIC statements needed to control the output are as follows:

POKE 59467,16 Sets shift register to free running output mode.
POKE 59464,C Sets shift rate. C is an integer of 0 to 255.
POKE 59466,D Loads shift register. D should be 15, 51, or 85 for a square wave output.

The frequency of the square wave can be determined from the following equation:

$$\text{FREQUENCY} = \frac{500000 \text{ HZ}}{(C+2)(D1)}$$

Where: D1 = 8 for D= 15
 D1 = 4 for D= 51
 D1 = 2 for D= 85

Reading or writing the shift register must be done last as this initiates the shifting operation. The control register at 59467 must be reloaded with 0 for the tape drive to write correctly.

Do not connect a speaker or earphones directly to the CB2 output of the PET. An amplifier is necessary to isolate the 6522 from inductive loads.

The TAPE #2 READ signal on User Port pin 8 (TAPE #2 READ and TAPE WRITE are reversed in the introductory manual) appears to be the CB1 line from the 6522 and carries the shift clock signal. With both CB1 and CB2 available, it may be possible to use the I/O port expansion scheme described in the MCS6522 Data Sheet.

PET CODES:

There are two ways to write to the screen of your PET: either POKEing screen memory (32768-33767) or by PRINTing. Besides this, there are two ways in which memory is interpreted by the character generator: standard mode (location 59468 = binary XXXX110X) or lower case mode (location ~~XXXX110X~~). Some of this confusion may be simplified with a character code chart. In the chart the OFF and RVS columns refer to values POKEed in screen memory whereas CHR\$ refers to the PRINT statement. Thus, either POKE 32768, 129 or PRINT CHR\$(18) CHR\$(65) will show a reverse A.

This simple program: 10 INPUT M,N:PRINT M;N,"AA"CHR\$(M) CHR\$(N)"BB":GOTO 10 allows you to explore the full CHR\$ set including cursor and reverse. The chart gives CHR\$ codes from (0-95) and (128-223). The missing values in the chart have the equivalents: (96-127) = (32-63), (224-254) = (160-190), and 255 = 222.

By referring to an ASCII code chart we see that ASCII (0-95) = PET(0-95) but that ASCII(96-127) = PET(192-223), ie. displaced by 96.

The printing mode (standard or lower case) is set by POKEing an address. So as not to disturb any of the other bits in the peripheral control register a safe way to set the lower case mode would be: POKE 59468, PEEK(59468) OR 14 and reset it to standard mode with POKE 59468, PEEK(59468) AND 253 OR 12.

Standard Mode: Location 59468 = XXXX110X

| | OFF | RVS | CHRS |
|--|-----|-----|------|
| | 64 | 192 | 192 |
| | 0 | 128 | 64 |
| | 65 | 193 | 193 |
| | 1 | 129 | 65 |
| | 66 | 194 | 194 |
| | 2 | 130 | 66 |
| | 67 | 195 | 195 |
| | 3 | 131 | 67 |
| | 68 | 196 | 196 |
| | 4 | 132 | 68 |
| | 69 | 197 | 197 |
| | 5 | 133 | 69 |
| | 70 | 198 | 198 |
| | 6 | 134 | 70 |
| | 71 | 199 | 199 |
| | 7 | 135 | 71 |
| | 72 | 200 | 200 |
| | 8 | 136 | 72 |
| | 73 | 201 | 201 |
| | 9 | 137 | 73 |
| | 74 | 202 | 202 |
| | 10 | 138 | 74 |
| | 75 | 203 | 203 |
| | 11 | 139 | 75 |
| | 76 | 204 | 204 |
| | 12 | 140 | 76 |
| | 77 | 205 | 205 |
| | 13 | 141 | 77 |
| | 78 | 206 | 206 |
| | 14 | 142 | 78 |
| | 79 | 207 | 207 |
| | 15 | 143 | 79 |

| | OFF | RVS | CHRS |
|--|-----|-----|------|
| | 80 | 208 | 208 |
| | 16 | 144 | 80 |
| | 81 | 209 | 209 |
| | 17 | 145 | 81 |
| | 82 | 210 | 210 |
| | 18 | 146 | 82 |
| | 83 | 211 | 211 |
| | 19 | 147 | 83 |
| | 84 | 212 | 212 |
| | 20 | 148 | 84 |
| | 85 | 213 | 213 |
| | 21 | 149 | 85 |
| | 86 | 214 | 214 |
| | 22 | 150 | 86 |
| | 87 | 215 | 215 |
| | 23 | 151 | 87 |
| | 88 | 216 | 216 |
| | 24 | 152 | 88 |
| | 89 | 217 | 217 |
| | 25 | 153 | 89 |
| | 90 | 218 | 218 |
| | 26 | 154 | 90 |
| | 91 | 219 | 219 |
| | 27 | 155 | 91 |
| | 92 | 220 | 220 |
| | 28 | 156 | 92 |
| | 93 | 221 | 221 |
| | 29 | 157 | 93 |
| | 94 | 222 | 222 |
| | 30 | 158 | 94 |
| | 95 | 223 | 223 |
| | 31 | 159 | 95 |

| | OFF | RVS | CHRS |
|--|-----|-----|------|
| | 96 | 224 | 160 |
| | 32 | 160 | 32 |
| | 97 | 225 | 161 |
| | 33 | 161 | 33 |
| | 98 | 226 | 162 |
| | 34 | 162 | 34 |
| | 99 | 227 | 163 |
| | 35 | 163 | 35 |
| | 100 | 228 | 164 |
| | 36 | 164 | 36 |
| | 101 | 229 | 165 |
| | 37 | 165 | 37 |
| | 102 | 230 | 166 |
| | 38 | 166 | 38 |
| | 103 | 231 | 167 |
| | 39 | 167 | 39 |
| | 104 | 232 | 168 |
| | 40 | 168 | 40 |
| | 105 | 233 | 169 |
| | 41 | 169 | 41 |
| | 106 | 234 | 170 |
| | 42 | 170 | 42 |
| | 107 | 235 | 171 |
| | 43 | 171 | 43 |
| | 108 | 236 | 172 |
| | 44 | 172 | 44 |
| | 109 | 237 | 173 |
| | 45 | 173 | 45 |
| | 110 | 238 | 174 |
| | 46 | 174 | 46 |
| | 111 | 239 | 175 |
| | 47 | 175 | 47 |

| | OFF | RVS | CHRS |
|--|-----|-----|------|
| | 112 | 240 | 176 |
| | 48 | 176 | 48 |
| | 113 | 241 | 177 |
| | 49 | 177 | 49 |
| | 114 | 242 | 178 |
| | 50 | 178 | 50 |
| | 115 | 243 | 179 |
| | 51 | 179 | 51 |
| | 116 | 244 | 180 |
| | 52 | 180 | 52 |
| | 117 | 245 | 181 |
| | 53 | 181 | 53 |
| | 118 | 246 | 182 |
| | 54 | 182 | 54 |
| | 119 | 247 | 183 |
| | 55 | 183 | 55 |
| | 120 | 248 | 184 |
| | 56 | 184 | 56 |
| | 121 | 249 | 185 |
| | 57 | 185 | 57 |
| | 122 | 250 | 186 |
| | 58 | 186 | 58 |
| | 123 | 251 | 187 |
| | 59 | 187 | 59 |
| | 124 | 252 | 188 |
| | 60 | 188 | 60 |
| | 125 | 253 | 189 |
| | 61 | 189 | 61 |
| | 126 | 254 | 190 |
| | 62 | 190 | 62 |
| | 127 | 255 | 191 |
| | 63 | 191 | 63 |

| | |
|--|-----|
| | 141 |
| | 13 |

| | |
|--|-----|
| | 147 |
| | 19 |
| | 146 |
| | 18 |

| | |
|--|-----|
| | 145 |
| | 17 |
| | 157 |
| | 29 |

| | |
|--|-----|
| | 148 |
| | 20 |
| | 131 |
| | 3 |

Lower Case Mode: Location 59468 ≠ XXXX110X, Same Except 193 to 218 Prints as Lower Case a to z Plus Different Graphics:

| | | | |
|--|-----|-----|-----|
| | 105 | 233 | 169 |
|--|-----|-----|-----|

| | | | |
|--|-----|-----|-----|
| | 122 | 250 | 186 |
|--|-----|-----|-----|

| | | | |
|--|----|-----|-----|
| | 94 | 222 | 222 |
|--|----|-----|-----|

| | | | |
|--|----|-----|-----|
| | 95 | 223 | 223 |
|--|----|-----|-----|

Number to Keyboard Conversion

| | | |
|----------------|--------------|-----------------|
| 0: End of line | 89: Y | 148: SAVE |
| 1-31: unused | 90: Z | 149: VERIFY |
| 32: space | 91: [| 150: DEF |
| 33: ! | 92: \ | 151: POKE |
| 34: " | 93:] | 152: PRINT# |
| 35: # | 94: ↑ | 153: PRINT |
| 36: \$ | 95: ← | 154: CONT |
| 37: % | 96: space | 155: LIST |
| 38: & | 97: ! | 156: CLR |
| 39: ' | 98: " | 157: CMD |
| 40: (| 99: # | 158: SYS |
| 41:) | 100: \$ | 159: OPEN |
| 42: * | 101: % | 160: CLOSE |
| 43: + | 102: & | 161: GET |
| 44: , | 103: ' | 162: NEW |
| 45: - | 104: (| 163: TAB(|
| 46: . | 105:) | 164: TO |
| 47: / | 106: * | 165: FN |
| 48: 0 | 107: + | 166: SPC(|
| 49: 1 | 108: , | 167: THEN |
| 50: 2 | 109: - | 168: NOT |
| 51: 3 | 110: . | 169: STEP |
| 52: 4 | 111: / | 170: + |
| 53: 5 | 112: 0 | 171: - |
| 54: 6 | 113: 1 | 172: * |
| 55: 7 | 114: 2 | 173: / |
| 56: 8 | 115: 3 | 174: ↑ |
| 57: 9 | 116: 4 | 175: AND |
| 58: : | 117: 5 | 176: OR |
| 59: ; | 118: 6 | 177: > |
| 60: < | 119: 7 | 178: = |
| 61: = | 120: 8 | 179: < |
| 62: > | 121: 9 | 180: SGN |
| 63: ? | 122: : | 181: INT |
| 64: @ | 123: ; | 182: ABS |
| 65: A | 124: < | 183: USR |
| 66: B | 125: = | 184: FRE |
| 67: C | 126: > | 185: POS |
| 68: D | 127: ? | 186: SQR |
| 69: E | 128: END | 187: RND |
| 70: F | 129: FOR | 188: LOG |
| 71: G | 130: NEXT | 189: EXP |
| 72: H | 131: DATA | 190: COS |
| 73: I | 132: INPUT# | 191: SIN |
| 74: J | 133: INPUT | 192: TAN |
| 75: K | 134: DIM | 193: ATN |
| 76: L | 135: READ | 194: PEEK |
| 77: M | 136: LET | 195: LEN |
| 78: N | 137: GOTO | 196: STR\$ |
| 79: O | 138: RUN | 197: VAL |
| 80: P | 139: IF | 198: ASC |
| 81: Q | 140: RESTORE | 199: CHR\$ |
| 82: R | 141: GOSUB | 200: LEFT\$ |
| 83: S | 142: RETURN | 201: RIGHT\$ |
| 84: T | 143: REM | 202: MID\$ |
| 85: U | 144: STOP | 203-254: unused |
| 86: V | 145: ON | 255: ⌘ |
| 87: W | 146: WAIT | |
| 88: X | 147: LOAD | |

USING THE PET COMPUTER

Note:- Words in a rectangle indicate a key to be pressed on the PET keyboard, capitals are individual letters to be typed.

To enter and run a program

1. Type NEW then **RETURN** . This will clear out any old programs in the useable memory.
2. Now type the program letter by letter on the keyboard, it will go into memory and also appear on the screen. At the end of each line you must press **RETURN** .
3. After you have typed the entire program correctly you are ready to run it. Type RUN then press **RETURN** and the program will start to operate.

To correct errors

1. The flashing cursor tells you where the next letter will go.
 - (a) to move the cursor to the right press **CURSOR** once for each space.
 - (b) to move the cursor to the left hold **SHIFT** down and press **CURSOR** once for each space.
 - (c) to move the cursor down press **CURSOR** once for each line.
 - (d) to move the cursor up hold **SHIFT** down and press **CURSOR** once for each line.
2. To change an instruction or part of one, move the cursor to the last correct part of the instruction and then type the correct part on top of the incorrect characters, to the end of that line, Be sure to put spaces on top of any letters left after the correction is made, press **RETURN** at the end.
3. To delete characters. Move the cursor to right of last character you wish to delete then press **INST DEL** . This removes the character it is over and moves everything after it left one space.
4. To insert characters. Move the cursor until it is located where the insertion is needed, then hold **SHIFT** down, press **INST BEL** followed by the character you wish to insert. You need to hold **SHIFT** down and press **INST DEL** for each letter you insert.

To load a program from tape

1. Place the cassette in the recorder but do not press any buttons on the tape recorder.
2. Type LOAD followed by the name of the program, if no name is typed it will load the next program on tape.
3. When the computer is ready it will ask you to press the play button on the tape recorder.
4. If the tape loads correctly the computer will then start to run the program. If there is just a ready indication you may have to type RUN. If there is an error in loading from tape, rewind the tape and start the loading instructions over.

Information on the screen

1. You may clear the screen and send the cursor to the upper left corner by holding down **SHIFT** and pressing **CLR HOME**.
2. If you wish to get a listing of the program in memory at any time type LIST then **RETURN**.
3. When the screen is full the first lines get lost at the top and new lines continue to be added at the bottom. If this happens during the listing of a program and you wish to examine some steps before they go off the top pressing **RUN/STOP** will stop the listing. Actually pressing **RUN/STOP** at any time stops the computer and if during the operation of your program it will tell you at what step you stopped it.

TEST FOR A PRIME NUMBER (BASIC)

```
10 PRINT "TYPE A WHOLE NUMBER"  
20 INPUT A  
30 IF A = 0 GO TO 999  
40 IF A = 1 GO TO 200  
50 IF A = 2 GO TO 220  
60 B = 2  
70 C = A/2  
80 D = A/B  
90 E = INT(D)  
95 F = B * E  
100 IF F = A GO TO 240  
110 IF B > C GO TO 260  
120 B = B + 1  
130 GO TO 80  
200 PRINT " 1 IS SPECIAL"  
210 GO TO 10  
220 PRINT " 2 IS A PRIME"  
230 GO TO 10  
240 PRINT A " IS NOT A PRIME"  
250 GO TO 10  
260 PRINT A " IS A PRIME"  
270 GO TO 10  
999 END
```

TO SORT UP TO 20 NUMBERS INTO ASCENDING ORDER

```
10 DIM A(20)  
20 PRINT "HOW MANY NUMBERS TO SORT"  
30 INPUT N  
40 IF N = 0 GO TO 230  
50 PRINT " GIVE ME" N " NUMBERS"  
60 FOR K = 1 TO N  
70 INPUT A(K)  
80 NEXT K  
90 J = N - 1
```

TO SORT UP TO 20 NUMBERS INTO ASCENDING ORDER ...continue

```
100 FOR M = 1 TO J
110 FOR I = 1 TO J
120 IF A(I) < A(I+1) GO TO 160
130 B = A(I+1)
140 A(I+1) = A(I)
150 A(I) = B
160 NEXT I
170 NEXT M
180 PRINT "THE NUMBERS IN ASCENDING ORDER"
190 FOR K = 1 TO N
200 PRINT A(K)
210 NEXT K
220 GO TO 20
230 PRINT "FINIS"
240 END
```

TABLE OF SQUARES, CUBES, ROOTS

```
10 PRINT " TYPE A STARTING NUMBER"
20 INPUT A
30 PRINT " TYPE THE ENDING NUMBER"
40 INPUT B
50 PRINT "   "
60 PRINT "N      SQUARE      CUBE      ROOT"
70 FOR N = A TO B
80 C = N2
90 C = INT(C)
100 D = N3
110 D = INT(D)
120 E = SQR(N)
130 PRINT N;
140 PRINT TAB(6);C;
150 PRINT TAB(16);D;
160 PRINT TAB(26);E
170 NEXT N
999 END
```

DIFFERENCES BETWEEN PET BASIC AND FORTRAN

In General

BASIC is an interpreter, interpreting and executing each statement as it comes to it.

FORTRAN is a compiler and makes two compiling passes before it attempts to execute.

Statements

All statements must be numbered in BASIC as it executes them in numerical order.

Only statements to which you transfer may be numbered and must be numbered, numerical order means nothing.

Constants and Variables

All constants and variables are real until you use the Integer function
A = INT(B) truncates the value of B and puts it in A.
Can get garbage in 9th digit

Has both integer and real arithmetic and variable names. Must be careful of mixed mode. Has double precision for extra accuracy.

Arithmetic

BASIC uses $A^{\uparrow}2$ for A^2

FORTRAN uses $A * 2$ for A^2

Decisions

BASIC IF is a logical if
e.g. IF A = B
IF A < B
IF A >= B etc

FORTRAN is a logical if
e.g. IF(A-B) 2,3,4

Looping

BASIC FOR N= 1 TO 20
...
...
NEXT N

FORTRAN DO 10 N= 1, 20
...
...
CONTINUE

Input

INPUT A,B stops calculating and waits for two numbers to be typed on keyboard and return button to be pushed
READ A,B takes information from a DATA statement in sequence

READ(8,2)A,B
2 FORMAT (2F8.2) for card read
Some versions of FORTRAN have simpler unformatted reads.

Output

PRINT A,B,C prints values
at 10,20,30,40

PRINT A;B;C prints 3 spaces
apart

PRINT "HELLO" prints characters
in quotes

BASIC also has TAB and signals
at the end of the statement
for more detailed spacing

WRITE(3,5)A,B,C
5 FORMAT(1X,3F15.2) format is
more complicated but allows
for great flexibility

Some FORTRAMS have an output
much like BASIC.

Subscripts

Dimension is DIM A(n)

Can have up to 4 subscripts
in PET BASIC and 256 limit to
their size.

Most FORTRAMS only allow triple
subscripts, Dimension statement
is DIMENSION

Subroutines

GOSUB L in BASIC

Must know the statement number
of the routine, and be sure
you use the same names of
variables to communicate with
a subroutine and subroutine
may destroy your variables if
you use the same names

CALL NAME(a,b,c...) in FORTRAN
Linkage is through the calling
sequence and the subroutine is
compiled at a different time, hence
the actual variable name is not
significant. Better for large
programs.

Comments

REM followed by printing is
not executed by the program,
only used to help the reader
and programmer tell what is
taking place in the program

C. COMMENT followed by printing is
not executed by the program, only
to give notes to those who read
the program.

```

1 OPEN5,4:CMD5
2 PRINTCHR$(26):PRINTCHR$(7):PRINTCHR$(7)
10 DIMA$(75),B$(75)
20 FORI=0TO74
30 READA$(I),B$(I)
40 NEXT
50 PRINT"THIS PROGRAM WILL TEST YOUR KNOWLEDGE OFWORLD CAPITAL CITIES."
60 PRINT:PRINT:PRINT"AFTER EACH STATE (SELECTED AT RANDOM)"
65 PRINT"PLEASE TYPE IN THE APPROPRIATE CAPITAL FOLLOWED BY A 'RETURN'."
80 PRINT:PRINT
90 N=0:C=0:W=0
100 N=N+1:IFN>10THEN400
105 I=74*RND( 6)
106 I=INT(I)
108 Q=0
110 PRINT:PRINT:PRINT"WHAT IS THE CAPITAL OF " ;A$(I);
115 PRINT:INPUTZ$
120 IFZ$=B$(I)THEN200
130 PRINT"NOT CORRECT!...TRY AGAIN":W=W+1
140 Q=Q+1:IFQ>=2THEN800
150 GOTO110
200 R=4*RND( 1) :C=C+1
201 R=INT(R)
210 IFR=1THEN300
220 IFR=2THEN301
230 IFR=3THEN302
240 IFR=4THEN303
250 PRINT"CORRECT..YOU'RE A GENIUS!":GOTO100
300 PRINT"RIGHT ON, BABY!":GOTO100
301 PRINT"ALL RIGHT!":GOTO100
302 PRINT"YES SIR!":GOTO100
303 PRINT"YOU'RE TOO MUCH!":GOTO100
400 PRINT"YOUR SCORE IS";C;"CORRECT,";W;"WRONG"
401 PRINT:PRINT"RATING";C/(C+W)*100;"%"
402 PRINT:PRINT:PRINT"DO YOU WISH TO CONTINUE THE LESSON (TYPE YES OR NO)?
"
403 INPUTC$
404 IFC$="YES"THEN90
405 IFC$="NO"THENS30
406 GOTO402
600 DATAAFGHANISTAN,KABUL,ANGOLA,LUANDA,ALGERIA,ALGIERS,ARGENTINA,BUENOS AIRES
610 DATAAUSTRALIA,CANBERRA,AUSTRIA,VIENNA,BELGIUM,BRUSSELS,BOLIVIA,SUCRE
620 DATABRAZIL,BRASILIA,BULGARIA,SOFIA,BURMA,RANGOON,CAMBODIA,PHNOM PENH
630 DATACANADA,OTTAWA,CHILE,SANTIAGO,COLUMBIA,BOGOTA,COSTA RICA,SAN JOSE
640 DATACUBA,HAVANA,CYPRUS,NICOSIA,CZECHOSLOVAKIA,PRAGUE,DENMARK,COPENHAGEN
650 DATADOMINICAN REPUBLIC,SANTO DOMINGO,ECUADOR,QUITO,EGYPT,CAIRO
660 DATAEL SALVADOR,SAN SALVADOR,ETHIOPIA,ADDIS ABABA,FINLAND,HELSINKI
670 DATAFRANCE,PARIS,W.GERMANY,BONN,E.GERMANY,BERLIN,GREECE,ATHENS
680 DATAGUATEMALA,GUATEMALA CITY,GUYANA,GEORGEROWN,HAITI,PORT-AU-PRINCE
690 DATAHONDURAS,TEGUCIGALPA,HUNGARY,BUDAPEST,ICELAND,REYKJAVIK
700 DATAINDIA,NEW DELHI,IRAN,TEHRAN,IRAQ,BAGHDAD,IRELAND,DUBLIN
710 DATAISRAEL,JERUSALEM,ITALY,ROME,JAMAICA,KINGSTON,JAPAN,TOKYO,JORDAN,AMMAN
720 DATAKENYA,NAIROBI,LEBANON,BEIRUT,LIECHTENSTEIN,VADUZ,LUXEMBOURG,LUXEMBOURG
730 DATAMALTA,VALLETTA,MEXICO,MEXICO CITY,MOROCCO,RABAT,NETHERLAND,AMSTERDAM
740 DATANEW ZEALAND,WELLINGTON,NICARAGUA,MANAGUA,NIGERIA,LAGOS,NORWAY,OSLO
750 DATAPARAGUAY,ASUNCION,PERU,LIMA,POLAND,WARSAW,PORTUGAL,LISBON,SPAIN,MADRID
760 DATASUDAN,KHARTOUM,SWEDEN,STOCKHOLM,SWITZERLAND,BERN,SYRIA,DAMASCUS
770 DATATHAILAND,BANGKOK,TURKEY,ANKARA,UGANDA,KAMPALA,USSR,MOSCOW,USA,WASHINGTO
N
780 DATAU.K.,LONDON,URUGUAY,MONTEVIDEO,VENEZUELA,CARACAS,YUGOSLAVIA,BELGRADE
800 PRINT"WELL,I GUESS YOU REALLY DON'T KNOW IT!! (SHAME)"
810 PRINT:PRINT:PRINT"THE CORRECT ANSWER IS ";B$(I);
815 PRINT:PRINT"NOW I WILL ASK YOU AGAIN!"
820 GOTO100
830 END
READY.

```

1) ROM112.MIC;1

| | | | | | |
|-----|------|--------|-----|-----|------|
| 799 | C589 | 8DFD01 | M=A | STA | 01FD |
| 800 | C58C | 6B | (| PLA | |
| 801 | C58D | 8DFE01 | M>A | STA | 01FE |
| 802 | C590 | A2FC | *< | LIX | #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 | AB | (| TAY | |
| 800 | C58A | 6B | (| PLA | |
| 801 | C58B | A2FE | *> | LIX | #FE |
| 802 | C58D | 9A | Z | TXS | |
| 803 | C58E | 4B | h | PHA | |
| 804 | C58F | 9B | X | TYA | |
| 805 | C590 | 4B | 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 | a | EDR | (60),Y |
| 811 | C59A | 18 | X | CLC | |

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

```

1 REM*****SQUIGGLE VERSION 2.0*****
2 REM IDEA BY PAUL HITTLE
3 REM PROGRAMMING BY B.SEILER, G.YOB
4 REM CLEAR SCREEN TO START
5 PRINT"Q";
9 REM SQUIGGLE GRAPHICS CHARACTERS
10 DATA"1","-","J","L","P","_";
18 REM CHARACTERS FOR EACH DIRECTION
19 REM 20-UP,30-DOWN,40-RIGHT,50-LEFT
20 DATA1,0,5,6
30 DATA0,1,4,3
40 DATA3,6,2,0
50 DATA4,5,0,2
55 REM A$ HOLDS CHARS, B HOLDS PTRS
56 REM FOR EACH DIRECTION
60 DIMA$(5),B(5,5)
65 REM SET UP A$ AND B()
70 FORI=0TO5
80 READA$(I)
90 NEXT I
100 FORI=1TO4
110 FORJ=1TO4
120 READB(J,I)
130 NEXT J
140 NEXT I
150 REM INITIAL VALUES
160 REM T1,T2 = DIRECTION OF TRAVEL
161 REM T1 IS CURRENT DIRECTION
162 REM T2 IS PREVIOUS DIRECTION
163 REM 1=UP, 2=DOWN, 3=RIGHT, 4=LEFT
170 REM
180 REM X,Y ARE POSITION OF WRIGGLER
181 REM ON SCREEN, 0,0 IS UPPER LEFT
182 REM CORNER (CURSOR HOME)
183 REM 20,12 IS CENTER OF SCREEN
190 T1=1
200 T2=1
210 X=20
220 Y=12

```

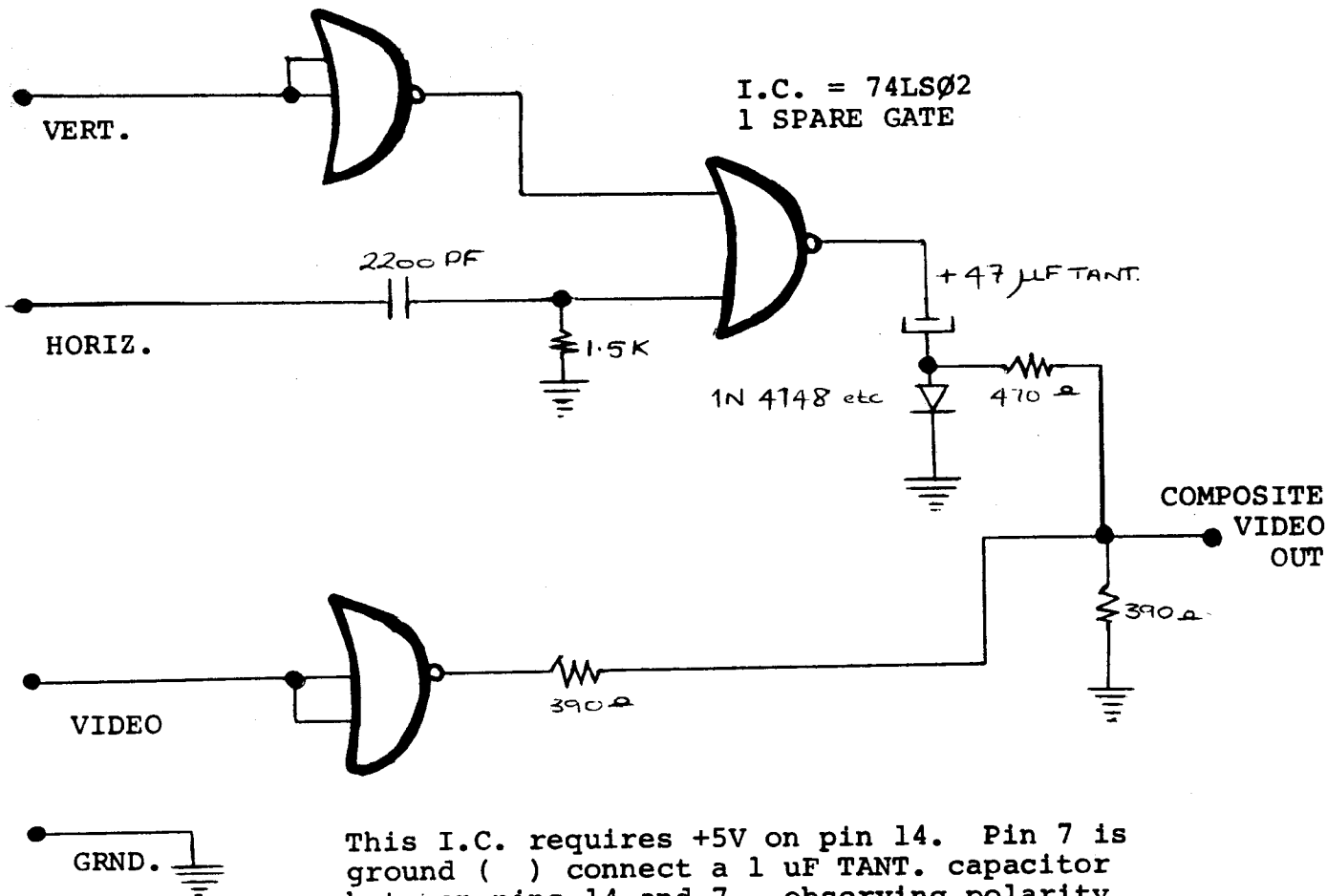


```

250 REM GET RANDOM FACTOR
260 PRINT "WIGGLE FACTOR(0-9)";
265 PRINT "███" : FOR J=1 TO 130 : NEXT J
266 PRINT "██" : FOR J=1 TO 100 : NEXT J
270 GET W$: IF W$="" THEN 265
275 IF W$ < "0" OR W$ > "9" THEN 260
280 W=VAL(W$)/10+.1
290 PRINT "C";
300 REM ***MAIN LOOP***
301 REM ** TURN OR NOT??
305 IF RND(1) > W*W THEN 325
306 REM YES, DO TURN
310 T1=4*RND(1)+1
320 IF B(T1,T2)=0 THEN 310
324 REM DRAW MOVE ON SCREEN
325 GOSUB 2000
330 T2=T1
339 REM UPDATE POSITION
340 ON T1 GO TO 400,410,420,430
400 Y=Y-1 : GOT 0500
410 Y=Y+1 : GOT 0500
420 X=X+1 : GOT 0550
430 X=X-1 : GOT 0550
490 REM ADJUST FOR WRAP-AROUND
500 IF Y < 1 THEN Y=23 : GOT 0300
510 IF Y > 23 THEN Y=1 : GOT 0300
550 IF X < 1 THEN X=39 : GOT 0300
560 IF X > 39 THEN X=1 : GOT 0300
570 GOT 0300
1990 REM ***DRAWING SUBROUTINE***
1991 REM POSITION CURSOR AT X,Y
2000 PRINT "█";
2010 FOR I=1 TO Y
2020 PRINT "█";
2030 NEXT I
2040 FOR I=1 TO X
2050 PRINT "█";
2055 NEXT I
2060 REM PRINT THE CHARACTER
2080 PRINT A$(B(T1,T2)-1);
2090 RETURN
READY.

```

ATTACHING A VIDEO MONITOR TO PET



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.

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.

```
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.
```

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 IF TI-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  
10 PRINT"  
20 PI=3.14159265  
30 FORA=0TO4*PI STEP(4*PI)/39  
40 Y=INT(SIN(A)*12+12):X=X+1  
50 GOSUB80  
60 NEXT  
70 FORA=33568TO33574:READZ:POKEA,Z:NEXT  
75 GOTO75  
80 POKE((24-Y)*40+32768)+X,46:RETURN  
READY.
```

The line that does the actual XY co-ordinate conversion is line 80. For the sake of clarity line 80 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 70 should be included when you test this program out but may be omitted subsequently. X has a range of 0-39 and Y has a range of 0-24.

DATA FILE ERRORS

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:

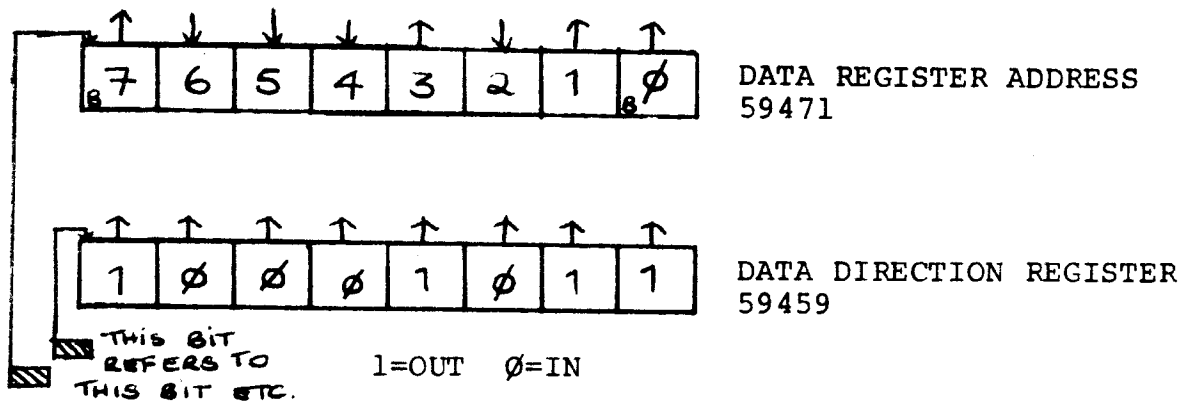
```
POKE 243,122
POKE 244,2
```

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).

More User Port Info



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 0, 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 (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 0 then bit 2 of the data register (59471) has been held at 0 volts by the outside world.

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 1025 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 6000 everything from 6001 upwards, as far as PET is concerned, does not exist. e.g. strings will be stored from 6000 downwards etc. and machine code programs can be safely put in location 6001 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 6000 or whatever to binary before POKING locations 134 and 135 with the information. In the standard 8K PET 134 will be 0 and 135

MACHINE CODE ENVIRONMENT (cont.)

will be 32 (32 x 256 = 8192) Remember that 1025 bytes are used for house keeping by the PET (8192 - 1025 = 7167) However to give the PET a ceiling of 6000 we convert 6000 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 (1000 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 1930 and also 1954 and 1970 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 disastrous 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 es-

entially 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 subroutines. 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).

SCRIMP (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,

| 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 | | SCR LH | * | \$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 | SCR TMP | |
| 1910 | 20 E6 19 | GEN | JSR | TMP RCS | |
| 1913 | 20 00 1A | | JSR | GENER | |
| 1916 | 20 70 19 | | JSR | TMP SCR | |
| 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 | SCR LH |
| 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  SCRLLH 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.

```

100 READL
110 READ A$:C=LEN(A$):IFA$=""*THENEND
120 IFC<10RC>2THEN200
130 A=ASC(A$)-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
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,EB,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 DATAA2,08,85,29,10,15,49,FF,85,37,38,A5,39,E5,37,85,22,A5,3A,85,23,B0,11
460 DATA C6,23,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: - (*)

$$\text{ACS } X = \text{ATN}(\text{SQR}(1-X^2)/X) + (1-\text{SGN}(X)) * \pi / 2$$

this correctly gives (unless $X=0$) arc cos ($-\phi.5$) as

Contf...

YOUR LETTERS (cont.)

$2\pi/3$ (120°) ; your formula gives

$\arccos(-0.5)$ as -60° 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

$$1-\text{SGN}(X) = 2$$

& if X is positive

$$1-\text{SGN}(X) = 0$$

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

$$P = 180/\pi \quad (\text{before FNS is used})$$

$$\text{DEFNS}(V) = \text{SIN}(V/P) \quad \text{etc.}$$

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 40 numbers per second from a $3\frac{1}{2}$ digit voltmeter (Hewlett Packard 3437A). Also using BASIC, the user port can generate an output trigger (e.g. to a measuring device)

YOUR LETTERS (cont.)

within about 10 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.

To input data in BASIC without returning to BASIC command mode on receipt of a null string then an input statement can be simulated by a GET loop which contains additional statements to cope with DEL codes. This has the additional advantage that if there is a displayed frame on screen the frame characters will not be accepted as part of the input.

The attached listing shows the above routine. This starts at line 9000 and to use it, instead of INPUT A\$ you put GOSUB 9000:A\$=IN\$.

```
8000 REM SUBROUTINE TO SIMULATE NON-PET          STANDARD INPUT STATEMENT
8010 REM STANDARD INPUT DOES NOT BREAK          ON RECEIPT OF A NULL STRING
8020 REM "TINPUT" COULD BE ALSO BE             SIMULATED EASILY
8030 REM ZA$ IS DEFINED IN LINE 10
9000 IN$="":PRINT "? ";
9010 GOSUB9070:PRINTZA$(ZB):IFZ$=""THEN9010
9020 IFZ$=CHR$(13)THENPRINT" ":RETURN
9030 IFZ$=CHR$(20)THENONSGN(LEN(IN$))+1GOTO9010,9060
9040 PRINTZ$:IN$=IN$+Z$:GOTO9010
9050 PRINTZ$:IN$=MID$(IN$,1,LEN(IN$)-1):GOTO9010
9070 ZB=1+(ZB=1):FORZA=1TO60:GETZ$:IFZ$<>" "THENRETURN
9080 NEXT:RETURN
```

Snowless Version of Life:

The "Life" program listed in Transactor #5 can be further refined to eliminate the snowy effect on the screen. Mark Taylor, in the U.K., has written to us with a snowless version of Life. He also included another interesting program illustrated in example 2:

1. If you are plagued by snow on the screen during POKE operations to the screen RAM or with machine code programmes, then if in BASIC location 59409 is POKEd with 52 then that will inhibit the character generator. Any transfer operations can then be carried out with the screen totally blank. To restore normal operation the above location should be POKEd with 60. For machine code programmes then LDA 52 & STA ABSOLUTE the above location will be somewhat faster.

The attached listing is a snowless version of "Life".

```
100 READL
110 READA$:C=LEN(A$):IFA$="*"THENEND
120 IF C<10RC)2THEN200
130 A=ASC(A$)-48:B=ASC(RIGHT$(A$,1))-48
140 N=B+7*(B)9)-(C=2)*(16*(A+7*(A)9))
150 IF N<0OR N>255THEN200
160 POKE L,N:L=L+1:GOTO110
200 PRINT"BYTE"L"="A$"] ???":END
300 DATA6400
310 DATA20,30,19,20,8A,19,20,E6,19,20,00,1A,A9,34,8D,11,E8
320 DATA20,70,19,A9,3C,8D,11,E8,A9,FF,CD,12,E8,F0,E6,4C,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
360 DATAEA,EA,EA,EA,EA,20,A6,19,B1,26,D8,06,A9,20,91,20,D8,04,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,D8,06,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,D8,06,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,D8,14,A9,01,91,26,D8,0E,A5,32,C9,03,F0,03,C9,02
440 DATAF0,04,A9,00,91,26,20,BD,19,F0,D8,20,A6,19,60,98,48,8A,48,A8,00,84,32
450 DATAA2,00,B5,29,10,15,49,FF,35,37,38,A5,39,E5,37,85,22,A5,3A,85,23,B0,11
460 DATA C6,23,D0,0D,13,65,39,85,22,A5,3A,85,23,90,02,E6,23,B1,22,C9,51,D0,02
470 DATAE6,32,CA,D0,CF,68,AA,63,A8,60,*
```

Programme Overlays on a PET - Supplied by Mike Stone

1. The 8K core of a PET is not usually a limitation in the home computer and hobbyist world, nor even in an educational environment where students are just creating small exercise programs. With the devices now available for attachment - the second cassette, (the printer, and floppy discs shortly) - the PET becomes a valid and genuine data processing machine, and complex string handling programs with files may well run out of space.
2. Programmers with experience on other computers know that one answer to this kind of problem is to break the program down into segments, so that only part of it is occupying memory at any time, and all or part is "overlaid" by other segments as required. When the program segments are on a disc, direct access features normally permit great flexibility, in that any required segment can be loaded; for tape only systems, the segments have to be arranged in order of need - e.g. job initialisation, main coding, and termination segments.
3. Since PET's BASIC includes a LOAD instruction to acquire dynamically a new program from tape, and (provided the new program is no longer than the one issuing the LOAD) all data areas remain available to the loaded program, the basis exists for an overlay system. However, for true overlaying, it is essential that some of the original program (e.g. control of the program flow, common subroutines, etc.) be retained throughout, whatever new segments are loaded. PET does not do this automatically; this paper tells you how to do it.

4. PET stores BASIC programs in location 1024 upwards.
Note that the pointers, and line numbers, are pairs of bytes giving low/high. The high must be multiplied by 256 and added to the low to give the actual quantity.
5. Whenever a line of code is entered from the keyboard, PET moves every statement around as necessary and re-adjusts all the chaining, so that statements are always stored in strict sequence of line number.
6. When your program contains a LOAD statement, this does NOT imply either CLR or NEW. The new program is simply loaded in at (and then executed from) location 1024, for as much space as it needs. The new program does NOT (as with some BASIC's) just replace those statements with identical line numbers; it is strictly a new program in its own right. However, any program statements at the end of the LOADING program whose space is not required by the LOADED program do remain unscathed by the LOAD. The problem is that the new program has no (forward-) chain into them, so PET knows nothing about them.
7. It follows from the above that if we code the instructions-to-be-preserved with high line numbers; and if the space needed by the newly-loaded segment does not over-write them; and if we can force the new segment to chain into the old instructions; then we have a real overlay system. So, if during the original program you can find in memory the last statement not to be preserved, you know it forward chains into the next highest line number, i.e. the first of the statements you do want preserved. Then when the overlay arrives, you need only find its very last statement and replace its forward-chain by the one you previously found, and both new and old code form a contiguous program.

8. A very simple illustration follows

Enter this program (do NOT put any spaces, except after line numbers):

```
10 A=A+1
20 GOSUB50
30 LOAD"NEWPROG"
50 PRINTA*2
55 RETURN
```

This is stored as follows ("PEEK" values):

```
1024) 0
      5) 11      } forward chain; 4 x 256 = 1024 + 11 = 1035
      6) 4      }
      7) 10     } line number 10
      8) 0      }
      9) 65     } A
1030) 178     } =
      1) 65     } A
      2) 170    } +
      3) 49     } 1
```



```

4)      0
→ 5)    19      } forward chain; 4 x 256 = 1024 + 19 = 1043
6)      4
7)     20      } line number 20
8)      0
9)    141      GOSUB
1040)   53      5
1)     48      0
2)      0
→ 3)    34      } forward chain; 4 x 256 = 1024 + 34 = 1058
4)      4
5)     30      } line number 30
6)      0
7)    147      LOAD
8)     34      "
9)     78      N
1050)   69      E
1)     87      W
2)     80      P
3)     82      R
4)     79      O
5)     71      G
6)     34      "
7)      0
→ 8)    43      } forward chain; 4 x 256 = 1024 + 43 = 1067
9)      4
1060)   50      } line number 50
1)      0
2)    153      PRINT
3)     65      A
4)    172      *
5)     50      2
6)      0
7)     49      } forward chain; 4 x 256 = 1024 + 49 = 1073
8)      4
9)     55      } line number 55
1070)   0
1)    142      RETURN
2)      0
→ 3)    0

```

If we want lines 50 and 55 to be available to an overlay, the important information is the forward chain in line 30, i.e. locations 1043 and 1044.

9. To see how it works, SAVE"PROG" and leave the cassette Record and Play keys down.

Enter NEW; then the following (again, no spaces):

```
5  A=A*2
10 GOSUB50
15  STOP
```

LIST if you like, to confirm that there are no lines 50 and 55.

SAVE"NEWPROG".

Now rewind your tape, and press RUN.

PROG will be loaded, will print "2", and continue up the tape. When NEWPROG has been loaded, you will get

```
?UNDEF'D STATEMENT ERROR IN 10
```

That is because the overlay looks like this:

```
1024)  0
      5)  11      } forward chain to 1035
      6)  4
      7)  5      } line number 5
      8)  0
      9)  65      A
1030) 178      =
      1)  65      A
      2) 172      *
      3)  50      2
      4)  0
→ 5)  19      } forward chain to 1043
      6)  4
      7) 10      } line number 10
      8)  0
      9) 141      GOSUB
1040)  53      5
```

```

1) 48 0
2) 0
→ 3) 25 } forward chain to 1049
4) 4 }
5) 15 } line number 15
6) 0 }
7) 144 STOP
8) 0
→ 9) 0

```

10. The last line, 15, does not chain into the old line 50. But that line 50 is still there, in location 1058 et seq. So, do this:

```

POKE 1043,34
POKE 1044, 4

```

LIST - and behold, NEWPROG now includes lines 50 and 55!

You can RUN if you like, to prove it.

What we have done is to use what we discovered about the first program (last sentence of paragraph 9) to modify the second program.

11. How do you program all this to happen automatically? It is not at all difficult. Let us assume that the statements-to-be-preserved are at lines 5000 and upwards. So, just before that, code this (NO SPACES):

```

4997 N1=PEEK(201) Get (low) address of line 4998
4998 N2=PEEK(202) Get (high) address of line 4999
4999 RETURN
5000 .....

```

(Locations 201, 202 always contain, during instruction execution, the address of the next instruction - strictly, the "Ø" between instructions.)

12. Now, just before your program wants to load in the overlay program, code this (spaces if you like!):

```
850 GOSUB 4997
860 N1 = N1 + 14           Low address of 4999 (the length
                          of 4998 is 14 bytes)
870 N2 = N2 * 256        Actual high address of 4999
880 If N1 < 256 THEN 900 Adjust low for page boundary
890 N1 = N1 - 256
900 BC = N1 + N2 + 1     BC is now actual machine address
                          of line 4999
910 Z1 = PEEK(BC):Z2=PEEK(BC+1) Hold the forward-chain
                          locations out of 4999
920 LOAD "NEWPROG"
```

13. As the first instructions of NEWPROG, the chain-adjusting must be done. The necessary code is very similar:

At the end of NEWPROG, as the very last statements, code (NO spaces):

```
3997     N1=PEEK(201)
3998     N2=PEEK(202)
3999     RETURN
```

And at the beginning code:

```
10     GOSUB3997
20     N1=N1+14
30     N2=N2*256
40     IF N1<256 THEN 60
50     N1=N1-256
60     BC=N1+N2+1     BC is now actual machine
                          address of 3999
70     POKE BC,Z1:POKE BC+1,Z2
```


14. It is worth just reiterating that the total size of the incoming overlay (irrespective of line numbers; just its size in bytes occupied) must be less than the total size of the instructions being overlaid.

Mike Stone

ABBREVIATING BASIC WORDS

As explained in the instruction manual, any BASIC word takes up 1 byte of memory storage space. It has been stated that the word "PRINT" can be abbreviated to "?" which saves time on entering programs. When listed, the word is expanded to its full form. Both forms take 1 byte per word.

We now have information on how to abbreviate the complete list of BASIC words. The algorithm to remember is as follows:

1. For any BASIC word, type in the first letter of the word (e.g. V for VERIFY).
2. Hold down the 'Shift' key and type in the second letter. If you are in graphics mode, this will appear as a graphic character (e.g.  for E). It is a good idea to go into lower case mode as the two letters are then easy to read. (Poke 59468, 14 → 12 for PET to graphics).

In some cases, this two-letter method gives a possibility of more than one BASIC word (e.g. READ and RESTORE). For one of the words (usually the longer) it will be necessary to type the first two letters and the shifted third. All these abbreviations are converted to full words upon the command LIST.

Below is a complete list of the words and abbreviations:

| <u>BASIC</u> | <u>ABBREV</u> | <u>BASIC</u> | <u>ABBREV</u> | <u>BASIC</u> | <u>ABBREV</u> |
|--------------|---------------|--------------|---------------|--------------|---------------|
| LET | Le | DEF | De | RUN | Ru |
| READ | Re | RETURN | REt | CLR | Cl |
| PRINT | ? | STOP | St | LIST | Li |
| PRINT# | Pr | STEP | StE | CONT | Co |
| DATA | Da | INPUT# | In | FRE | Fr |
| THEN | Th | SGN | Sg | TAB (| Ta |
| FOR | Fo | ABS | Ab | SPC (| Sp |
| NEXT | Ne | SQR | Sq | PEEK | Pe |
| DIM | Di | RND | Rn | POKE | Po |
| END | En | SIN | Si | USR | Us |
| GOTO | Go | ATN | At | SYS | Sy |
| RESTORE | REs | EXP | Ex | WAIT | Wa |
| GET | Ge | AND | An | LEFT\$ | LEf |
| GOSUB | GOs | NOT | No | RIGHT\$ | Ri |
| OPEN | Op | VAL | Va | MID\$ | Mi |
| CLOSE | CLo | ASC | As | CHR\$ | Ch |
| SAVE | Sa | CMD | Cm | STR\$ | STr |
| LOAD | Lo | VERIFY | Ve | | |

SIMULATING A CALCULATOR ON YOUR PET

Many users have asked whether the PET can do live calculations. Although a simple sum such as $2 + 3$ can be performed thus:

```
PRINT 2 + 3 'RETURN'
```

it would be more convenient if the operation of a calculator could be simulated directly. The following program should give you an idea of how this can be achieved:

```

5 REM GRAPHICS
10 PRINT"C
20 PRINT"
25 PRINT"
30 PRINT"
40 FOR I=1 TO 19
50 PRINT"
60 NEXT
1000 REM CONTROLLER / INPUT
1010 GET A$: IFA$="" GOTO 1010
1020 A=ASC(A$)
1030 IFA>57 THEN 4000
1040 IFA<48 AND A<>46 THEN 2000
1050 IF T=1 THEN X$="" : T=0
1055 IF LEN(X$)=9 THEN D$="ERROR " : GOSUB 5115 : T=1 : GOTO 1000
1060 X$=X$+A$ : X=VAL(X$) : GOSUB 5020
1070 GOTO 1000
2000 REM OPERATORS
2010 IFA<48 OR A=44 THEN D$="ERROR " : GOSUB 5115 : CLR : GOTO 1000
2020 IFA=48 THEN N=N+1 : B(N)=X : X=0 : Y=0 : O$(N)=O$ : O$="" : T=1 : GOSUB 5000 : GOTO 1000
2030 IF O$="*" THEN X=X*Y
2040 IF O$="/" THEN X=Y/X
2050 IF O$="+" THEN X=X+Y
2060 IF O$="-" THEN X=Y-X
2065 Y=X : O$=A$ : T=1
2070 IFA=41 THEN Y=B(N) : O$=O$(N) : N=N-1 : T=0
2080 GOSUB 5000 : GOTO 1000
4000 IFA$="S" THEN X=SIN(X)
4010 IFA$="C" THEN X=COS(X)
4020 IFA$="T" THEN X=TAN(X)
4030 IFA$="L" THEN X=LOG(X)
4040 IFA$="E" THEN X=EXP(X)
4042 IFA$="" GOTO 2030
4043 IFA$="←" THEN CLR : T=1
4045 GOSUB 5000
4050 GOTO 1000
5000 REM DISPLAY
5010 X$=STR$(X)
5020 D$=RIGHT$(" "+X$,11)+" "
5030 IF X<=9999999999 AND X<.01 GOTO 5115
5040 IF X=0 GOTO 5115
5050 IF ABS(X)>1E38 OR ABS(X)<1E-38 THEN D$="ERROR " : GOTO 5115
5100 R$=RIGHT$(" "+X$,15)
5110 D$=LEFT$(R$,11)+" "+RIGHT$(R$,3)
5115 PRINT"#####" D$
5120 RETURN
READY.

```



{ "home" 2x "CURSOR DOWN" 12x "CURSOR RIGHT" }

Although the program is by no means perfected, the framework exists for a versatile program. Lines 4000 onwards determine the functions so that when 'S' is pressed the sine of the number on display is calculated and to clear all registers '+' is pressed. The normal operation for +, -, x and ÷ is the same as a straightforward calculator, and there are multiple sets of brackets.

This idea could be used to simulate actual models - including programmable calculators, thus giving access to a wide range of ready-written low key software. We would like to hear from any user who succeeds in doing this.

BITS AND PIECES

Some more hints and tips to help you write efficient programs:

When writing REMark statements, graphics and lower case can be included if they are put inside inverted comma's. This enables separating lines such as:

```
10 REM "_____"
```

```
* * * * *
```

When using subscripted variables such as A(4) the operating system automatically reserves 10 elements without having to declare a dimension with DIM. If, however, you are using a very long program and are using less than 10 elements per variable - say 4 - it will save space to declare the dimension's length. For example:

```
10 DIM A(4), C(3)
```

```
* * * * *
```


To display a number (N) to D decimal places, use the following routine:

```
1Ø M = INT(N*1Ø↑D+Ø.5)/1Ø↑D
```

```
2Ø PRINT M
```

```
* * * * *
```

For an intriguing display of graphics, try running this one line program entitled "BURROW"

```
1 A$="↑↓⇒⇐":PRINTMID$(A$,RND(.5)*4+1,1)"*⇐";:FORT=1TO3Ø:  
NEXT:PRINT"☐⇐";:GOTO1
```

```
* * * * *
```

STANDARD SYMBOLS

I have assembled a small table of symbols that are not hard to obtain from a typewriter (if you are using one) and are quite distinguishable if you write out your programs in capital block letters. It would be appreciated if you use these when submitting software for publication; especially programs containing cursor control.

SYMBOLS

| | |
|------------------|---------------------|
| h - cursor home | ¢ - carriage return |
| c - clear screen | ␣ - space (blank) |
| cr- cursor right | ! - RVS on |
| cl- cursor left | / - RVS off |
| cu- cursor up | @ - Shift on |
| cd- cursor down | |
| d - delete | |
| i - insert | |

To represent any graphic characters clearly, the character below should be followed by the '@'. For example:

☐ = M@ (M w/Shift on)

Following this standard should make programs fairly easy to read however any suggestions are quite welcome.

INTERFACERS.....

This month I received a letter from Andrew Hwang of Concordia Designs in Toronto. Andrew has successfully interfaced a PET to an X-Y Plotter. Excellent! A copy of his response follows. Thank you Andrew.

If anyone else has gizmos or gadgets interfaced to their PETs, be it practical or unusual, write in and tell us about it! A brief note is sufficient (such as Andrew's), and businessess... the "TRANSACTOR" is sent to over 500 subscribers. Get the hint?

**CONCORDIA
DESIGNS**

INNOVATIVE
CUSTOM DESIG
SPECIALISTS

INTERFACING DIVISION

P.O.Box 219, Station D, Scarborough, Ontario, Canada. M1R 5B7

CPU Club Members

c/o Commodore Business Machines Ltd.
3370 Pharmacy Ave., Agincourt,
ONTARIO. M1W 2K4

13 Dec.1978

RE: Full Graphical Plotting Capability
with the PET and a Digital Plotter

Dear Fellow CPU Members:

We have succeeded in interfacing the PET with a Digital Plotter made by Houston Instruments. Full hard-copy graphs of data or functions can now be easily plotted with pen accuracy of up to 0.005 inches. Softwares are also available to issue plotting commands in simple basic steps. Anyone wishing more information may write to: Concordia Designs, Interfacing Div., P.O.Box 219, Station D, Scarborough, Ontario, Canada. M1R 5B7.

Faithfully,



Mr. Andrew Hwang, M.Sc.
PET User Club Member #65

Jim Butterfield, well known PET enthusiast, has sent to Commodore a couple of interesting items which have also been passed on to PET User Notes. Thanks Jim...and thanks for recognizing Brad as a source--maybe he'll send in something else as well.

Summary of Cassette data file "patches"

The following information has been passed around users, and is now "official" with the issuance of a Commodore bulletin. It seems worth while to summarize briefly:

1. Opening a file for writing: an omission in current ROM programs makes it highly desirable to precede all OPEN statements with a couple of POKES:

```
before OPEN x,1... for writing: POKE 243,122 : POKE 244,2
before OPEN x,2,.. for writing: POKE 243,58 : POKE 244,3
```

2. When writing tapes, it is useful to increase the spacing between tape blocks; otherwise you might miss a block during subsequent reading. There are several approaches to this; my technique is to call the following subroutine immediately after each PRINT#:

| Cassette #1 | Cassette #2 |
|------------------------------|------------------------------|
| 950 IF Z9<=PEEK(625)GOTO 990 | 950 IF Z9<=PEEK(626)GOTO 990 |
| 960 POKE 59411,53 | 960 POKE 59456,207 |
| 970 FOR Z9=1 TO 60:NEXT Z9 | 970 FOR Z9=1 TO 60: NEXT Z9 |
| 980 POKE 59411,61 | 980 POKE 59456,223 |
| 990 Z9=PEEK(625):RETURN | 990 Z9=PEEK(626):RETURN |

3. Even with the above coding, it seems wise to guard against a potential "dropped block". Think in terms of writing a "number of items" total on tape so that when reading, you can check that nothing has been lost.
4. Don't PRINT# a line of over 80 characters unless you're prepared to do some careful work with GET# statements when you read it back in. In general, avoid "print punctuation" when writing (PRINT#1, A;B ... PRINT#2, X\$,Y\$); each data element can be written as a separate "line". Watch for long strings.
5. Either: check the value of ST after every read, or use your own checking routines on your data. ST can be useful, but doesn't guarantee your data is 100% good. IF ST=0 .. no errors are seen; IF ST>63 .. you have come to the end of file; if anything else, an error has been detected.
6. Always CLOSE your cassette files after you're finished with them. When writing, your data is accumulated into a buffer .. if you don't CLOSE, it may not go onto tape.

Most Basic errors abort the cassette file without CLOSEing it; if this happens while you have a cassette tape open for writing, better start over .. your tape will likely have data missing.

Larry Tessler's UNLIST appeared in User Notes without much fanfare. If you can read between the lines, however (and cope with the typos) it's quite a blockbuster of a program.

In general, it allows a program to be handled as data ... using the UNLIST key, you can re-process programs as if they were data files, and create such things as program-writing programs, language translators, and many other startling things.

Perhaps the most immediate use of UNLIST for the casual computerist is to merge two programs together. One program could contain subroutines; and these could be merged with other programs to save a lot of typing. It's especially useful to be able to merge a single set of DATA statements into several programs, each of which is set up to process the DATA in different ways.

Brad Templeton (Toronto) has passed me an even more concise way of doing the same thing. I'll go through the whole operation, step by step.

First, prepare the program you will want to merge in the following manner. Load the program. Put a blank tape into the cassette and rewind. Enter OPEN 1.1.1 : CMD 1 : LIST. Be sure to put this on a single line, using colons as indicated. Press RECORD and PLAY as instructed. The tape will move. When it stops, type ?"POKE611,0": PRINT#1:CLOSE1. Your tape is now ready, and PET should be back to normal operation. You may file this tape and use it at any future time.

Now for the merge. When you have your second program loaded into the PET, mount the tape you have previously written. Type OPEN1, press PLAY as requested, and wait for the tape to stop.

Here comes the tricky bit. Clear the screen, give 4 cursor down's, and type the following line, but DO NOT HIT RETURN:

```
POKE611,1:POKE525,1:POKE527,13:?"h" (h is Cursor Home, displaying reverse S)
```

Don't hit return. Instead, press cursor home and 6 cursor downs ... then type the identical line. This time, hit RETURN at the end of the line and listen to the tape move.

Eventually, things will stop with a ?SYNTAX ERROR or ?OUT OF DATA printed between the two lines, and the tape should stop. (If it doesn't, stop it with the RUN-STOP key). The merge is now complete. Type CLOSE 1 to close the file.

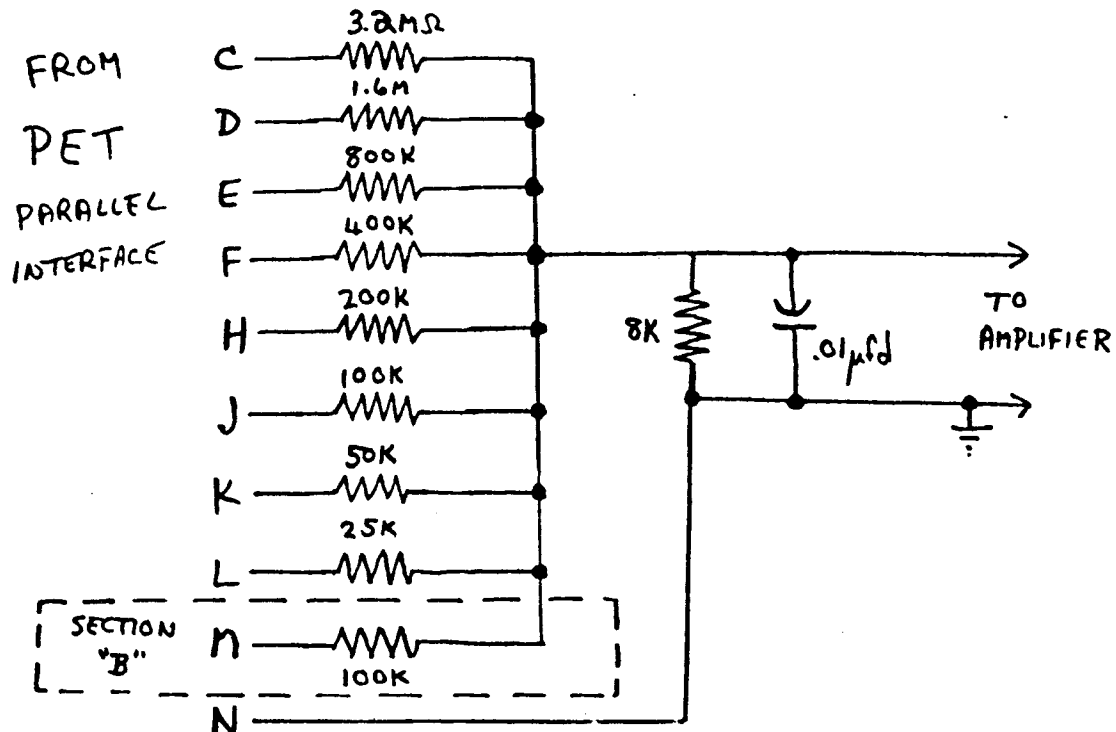
(The ?"POKE611,0" in paragraph 5 may be unnecessary .. I put it there to guard against a processor crash situation I encountered during early testing ...)

Jim has also submitted a schematic of a circuit he demonstrated at the first meeting of the Toronto PET Users Group. The schematic has been published in the club newsletter and Lyman Duggan suggested I pass it on to Transactor subscribers. Thanx again Jim. -Karl J.

Poor Man's D/A Converter

Cheap; good for generating Chamberlin/style music. Precision resistors are preferred, but most anything will generate a recognizable sound.

Section B of the diagram supports the CR2 sound effects - so that one interface board covers most sound requirements.



The capacitor provides some reduction of the sampling frequency (when generating music) .. tone controls on the amplifier will also help, if available.

The HUH "Petunia" gives a high quality equivalent of this converter. Toronto price - about \$40.

Reference: see BYTE, September 1977, lengthy article by Hal Chamberlin on computer-generated music. 6502 programs are given.

The "ON...GOTO" Statement

e.g. 100 ON I GOTO 10, 20, 146, 2040

The above statement, when encountered, will cause program execution to branch to the line indicated by the I'th number after the GOTO. That is:

```
    If I=1    GOTO 10
    If I=2    GOTO 20
    If I=3    GOTO 146
    If I=4.4  GOTO 2040
```

I is truncated to an integer value. If I is equal to zero, or attempts to select a non-existent line (greater than or equal to 5), in this case the statement after the ON...GOTO is executed. As many line numbers as will fit on a line can follow an ON...GOTO. Thus the main purpose is to eliminate successive IF...GOTO statements and save on memory consumption.

e.g. 200 ON SGN(X)+2 GOTO 40, 50, 310

In this case, execution will branch to line 40 if the SGN(X) expression is less than zero, line 50 if equal to zero and line 310 if equal to one.

When using expressions in the ON...GOTO statement, do not allow the final result to be negative. Implement an ABS function into the expression, else an ILLEGAL QUANTITY ERROR will result.

The "ON...GOSUB" Statement

Identical to the ON...GOTO statement except that a subroutine call is executed instead of an absolute GOTO. On return from the subroutine, execution continues at the line following the ON...GOSUB.

A Short Note On Subroutines

Jumping out of a subroutine can be HAZARDOUS! That is, subroutines containing IF...GOTO's or ON...GOTO's can cause an OUT OF MEMORY ERROR for this reason: When a program encounters a GOSUB, the machine loads the return address into a stack area. The subroutine is then executed and let's say 'jumps out' on a GOTO statement within the subroutine. Thus the RETURN statement is never executed and the return address is not unloaded from the stack. The stack will fill to the limit and the error message results.

To avoid this problem the ON...GOTO statement can be used instead of RETURN. Thus all GOSUB statements directed at that particular subroutine must also be replaced with absolute GOTO's. By implementing a control variable just prior to the GOTO, the return addresses can be placed after the ON...GOTO in the subroutine and the hazard is eliminated. For example:

```
100 C=1:GOTO 5000          Subroutine address
110 REM RETURN FROM SUB.
```

```
500 C=2:GOTO 5000          Same subroutine
510 REM RETURN FROM SUB.
```

```
5000 REM SUBROUTINE
```

```
5010 } 'SUBROUTINE'
```

```
⋮
```

```
5080 }
```

```
5090 ON C GOTO 110, 510      Instead of RETURN
```

NEWSLETTER ADDRESSES

1. The "Tranasctor" (subs. \$10.00 Cdn.)
3370 Pharmacy Avenue
Agincourt, Ont.
M1W 2K4
2. PET User Notes (subs. \$6.00 U.S.) †
P.O. Box 371
Montgomeryville, PA 18936
3. THE PET PAPER (subs. \$15.00 U.S.) †
P.O. Box 43
Audubon, PA 19407
4. Cursor (subs. \$24.00 U.S.) †
P.O. Box 550
Goleta, Calif. 93017
5. The PET Gazette (Free w/lg. S.A.S.E. U.S. postage) †
c/o Len Lindsay
1929 Northport Drive
Room 6
Madison, Wisconsin 53704

FAILSAFING

Recently I have received a number of inquiries on how to avoid entering the command mode by hitting "RETURN" after an input statement without data entry. I know of three such methods:

1. The "GET" loop instead

e.g. 20 GET A\$: IF A\$=" " THEN 20
 30 PRINT A\$

The above routine will loop continuously in line 20 until a key is depressed. Once entry is made, the routine will print the entry be it alphabetic, numeric or graphic character. Use of the numeric variable is confusing because even if no key has been struck, the value returned is zero. That is:

```
20 GET A: IF STR$(A)=" " THEN 20
30 PRINT A
```

will return 0 immediately.

2. Forced Input

If after an INPUT statement you arrange an invalid input to the right of the '?', hitting "RETURN" will result in a ? REDO FROM START and go back to the INPUT statement. For example:

```
10 INPUT "A VALUE xxxx ? clclclcl";A
20 PRINT A
```

The cursor is left 2 character spaces beyond the '?'. Therefore you must arrange your 'invalid input' such that it will be erased by the entry else it will be included (be it to the right or left of the entry), and a ?REDO FROM START will be returned.

The above routine does not work with string variables because ' ' will be accepted as an alphabetic input if "RETURN" is hit. Therefore a test statement must be added:

```
10 INPUT "A CHAR. clclclcl";A$
15 IF A$ = " " THEN 10
20 PRINT A$
```

3. Opening the Keyboard as a File

By assigning the keyboard a file number using the OPEN statement and opening that file for reading, all input statements will accept data only as an entry. The following program will demonstrate this.

```
1 OPEN 1,0,0 (last 0 optional)
.
.
.
500 PRINT "A VALUE?";
510 INPUT #1,A$
520 PRINT: PRINT A$
530 END
```

The open statement may be placed at the very beginning of the program and may even be line 0. Other programming can be inserted (between 1 and 500). The preceding words and '?' must be displayed using a PRINT statement (line 500). The double PRINT's in line 520 are required to get A\$ to print on the next screen line. Otherwise A\$ will be displayed just to the right of the entry.

If anyone should find bugs with this due to not closing the file or otherwise, please let me know so that I may pass it on.

HARDWARE FIX

Most PET Users involved with data files are aware of the problem concerning file reads. That is after a block READ the tape motor does not stop instantaneously and tends to roll the tape past the beginning of the next block resulting in 'LOST DATA'.

Realizing this, Richard Leon and Larry Phillips of the Vancouver PET Users Group devised a hardware fix. It consists of a resistor, a capacitor, a diode and the unused half of the DPDT switch connected to the record button of the cassette deck (Figure 1).

0.5v Motor Source

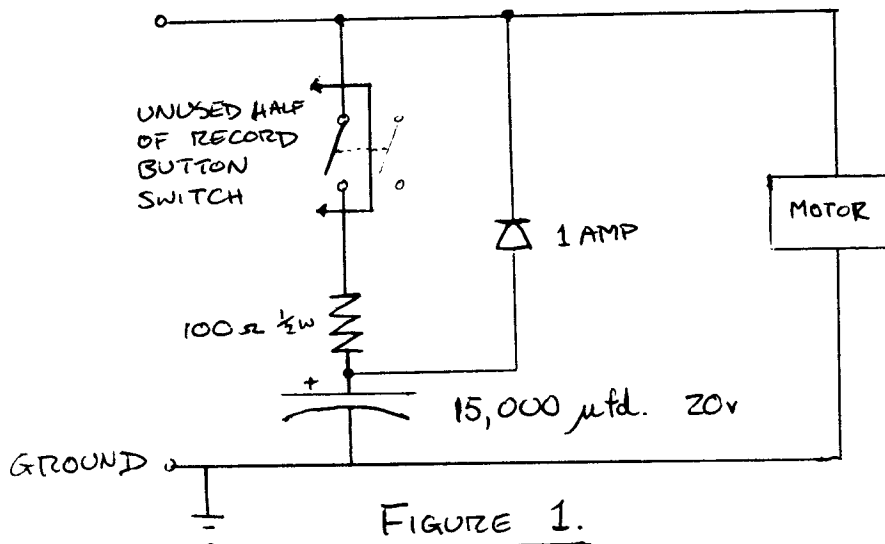


FIGURE 1.

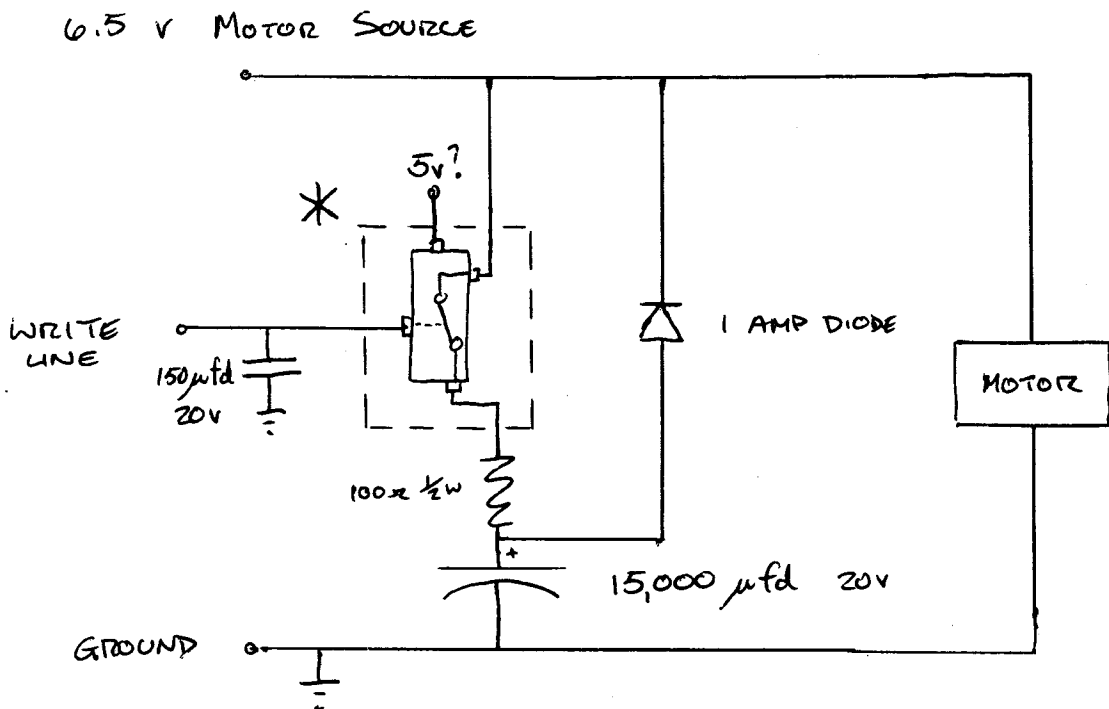
Operation is relatively simple. During Data writes the record switch is closed and the capacitor charges through the resistor. When the block is finished the motor voltage turns off and C1 discharges into the motor through the diode causing the armature to rotate that extra little bit. This allows a larger gap between blocks. During reads, the motor still does not stop instantaneously but it won't roll far enough for the tape head to encounter the next block. (During reads the record switch is up and the charging network is disabled).

The fix itself poses only one problem. Installation. Many of you may not be willing to dismantle your cassette decks to access the record button DPDT switch. Therefore I have a second idea. Using the same concept as Richard Leon, build an 'INTERFACE' for your tape deck and install it between the 6 pin molex plug and the printed circuit board edge connector. The same circuitry is used with one major difference. The enable/disable switch must be simulated such that identical operation is obtained without using a mechanical switch. Therefore the switch needs control logic such that it enables the circuit during data writes and disables it during reads. Ideally, this could be done with logic gates, however most TTL logic chips require +15 and -15 volt supplies. There is "logic" available on the PCB cardedge. Here are the characteristics:

| PIN on PCB Cardedge | OPERATION | | |
|---------------------------|--------------------|--------|--------|
| | NONE | READ | WRITE |
| READ | Low | Active | Active |
| WRITE | High | High | Active |
| SENSE | High | Low | Low |
| MOTOR | Open (floating) | 6.5v | 6.5v |

Using these lines a control must be designed to either open the switch during reads or close it during writes. This is where I stand right now. The 'write' line seems to be the most likely candidate for the control input since 'write' is held high constantly and goes low only during writes. Therefore it could be used to hold open a N.C. (normally closed) switch. However, when data is passed to the tape, 'write' is active (high and low) which means the switch will be on, off or unstable anyways.

If this signal could be 'filtered' and seen by the switch input as low during writes the switch would stay closed and the design is complete. Here's what I have so far. Will it work?



* N.C. integrated switch. Part number anyone?

FLASH

Richvale Telecommunications, 10610 Bayview Ave; Richmond Hill, Ontario L4C 3N8 884-4165 are pleased to announce "Hampet": Morse and RTTY interface.

The "Hampet" is a morse and RTTY(Baudot) interface. You simply plug the "Hampet" into the Pet, and your ready to display, transmit and receive morse and RTTY at rates upto 100WPM. The interface will also send random words and characters, displayed and with audio from a built in side tone oscillator and speaker. In addition the "Hampet" can re-transmit 400 characters from memory for "brog" transmissions.

The "Hampet" is sold wired and tested, and comes in an attractive, blue and sand, sloping front, aluminum case, with on-off, volume control, frequency control, PLL control, tune LED, on LED, headphone muting jack and four RCA phone jacks. The "Hampet" is capable of perfect copy with an "S4" signal. The "Hampet" retails for 189 dollars with immediate delivery.

Dealer enquiries are invited.

ADDRESSING

Every memory location in your PET contains one byte of information. In order for PET to get at these bytes it must have a means of accessing them. Therefore each and every memory location has its own individual address; all 65536 of them. The microprocessor places these addresses on the address buss which immediately enables one memory location to the data buss. Bearing that in mind, one of two operations can happen now. PET can either place a byte into that location (i.e. POKE) or "look" at what's already there (i.e. PEEK). When performing the first operation the microprocessor places a byte on the data buss and transfers it along the buss and into the enabled memory location.

In the second operation, the information or byte in the enabled location is transferred onto the data buss and along

the data buss back to the microprocessor. This location is not "emptied" but rather only a duplicate or copy of the information is transferred. Once either of these operations is complete the microprocessor then places a new address on the address buss and another location is enabled. This process repeats thousands of times every second, however these operations aren't possible on all memory locations, but I'll explain this later.

The microprocessor has control of 99.9% of the addresses being placed on the address buss. That extra 0.1% control was left for the user and can be obtained through use of the PEEK, POKE and SYS commands. When executing these commands the user must choose an address. This address will be one of the 65,536 memory locations (i.e. 0 to 65535). This is where the memory map enters the picture. The memory map may well be your most powerful tool for choosing addresses. If you look at the map you'll see that all of the addresses are listed in ascending order down the left hand side; first in hexadecimal and then in decimal. (See section on hexadecimal and binary for explanation of this conversion) the decimal address is the one you use when executing the above 3 BASIC commands. To the right are the descriptions of what you can expect to find at the corresponding addresses. If we then PEEK these addresses we are returned the actual bytes that are in those particular memory locations. For example, let's say during a program we hit the STOP key and got:

```
BREAK IN 600  
READY.  
0
```

PET gets '600' from a storage register at addresses 138 and 139. We could also PEEK these locations and find that 600 is indeed stored in 138, 139. However it is not stored as a six, a zero and a zero. Instead it is stored as the decimal conversion of

the line numbers representation in hexadecimal. All information of this type is returned in this manner. Now that we know what the memory map will help us do let's cover some of the rules.

RAM and ROM

We all go through life with basically 3 types of memory:

1. MEMORY PRESENT: This memory we use to remember things like **what** street we're driving on or our present location.
2. MEMORY PERMANENT: Things like our names and fire is hot we never forget.
3. MEMORY PAST: Recent occurrences and not so recent such as things we did 10 or 12 years ago.

In the PET there are only two:

1. RAM Random Access Memory: This type of storage is used for our programs and things that change such as the clock and previous line number.
2. ROM Read Only Memory: This is PET's permanent memory. In ROM are the addition routines, clock updating routines and loading routines to name a few. These functions would have to programmed into PET on each power up if they weren't permanently 'burnt in'.

The third type, memory past, is instantly 'forgotten' on power down. The only way to recall it is to first save it on tape, disc, etc.

Recall earlier I mentioned that POKE and PEEK aren't possible on all memory locations for several reasons:

- A. Not all PET memory locations actually exist. On the memory map, locations 1024 to 32767 is the 'available RAM including expansion'. If you have a PET with 8K, simple arithmetic

shows that 3/4 of the available RAM space is non-existent. If you decide to expand your system, PET will 'fit' the added RAM into this area. However POKing or PEEKing this space (i.e. 8192 to 32767) will return invalid results on 8K PETs.

- B. The same concept applies to locations 36864 to 49151. This is the available ROM expansion area.
- C. Next on the memory map is the Microsoft BASIC area; locations 49152 to 57463. This is the memory that recognizes and performs your commands. Changing the contents of these locations is impossible because it is Read Only Memory and is actually 'burnt in' at the factory. Therefore, POKing these locations will simply do nothing. Also, Microsoft requested that these locations return zeros if PEEKed (for copyright reasons).

With these 3 rules and your memory map you are now equipped to explore capabilities of your PET that you probably never thought possible. Before we try some examples let's go into one more important occurrence that may have had you scratching your head ever since that first power up.

MISSING MEMORY?

When you turn on your 8K (where **K** = 1024) PET, the first thing it tells you is 7167 BYTES FREE; a reduction of almost 12%.

- Q. Where did the missing 1024 bytes go?
- A. It's still there...right below the available RAM space (notice it starts at location 1024). PET uses this memory to do some very useful operations for you which you can find and access by looking them up on the memory map.
- Q. But why not do this in ROM space?
- A. PET needs RAM type memory to store this data because it is always changing. The information in this "low" end of memory is actually produced by routines found in ROM.

Take for example the built-in clock. The clock or time is stored in locations 512, 513 and 514 of RAM. However the data comes from a routine found in ROM at location F736_{hex}. The time is of course always changing, therefore it must be stored in RAM. But because it is in RAM, you may also change it; either by setting TI or TI\$ or you can POKE the above 3 locations. Try it.

Now let's try some examples.

1. Location 226 (00E2 in HEX) holds the position of the cursor on the line. Try these:

```
POKE 226,20:"PRINTS AT NEXT SPACE  
?"123456789";:?PEEK(226)
```

2. Location 245 (00F5 in HEX) stores the line the cursor is presently on (0 to 24). POKing this location will move the cursor to the specified line after a display execution. For example try:

```
?"A": POKE 245,10:"B":?"C"
```

```
POKE 245,21-1:"cu":POKE 226,20:"PRINTS HERE"
```

The above will move the cursor to line 20 (21-1), print a 'cursor up' on line 21 and display your message starting at column 21, line 20.

While experimenting with out-of-range values I obtained some rather interesting results. Try POKing location 245 with a number greater than 24, say 40 or 60, and hit the cursor up/down key a number of times. Also, experiment with unusual numbers in location 226 such as:

```
POKE 226,100:"123456789"
```


In the command mode (i.e. when you're operating PET directly all typed keys go first into the keyboard buffer and then into screen memory or VIDEO RAM. However you may also load the buffer under program control by POKing the ASCII representations of the characters into sequential locations of the buffer. You must also increment by 1 the contents of 525 each time another character is POKed in, but remember -- not past 9. Page 6 of "Transactor" #2 contains a table of all the values for characters and commands. "Transactor" #1, page 12 lists some extras such as cursor controls and the RETURN key (13). Try the following endless loop. 145 is a cursor up

```
POKE 525,4:POKE527,145:POKE528,145:POKE529,145:POKE530,13
```

Some other interesting items are:

- POKE59409,52 - Blanks screen
- POKE59409,61 - Screen back on
- POKE59411,53 - Turns cassette motor on
- POKE59411,61 - Turns motor off
- POKE59468,14 - Lower case mode
- POKE59468,12 - Graphics mode
- POKE537,136 - Disables STOP key and clock

If anyone knows of or discovers any peculiarities by "POKing" around, please send them in. When I receive enough of them a handy dandy 'PETRIX' card will be included in a future "Transactor" bulletin.

THE SYSTEM COMMAND

On the last three pages of the memory map are listings of the subroutines stored in PET ROM that perform your commands and programs. These subroutines are stored as machine language.

When a SYS command is executed PET jumps to the specified decimal address and continues from there in machine language. Take for example the Machine Language Monitor program. This is a machine language program and is initialized by a SYS command stored as a BASIC program line. LOAD and RUN your M.L.M. then type 'X' and hit 'RETURN' to exit to BASIC. Now list. What you'll see is:

```
10 SYS (1039)
```

Location 1039 is the address to which PET will jump and also the address at which the first machine language instruction is stored. (A listing of all of the M.L.M. instructions is in "Transactor" #5, pages 5A and 5B) . When this BASIC line is executed PET operates in machine code beginning with address 1039.

The SYS command does not require brackets around the specified address.

Since PET has its subroutines stored in machine language you can use the SYS command to access and execute them. However you may come up with some rather peculiar if not disastrous results. When jumping into ROM you may find yourself in the middle of a subroutine or at the beginning of a subroutine belonging to a major function routine. Often PET will 'hang-up' or crash and you will be forced to power down to resume normal operation. To demonstrate jumping into the middle of a routine, try the following examples:

1. SYS52764 (CE1C)
2. SYS62498 (F422)
3. POKE523,1:SYS62498 (F422)
4. SYS62463 (F3FF)
5. SYS64824 (FD48)

The numbers on the right are the addresses of the above sub-routines in hexadecimal. Compare them to the memory map, especially for e.g. #1. Also take a look at 523.

The following are examples of valid locations which you can use with the SYS command to access useful routines, however these routines are already accessible through BASIC.

1. SYS62651 (F346)
2. SYS62278 (F4BB)
3. SYS63134 (F69E)

Example #3 will perform a 'SAVE' but will not produce a tape header.

Experiment with your memory map. Hex to decimal conversions can be obtained using the method following this article.

SUMMARY

This has been merely 'a scratch on the surface' of the extremely complex inner workings of PET. Do not be afraid to experiment with the POKE and SYS commands. There is absolutely nothing you can do to harm PET from the keyboard that turning power off and on won't fix. Also do some PEEKing around especially in low end memory. One good way is to write a small monitor program:

```
10?"c"PEEK(516):GOTO 10
```

The above will monitor the 'SHIFT' key. Try running it and depress 'SHIFT'. Compare the map.

When POKing or SYSing to random addresses, remember the address you choose. Often PET will do something which may erase the address from the screen (e.g. SYS64840).

The addresses that have been listed here are only a few of many that are already known and only a minute percentage of the ones not known. Probe around and send in any discoveries, useful, peculiar or otherwise. They will be collected together and published in a future "Transactor" bulletin.

BINARY to HEXADECIMAL to DECIMAL

We all know how to count in base 10 or decimal. We start at zero and count one...two...three and so on to nine. Once nine is reached we've run out of numbers, that is single digit numbers. So in order to continue we must now make use of two digits; we place a "1" in the 10's column and reset the 1's column back to zero. Continuing from here, sooner or later we would reach 99. Adding "1" would generate a carry into the 10's column and this in turn will generate a carry into the 100's columns to zeros.

This explanation of base 10 was given simply to demonstrate how we actually do our counting that we just do naturally. Binary is much simpler than decimal because there are only two numbers to worry about; zero (0) and one (1).

| | |
|-------------------|------------------------------|
| Base 2 number set | Base 10 number set |
| 0, 1 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |

With a little practise you'll see that counting in Binary is just as easy as counting in decimal.

Binary is base 2 ('Bi') just as decimal is base 10 ('Deci') just as hexadecimal is base 16 ('Hexadeci') but I'll talk about the "HEX" numbering system later.

In base 10 we are 'allowed' to count up to 9 before carrying the "1" into the next column. Generally in any base we count to one less than the base # and generate a carry into the next column. In base 2 we count up to "1" and do our carry. Just as we cannot fit a "10" base ten into one column we cannot fit a "2" base two into one column. The base # is most important. Let's illustrate by comparison.

| NUMBER | NUMBER REPRESENTATION IN: | |
|--------|---------------------------|---------|
| | BASE 2 | BASE 10 |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0002 |
| 3 | 0011 | 0003 |
| 4 | 0100 | 0004 |
| 5 | 0101 | 0005 |
| 6 | 0110 | 0006 |
| 7 | 0111 | 0007 |
| 8 | 1000 | 0008 |
| 9 | 1001 | 0009 |
| 10 | 1010 | 0010 |
| 11 | 1011 | 0011 |

Notice how in binary, on every multiple of 2 a carry is generated whereas in decimal the carry is generated upon multiples of 10.

Let's now define the columns of the two number bases. In base 10 we have the 1's column, 10's column, 100's column and so on. Each column is the previous column times ten; $1 = 0.1 \times 10$, $10 = 1 \times 10$, $100 = 10 \times 10$ and so on. We can also represent these using exponents; $1 = 10^0$, $10 = 10^1$, $100 = 10^2$ (10 'squared'), $1000 = 10^3$ (10 'cubed'), and so on. In base two each column is the previous column times two; we have the 1's column, 2's column, 4's column, 8's, 16's, 32's and on. Using exponent representation, $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, and so on. Now let's represent some numbers of the two bases using their column breakdown:

$$2_{\text{base } 10} = 0002 = 0 \times 1000 + 0 \times 100 + 0 \times 10 + 2 \times 1$$

$$2_{\text{base } 2} = 0010 = 0 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$$

$$7_{10} = 0007 = 0 \times 1000 + 0 \times 100 + 0 \times 10 + 7 \times 1$$

$$7_2 = 0111 = 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$$

$$12_{10} = 0012 = 0 \times 1000 + 0 \times 100 + 1 \times 10 + 2 \times 1$$

$$12_2 = 1100 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1$$

The same three examples using exponent representation will be:

$$2 = 0002 = 0 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 2 \times 10^0$$

$$2 = 0010 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$7 = 0007 = 0 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$$

$$7 = 0111 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$12 = 0012 = 0 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$$

$$12 = 1100 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

Use this table as a reference for the following exercises.

Try the following example on representing decimal numbers in binary by placing a 1 or a 0 in the correct column position.

| NUMBER | | 2^3 | 2^2 | 2^1 | 2^0 |
|--------|---|-------|-------|-------|-------|
| 4 | = | — | — | — | — |
| 12 | = | — | — | — | — |
| 13 | = | — | — | — | — |
| 14 | = | — | — | — | — |
| 15 | = | — | — | — | — |

What must be done to represent the number 16 in binary. If you said "A fifth digit must be used at the leftmost position", then you're absolutely right. Except for one thing: digit is a word we use in decimal. In binary we use the word BIT derived from Binary digIT. By implementing a fifth bit it is now possible to represent numbers greater than 16 but only up to 31. Once past 31, a sixth bit position must be used. Continue with the exercise. Notice the leftmost column values have changed.

| NUMBER | | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
|--------|---|-------|-------|-------|-------|-------|-------|
| 16 | = | — | — | — | — | — | — |
| 21 | = | — | — | — | — | — | — |
| 28 | = | — | — | — | — | — | — |
| 32 | = | — | — | — | — | — | — |
| 51 | = | — | — | — | — | — | — |
| 62 | = | — | — | — | — | — | — |
| 63 | = | — | — | — | — | — | — |

What would be the highest possible number you could represent using only 6 bits?

$$\begin{array}{cccccccc} & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ ? = & & & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} \end{array}$$

7 bits?

$$? = \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1}$$

8bits?

$$? = \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1} \quad \underline{1}$$

If your answers were 63, 127 and 255, you're correct. Notice how these values are 1 less than the value of the next bit position to the left. ($2^8=256$)

The BYTE

Every memory location in PET is actually one byte. A byte consists of 8 bits. In computer electronics the binary number system is used. This way we can use a high voltage to represent a "1" and a low voltage to represent a "0". Can you imagine the circuitry that would be required to operate a computer in decimal or base 10? Ten unique voltages would have to be used to represent each of the ten digits. Then a separate computer would probably be required to distinguish between them all. By using binary PET must only distinguish between two voltages. Since a 5 volt supply is used for the logic circuitry, anything over 2.4 volts is considered high or a "1" and anything under is considered low or a "0". These voltages are typically 4.8 volts and 0.2 volts, respectively. Each bit of every byte in memory holds one of these voltages. With 8 bits in each byte, 256 combinations can be obtained (0-255) as you can see from the above exercise. If you look at the table on page 6, "Transactor #2, you'll see that all the keys can be encoded into one of these combinations. PET uses some combinations to represent the commands so that they only take up one byte in memory. PET also uses some of these combinations twice to represent graphics as you'll see by comparing the table to page 12 of "Transactor" #1. PET ROM routines distinguish between commands and graphics.

Try POKing a RAM location, say 6000, with a number greater than 255, say 256. A ?ILLEGAL QUANTITY ERROR will be returned because more than 8 bits are required to represent 256 in binary.

$$\begin{array}{cccccccc} 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 256 = & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Essentially, 256 won't 'fit' into a single byte.

Try PEEKing a non-existent memory location, say 10,000:

?PEEK(10000)

A 255 will be returned. A unconnected or open line is considered high by PET. Since the byte is not really there, the data buss lines will be open and read as high or all 1's by the micro-processor.

Hexadecimal or "HEX"

Hexadecimal means base 16. This means we can count up to 15 before generating a carry. However we can't use the numbers 10, 11, 12, 13, 14 and 15; these take up two columns. We need to represent these numbers using a single character. Therefore we use the first 6 letters of the alphabet.

Hexadecimal number set

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

$A_{16} = 10_{10}$

$D_{16} = 13_{10}$

$B_{16} = 11_{10}$

$E_{16} = 14_{10}$

$C_{16} = 12_{10}$

$F_{16} = 15_{10}$

When counting in HEX we generate our carry upon 16:

$$\begin{array}{rcl}
 14_{10} & = & E_{16} \\
 \downarrow & & \downarrow \\
 15_{10} & = & F_{16} \\
 \downarrow & & \downarrow \\
 16_{10} & = & 10_{16}
 \end{array}$$

Recall in binary, 4 bits will yield a maximum of 15

$$15_{10} = 1111_2 = F_{16}$$

Now since a byte has 8 bits, we can split it up into two fields of our and then represent it as two hexadecimal characters.

$$\begin{array}{rcl}
 4_{10} & = & 0000\ 0100_2 = 04_{16} \\
 12_{10} & = & 0000\ 1100_2 = 0C_{16} \\
 255_{10} & = & 1111\ 1111_2 = FF_{16}
 \end{array}$$

HEX Addresses

We won't discuss how a byte recognizes its own address; this is buried deep inside the integrated electronics of the IC chips. The address buss consists of 16 lines, 0 through 15. PET needs this many lines to address all 65,536 bytes. Because location 0 (zero) is included, the maximum address obtainable is 65,535 in decimal. When this location is addressed, all 16 lines of the address buss will have a high voltage. In other words logic 1.

$$\begin{array}{cccccccccccccccc}
 2^{15} & 2^{14} & 2^{13} & 2^{12} & 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 65,535 = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 \\
 & = & 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0
 \end{array}$$

On PET = $2 \uparrow 15 + 2 \uparrow 14 + \dots + 2 \uparrow 0$ Try it.

If we now split the 16 columns into four fields of four we can also represent each field using a hexadecimal character thus converting decimal to hexadecimal as Jim has on the left of the memory map.

$$65,535_{10} = 1111\ 1111\ 1111\ 1111\ (2)$$

$$= \text{FFFF}_{16}$$

Recall e.g. #1 of the SYS command (pg.8)

$$52764_{10} = 1100\ 1110\ 0001\ 1100\ (2)$$

$$= \text{CE1C}_{16}$$

When operating PET, the decimal addresses are used for PEEK, POKE and SYS. Therefore you probably won't find yourself converting from decimal to HEX when using BASIC. However you will need to convert from HEX to decimal when you want to SYS to those ROM subroutines.

$$\text{CE1C}_{16} = 1100\ 1110\ 0001\ 1100$$

$$= 2^{15} + 2^{14} + 2^{11} + 2^{10} + 2^9 + 2^4 + 2^3 + 2^2$$

$$\text{ON PET} = 2 \uparrow 15 + 2 \uparrow 14 + 2 \uparrow 11 + 2 \uparrow 10 + 2 \uparrow 9 + 2 \uparrow 4 + 2 \uparrow 3 + 2 \uparrow 2$$

$$= 52764_{10}$$

$$\text{F422}_{16} = 1111\ 0100\ 0010\ 0010$$

$$= 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{10} + 2^5 + 2^1$$

$$\text{On PET} = 2 \uparrow 15 + 2 \uparrow 14 + 2 \uparrow 13 + 2 \uparrow 12 + 2 \uparrow 10 + 2 \uparrow 5 + 2 \uparrow 1$$

$$= 62498_{10}$$

Try verifying some of the other examples using the same conversion method. With a little practice HEX conversions will be as easy as counting to 'F'.

COMMERCIAL CONFUSION,
or,
"Where'd the penny go?"

Jim Butterfield
Toronto

PET is certainly the greatest business tool since electric pencil sharpeners, and printers and floppy disks will herald an explosion of commercial applications.

Basic seems like the ideal language for a small business system - but it has a hidden "gotcha" that will give you problems if you don't know how to handle it. I call it, "the missing pennies problem", and it's common to almost all Basic implementations.

Crank up your PET and try this: PRINT 2.23 - 2.18 -- it's a simple business calculation and the answer has gotta be a nickel, right? So how come PET says .0499999998?

Think of the mess this could cause if you're printing out neat columns of dollar-and-cent results. Think of the problems if you arrange to print the first two places behind the decimal point: you'll print .04 instead of .05! Think of what the auditor will say when he finds that the totals don't add up correctly!

In a moment we'll discuss how to get rid of this problem. First, though, let's see how it happens.

PET holds numbers in floating binary. That means certain fractions don't work out evenly. Just as, in decimal, one third works out to .333333..., an endless number, PET sees fractions like .10 or .68 as endless repeating fractions - in binary. To fit the fraction in memory, it must trim it. Thus, many fractions such as .37 are adjusted slightly before storage.

Try this program: it will tell you how numbers are stored inside PET:

```
100 INPUT"AMOUNT";A:B=INT(A):C=A-B:?A;"=";B;" ";
110 FORJ=1TO10:C=C*10:D=INT(C):C=C-D:?D;:IFC>0 THEN NEXTJ
120 ?:GOTO100
```

If you try entering numbers in our above example, 2.23 and 2.18, you'll see how PET stores them - and why the problems happen.

How to fix the problem. Easy. Change all numbers to pennies - which eliminates fractions - and your troubles disappear. For example:

```
340 INPUT"AMOUNT";A : A=INT (A*100 +.5)  converts A to pennies;
.....
760 PRINT A/100      outputs pennies in dollars-and-cents.
```

Micro Magazine

Published monthly, MICRO is the only mag. devoted entirely to 6502 based systems. It covers software and hardware on PET, KIM-1, APPLE, AIM, SYM and virtually anything to do with the MOS 6502. MICRO has run out of back issues but they are offering "The Best of MICRO Volume 1", which covers issues 1 through 6. Cost is \$6.00 plus \$1.00 postage and handling, payable in US Funds. Subscriptions are also available; \$15.00 (US) for 12 issues. Send your name and address, etc., to:

MICRO Magazine
P.O.Box 3,
SO. CHELMSFORD, MA
01824.

Specify Best of MICRO and/or subscription and at which issue you wish your subscription to start. Highly recommended.

T I S Workbooks

Total Information Services now has 5 workbooks relating to the PET. All are excellent, particularly PET Cassette on data file writing.

| | | |
|-------------------------------|---|---------|
| Getting Started With Your PET | - | \$5.00* |
| PET String and Array Handling | - | \$5.00* |
| PET Graphics | - | \$6.00* |
| PET Cassette | - | \$6.00* |
| PET Miscellaneous | - | \$5.00* |

* payable in US Funds. Write To:

Total Information Services
P.O.Box 921,
LOS ALAMOS, NM
87544


```

600 DATA3042, 426, 18
610 DATA1521, 167, 17
620 DATA761, 46, 16
630 DATA-1E10, 0, 6
640 DATA-1
650 READX, Y, P: IFIK<X*100GOTO6550
660 T=Y*100:PRINT"ON FIRST $";X:"TAX IS ";Y
670 J=I-X*100:I=FNP(J):PRINT"ON RMG $";J/100:"TAX AT";P;"% IS $";I/100
680 I=I+T:C=3:F=1:I#="TOTAL FED INCM TAX":GOSUB2150
690 S=I:P=25:I=FNS(FNP(D(0))):D(11)=I
695 IFI>0THENF=-1:I#="DIV TAX CREDIT":GOSUB2150
700 I=C(C):I#=" *BASIC FEDERAL TAX*":GOSUB2200:PRINT:D(6)=I
710 IFIK<3E4GOTO740
720 IFIK<=3333E2THENI=3E4:GOTO740
730 P=9:I=FNP(I)
740 C=4:C(C)=0:F=1:I#="GENERAL TAX REDUCTION":GOSUB2150
750 I#="REDUCTION FOR CHILDREN":GOSUB2100
760 S=5E4:I=FNS(C(C)):C(C)=0:C=3:F=-1:I#="TOTAL REDUCTIONS":GOSUB2150
770 GOSUB2300:I#=" **FEDERAL TAX**":GOSUB2200:D(7)=I:D(8)=I
780 C=4:I#="FOREIGN TAX PAID":GOSUB2100:IFI=0GOTO850
790 W=I:I#="FORGN INCOME":GOSUB2100:K=I:X=(D(2)-D(10))/100:Y=(D(7)+D(11))/100
800 S=INT(K/X*Y):PRINTK/100:"/";X:"*";Y:"=":S/100
810 I#="--DEDUCT":I=FNS(W):GOSUB2200:D(8)=D(8)-I
820 PRINT". . ANOTHER COUNTRY. . ":GOTO780
850 I=D(8):I#="FEDERAL TAX PAYABLE":PRINT:GOSUB2200:PRINT
860 P=44:I=FNP(D(6)):I#="BASIC ONTARIO TAX":GOSUB2200:D(9)=I
1000 READX:IFX<>-1GOTO1000
1010 PRINT"==ONTARIO PROPERTY TAX=="
1020 I#="TOTAL RENT PAYMENTS":GOSUB2100:IFI=0GOTO1040
1030 P=20:I=FNP(I):I#="*20% OF RENT":GOSUB2200
1040 C=1:C(C)=I:F=1:I#="PROPERTY TAXES&COLLG RES":GOSUB2100
1050 I=C(C):P=10:X=FNP(I):I#="*OCCUPANCY COST*":GOSUB2200:PRINT
1060 S=18E3:I=FNS(I):I#=" ADD ":GOSUB2200:C(C)=I:I=X:I#=" TO. ":GOSUB2150
1070 I#="PROPERTY TAX CREDIT":I=C(C):GOSUB2200
1080 P=1:I=FNP(D(12)):I#="SALES TAX CREDIT":GOSUB2150
1090 I#="PENSIONER CREDIT":GOSUB2100:I=C(C):I#="TOTAL CREDITS":GOSUB2200
1100 P=2:I=FNP(D(5)):I#="LESS--":F=-1:GOSUB2150
1110 GOSUB2300:S=5E4:I=FNS(I):I#="ONTARIO P S & P CREDITS":GOSUB2200
1120 C(C)=I:I#="POLITICAL TAX CREDIT":GOSUB2100
1130 I=C(C):I#="*TOTAL ONT TAX CREDITS":D(13)=I:GOSUB2200
1140 P1=4:GOSUB2000:I=D(8):I#="FEDERAL TAX PAYABLE":GOSUB2200
1150 I#="POLIT/BUS/EMPLMT CREDIT":GOSUB2100:X=D(8)+D(9)-I
1160 I#="ONTARIO TAX PAYABLE":I=D(9):GOSUB2200
1170 I#="TOTAL PAYABLE":I=X:GOSUB2200:PRINT
1180 C=1:C(C)=0:F=1:I#="TAX DEDUCTED PER SLIPS":GOSUB2100
1190 I#="ONTARIO TAX CREDITS":I=D(13):GOSUB2150
1200 I#="OVERPAYMENTS/INSTALMENTS":GOSUB2100
1210 I=C(C):I#="**TOTAL CREDITS**":GOSUB2200:PRINT
1220 I#="BALANCE DUE":I=X-I:IFI<0THENI#="REFUND":I=ABS(I)

```

```

1230 GOSUB2200:PRINT
1999 END
2000 PRINT:PRINT"===PAGE":GOTO2020
2010 PRINT:PRINT"===SCHEDULE";
2020 PRINTP1:"OF RETURN===":RETURN
2100 I=0:GETZ$:PRINTI$;"? ";
2110 Y$="":PRINT"&";
2120 GETZ$:IFZ$=""GOTO2120
2130 Z=ASC(Z$):IFZ>47ANDZ<58GOTO2145
2135 IFZ$="-"ANDY$=""GOTO2145
2136 IFZ$="."GOTO2145
2139 IF(Z=157ORZ=20)ANDY$<>""THENY$=LEFT$(Y$,LEN(Y$)-1):PRINT" ";:GOTO2147
2140 IFZ$="+"THENPRINT" ";:I=I+VAL(Y$):FORJ=1TOLEN(Y$):PRINT" ";:NEXTJ:GOTO2110
2142 IFZ=13ANDI=0THENPRINT" ";
2143 IFZ=13THENI=FNC(I+VAL(Y$)):PRINT:GOTO2150
2144 GOTO2120
2145 Y$=Y$+Z$
2147 PRINTZ$:GOTO2120
2150 C(C)=C(C)+I*F
2200 PRINTI$:M=1E8:FORJ=LEN(I$)TO25:PRINT" ";:NEXTJ:J=ABS(I):Z$=" ":Z=0
2210 D=INT(J/M):J=J-D*M:IFD=ZTHENPRINT" ";:GOTO2230
2220 Z$=" ";Z=19:PRINTCHR$(D+48);
2230 M=M/10:IFM=1E4THENPRINTZ$:
2240 IFM=10THENPRINT" ";:Z=M
2250 I+D=1GOTO2210
2260 IFI<0THENPRINT"CR";
2270 PRINT:RETURN
2300 B=0:I=FNB(C(C)):C(C)=I:RETURN
READY.

```

TAX ONTARIO 1978

Last months listing of the Income Tax program requires a few revisions due to the fact that the Centronics 779 does not recognize PET graphics. The corrections are as follows:

```

111 INPUT"cdINSTRUCTI.....
112 PRINT"cdONTARIO IN.....
113 PRINT"cdFOLLOW YOUR.....
115 PRINT"cdFOR 'NIL' ITEMS.....
116 PRINT"cdFOR 'MULTIPLE' ITEMS.....
310 PRINT"!EXEMPTIONS.....
470 IF I 231E3 THEN PRINT"!NO TAX PAYABLE.....
2110 Y$="":PRINT"&@c1";
2139 IF (Z=157 OR Z=20) AND Y$ "" THEN Y$=LEFT$(Y$,
LEN(Y$)-1):PRINT"c1␣";:GOTO2147
2140 IF Z$="+" THEN PRINT"cd";:I=I+VAL(Y$):FORJ=1TO
LEN(Y$):PRINT"c1";:NEX.....
2142 IF Z=13 AND I=0 THEN PRINT"cu";

```

cd-Cursor Down, !-RVS On, &@-Shifted '&', c1-Cursor Left,
␣-Blank or Space, cu-Cursor Up.

When operating in machine language, PET is at top efficiency. Machine code programs can execute at speeds 10 to 1000 times that of the equivalent BASIC implementation. Also, depending on the operation, they may consume as much as 10 times less memory. The reason BASIC is so "slow" is that BASIC must first be interpreted into machine code such that the Microprocessor can handle it. In fact, about 90% of the total execution time is spent interpreting while only about 10% of the time is spent on the actual operation. In machine code programs the BASIC interpreter is bypassed hence the greatly increased speed of processing. This speed is realized most in programs where a lot of tests or comparisons are made. MicroChess^c is a prime example. At level 8 (playing at it's best) the machine can still spend as much as 10 minutes on a move in some situations. Imagine a chess program coded in BASIC!

A brief explanation of machine language would be highly impracticable because of the variations and possibilities of the concepts. Undoubtedly a lot of important information would be overlooked. However I have here an excerpt from PET User Notes, Issue #5, written by Jim Butterfield:

"A Little Exercise in PET Machine Language
Jim Butterfield Toronto, Canada

Clear the PET completely (NEW:CLR) and enter the following three lines of BASIC...Do not insert extra spaces!

```
100 SYS(1050)
110 GOT0100
120 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

The last line should not be less than 15 X's, and preferably a few more, say 25. You may list this program but do not try to run yet.

Now you have some POKeing to do, and unfortunately you can't have a program help you. First, make sure that the above lines are OK by ?PEEK(1050); this should return an 88 (X character). Now starting with POKe1050,32...input the following values:

```
Starting at 1050: 32 228 255 208 1 96 162 0 157 0
Starting at 1060: 128 157 0 129 157 0 130 157 0 131
Starting at 1070: 202 208 241 96
```

Double-check the above values by listing them with:

```
FOR J=1050 TO 1070 STEP 10:FOR K=0 TO 9:?PEEK(J+K);:
NEXTK:?:NEXTJ
```

It is vital that these numbers be correct - one mistake and your system will crash. Behind the 96 you should see some leftover X's (88's).

Now type RUN. Try tapping a few keys and note how the screen changes. Stop the program with the STOP key.

What's it all about? We've written a program in machine language, the fundamental 6502 language of the PET. In working with the inner fabric of the machine we find we get: (i) compactness - we've fitted a whole program within one BASIC line: (ii) speed - no BASIC program could fill the screen that fast. We lose, however, in the need for preciseness; one mistake and the system crashes, and you have to switch off and on again. We also lose flexibility - adding an instruction isn't easy.

For those who would like to try tracking the machine language program above, a few brief notes. 32-228-255 calls the PET subroutine to get a character (something like BASIC GET). 208-1--96 exits if no character is seen (like IF X\$=" " THEN RETURN). Now we're ready to zip through the screen with the character we found. We set up for repetition with 162-0 which loads an internal (X) register for 256 repeats; much later we invoke the repetition with 202-208-241, and after the 256th time we return (96). Within the repetition itself, we set the four quarters of the screen with four 157-0-xx instructions."

Those interested in getting seriously involved in machine language should consider first the MOS KIM-1 Microcomputer Module and:

The First Book of KIM By: Jim Butterfield, Stan Ockers and
Eric Rehnke
Publisher: Hayden

The book is mostly machine language programs written for the KIM. Programming them into PET would be most difficult even with the Machine Language Monitor. KIM has numerous subroutines in ROM that aren't like PET's.

Other suggested reading:

Programming a Microcomputer By: Caxton C. Foster
Publisher: Addison-Wesley

MOS 6500 Programming Manual By: Commodore/MOS Technology

6502 User Notes By: Eric Rehnke
P.O. Box 33093
North Royalton, Ohio
44133
\$12.00/Yr.?

All four publications are excellent, but for beginners I suggest the last three in order (if using PET with the M.L.M.).

Besides the program outlined in J. Butterfields article are four more that also operate in machine language. The first three read from the DATA statements data which has already been converted into decimal. This data represents the machine language instructions which (in these three particular programs) are POKed into the second cassette buffer (826 to 1017). Since they are in decimal, a conversion program has been included so you may convert back to HEX and compare them to the table. However, not all of these will necessarily be instructions (as you will see when you find one that matches a "Future Expansion" code). Some may be addresses of direct data depending on the preceeding instruction. Addresses will appear as low order first, high order second. For example:

JSR 00 05

....will jump to the subroutine starting at location 0500.

The DATA statements in Life contain the actual hexadecimal representations of the instructions and addresses. They are read by the program (line 110), tested for validity (lines 120 and 150), converted to decimal (lines 130 and 140) and POKed into memory (line 160) starting at decimal location 6400 (HEX 1900). SYS 6400 executes the program.

View By Jim Butterfield
from an idea by Brad Templeton

```
10 PRINT"SYS826 TURNS PAGEVIEW ON AND/OR OFF
20 PRINT:PRINT"SELECT PAGES WITH 'POKE849,X'
30 PRINT TAB(10)"TRY X=0, 2, 4, 31, 232
40 FOR J=826 TO 858:READ X:POKE J,X:NEXT: END
50 DATA 120,173,25,2,73,200,141,25,2
60 DATA 173,26,2,73,229,141,26,2,88,96
70 DATA 162,0,189,0,0,157,0,128,202
80 DATA 208,247,76,133,230
```

Non-Stop By Jim Butterfield

```
0 REM**MACHINE LANGUAGE STOP KEY DISABLE**
1 GOSUB 63520:END
63520 DATA 120,169,96,141,25,2,169,3
63521 DATA 141,26,2,88,96,0,0,0
63522 DATA 120,169,133,141,25,2,169,230
63523 DATA 141,26,2,88,96,0,0,0
63524 DATA 32,234,255,169,255,141,9,2
63525 DATA 76,136,230
63526 FOR L=832 TO 874:READ K:POKE L,K:NEXT
63527 RETURN
```

Auto-Repeat By The Software Shoppe
From 'The Paper' Volume 1, Issue #10

```
5 REM**MACHINE LANGUAGE AUTO-REPEAT**
10 DATA 120,56,169,233,237,26,2,141
20 DATA 26,2,88,96,173,35,2,201,255
30 DATA 208,12,169,0,141,119,3,169
40 DATA 90,141,120,3,208,25,238,119
50 DATA 3,173,120,3,205,119,3,176,14
60 DATA 169,6,141,120,3,162,255,142
70 DATA 3,2,232,142,119,3,76,133,230
80 FOR I=889 TO 947
90 READ J
100 POKE I,J
110 NEXT I
120 PRINT"SYS889 WILL ENABLE AND DISABLE
130 PRINT"THE AUTO REPEAT FUNCTION
140 END
200 REM**TRY THIS AUTO-REPEAT IN BASIC**
210 GET H#
220 PRINT H#;
230 POKE 515,255
240 GOTO 200
```

Life By Mark Taylor

```
100 READL
110 READA$:C=LEN(A$):IFA$="*"THENEND
120 IFCC(10RC)2THEN200
130 A=ASC(A$)-48:B=ASC(RIGHT$(A$,1))-48
140 N=B+7*(B/9)-(C=2)*(16*(A+7*(A/9)))
150 IFN(00RN)255THEN200
160 POKEL,N:L=L+1:GOTO110
200 PRINT"BYTE"L="[A$]" ???":END
300 DATA6400
310 DATA20,30,19,20,8A,19,20,E6,19,20,00,1A,A9,34,8D,11,E8
320 DATA20,70,19,A9,3C,8D,11,E8,A9,FF,CD,12,E8,F0,E6,4C,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,F3,60,00,80,00,15,00
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
360 DATAEA,EA,EA,EA,EA,20,A6,19,B1,26,D8,06,A9,20,91,20,D8,04,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,EA,20,A6,19,B1,26,D8,06,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,D8,14,A9,01,91,26,D8,0E,A5,32,C9,03,F0,03,C9,02
440 DATAF0,04,A9,00,91,26,20,BD,19,F0,D8,20,A6,19,60,98,48,3A,48,A0,00,84,32
450 DATAA2,00,85,29,10,15,49,FF,85,37,38,A5,39,E5,37,85,22,A5,3A,85,23,B0,11
460 DATA06,23,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,*
```

```

100 DIM A$(16)
110 FOR K=0 TO 39:PRINT ". ";NEXT:A=0:B=0:C=0:L=0:M=0:N=0:X=0
120 PRINT:INPUT"HEX# crcrcrcrcrcrcrcrcr ";HEX$:L=LEN(HEX$)
130 IF L>4 AND L<>2 THEN PRINT" 2 OR 4 HEX DIGITS ONLY":GOTO 110
140 IF L=4 THEN C=2:A$(2)=LEFT$(HEX$,2)
150 A$(1)=RIGHT$(HEX$,2):C=2
160 X=X+1
170 A=ASC(A$(X))-48
180 IF A<0 OR A>22 THEN PRINT"! HEX DIGITS ONLY":GOTO 110
190 B=ASC(RIGHT$(A$(X),1))-48
200 IF B<0 OR B>22 THEN PRINT"! HEX DIGITS ONLY":GOTO 110
210 N=B+7*(B>9)-(C=2)*((16*(A+7*(A>9)))
220 IF L=0 THEN M=M+N:PRINT"DEC ADDRESS = "M:GOTO 110
230 IF L=4 THEN L=0:M=N*256:GOTO 160
240 PRINT"IN DEC = crcrcrcrcrcr "N:GOTO 110

```

100 REM DECIMAL TO BINARY TO HEXADECIMAL CONVERTER FROM TRANSACTOR ISSUE NO. 10
110 REM WRITTEN BY KARL J. HILDON ABSOLUTELY NO COPYRIGHTS

```

120 DIM A$(16)
130 INPUT"DEC VALUE ";D$
140 A=VAL(D$)
150 IF A-INT(A)>0 THEN PRINT:PRINT"! INTEGERS ONLY ";PRINT:GOTO130
160 IF 0>65535 THEN PRINT:PRINT"! 65535 MAXIMUM ";PRINT:GOTO130
170 PRINT:PRINT"BINARY: ";
180 REM *****
190 REM ***BINARY CONVERT***
200 REM *****
210 FOR X=15 TO 0 STEP-1
220 Y=A-2↑X
230 IF Y<0 THEN A$(X)="0"
240 IF Y>=0 THEN A$(X)="1":A=A-2↑X
250 PRINT A$(X);
260 C=C+1:IF C=4 THEN PRINT" ";:C=0
270 NEXT
280 REM *****
290 REM ***HEX CONVERT***
300 REM *****
310 S$=" ";K=3
320 PRINT:PRINT:PRINT"HEX: ";
330 FOR H=15 TO 0 STEP-1
340 G=VAL(A$(H))
350 IF G=1 THEN T=T+2↑K
360 K=K-1
370 IF K<0 THEN GOSUB 400:K=3:T=0
380 NEXT
390 PRINT:FOR L=0 TO 39:PRINT". ";NEXT:PRINT:GOTO130
400 ON T+1 GOTO 410,420,430,440,450,460,470,480,490,500,510,520,530,540,550,560
410 PRINT "0"+S$:RETURN
420 PRINT "1"+S$:RETURN
430 PRINT "2"+S$:RETURN
440 PRINT "3"+S$:RETURN
450 PRINT "4"+S$:RETURN
460 PRINT "5"+S$:RETURN
470 PRINT "6"+S$:RETURN
480 PRINT "7"+S$:RETURN
490 PRINT "8"+S$:RETURN
500 PRINT "9"+S$:RETURN
510 PRINT "A"+S$:RETURN
520 PRINT "B"+S$:RETURN
530 PRINT "C"+S$:RETURN
540 PRINT "D"+S$:RETURN
550 PRINT "E"+S$:RETURN
560 PRINT "F"+S$:RETURN

```


SHIFTED CAPITALS

When writing programs with a great deal of text, it is sometimes best to use the lower case character set. This makes for easier reading. The game Hammurabi is a good example.

To obtain the lower case mode on PET, location 59468 must be POKED with 14 (12 returns it to graphics). Now lower case letters, plus some other graphics are available on the PET by typing the desired letter while depressing the 'shift' key. However when a lot of text is involved, constantly holding down the shift key can become rather awkward. A 'keyboard inverter' would certainly be desirable.

P.T. Spencer, a high school teacher in Agincourt, enquired about this possibility and then answered his own question with the following program which he has submitted for the Transactor.

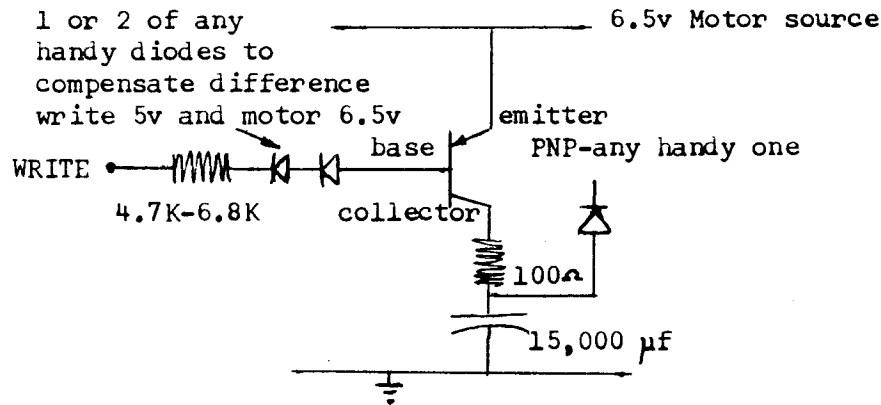
```
10000 REM**KEYBOARD INVERTER SUBROUTINE**
10010 POKE 59468,14
10020 Z$=""
10040 GET Y$:IF Y$="" THEN 10040
10050 Y=ASC(Y$):IF Y=13 THEN 10120
10060 IF Y>64 AND Y<91 THEN Y=Y+128:GOTO 10090
10070 IF Y>192 AND Y<219 THEN Y=Y-128:GOTO 10090
10080 GOTO 10100
10090 Y$=CHR$(Y)
10100 PRINT Y$:Z$=Z$+Y$:GOTO 10040
10120 PRINT:RETURN
```

The program only affects the alphabetic characters and prints everything else out normally. Also, since the characters are being displayed under program control the lines won't be entered into memory, only displayed! To do this program execution would have to be halted and then line numbers followed by "?" inserted at the beginning of every second line of text. (Be careful not to type more than 75 characters per 2 lines or inserting is impossible. Also do not stop program execution less than 5 lines from the bottom or the "BREAK - READY" signal will cause text on the very top line to be scrolled off the screen.) Once this insertion is finished, hitting 'RETURN' enters the characters as program text. Only one pitfall is that these PRINT statements will return the text without the visual continuity (i.e. 'PRINT' starts at extreme left) unless the closing quotation marks are inserted and followed by semi-colons. Therefore it would be more desirable to enter the text into DATA statements and concatenate using the READ command. Instead you would insert the line numbers followed by DATA" or you could have the program do it for you using the following modifications:

```
10000 REM **KEYDATA KEYBOARD INVERTER**
10010 POKE 59468,14
10020 Z$=""
10030 L=0:GOTO 10040:PRINT PRINTED:"cl": "DATA": PRINT CHR$(34)
10040 GET Y$:IF Y$="" THEN 10040
10050 Y=ASC(Y$):IF Y=13 THEN 10120
10060 IF Y>64 AND Y<91 THEN Y=Y+128:GOTO 10090
10070 IF Y>192 AND Y<219 THEN Y=Y-128:GOTO 10090
10080 GOTO 10100
10090 Y$=CHR$(Y)
10100 PRINT Y$:Z$=Z$+Y$:L=L+1:IF L=70 THEN 10030
10110 GOTO 10040
10120 PRINT:STOP
```

Hardware Fix

In Transactor #8, a hardware fix was illustrated to overcome the data file write problem. However, to install it required dismantling the cassette deck so the interface idea was presented. This still had one problem area: the switch. Fortunately, I've received two responses regarding the 'INTERFACE'. The first was from Jim Yost in Somerville, MA. He writes..... "In reference to Bulletin #8: The capacitor charging switch you want is a simple transistor. The basic operational rule is that whatever current flows in the base allows β times that to flow into the collector. β 's typically are 100 or more nowadays. The circuit is thus:



The write pulses (lows) charge the 15,000μF cap in pulses so that it is charged at the end of a write sequence when the motor source drops. Don't use the 15μF - it would short the write pulses".

Secondly, Andrew Chiu of Toronto has designed, among other things, an interface type fix for file writing to the cassette deck. This and other devices are for sale through Rapid Electronics. Andrew has submitted a complete description of the cassette deck interface and its operation which follows.

Thank you, Andrew and Jim, and once again..... thanks to Richard Leon for launching the concept.

Andrew Chiu
 39 Farmview Cr.,
 WILLOWDALE, Ontario
 M2J 1G5

HARDWARE FIX ADAPTER

The object of this design is to adapt Richard Lean's Hardware Fix (ref: TRANSACTOR BULLETIN #8) without dismantling your cassette and using Karl's "Interface" idea we can install a circuit between a 6 pin Molex Plug and the printed circuit board edge connector (see Fig.6).

A quick look at the table provided by Karl J. in the Transactor Bulletin #8 (table 1) indicates that when the cassette is doing a write operation the write-line is 'active' and for other operations it is 'high', therefore based on this unique active state a simple switching circuit can be controlled.

TABLE 1

| PIN on PCB Cardedge | OPERATION | | |
|---------------------------|-----------|--------|--------|
| | NONE | READ | WRITE |
| READ | LOW | ACTIVE | ACTIVE |
| WRITE | HIGH | HIGH | ACTIVE |
| SENSE | HIGH | LOW | LOW |
| MOTOR | OPEN | 6.5V | 6.5V |

HOW IT WORKS. Most PET lovers are the software type, in order to understand the theory behind this circuit, let us review some characteristics of the key element, a diode.

The ideal-diode approximation strips away all but the bones of diode operation. What does a diode do? It conducts well in the forward direction and poorly in the reverse direction. Boil this down to its essence, and this is what you get: ideally, a rectifier diode acts like a perfect conductor (zero voltage) when forward-biased and like a perfect insulator (zero current) when reverse-biased as shown in Fig.3.

In circuit terms, an ideal diode acts like an automatic switch. When conventional current tries to flow in the direction of the diode arrow, the switch is closed (see Fig.3b). If conventional current tries flowing the other way, the switch is open (Fig. 3c). We cannot simplify the idea of the diode beyond this point.

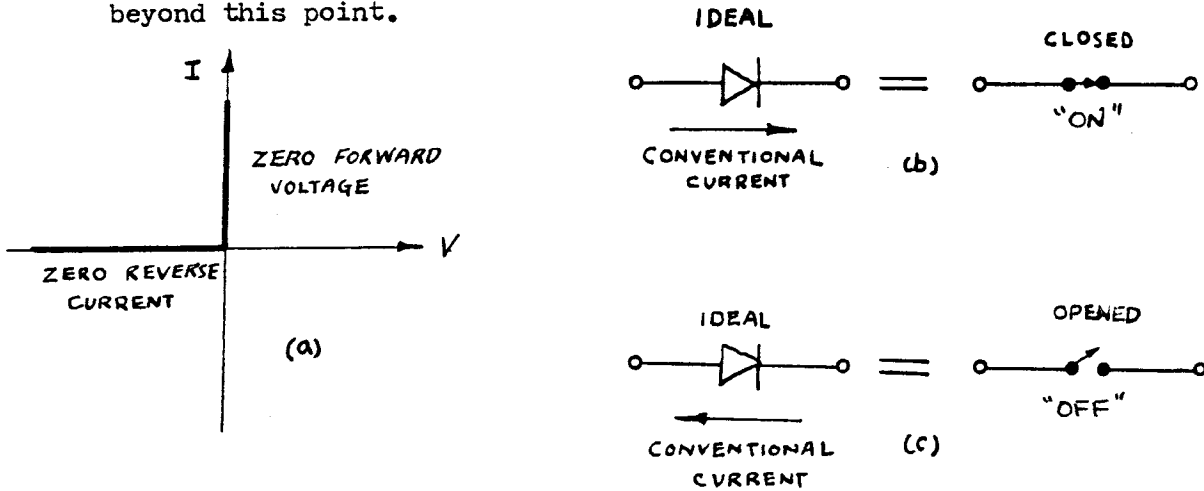


Fig 3. (a) Ideal-diode Curve. (b) Closed-switch analogy. (c) Open-switch analogy.

Once the operation of the diode is understood, then we can look at Fig.1. It is a complete diagram of the Hardware Fix Adapter. IC.1 is connected as a negative-recovery monostable circuit. As long as the write-line is active, the circuit stays triggered and the output remains High. (approximately equal supply voltage at no load). The current flows out from pin 3 and the capacitor C3 charges through the resistor R2 and diode D2. The capacitor is acting like a temporary quick charge battery. When the block is finished, the power to the motor is turned OFF and the write-line go High. The capacitor C3 will discharge through D3 to the cassette motor. The discharged energy provided power to the motor for a small period of time. This allows a larger gap between blocks.

Fig. 4 is a simplified circuit, only the diode and C3 is shown. The imaginary switches are the approximation of IC.1 and the transistor switch which is controlled by the internal operation of your PET. Fig. 4 a & b when the motor power switch is turned ON and write-line is active, D1 and D2 are both ON, but D3 is OFF. Therefore the current has to flow into the capacitor and the motor. Fig 4 c & d when motor power switch is OFF and write-line is High, D1 and D2 is Off, but D3 is ON. Therefore the current can discharge through D3 to the motor. The ideal diode model is a helpful tool to explain the operation of the circuit.

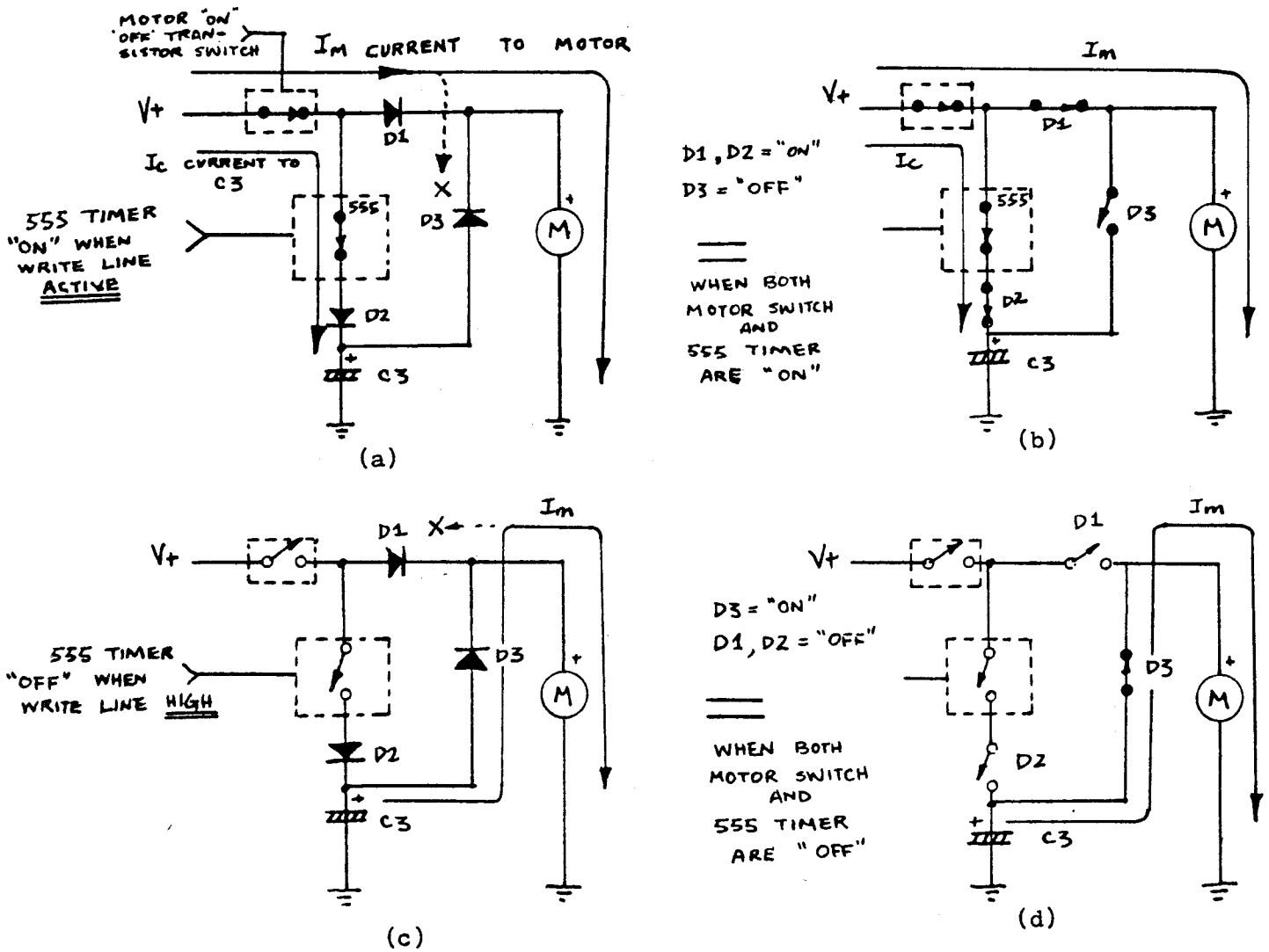


Fig. 4 Simplified circuit diagram of the Hardware Fix Adapter.

CONSTRUCTION. Wire-wrap, point-to-point or printed-circuit techniques can be employed in the construction of the Adapter. However a printed-circuit board definitely makes things easier once it has been made or obtained. I built my unit with all the components mounted on the board except the 20,000 μF capacitor. Watch the polarity of all the diodes and C3. The cathode (-Ve) end of the diode is marked with a Band or Bands. The polarity of the capacitor is marked on the label, some are colour coded. (red for positive). Transistor Q1 may be supplied in different shapes. To identify the transistor leads, please check the manufacturers data book. A component lay out diagram is shown in Fig. 2. Watch the spacing (0.156") of the card-edge connector. and the slot. The physical dimension must fit the 6 pin Molex Plug which the cassette uses.

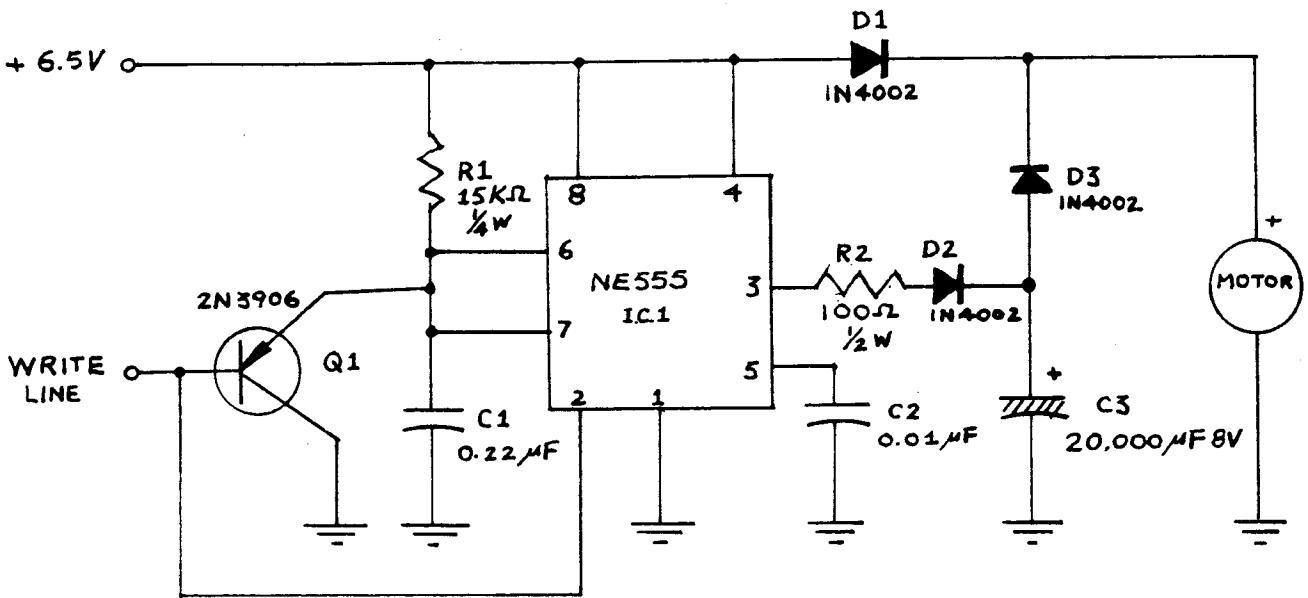


Fig 1. Circuit diagram of the Hardware Fix Adapter.

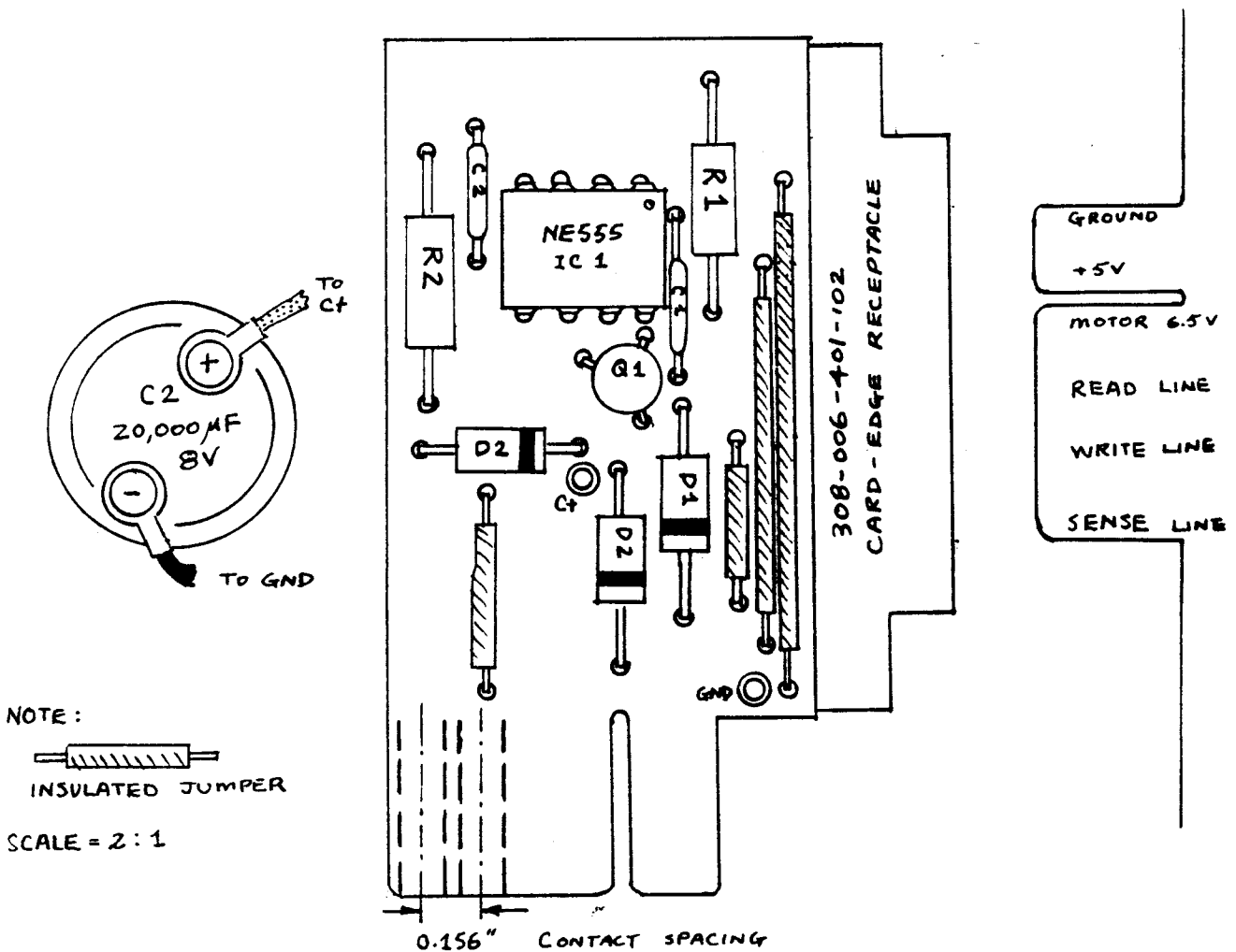


Fig 2. Parts placement, component-side view.

OPEN THE PET COMPUTER. (This is the procedure outlined in the service manual)

1. Press the rocker switch to the OFF position.
2. Remove the power cord from the wall socket to avoid possible electrical shock.
3. Remove the two screws located on each side of the unit under the lip of the cover.
4. Lift the cover slowly - a few inches. When you locate the cable leading to the cassette, remove the connector at the main board. Then lift the cover all the way up and engage the supporting rod located on the left side of the cover.

INSTALLATION PROCEDURE OF THE ADAPTER. (see Fig. 5,6 & 7)

1. Plug the 6 pin Molex Plug to the Adapter's card-edge connector.
2. Plug the Adapter to the main board's card-edge connector.
3. The big 20,000 uF capacitor can be mount in front of the main board by locking cable ties and adhesive backed mounts.

FINAL TESTING. Murphy's law states that "If anything can go wrong, IT WILL". So please check the polarity of all the components again before you touch the power switch. One thing will damage your PET for sure is a short circuit between ground and +5V line or +6.5V motor control line. When you think everything is OK. Turn On your PET, and use a voltmeter to measure the voltage across C3, it should give you a very small reading (almost zero). If the voltage is right, type in the testing program. After you finished, SAVE the program, while your PET is saving the program, monitor the voltage across C3 again, it should rise slowly to approximately to 4.5V and stop there. When finished saving, the voltage will drop to zero again (0.6V). Next, put a new tape in the cassette and RUN the test program. The test program will check and print the number of drop out errors found - you should have none.

PARTS AVAILABILITY. Essential parts, including 6 pin card-edge receptacle (# K60013PCSCGD6), Universal PC Board and 20,000 uF are available from ARKON ELECTRONICS LTD., 91 Queen Street E., TORONTO, Ontario, M5C 1S1. A assembled and tested unit for \$26.00 is available from RAPID ELECTRONIC, P.O. Box 1031, Station 'B', WILLOWDALE, Ontario, M2K 2T6.

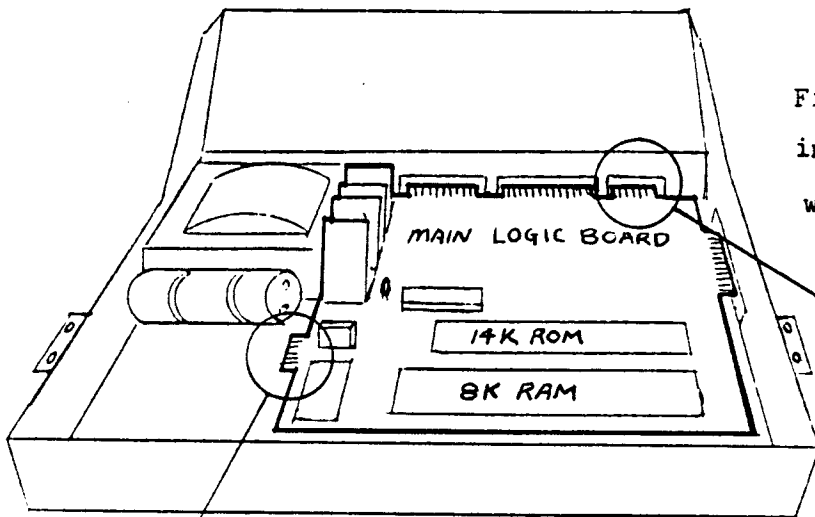


Fig 5. Simplified inside view of the PET with the cover removed.

Card-edge connector for cassette No.2 6 pin Molex Plug.

Card-edge connector for cassette No.1 6 pin Molex Plug.

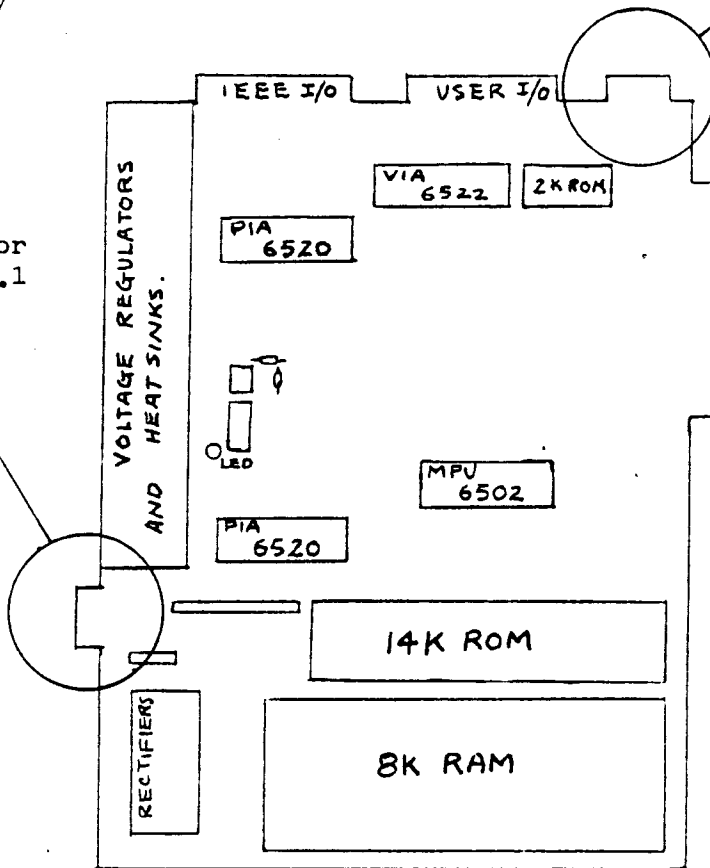
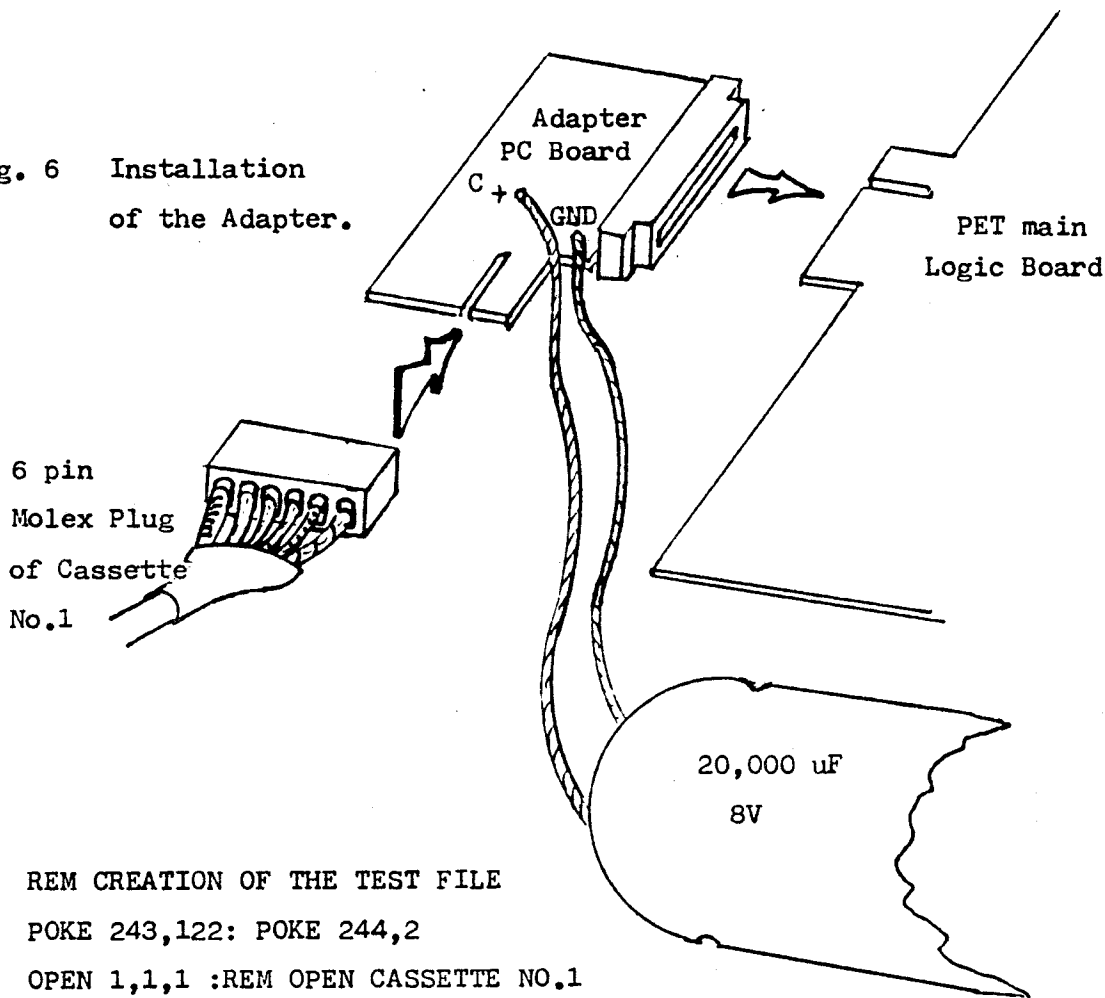


Fig 7. Simplified view of the main logic board.

Fig. 6 Installation of the Adapter.



```

1  REM CREATION OF THE TEST FILE
10 POKE 243,122: POKE 244,2
20 OPEN 1,1,1 :REM OPEN CASSETTE NO.1
30 FOR I=1 TO 300
40 PRINT I
50 PRINT#1, I
60 NEXT I
70 PRINT#1: CLOSE1
80 STOP

```

NOTE: For Cassette NO.2 the following lines have to be changed to :-
 10 POKE 243,58:POKE 244,3
 20 OPEN 1,2,1
 105 OPEN 1,2

```

100 REM RECOVER AND TEST THE TEST FILE
105 OPEN 1
110 FOR I=1 TO 300
120 INPUT#1,A
130 IF (ST) AND 64 THEN GOTO 170
140 PRINT A;
150 IF A B+1 THEN PRINT "ERROR": J=J+1
160 B=A: NEXT I
170 CLOSE 1: PRINT J;"ERRORS FOUND"

```

INTERUPTS ON THE

COMMODORE PET

(c) 1979 Brad Templeton

One of the most important features of the COMMODORE PET operating system is the use of interrupts. They are used to reset the PET, and they handle most of the tape and all of the keyboard i/o. This article will provide an introduction to interrupts on the 6502 (The PET's cpu) and a description of how the PET handles them. For your information, pseudo source listing is provided for the interrupt software of the PET, as produced by my disassembler.

Under normal conditions, a processor executes machine code in a linear fashion. It moves through memory, obtaining instructions (which can be one, two or three bytes long) and executing them. Sometimes, certain programmed instructions cause jumps to other places, just like GOTO and GOSUB of basic. To make a machine more flexible, however, interrupts are provided to do jobs that would be very expensive to do in software.

Essentially, an interrupt is controlled by a line right into the processor. When the processor detects the correct voltage on this line, an interrupt may be generated. First, in order to simplify matters, the processor finishes the instruction it is presently carrying out. Then, if the in-

interrupt is ok (interrupts can be masked), The processor saves the program location it was at, and the contents of its flags onto the stack. It then goes to a special reserved area of memory (in ROM on the PET) and pulls out two bytes indicating what location it should start executing from. It then goes there and executes machine code until the instruction RTI (Return from Interrupt \$40) is encountered. It then goes back to the stack and restores its flags, and loads the location it saved to the instruction counter. It then goes and executes the code after where it stopped as though nothing had occurred. (If the interrupt program was correctly written)

On the 6502, three types of hardware interrupts can occur, as well as a fourth special type. The locations they branch to are kept in byte pairs called vectors at the end of memory. One of these interrupts, NMI or Non Maskable Interrupt, can not be used on the PET. Its vector, \$FFFA-B, points to \$CA60, which is the middle of a subroutine. The line for this is also fixed off by a resistor on the pc board. Later PETs may plan to include this.

The interrupt called for power up is named RES. It branches to a routine which sets up basic and the operating system. It also, through what I consider to be one of the PET's worst design flaws, branches to the routine to destructively test how much memory is in the machine. At the very start, it also tests the condition of the diag-

nostic sense (MSB of \$E810), and goes to the diagnostic routine if this is set. RES is fired by power up, or by grounding pin 27 on the bottom of your memory expansion bus. If you set it by touching that pin, it does not clear memory below \$400, so programs there (the tape buffers) are safe. This is, unfortunately, a very small area. It vectors through \$FFFC-D.

The general use, hardware interrupt is the IRQ. IRQ vectors through \$FFFE-F, as does BRK. This points to location \$E66B in the PET. It is generated every 60th of a second by the tv hardware, and can also be generated from the memory expansion bus, on pin 28. It is also connected to the 6522 versatile interface adaptor. I will discuss the 60 per second interrupts here in detail. For information of generation by the 6522 (there is another whole article's worth of material in there) you can write MOS^① for the manual on it. Interrupts can be generated from it at exactly timed intervals, and by certain i/o conditions on the user port and IEEE bus. The exactly timed intervals are used to send precise frequency signals to the tape. (In fact, the 6522 is the PET's tape interface!)

The 60 per second interrupts do the following:

- Scan the keyboard, checking for new keys and decoding them.
- Increment the real time clock, and check for midnight
- Flash the cursor if it is on. (\$0224 > 0)
- Test tape recorder status for stop-start

Copy a byte for the break key test.

Whatever else you want them to do.

When the IRQ occurs, the code at \$E66B (see source) saves the processor register A, X and Y on the stack. It then checks, by loading back from the stack, the flags, to see if the BRK flag was set. The BRK, a software IRQ, vectors through the same place, but sets the BRK flag. This is handy to test what type of interrupt occurred. It then does a jump indirect to one of two places in RAM (\$219 or \$21R) depending on the type of interrupt.

Normally, the RAM IRQ vector is set to \$E685, which is the standard IRQ code. BRK has no default setting. The small piece of code you see after the JMP indirects is the return code, which restores the registers and does the RTI. The first thing INT_CODE does is the JSR INCR_CLOCK which increments the clock and copies the PIA register the break key test uses. When Steve Punter of Mississauga saw this with the disassembler, he devised an ingenious way to disable the BREAK key of the PET. By telling the PET to branch to \$E688 instead of \$E685 by means of a POKE 537, 136 statement, the PET bypasses the INCR_CLOCK subroutine, and does not test the break key. (Note INCR_CLOCK passes through a JMP vector table in high ROM at \$FFEA) This has the side effect of turning off the real time clock. When this statement is not used the clock proceeds normally. After it is updated, it is compared with a three byte table that con-

tains the value for midnight. If it is midnight on the clock, it is zeroed. The PET also keeps a secondary clock just after the main one. This is used for calibrating the real time clock. About every 6 seconds, this clock reaches a special limit, and when it does, it is zeroed, and the main clock is not incremented on this cycle. This is because the interrupt generator runs slightly faster than exactly 60 times per second. Even with this compensation, you may have noticed the clock is a few seconds off after several hours of PET operation. If they had used the 60 hz ac power line for the interrupt, it would have been more accurate, but that would have caused problems for PETs sold abroad.

After doing the clock, it proceeds to flash the cursor, once every third of a second, if the location FLASHING (\$224) is set to zero. (POKE 548,0 in a program turns the cursor on, but with some bugs - try it and see.) It does it with a very silly method that has no apparent purpose, instead of the standard method, a EOR \$80. It then sets up two keyboard test locations.

In using your PET, you may have noticed that if the tape drive is stopped by the machine itself, that you can push stop and play and the motor will run again. This is handled by the section of code at \$E6CD. After this comes the keyboard interpretation routines. The method of decoding the keyboard PIA has already been published in your

PET manual, and in PET user notes, so I will not dwell on it here. Once it has the matrix coordinate of the key, it waits for it to stabilize, to avoid bounce and repeating letters. (The TRS-80 does this poorly). It then converts the matrix number to an ascii character through the table at \$E75C. (You can use this table in your programs, if you want to account for how long a key is held down - a great real time feature!) It then puts the key in the correct place in the keyboard buffer starting at \$20F. Finally it goes back.

WHAT YOU CAN DO

Because the PET IRQ goes through RAM, it is one of the main links you have that can give you operating system control. You can insert your own programs before and after the interrupt code to have your PET do two jobs at once, like handle i/o while running basic. I have used interrupts to write programs to:

Interpret the PET keyboard and the full sized keyboard I attached to the PET like a regular keyboard.

Provide functions like repeat after a certain period of time and shift lock.

Turn the ! key to a statement number key, so that it would provide a line number 10 higher with every push.

Have upper case letter keys print out as full basic keywords.

Display whole pages of PET memory constantly on the screen.

Provide a non-destructive reset that works in special cases.

Much more is possible.

To use your own programs, you merely set them up in some convenient location (machine code only), preferably starting at location that ends in \$85, such as \$385 in the second tape buffer. Something located there can then be started with a POKE 538,3 and stopped with POKE 538,230, rather than having to write a special machine language program that disables the interrupt with SEI, changes the locations, and enables the interrupt with CLI. You do not

need to disable if you are only changing one byte of the location. Put some code there and follow it with a JMP \$E685. This way it does your code and proceeds on to do its own. If you put in the following series:

```
EE 50 80 40 85 E6
```

starting at \$385 (901 base 10), and initiate it with POKE 538,3 you will see a byte on the screen constantly increasing in "value", once every 60th of a second. The PET will also be doing everything else as usual. The following code:

```
A2 00 80 00 00 90 50 80 E8 D0 F7 40 85 E6
```

will dump a page of memory on the screen constantly. You can poke 905 with the page you wish to examine. Try 0,1,2,4,31,232. It starts with page 0. When scanning page 0, move the cursor and see what happens.

While doing this, you may have noticed that there is no flicker whatsoever on the screen despite the massive amount of writing to it being done. (Far faster than BASIC printing). This is because the interrupt is fired by the screen scan signal, and the screen is doing nothing shortly after the interrupt goes. This is also why the flashing cursor will never "snow" the screen. You can store almost half a screen without "snow" this way.

Sometimes it is important to put code in after the interrupt code of the PET. This can be done by manipulation of the stack, and is necessary for programs like the statement

numberer or keyword printer I included in my list above. I have included some code you can put in to allow you to do this. >PRNG means the high order byte of where your post interrupt code starts and PRGG is the low order byte. PCLO and PCHI are two locations for storing the correct pc you can use. The program works by altering the stack, so that the PET goes to your program when it RTIs. The second part of the program, which finishes your routine off (GOBACK) resets the stack and restores the proper program counter and machine registers. You should be able to have a lot of fun with it.

It should be noted that probably the only reason the IRQ vector is in RAM is that the PET does change it for tape i/o routines. There is a table of possible vectors starting at \$FD2A in the rom, and the table ends with the standard vector \$E685. If you ever change the high order byte of the IRQ RAM vector, you must reset it before tape i/o is done. If you don't, the PET will reset it anyway, but the tape i/o may not be done, and you may crash your PET.

Incidentally, the disassembler was written in the system language B (a very nice, much improved BCPL) here at the University of Waterloo where I go to school and work for the Mathematics Faculty Computing Facility. This article was also prepared and formatted on the same Honeywell 66/60. Many of the labels used in the disassembly were provided through the massive effort of examining the PETs ROMs done

by Jim Butterfield of Toronto. My next article for the Transactor will be on programming interactive games for the PET.

- ① The 6522 Data Sheets (24 pgs.) and other MOS publications are available through dealers.

Here is the code for the interrupts on the PET

```

E668 48          INTERRUPT  PHA
E66C 8A          TXA
E66D 48          PHA
E66E 98          TYA
E66F 48          PHA
E670 8A          TSX
E671 8D 04 01    LDA  $104,X
E674 29 10       AND  #$10
E676 F0 03       BEQ  $E678
E678 6C 18 02    JMP  [BRK_LOW]
E67B 6C 19 02    JMP  [IRQ_LOW]
E67E 68          RETURN_INT PLA
E67F A8          TAY
E680 68          PLA
E681 AA          TAX
E682 68          PLA
E683 40          RTI

E684 60          RTS

E685 20 EA FF    INT_CODE  JSP  INCR_CLOCK
E688 AD 24 02    LDA  FLASHING
E68B D0 23       RNE  $E68D
E68D CE 25 02    DEC  C_TIMER
E690 D0 1F       RNE  $E692
E692 A9 14       LDA  #$14
E694 8D 25 02    STA  C_TIMER
E697 A4 F2       LDY  C_COLUMN
E699 4E 27 02    LSR  C_STATE
E69C B1 F0       LDA  (C_ROWADR),Y
E69E 80 06       BCS  $E6A6
E6A0 FF 27 02    INC  C_STATE
E6A3 8D 26 02    STA  CHAR_UND_C
E6A6 0A          ASL
E6A7 80 03       BCS  $E6AC
E6A9 38          SEC
E6AA 80 01       BCS  $E6AD
E6AC 18          CLC
E6AD 6A          ROR
E6AE 91 E0       STA  (C_ROWADR),Y
E6B0 A2 FF       LDY  #$FF
E6B2 8E 23 02    STY  KEY_IMAGE
E6B5 F8          INY
E6B6 8E 04 02    STY  SHIFT_FL
E6B9 A2 50       LDY  #$50
E6BB AD 10 E8    LDA  PIA1
E6BE 29 F0       AND  #$F0

```

```

E6C0 8D 10 E8
E6C3 A0 00
E6C5 AD 10 E8
E6C8 0A
E6C9 0A
E6CA 0A
E6CB 10 07
E6CD 8C 07 07
E6D0 A9 3D
E6D2 00 07
E6D4 AD 07 07
E6D7 00 05
E6D9 A9 35
E6DB AD 13 E8
E6DE 90 0A
E6E0 8C 0A 02
E6E3 AD 40 E8
E6E6 09 10
E6E8 D0 0A
E6EA AD 0A 02
E6ED 00 0A
E6EF AD 40 E8
E6F2 29 EF
E6F4 AD 40 E8
E6F7 A0 0A
E6F9 AD 12 E8
E6FC CD 12 E8
E6FF 00 F6
E701 4A
E702 80 05
E704 48
E705 20 3F E7
E708 68
E709 CA
E70A FC 0A
E70C 88
E70D D0 F2
E70F FF 10 E8
E712 D0 E3
E714 AD 23 02
E717 CD 03 02
E71A F0 20
E71C 8D 03 02
E71F AA
E720 30 1A
E722 8D 5B E7
E725 4E 04 02
E728 90 02
E72A 09 80
E72C AF 0D 02
E72F 9D 0F 02
E732 F8
E733 E0 0A
E735 D0 02

```

```

STA PIA1
LDY #0
LDA PIA1
ASL
ASL
ASL
RPL $E6D4
STY C1_STAT
LDA #3D
BNF $E6DB
LDA C1_STAT
BNE $E6DE
LDA #35
STA PIA1_B4
BCC $E6EA
STY C2_STAT
LDA PORT_B
ORA #10
BNE $E6F4
LDA C2_STAT
ANE $E6F7
LDA PORT_B
AND #$EF
STA PORT_B
LDY #8
LDA KB_ROWIN
CMP KB_ROWIN
BNE $E6F7
LSR
BCS $E709
PHA
JSR DECODE_KRD
PLA
DEX
REQ $E714
DEY
BNE $E701
INC PIA1
RNE $E6F7
LDA KEY_IMAGE
CMP KEY_DOWN
BEQ $E73C
STA KEY_DOWN
TAX
BMI $E73C
LDA $E75B,X
LSR SHIFT_FL
BCC $E72C
ORA #80
LDX KEYCOUNT
STA KEY_BUFFER,X
INX
CPX #8A
BNE $E739

```

| | | | | | |
|------|----|----|----|------------|--------------|
| E737 | A2 | 00 | | LDX | #\$0 |
| E739 | 8E | 0D | 02 | STX | KEYCOUNT |
| E73C | 4C | 7E | E6 | JMP | RETURN_INT |
| E73F | 8D | 58 | E7 | DECODE_KRD | LDA \$E75B,X |
| E742 | 00 | 07 | | BNE | \$F748 |
| E744 | A9 | 01 | | LDA | #\$1 |
| E746 | 8D | 04 | 02 | STA | SHIFT_FL |
| E749 | 00 | 10 | | BNE | \$E75B |
| E74B | C9 | FF | | CMP | #\$FF |
| E74D | F0 | 0C | | BEQ | \$E75B |
| E74F | C9 | 3C | | CMP | #\$3C |
| E751 | 00 | 05 | | BNE | \$E75B |
| E753 | 2C | 11 | E8 | BIT | PIA1 + 1 |
| E756 | 30 | 03 | | BMI | \$E75B |
| E758 | 8E | 23 | 02 | STX | KEY_IMAGE |
| E75B | 60 | | | RTS | |

| | | | | | | |
|------|----|----|----|------------|-------------|---------|
| F736 | AD | 05 | 02 | UPDATE_CLK | LDA | CLOCK_2 |
| F739 | 69 | 01 | | ADC | #\$1 | |
| F73B | 8D | 05 | 02 | STA | CLOCK_2 | |
| F73E | 90 | 03 | | BCC | \$F743 | |
| F740 | EE | 06 | 02 | INC | CLOCK_2 + 1 | |
| F743 | C9 | 6F | | CMP | #\$6F | |
| F745 | 00 | 07 | | BNE | \$F74E | |
| F747 | AD | 06 | 02 | LDA | CLOCK_2 + 1 | |
| F74A | C9 | 02 | | CMP | #\$2 | |
| F74C | F0 | 26 | | BEQ | \$F774 | |
| F74E | FE | 02 | 02 | INC | M_CLOCK + 2 | |
| F751 | 00 | 0A | | BNE | \$F75B | |
| F753 | FE | 01 | 02 | INC | M_CLOCK + 1 | |
| F756 | 00 | 03 | | BNE | \$F75B | |
| F758 | FE | 00 | 02 | INC | M_CLOCK | |
| F75B | A2 | 00 | | LDX | #\$0 | |
| F75D | 8D | 00 | 02 | LDA | M_CLOCK,X | |
| F760 | DD | 88 | F7 | CMP | \$F788,X | |
| F763 | 90 | 17 | | BCC | \$F77C | |
| F765 | E8 | | | INX | | |
| F766 | F0 | 03 | | CPX | #\$3 | |
| F768 | 00 | F3 | | BNE | \$F75D | |
| F76A | A9 | 00 | | LDA | #\$0 | |
| F76C | 9D | FF | 01 | STA | \$1FF,X | |
| F76F | CA | | | DEX | | |
| F770 | 00 | FA | | BNE | \$F76C | |
| F772 | F0 | 08 | | BEQ | \$F77C | |
| F774 | A9 | 00 | | LDA | #\$0 | |
| F776 | 8D | 05 | 02 | STA | CLOCK_2 | |
| F779 | 8D | 06 | 02 | STA | CLOCK_2 + 1 | |
| F77C | AD | 12 | E8 | LDA | KB_ROWIN | |
| F77F | CD | 12 | E8 | CMP | KB_ROWIN | |
| F782 | 00 | F8 | | BNE | \$F77C | |
| F784 | 8D | 09 | 02 | STA | PIA_COPY | |
| F787 | 60 | | | RTS | | |

Here is the source for the post interrupt code program

```
START      LDA    $105,X          GET
           STA    PCL0          PROGRAM
           LDA    $106,X          COUNTER AND
           STA    PCHI          STORE IT
           LDA    PROG           PUT IN YOUR
           STA    $105,X         OWN CODE
           LDA    >PR0G          LOCATION
           STA    $106,X
           JMP    $E685          DO NORMAL INTERRUPT
GOBACK     REM    THIS CODE GOES AFTER YOUR CODE, TO RETURN
           LDA    PCHI          RESTORE
           PHA    DLD
           LDA    PCL0          LOCATION
           PHA
           TSX
           DEX
           DEX
           DEX
           DEX
           TXS
           JMP    $E65F          DO RTI
```

KVENICH & ASSOCIATES

International Trade Brokers

51 Carlingview Drive Unit 5 Rexdale, Ontario M9W 5E7 . Phone 675-7333
Telex 06-989100

ATTENTION: NEW PRODUCTS

KVENICH & ASSOCIATES HAVE BEEN ASKED TO DISTRIBUTE THE WAVECOM INTERFACE. DESIGNED BY THE MICRO-SYSTEMS ENGINEERING GROUP, THIS NEW PRODUCT WILL INTERFACE THE PET COMPUTER AND THE I.B.M. SELECTRIC TYPEWRITER FOR COMPREHENSIVE WORD PROCESSING.

THE WAVECOM INTERFACE IS A STAND ALONE DEVICE WHICH CONTAINS ITS OWN PROCESSOR AND ROM MEMORY. NO PROGRAMMING IS REQUIRED TO RESIDE IN THE PET'S 2nd CASSETTE BUFFER AS IS THE CASE WITH MANY INTERFACES PRESENTLY ON THE MARKET. THIS FEATURE ALLOWS THE USER THE 2nd CASSETTE FOR BUSINESS FILES OR ACCOUNTING PROGRAMS.

FEATURES OF THE WAVECOM INTERFACE

1. A stand alone interface.
 2. Plugs into the PET COMPUTER via the IEEE port.
 3. All parts are included with the WAVECOM interface (including plugs, and wire connectors).
 4. Plugs into the SELECTRIC typewriter.
 5. Installation of solenoids and plugs are required for the SELECTRIC by the user or dealer all parts are included. are provided.
 6. Can be used with a SELECTRIC terminal. Communication is via a telephone handset through an acoustic coupler. No modifications are necessary.
 7. Will also operate with many other computers such as TRS-80, SOCERER, and APPLE.
 8. Unplug the SELECTRIC typewriter from the WAVECOM Interface and the typewriter will return to normal manual operation.
 9. Allows any computer to completely control all user controlled key including, the TAB FUNCTION and the BACKSPACE key.
-

NEWS RELEASE



HOME COMPUTER CENTRE

Computers for Home & Small Business

(416) 222-1165

222-1166

6101 YONGE STREET, WILLOWDALE, ONTARIO M2M 3W2, CANADA

HOME COMPUTER CENTRE ANNOUNCES

THE NEW RELEASE OF PET SOFTWARE

The following programs are now officially released with complete documentation.

1. ENTRY - Used as a general purpose data entry program for business applications with user definable entry format, the program may be used for a Mail List, Daily Journal, General Ledger, Record Keeping etc. It works with cassette printer, and other IEEE devices.
List Price \$24.95

2. PROCESS - General purpose data process program. It is designed for limited data processing power on the PET. Basic operation includes SORT, EDIT, DELETE, INSERT, and MACRO. The program is particularly useful for merging large amounts of data from different input sources.
List Price \$24.95

3. INVENTORY - Inventory control program on the PET Data includes, item #, description, quantity on hand, reorder limit and prices. It generates inventory report and low inventory report. Handle up to 60 items on the 8K PET. Data may be insert, delete, charge, on the memory instantly.
List price \$24.95

All the HCC officially released programs come with complete documentation. The programs are intended for practical business applications, and special techniques are used to insure easy operation and data reliability. Special features include interactive message, error-free operation, recoverable operator errors, general I/O etc. The released programs have been tested for an extended period of time.

COMPUTER COURSES FOR EVERYONE

Human Computing Resources Corporation presents an ongoing program of courses on computers.

The courses have been created in response to the growing need for an objective, non sales oriented, viewpoint on how to evaluate personal computers, microcomputers and minicomputers. They will be attractive to people from many walks of life -- business people, professionals, artists, engineers, enthusiastic new users. They will be doubly attractive to people who have researched the computer market and find they lack the expertise to choose one system over another.

Being offered in spring/summer 1979 are:

- Introduction to Computing and Personal Computers (bimonthly; 9 hours)
- How to Buy a Computer for Small Business (23 May and 18 July; one day)
- Introduction to Microprocessors (23 and 30 June; 14 and 15 August; 2 days)
- Introductory Programming in BASIC (monthly; 18 hours)
- Programming in PASCAL (bimonthly, beginning in June; 18 hours)

In the works are courses on word processing, computers in the law office, and computers in medicine and in the medical office (two courses).

Our instructors are skilled educators, business people and creative computer professionals. They have had broad experience with all types of computers and computer applications.

All courses are held at HCR's offices, 10 St Mary Street, Suite 401, Toronto (near Yonge and Bloor), or in downtown Toronto hotel suites. Courses are priced at from \$55 to \$115. Fees for all courses are income tax deductible.

For more information, mail in the form below, or call us at 922-1937.

Please send me information about these courses:

- | | |
|---|---|
| <input type="checkbox"/> Introduction to Computing and Personal Computers | <input type="checkbox"/> Introduction to Electronic Troubleshooting |
| <input type="checkbox"/> How to Buy a Computer for Small Business | <input type="checkbox"/> Introduction to Computer Graphics |
| <input type="checkbox"/> Introduction to Microprocessors | <input type="checkbox"/> Introduction to Word Processing |
| <input type="checkbox"/> Introductory Programming in BASIC | <input type="checkbox"/> Computers in the Legal Office |
| <input type="checkbox"/> Programming in PASCAL | <input type="checkbox"/> Computers in the Medical Office |
| | <input type="checkbox"/> Frontiers of Medical Computing |

NAME _____

COMPANY & TITLE _____

ADDRESS _____

PHONE _____

NAKCOMM SYSTEMS INC.

80 HALE ROAD, UNIT 7, BRAMPTON, ONT. L6W 3M1 • (416) 459 7616

Nakcomm Systems Inc. wishes to extend thanks to Commodore for the opportunity to offer you several new PET compatible items.

You may find these units an economical and easy way to expand the capabilities of your PET:

Mini Printer Model TC-100

40 Character Per Second

96 Character set

5 X 7 dot matrix

List Price \$499.95

Full Size Keyboard Model 74-KB

List Price \$199.95

32 K Byte Expansion Board

Model PNE-32 List Price \$912.95

SPECIAL NOTE: We also have available, an Interface Model PTP-10, allowing the PET to operate with any Centronics Printer. List Price \$69.95

Orders placed directly on Nakcomm Systems Inc. will be dealt with promptly.

Our Terms and Conditions are as follows:

Payment - Cash (cheque or money order) with order.

Warranty - 90 Days parts and labour.

Delivery - 1 to 3 weeks depending on item and stock.

After Warranty Service - Done on the Nakcomm premises.

If you require any additional information on the above described units or further explanation of our offer, contact us at any time.

Yours very truly,
Nakcomm Systems Inc.

Donald R. Young,
Marketing Co-Ordinator.

Dealer Inquiries Invited

Convert Upper to Lower Case

The following is a program that will convert all upper case text to lower case. However, keep in mind that any graphics above the alphabetic keys will now be unusable if they are to appear simultaneously with lower case letters.

```
59030 FOR T=1024 TO 8006-FRE(0):A=PEEK(T)
59031 ON Z GOTO 59034,59037
59032 IF A=153 OR A=178 THEN Z=1
59033 NEXT
59034 IF A=34 THEN Z=2:NEXT
59035 IF A=58 OR A=0 THEN Z=0
59036 NEXT
59037 IF A 64 AND A 91 THEN POKE T,A+128
59038 B=PEEK(T+1):IF B=34 OR B=0 THEN Z=0
59039 NEXT:END
```

When writing the program use no spaces. The program will convert strings and PRINT statements but will not affect DATA statements. Also, it may terminate with a '?NEXT WITHOUT FOR ERROR IN 59036' but that's OK.

Of course you need not use the same line numbers. They were chosen due to their unusualness. The program was then recorded using the UNLIST routine in Transactor #7. It can then be merged with other programs with a good chance of not interfering with other program lines.

.....
To receive Transactor Volume 2 bulletins, please return this form with your cheque for \$15.00 annually renewable, to CBM 3370 Pharmacy Avenue, Agincourt, Ontario, M1W 2K4. Volume 1 back issues will be available at 10 dollars for a limited time only (while supplies last).

VOL 2

NAME.....

COMPANY (if applicable).....

ADDRESS.....

.....POSTAL CODE.....

RECEIPT REQUIRED? YES NO
(Invoices cannot be issued for the \$15.00 annual fee)

IDEAS & COMMENTS.....

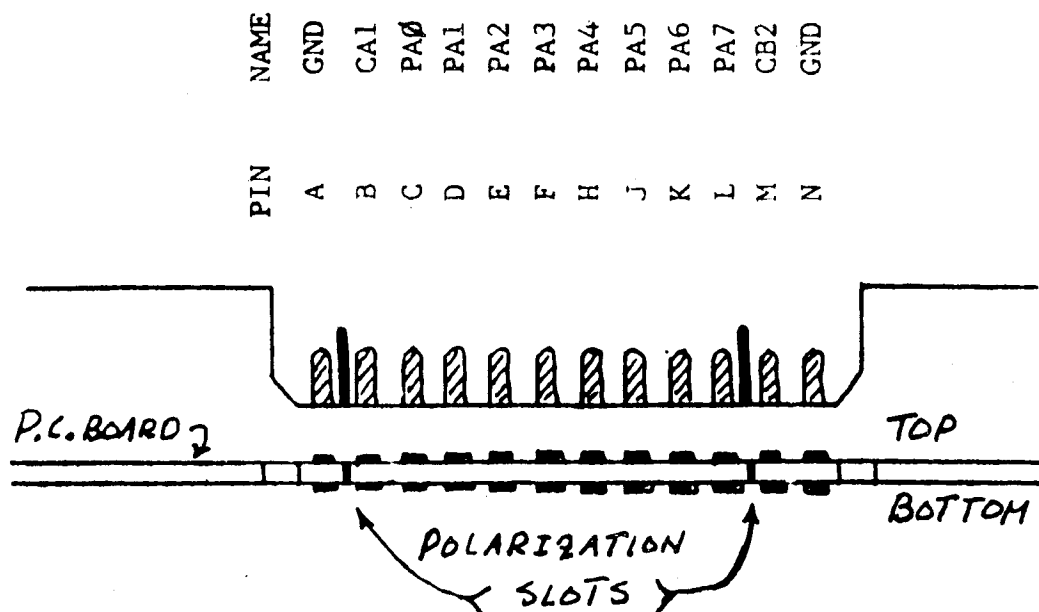
.....

THE USER PORT COOKBOOK

This bulleting describes how to use your PET's user port and how to interface it to real devices.

The pin-out of the user port is shown below:

(Viewed from the top)



The user port pins are on the bottom of the PET circuit board.

Page 22 of "An Introduction to Your New PET Personal Electronic Transactor" describes the manufacturer part # for several edge connectors which will fit the user port. If you cannot find a 12 position 24 contact connector, saw off a larger one to fit.

Note: Be sure that the upper and lower contacts in the connector are not electrically connected! Other signals reside on the top side which are not compatible with the user port.

The pin names correspond to the lines which connect to a MOS 6522 VIA, Versatile Interfree Adaptor.

The data sheet for this LSI chip is available from Commodore, 360 Euston Road, London NW1 3BL. (It is 24 pages long.) This bulletin is concerned with using the user port, and will not describe the 6522 in any more detail than is required.

Cont'd...

Pins CA1 and CB2 act as "handshake" lines. Pins PA0 through PA7 act as data lines. Electronically, these lines can drive one TTL load. If your cable is more than 24" long, you may have to buffer the lines.

A series of memory locations in the PET act as control and data registers for the 6522. These are accessed via PEEK and POKE in BASIC, and by the 6502 machine language instructions which read and write to memory.

The memory locations of use to us are:

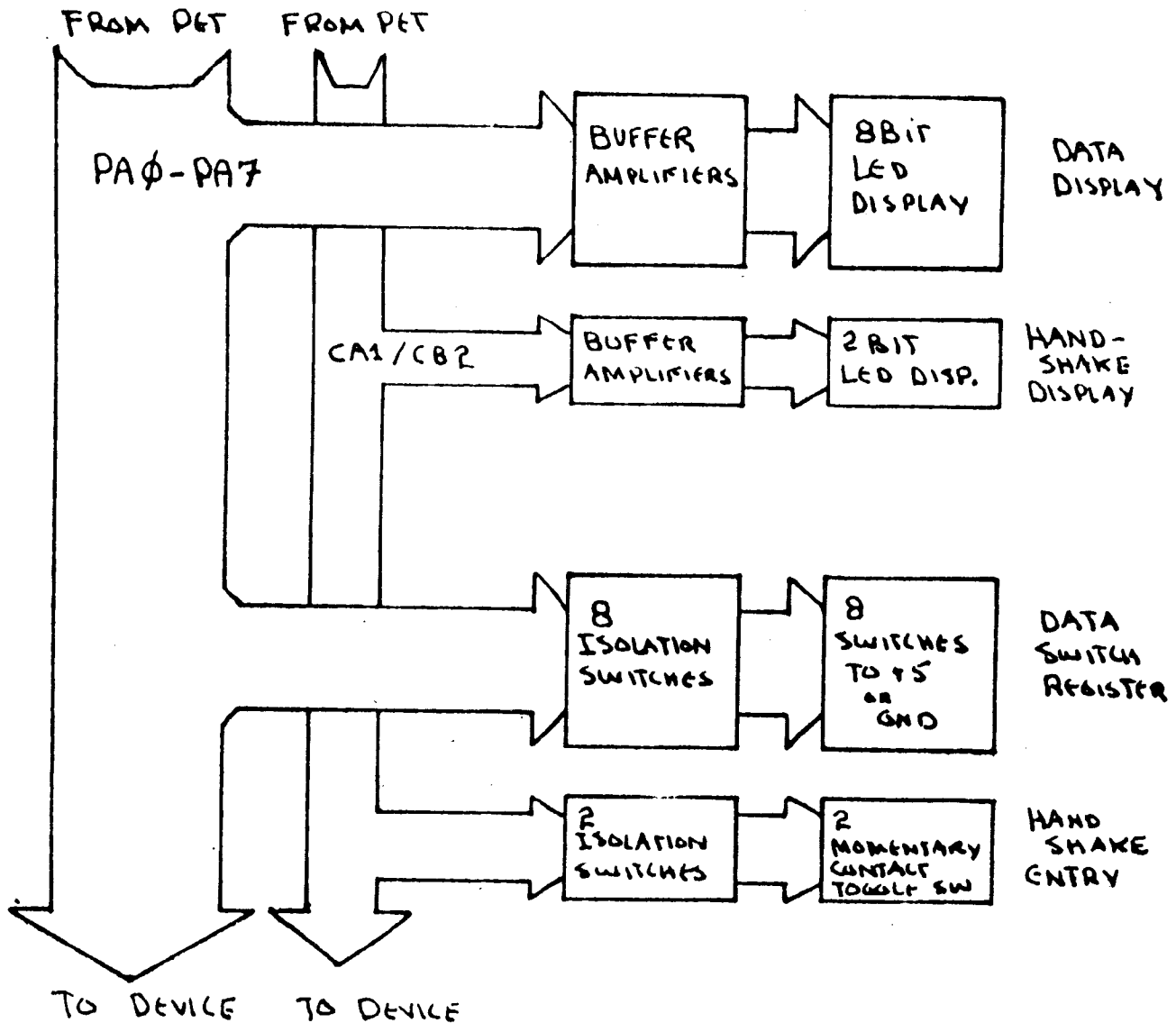
| <u>Name</u> | <u>Hexadecimal</u> | <u>Decimal</u> |
|-----------------------------|--------------------|----------------|
| Data Direction Register | E843 | 59459 |
| Data Register | E84F | 59471 |
| Data With Handshake | E841 | 59457 |
| Peripheral Control Register | E84C | 59468 |
| Auxillary Control Register | E84B | 59467 |
| Interrupt Flag Register | E84D | 1469 |

See the 6522 sepc. sheet for the exact definitions of these registers. The examples in this bulletin will cover most of your usual uses of the user port.

Cont'd...

THE "BLINKIN LIGHT" MACHINE

One way to get started with the user port is to build a device capable of showing the status of each line and to permit manual control of the lines. Here is a block diagram of a display/switch panel:

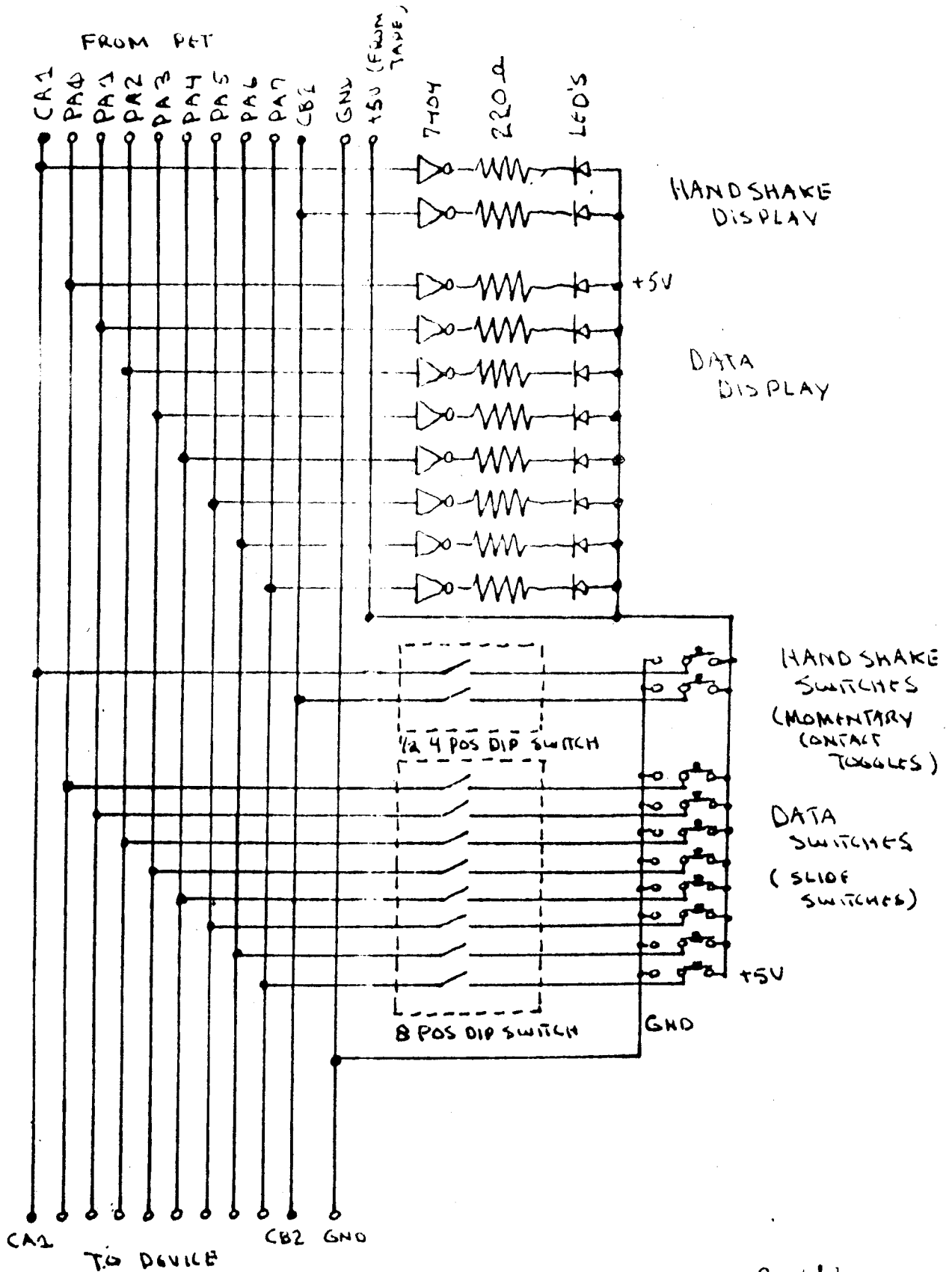


Some breadboard and about \$20.00 of parts (at very retail prices) resulted in the following circuit. 7404 inverters serve as the buffer amplifiers, the +5 is taken from the cassette #2 supply, dip switches are used for isolation, slide switches for the register, and some toggle switches for the handshakes.

Note: This circuit draws \approx 200mA which is close to the maximum available from the PET.

Cont'd...

Parallel User Port Indicator and Switch Register



Cont'd...

The following examples assume you have an equivalent device to attach to your user port. Be sure that (i) the device works correctly, and (ii) the connector is correctly plugged in. Sometimes the PET PCB has its pins offset slightly, making it possible to insert the connector so that one pin on the connector touches two on the PET, etc. The best solution is to use the polarizer slots.

1. Simple Output

Enter and run: (Be sure your isolation switches are all open)

```
10      REM SIMPLE OUTPUT
20      REM DATA DIRECTION = ALL OUTPUT
30      POKE 59459,255
40      FOR J= 0 to 255
50      POKE 59471, J
60      POKE = 1 to 100 : NEXT K
70      NEXT J
80      GO TO 40
```

You should see your data LEDs count in binary from 00 to FF (0-255) with PA0, the LSB, blinking at about 3HZ.

Line 30 sets the data direction register to all '1', which sets all data lines for output. Each bit in the D.D.R. corresponds to a given PA0-7 line. The PA line is:

Input if bit is zero
Output if bit is one

To see the effect of this, change line 30 to:

```
30 POKE 59459,15
```

Now, only the 4LSB (PA0-PA3) will blink. Lines PA4-PA7 will be lit with no change. (Note: The TTL input of the 7404 will force the input lines high. Therefore the 4 MSB will indicate A '1' state. If your display circuit is different the state of the MSB might be zero. In any case, they won't change.)

Try other masks to see other patterns.

Cont'd...

Line 50 POKES the data register with J (0-255) inside the For-Next loop in lines 40-70.

Line 60 is a delay. Try removing it and you will notice that the PA0 line will blink too fast for you to see. (Vut PA1 will flicker).

Line 80 starts the counting loop over again.

You can write other programs to make moving patterns, etc.

II. Simple Input

Enter and run:

```
10     REM SIMPLE INPUT           [v] Clear/home
20     REM D.D.R. = INPUT         [s] Home
30     POKE 59459, 0              [l] Cursor left
40     PRINT "[v] ";
50     PRINT "[s] " PEEK (59471) "[l] ";
60     GO TO 50
```

Connect your data switch register by closing the 8 data isolation switches. When these switches are closed, the switch register forces PA0-PA7 to the value selected by the switches.

A number will appear in the upper left corner of your PET's screen. Set your switches to all zero, and then set bit 0 to 1. A '1' will now appear on the PET, and on your date display.

Try one switch at a time to get: 1-2-4-8-16-32-64-128 and then try other combinations.

Notice that if you open an isolation switch for a given bit, it will become a '1' due to the 7404's.

Line 30 sets the D.D.R. to all zeros, naming PA0-PA7 inputs.

Line 40 clears the screen.

Line 50 homes the cursor

prints the data register's value

prints the CURSOR LEFT and 3 blanks.

Line 60 loops back to line 50.

Note: The CURSOR LEFT is required because numbers are printed on the PET in the form:

```
      123.45
     ┌───┴───┐
    BLANK  DIGITS  CURSOR
    OR "-"  RIGHT
```

Cont'd...

This 'trick' removes left over digits from the previous number - suppose you had set bits 7 and 3 (giving 136) and then you reset bit 7 (leaving 8). If you don't remove the characters, you will see 836!!

III. The Handshake Lines

The 6522 provides several options for the CA1 and CB2 lines. See the 6522 specification for full details.

Note: Several of the addresses mentioned below control other aspects of the 6522. If you can bits other than those mentioned, you may have an inoperable PET, as your PET uses the 6522 for internal uses as well. (The 6522 has CA2, CB1, PBO-7 lines which the PET uses for other I/O functions than the user port.) You are warned! (I wasn't able to SAVE a program until I had reset two of the registers which had been POKEd erroneously!!)

For our purposes, these registers control the CA1 and CB2 lines:

| | |
|-----------------------------|--------|
| Data Register | DATA |
| Data with Handshake | HDATA |
| Peripheral Control Register | PCREG |
| Auxillary Control Register | AUXKEY |
| Interrupt Flag Register | IFREG |

The acronyms in the right column above will be used from here.

CA1 CA1 is an input only line which can detect a "Data Ready" transition. When it does so, bit 1 of the interrupt flag register is set. *(Our convention is MSB is bit 7, LSB is bit 0)*

When the HDATA Register is read or written, the bit in the IFREG will be reset. Accessing the data register has no effect on the IFREG.

Detecting the flag bit is done by:

```
IF PEEK (59469) AND 2 THEN (Line #) or:
WAIT 59469, 2
```

Cont'd...

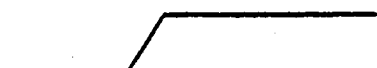
Bit 0 in the PCREG controls whether CA1 sets the flag in the IFREG. If this bit is zero, a negative transition sets the flag bit. If it is one, a positive transition will be detected

Negative Transition



POKE 59468, PEEK (59468) AND 254

Positive Transition



POKE 59468, PEEK (59468) OR 1

Use the expressions above to choose the transition you want.

The flag bit will remain set until the HDATA register is read or written. Bit 0 of the AUXKEY controls whether the data is latched when the flag bit is set.

| | | | |
|--------|--------------|-----------------|---|
| AUXKEY | <u>Bit 0</u> | 0 = No latching | The value of the DATA and HDATA registers follow the PA0-PA7 lines (those set for Input) regardless of the CA1 flag bit in the IFREG. |
| | <u>Bit 0</u> | 1 = Latching. | When the CA1 flag bit is set, DATA and HDATA will be latched. Their value remains the same, even though PA 0-PA7 may change. |

CB2 Using CB2 is more complex than CA1. The 6522 specification should be consulted for the more exotic ways of using CB2.

CB2 can be used as:

- A. Handshake output
- B. Handshake input
- C. Shift register I/O.

Bits 2, 3, and 4 of the AUXREG control whether handshake or shift register mode is to be used. If the bits are all zero, CB2 is in handshake mode. If any bit is not zero, CB2 is in a shift register mode.

Cont'd...

HANDSHAKE MODES

Output Mode

First you must set the AUXREG to disable the shift register. This is done with POKE 59467, PEEK (59467) AND 227.

Then you can force CB2 low with POKE 59468, PEEK ((59468) AND 31) OR 192 and you can force CB2 high with:

POKE 59468, PEEK (59468) OR 224.

Here is an example program which blinks CB2 at about 1 HZ:

```
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: NEXTJ
70 GO TO 30
```

Input Mode

Note: This section has not worked in practice. Toggling CB2 does not set the flag bit.

CB2 will set bit 3 in the IFREG if a transition occurs and the PCREG is set correctly.

First, set the AUXREG bits 2, 3 and 4 to zero,

POKE 59467, PEEK (59467) AND 227

If detection of a negative transition is wanted,

POKE 59468, PEEK (59468) AND 31

If a positive transition is wanted,

POKE 59468, PEEK (59468) AND 31 OR 64

Then, to detect a transition, check bit 3 of IFREG:

IF PEEK (59469) AND 16 THEN (line #)
or, WAIT 59469, 16

Cont'd...

To reset the flag bit, the B port register must be read.

```
X = PEEK (59456)
```

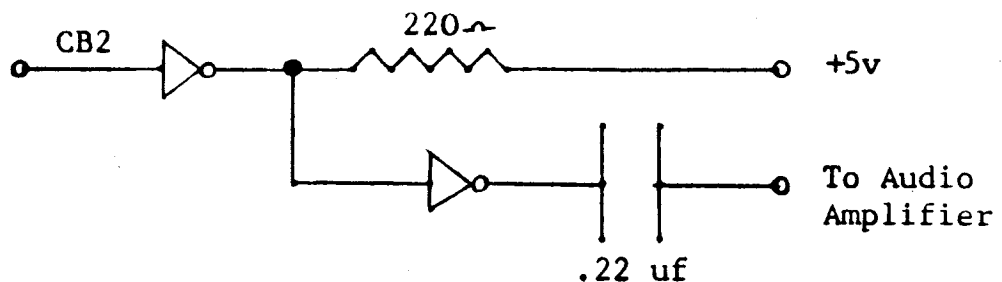
((Don't POKE this address!!))

SHIFT REGISTER MODES

If bits 2, 3 and 4 in the AUXREG are not zero, CB2 acts as a shift register. Set the 6522 specification for details.

Only one of these MODES can be conveniently handled in BASIC. The others require machine code to operate correctly.

The "Blinken Lites" can be given an audio capability with the following change:



Just add an extra inverter (two remain for use) and a capacitor to the CB2 Led driver's output.

Note: It is advisable to add an electrolytic capacitor, say 100 mfd to the Blinken Lite so that a sudden drain of power won't reset the PET.

Check the audio extension by toggling the CB2 line in output handshake mode:

```
10     POKE 59467, PEEK (59467) AND 227
20     A = 59468:X = PEEK (A) AND 131 OR 192
30     Y = PEEK (A) OR 224
50     POKE A, X: POKE A, Y: GO TO 50
```

LINE 10 sets up AUXREG to disable shift registers,
LINE 50 turns CB2 on and off.

Cont'd...

The reason for making the variables A, X and Y is because BASIC references variables much faster than converting constants. This maximizes BASIC's speed.

The PET keyboard can change the tones by using these changes:

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

Pressing different keys will give different rates of clicking. You now have a low fidelity sound-maker.

FREE RUNNING MODE

When AUXREG bits 4-2 are "100", the shift register cyclically outputs its contents on CB2 at a rate determined by the Timer 2. The addresses are:

```
SHIFT REGISTER:      59466
TIME 2              59464
```

Time 2 decrements to zero and then shifts the shift register once. Timer 2 is reloaded and this goes on. The output bit of the shift register is put in bit 7, causing the register to "rotate right".

Here is a simple "Music Maker" program:

```
10      REM MUSIC MAKER
20      POKE 59467, PEEK (59467) AND 227 OR 16
30      PRINT "TONE COLOR";
40      INPUT TC
50      IF TC < 1 OR TC > 254 THEN 30
60      POKE 59466, TC
70      PRINT "PRESS KEYS FOR TONES"
80      GET A $: IF A $ = " " THEN 80
90      POKE 59464, ASC (A $)
100     GO TO 80
```

It is straightforward to use the letters to make a true "keyboard" - choose notes for each key and make a table which is indexed by the ASCII value of the key. This is left as an exercise. (With only 256 possible frequencies, the options are somewhat limited.)

Cont'd...

IV. Some Interfacing examples

The following program lets you monitor the user part and modify any registers you wish as required. Be sure to save it on tape BEFORE running it. (Also, be sure you have not run any other programs first, i.e. turn the power on/off to initialise properly!!)

Once you have it saved, RUN it. Its operation is quite simple. The registers are named according to the 6522 specification with one exception, "DATA" is ORA without handshake. Use the "Blinken Lights" and this program to see how all parts except the shift register works.

```
10 REM PET 6522 VIEWING PROGRAM
20 REM BY: GREGORY YOB, COMMODORE
30 REM
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 "TILC-L", "TIC-H", "T1L-L", "T1L-11"
90 DATA "T2LC-L", "T2C-11", "SR", "ACR"
100 DATA "PCR", "IFR", "IER", "DATA"
110 REM 'DATA' IS ORA WITHOUT HANDSHAKE
120 DIM R $ (16), F (16), A (16)

200 A=59456: FOR J =1 TO 16
210 READ A $: R $ (J) = LEFT $ (A $ + "eight blanks",
E) + ":"
220 A(J) = A: A=A+1
230 NEXT J
240 F(4) = 1: F (12) = 1: F (13) = 1: F(14) = 1:
F(16) = 1

300 REM SET UP DISPLAY
310 PRINT "  7 6 5 4 3 2 1 0"
320 PRINT "            ";
330 PRINT "D = DATA P = POKE S = SHOW"
340 PRINT "II = HELP Q = QUIT"
```

= Clear/Home
 = Cursor Down
 = Home
 = Cursor Left

Cont'd...

```

400 REM DISPLAY Y LOOP
410 PRINT ' S Q Q ' ;
420 FOR J = 1 TO 16
430 IF F(J) = 0 THEN 450
440 Z = PEEK (A(J)): PRINT R $ (J);: COSUR 1000
450 NEXT J
460 REM IF NO INPUT DO LOOP AGAIN  = Cursor up
470 GET A $ : IF A $ = " " THEN 410

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 $ = "Q" THEN END

700 CO TO 310

1000 REM BINARY DISPLAY
1010 Z1 = 128
1020 FOR Z2 = 1 TO 8
1030 PRINT SGN (Z AND E1);
1040 IF Z2 = 4 THEN PRINT " ";
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 " 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 " D = DATA READS ORA WITH HANDSHAKE

```

Cont'd...

```

3080 PRINT " P = POKE LETS YOU POKE A REGISTER
3090 PRINT " S = SHOW SELECTS REGISTERS TO DISPLAY
3100 PRINT " Q = QUIT STOP PROGRAM
3300 PRINT "[ ] [ Q ]";: GOSUB 4990: RETURN
3500 REM CHANGE DISPLAYED REGISTERS
3510 PRINT "[ v ]" SHOW REGISTERS [ Q ] [ Q ] [ Q ]
3520 GOSUB 4000
3530 PRINT "S = SHOW, E = ERASE, X = FINSISH";:
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 "[ S ] [ Q ] [ Q ] [ Q ]";
3580 GO TO 3520

4000 REM GET REGISTER NAME, RETURN Z = INDEX
4010 PRINT "QQ REGISTER NAME: [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]";: INPUT AS
4020 RESTORE: FOR Z = 1 TO 16: READ B S
4030 IF B$ = A$ THEN RETURN
4040 NEXT Z: PRINT "[ Q ] [ Q ] [ Q ] THE REGISTERS ARE CALLED:
4050 FOR J = 1 TO 16: PRINT LEFT $ (R$ (J), 6)" " ";:
NEXT J
4060 PRINT "[ ● ] [ ● ] [ ● ] [ ● ] [ ● ] [ ● ] [ ● ] [ ● ]";: GO TO 4010
4500 REM GET BINARY NUMBER
4510 PRINT "BINARY VALUE: " ;: INPUT A$: Z1 = 128:
B = Ø
4520 IF LEN (A$) < 8 THEN PRINT " ● "; GO TO 4510
4530 FOR J = 1 TO 8
4540 IF MID $ (A$, J, 1) = "1" THEN B = B OR Z1
4550 Z1 = Z1/2: NEXT J
4560 RETURN

5000 GET A$: PRINT "[ & ] [ ]";: FOR K = 1 TO 20: NEXT K
5010 PRINT "[ ] [ ]"; FOR K = 1 TO 20: NEXT K
5020 IF A $ = " " THEN 5000
5030 RETURN

```

EXAMPLE 1 An Encoded ASCII Keyboard.

A surplus encoded ASCII keyboard was found with the following pinout:

| | <u>Keyboard</u> | <u>Wired to PET'S</u> |
|-----|---------------------|-----------------------|
| Pin | 1 INT KEY 1 | |
| | 2 RPT KEY | |
| | 3 — (NO CORRECTION) | CB2 |
| | 4 — | |

Cont'd...

Keyboard

5 GND
 6 +5v
 7 STROBE
 8 PARITY
 9 B4
 10 B3
 11 B1
 12 B7
 13 B2
 14 B6
 15 B5

Wired to PET'S

GND
 (SEPARATE +5 SUPPLY)
 CA1
 PA7
 PA3
 PA2
 PA0
 PA6
 PA1
 PA5
 PA9

First, the keyboard was connected to the "Blinken Lights" to check what it did, The "Blinken Lights" power was provided by the supply for the keyboard.

Watching the LEDs for PA0-7 and CA1 it was found that:
 (1) the correct ASCII code with parity appeared, and (2) the CA1 went high when a key was depressed.

The CA1 LED flickered when roll-over was tried (press one key, press 2nd key, release first key), showing that the keyboard had this feature.

Next, the keyboard was attached to the PET and the following program entered:

```
10      PRINT " [S] " PEEK (59471) and 127 " [ ] ";
20      GO TO 10
```

The ASCII values now appear in decimal on the PET's screen.

Now to us the CA1 input, and write characters on the screen, we have to:

1. Enable latching
2. Set CA1 to positive transition
3. Wait for CA1 FLAC bit
4. Get the data and print it as a character
5. Go to 3.

In Basic:

```
10      REM PRINT ON SCREEN FROM USER PORT
20      PRINT " [V] ";
30      POKE 59468, PEEK (59468) OR 1: REM PCR +
```

Cont'd...

```

40      POKE 59467, PEEK (59467) OR 1:  REM ACR LATCH
50      IF PEEK (59469) AND 2 THEN 70
60      GO TO 50
70      PRINTS CHR$( PEEK (59457) AND 127);
80      GO TO 50

```

Of course (!!) you must enter your characters in UPPER case - so press SHIFT if you have a FULL ASCII keyboard.

Note: When I was doing this, I would plug the unit in and nothing would happen!! Using the Blinken Lights, I saw the keyboard worked just fine. The actual problem? Be sure your socket is CORRECTLY inserted and is LINED UP with the pins!

DIGRESSION

How to represent the PET character set using ASCII. A study of PET & ASCII reveals that the PET recognizes 138 symbols and functions while ASCII recognizes 128 combinations.

Here is a solution to this problem.

1. ASCII Characters 0 - 31 are ignored except for these:
(all values are in decimal)

| | | | |
|----|--|--------|--------|
| 1 | A → RVS ON | 17 → Q | DELETE |
| 2 | B → RVS OFF | 18 → R | INST |
| 3 | C → HOME | 13 → | RETURN |
| 4 | D → HOME/CLR | | |
| 5 | E → CRSR DOWN | | |
| 6 | F → CRSR UP | | |
| 7 | G → CRSR RIGHT | | |
| 8 | H → CRSR LEFT | | |
| 27 | (ESCAPE) Puts the conversion into "Graphics Mode" | | |
| 10 | (LINE FEED) Puts the conversion into "Normal Mode" | | |

2. Normal Mode

Characters 0 - 31 Are the same
 Characters 32 - 93 Are Unchanged
 Characters 96 - 127 Are changed to 64 - 95 (Convert to Upper Case)

3. Graphics Mode

Characters 0 - 31 Are the Same
 Characters 32 - 95 Are Changed to 160 - 223
 Characters 96 - 127 Are Changed to 192 - 223

Cont'd...

This subroutine fetches an ASCII character and converts it to a PET character by the above rules. It is assumed that the initialization is performed and the mode flag, MF, is not changed.

```
1000 REM - INITIALIZE ROUTINE
1010 DATA Ø, 10, 146, 19, 147, 17, 145, 29, 157,
      Ø, Ø, Ø, Ø, 13, Ø, Ø, Ø, 20, 148
1020 DIM TT (31): FOR J = Ø TO 18: READ TT (J):
      NEXT J
1030 MF = Ø
1040 POKE 59468, PEEK (59468) OR 1
1050 POKE 59467, PEEK (59467) OR 1: RETURN

2000 REM - CONVERSION ROUTINE, RETURNS A$
2010 IF (PEEK (59469) AND 2) = Ø THEN 2010
2020 CH = PEEK (59457) AND 127
2030 IF CH > 31 THEN 2100
2040 REM CTRL CHAR
2050 IF CH = 10 THEN MF = Ø
2060 IF CH = 27 THEN MF = 1
2070 IF TT (CH) = 0 THEN 2010
2080 A $ = CHR$ (TT(CH)): RETURN
2090 REM CASE CONVERT
2100 IF CH > 95 THEN CH = CH-32
2110 REM MODE CONVERT
2120 CH = CH + MF * 128
2130 A $ = CHR$ (CH): RETURN
```

Try it out and see!! Look at your PET keyboard if you are confused.

Note: Don't forget the parenthesis in line 2020

EXAMPLE 2 The Writehander

The Writehander TM is a one-handed keyboard described in INTERFACE AGE, January 1978, and is manufactured by the NEWO Company, 246 Walter Hays Drive, Palo Alto, California 94303. We interfaced it to the PET to try it out...



Cont'd...

The Writehander has a 16 line rainbow ribbon cable with this pinout:

| WRITEHANDER | | | PET |
|-------------|--------------|---------------|----------------|
| <u>Line</u> | <u>Color</u> | <u>What</u> | |
| 1 | Brown | Bit 1 | PA \emptyset |
| 2 | Red | +7-+23v Power | |
| 3 | Orange | Bit 2 | PA1 |
| 4 | Yellow | GND | GND |
| 5 | Green | Bit 3 | PA2 |
| 6 | Blue | +5v | +5 (Separate) |
| 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 | ACK | CB2 |

These were wired to the PET as indicated, with the ground and +5V connected to a separate power supply.

The Writehander was wired with these options:

- | | | |
|---|-------------------------------------|--|
| 1 | Strobe goes active low | + to -  |
| 2 | Acknowledge is active low | + to -  |
| 3 | Parity fixed at low (\emptyset) | |

This means the following sequence is required:

- | | |
|----|--|
| 1 | POKE DDR TO ALL INPUT |
| 2 | CA1 TO HI-LD TRANSITION |
| 3 | DISABLE SHIFT REGISTER CB2 MODE |
| 4 | ENABLE CA1 LATCHING |
| 5 | TURN CB2 ON |
| 6 | WAIT FOR INTERRUPT FLAG |
| 7 | READ DATA WITH HANDSHAKE, MASK OFF, PARITY BIT |
| 8 | TURN CB2 OFF |
| 9 | DISPLAY VALUE ON SCREEN |
| 10 | GO TO 5 |

Cont'd...

This was turned into a basic program:

```
5      PRINT "v";
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    GO TO 50
```

This program shows the ASCII codes input by the Writehander. To show the characters, change line 90 to:

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

Three items are worth noting!!

1. The Writehander would work well with the Blinken Lightes and refuse to work with the PET! Eventually it was learned that:

CB2 (ACK) must be high when the Writehander brings CA1 (strobe) low. The Writehander won't strobe unless (ACR) is high.

2. OR is evaluated before AND by the PET! Line 80 was first written as:

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

And it was discovered that the data went into the PET when CB2 was toggled manually! A PEEK of 59468 revealed bit 0 was set, i.e. positive CA1 transition. When parenthesis were inserted it worked!!

So - interfacing has its hazards!!

3. The CHR\$ function in PET does not correspond to the ASCII code. To get the corresponding graphic character for an ASCII LOWER case,

```
90     X = X AND 127; IF X > 95 THEN X = X + 96
100    PRINT CHR$(X); GO TO 50
```


MCS6501-MCS6505 MICROPROCESSOR INSTRUCTION SET — ALPHABETIC SEQUENCE

| | | | |
|------------|--|--|---|
| ADC | Add Memory to Accumulator with Carry | | |
| AND | "AND" Memory with Accumulator | | |
| ASL | Shift Left One Bit (Memory or Accumulator) | | JSR Jump to New Location Saving Return Address |
| BCC | Branch on Carry Clear | | LDA Load Accumulator with Memory |
| BCS | Branch on Carry Set | | LDX Load Index X with Memory |
| BEQ | Branch on Result Zero | | LDY Load Index Y with Memory |
| BIT | Test Bits in Memory with Accumulator | | LSR Shift Right One Bit (Memory or Accumulator) |
| BMI | Branch on Result Minus | | NOP No Operation |
| BNE | Branch on Result not Zero | | ORA "OR" Memory with Accumulator |
| BPL | Branch on Result Plus | | PHA Push Accumulator on Stack |
| BRK | Force Break | | PHP Push Processor Status on Stack |
| BVC | Branch on Overflow Clear | | PLA Pull Accumulator from Stack |
| BVS | Branch on Overflow Set | | PLP Pull Processor Status from Stack |
| CLC | Clear Carry Flag | | ROL Rotate One Bit Left (Memory or Accumulator) |
| CLD | Clear Decimal Mode | | ROR Rotate One Bit Right (Memory or Accumulator) |
| CLI | Clear Interrupt Disable Bit | | RTI Return from Interrupt |
| CLV | Clear Overflow Flag | | RTS Return from Subroutine |
| CMP | Compare Memory and Accumulator | | SBC Subtract Memory from Accumulator with Borrow |
| CPX | Compare Memory and Index X | | SEC Set Carry Flag |
| CPY | Compare Memory and Index Y | | SED Set Decimal Mode |
| DEC | Decrement Memory by One | | SEI Set Interrupt Disable Status |
| DEX | Decrement Index X by One | | STA Store Accumulator in Memory |
| DEY | Decrement Index Y by One | | STX Store Index X in Memory |
| EOR | "Exclusive-Or" Memory with Accumulator | | STY Store Index Y in Memory |
| INC | Increment Memory by One | | TAX Transfer Accumulator to Index X |
| INX | Increment Index X by One | | TAY Transfer Accumulator to Index Y |
| INY | Increment Index Y by One | | TSX Transfer Stack Pointer to Index X |
| JMP | Jump to New Location | | TXA Transfer Index X to Accumulator |
| | | | TXS Transfer Index X to Stack Pointer |
| | | | TYA Transfer Index Y to Accumulator |

| | |
|-------------------------|-------------------------|
| 00 - BRK | 20 - JSR |
| 01 - ORA - (Indirect,X) | 21 - AND - (Indirect,X) |
| 02 - Future Expansion | 22 - Future Expansion |
| 03 - Future Expansion | 23 - Future Expansion |
| 04 - Future Expansion | 24 - BIT - Zero Page |
| 05 - ORA - Zero Page | 25 - AND - Zero Page |
| 06 - ASL - Zero Page | 26 - ROL - Zero Page |
| 07 - Future Expansion | 27 - Future Expansion |
| 08 - PHP | 28 - PLP |
| 09 - ORA - Immediate | 29 - AND - Immediate |
| 0A - ASL - Accumulator | 2A - ROL - Accumulator |
| 0B - Future Expansion | 2B - Future Expansion |
| 0C - Future Expansion | 2C - BIT - Absolute |
| 0D - ORA - Absolute | 2D - AND - Absolute |
| 0E - ASL - Absolute | 2E - ROL - Absolute |
| 0F - Future Expansion | 2F - Future Expansion |
| 10 - BPL | 30 - BMI |
| 11 - ORA - (Indirect),Y | 31 - AND - (Indirect),Y |
| 12 - Future Expansion | 32 - Future Expansion |
| 13 - Future Expansion | 33 - Future Expansion |
| 14 - Future Expansion | 34 - Future Expansion |
| 15 - ORA - Zero Page,X | 35 - AND - Zero Page,X |
| 16 - ASL - Zero Page,X | 36 - ROL - Zero Page,X |
| 17 - Future Expansion | 37 - Future Expansion |
| 18 - CLC | 38 - SEC |
| 19 - ORA - Absolute,Y | 39 - AND - Absolute,Y |
| 1A - Future Expansion | 3A - Future Expansion |
| 1B - Future Expansion | 3B - Future Expansion |
| 1C - Future Expansion | 3C - Future Expansion |
| 1D - ORA - Absolute,X | 3D - AND - Absolute,X |
| 1E - ASL - Absolute,X | 3E - ROL - Absolute,X |
| 1F - Future Expansion | 3F - Future Expansion |

40 - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
50 - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion

60 - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
70 - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion

| | |
|-------------------------|-------------------------|
| 80 - Future Expansion | A0 - LDY - Immediate |
| 81 - STA - (Indirect,X) | A1 - LDA - (Indirect,X) |
| 82 - Future Expansion | A2 - LDX - Immediate |
| 83 - Future Expansion | A3 - Future Expansion |
| 84 - STY - Zero Page | A4 - LDY - Zero Page |
| 85 - STA - Zero Page | A5 - LDA - Zero Page |
| 86 - STX - Zero Page | A6 - LDX - Zero Page |
| 87 - Future Expansion | A7 - Future Expansion |
| 88 - DEY | A8 - TAY |
| 89 - Future Expansion | A9 - LDA - Immediate |
| 8A - TXA | AA - TAX |
| 8B - Future Expansion | AB - Future Expansion |
| 8C - STY - Absolute | AC - LDY - Absolute |
| 8D - STA - Absolute | AD - LDA - Absolute |
| 8E - STX - Absolute | AE - LDX - Absolute |
| 8F - Future Expansion | AF - Future Expansion |
| 90 - BCC | B0 - BCS |
| 91 - STA - (Indirect),Y | B1 - LDA - (Indirect),Y |
| 92 - Future Expansion | B2 - Future Expansion |
| 93 - Future Expansion | B3 - Future Expansion |
| 94 - STY - Zero Page,X | B4 - LDY - Zero Page,X |
| 95 - STA - Zero Page,X | B5 - LDA - Zero Page,X |
| 96 - STX - Zero Page,Y | B6 - LDX - Zero Page,Y |
| 97 - Future Expansion | B7 - Future Expansion |
| 98 - TYA | B8 - CLV |
| 99 - STA - Absolute,Y | B9 - LDA - Absolute,Y |
| 9A - TXS | BA - TSX |
| 9B - Future Expansion | BB - Future Expansion |
| 9C - Future Expansion | BC - LDY - Absolute,X |
| 9D - STA - Absolute,X | BD - LDA - Absolute,X |
| 9E - Future Expansion | BE - LDX - Absolute,Y |
| 9F - Future Expansion | BF - Future Expansion |

C0 - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
D0 - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion

E0 - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
F0 - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

| | | |
|-----------|---------|---|
| 0000-0002 | 0-2 | USR Jump instruction |
| 0003 | 3 | Current I/O Device for prompt-suppress |
| 0005 | 5 | Cursor position for Input & Print |
| 0008-0009 | 8-9 | Integer address from Basic (for SYS, GOTO, etc.) |
| 000A-0059 | 10-89 | Basic input buffer; # of array subscripts |
| 005A | 90 | Search character (usually ':' or end-of-line) |
| 005B | 91 | Scan-between-quotes flag |
| 005C | 92 | Basic input buffer pointer; number of subscripts |
| 005D | 93 | First-character of array-name; default DIM flag |
| 005E | 94 | Type: FF=string; 00=numeric |
| 005F | 95 | Type: 80=integer; 00=floating point |
| 0060 | 96 | 'DATA' scan flag; LIST quote flag; memory flag |
| 0061 | 97 | Subscript flag; FNx flag |
| 0062 | 98 | 0=input, 64=get, 152=read (flag) |
| 0063 | 99 | flag for trigonometric signs/comparison evaluation flag |
| 0064 | 100 | input flag (suppress output if negative) |
| 0065 | 101 | variable pseudo-stack pointer |
| 0066 | 102 | fixed-point pseudo-stack pointer |
| 0067 | 103 | dummy value (0) |
| 0068-0070 | 104-112 | variable x pseudo-stack |
| 0071-0072 | 113-114 | pointer for number transfer |
| 0073-0074 | 115-116 | number pointer |
| 0075-0078 | 117-120 | product staging area for multiplication |
| 007A-007B | 122-123 | start of basic pointer |
| 007C-007D | 124-125 | end of basic/start of variables pointer |
| 007E-007F | 126-127 | end of variables/start of arrays |
| 0080-0081 | 128-129 | start of available space pointer |
| 0082-0083 | 130-131 | bottom of strings (moving down) pointer |
| 0084-0085 | 132-133 | top of strings (moving down) pointer |
| 0086-0087 | 134-135 | limit of Basic memory pointer |
| 0088-0089 | 136-137 | current program line number |
| 008A-008B | 138-139 | previous line number |
| 008C-008D | 140-141 | previous line address (for CONT) |
| 008E-008F | 142-143 | line number of DATA line |
| 0090-0091 | 144-145 | memory address of DATA line |
| 0092-0093 | 146-147 | input vector (DATA etc.) |
| 0094-0095 | 148-149 | current variable name |
| 0096-0097 | 150-151 | current variable address |
| 0098-0099 | 152-153 | variable pointer for current FOR/NEXT |
| 009A | 154 | Y save register; new operator save |
| 009C | 156 | comparison symbol accumulator: < 1 =2 > 4 |
| 009D-00A1 | 157-161 | number work area for SQR, etc. |
| 00A2 | 162 | pseudo-stack yardstick (3 or 7) |
| 00A3-00A5 | 163-165 | jump vector for functions |
| 00A6-00AA | 166-170 | numeric store area |
| 00AB-00AF | 171-175 | numeric store area |
| 00B0-00B5 | 176-181 | primary accumulator E,M,M,M,H,S |
| 00B6 | 182 | Taylor series constant counter |
| 00B7 | 183 | accumulator high-order propagation word |
| 00B8-00BD | 184-189 | secondary accumulator |
| 00BE | 190 | sign comparison, primary/secondary |
| 00BF | 191 | low-order rounding byte for primary acc |

| | | |
|-----------|---------|--|
| 00C0-00C1 | 192-193 | Cassette buffer length/Taylor constant pointer |
| 00C2-00D9 | 194-217 | Subrtn: Get Basic Char; C9.CA=pointer |
| 00DA-00DE | 218-222 | RND storage and work area |
| 00E0-00E1 | 224-225 | Pointer to screen cursor line |
| 00E2 | 226 | Position of cursor on line |
| 00E3-00E4 | 227-228 | Utility pointer; tape buffer, scrolling |
| 00E5-00E6 | 229-230 | End of current program/tape end address |
| 00E7-00E8 | 231-232 | Tape timing constants |
| 00E9 | 233 | Tape buffer character |
| 00EA | 234 | Direct/programmed cursor; 00=direct |
| 00EB | 235 | Tape read/verify flag |
| 00EC | 236 | Tape write flag |
| 00ED | 237 | |
| 00EE | 238 | Number of * characters in file name |
| 00EF | 239 | Logical file number |
| 00F0 | 240 | File command (from OPEN) |
| 00F1 | 241 | Device number |
| 00F2 | 242 | Maximum line length (40 or 80) |
| 00F3-00F4 | 243-244 | Tape buffr address (start of buffer) |
| 00F5 | 245 | Line where cursor lives |
| 00F6 | 246 | Last key pushed (ASCII); buffer checksum |
| 00F7-00F8 | 247-248 | Tape start address/tape pointer |
| 00F9-00FA | 249-250 | File name pointer |
| 00FB | 251 | Number of "insert" keys pushed |
| 00FC | 252 | Serial bit shift word |
| 00FD | 253 | # blocks remaining to write |
| 00FE | 254 | Serial word buffer |
| 0100-010A | 256-266 | Binary to ASCII conversion area |
| 010E-01FF | 267-511 | Stack area |
| 0200-0202 | 512-514 | T1 and T1S clock - jiffies |
| 0203 | 515 | Which key depressed: 255 = no key |
| 0204 | 516 | Shift key: 1 if depressed |
| 0205-0206 | 517-518 | Clock (unused?) |
| 0207 | 519 | Cassette 1 status switch |
| 0208 | 520 | Cassette 2 status switch |
| 0209 | 521 | Keyswitch BIA: STOP & RVS flags, etc. |
| 020A | 522 | |
| 020B | 523 | Load=0, Verify=1 |
| 020C | 524 | Status |
| 020D | 525 | # characters in keyboard buffer |
| 020E | 526 | Reverse flag |
| 020F-0218 | 527-536 | Keyboard buffer |
| 0219-021A | 537-538 | Hardware interrupt vector |
| 021B-021C | 539-540 | Break interrupt vector |
| 021D | 541 | |
| 021E | 542 | End-of-line-for -input pointer |
| 0220-0221 | 544-545 | Cursor log (row, column) |
| 0222 | 546 | PBD image for tape I/O |
| 0223 | 547 | Key image |
| 0224 | 548 | 0=flashing cursor; else no cursor shows |
| 0225 | 549 | Cursor timing countdown |
| 0226 | 550 | Character under cursor |
| 0227 | 551 | Cursor blink flag |
| 0228 | 552 | Tape write |
| 0229-0241 | 553-577 | Line address high & screen line wrap table |

| | | |
|-----------|-------------|---|
| 0242-024B | 578-587 | Logical numbers of open files |
| 024C-0255 | 588-597 | Device numbers of open files |
| 0256-025F | 598-607 | Command/Secondary address of open files |
| 0260 | 608 | Input from screen/Inpout from keyboard |
| 0261 | 609 | X-save flag |
| 0262 | 610 | How many open files |
| 0263 | 611 | Input device, normally 0 |
| 0264 | 612 | Output CMD device, normally 3 |
| 0265 | 613 | Tape parity |
| 0266 | 614 | |
| 0268 | 616 | Pointer in filename transfer |
| 026A | 618 | |
| 026C | 620 | Serial bit count |
| 026F | 623 | |
| 0270 | 624 | Tape write countdown |
| 0271 | 625 | Tape buffer #1 count |
| 0272 | 626 | Tape buffer #2 count |
| 0273 | 627 | Leader counter |
| 0274 | 628 | Flag for tape error |
| 0275 | 629 | 0 if 1st $\frac{1}{2}$ -byte cntr not written |
| 0276 | 630 | 2nd $\frac{1}{2}$ -byte cntr/tape error count |
| 0277 | 631 | |
| 0278 | 632 | Cassette read flag |
| 0279 | 633 | Checksum working word |
| 027A-0339 | 634-825 | Tape #1 buffer |
| 033A-03F9 | 826-1017 | Tape #2 buffer |
| 0400-7FFF | 1024-32767 | Available RAM including expansion |
| 8000-8FFF | 32768-36863 | Video RAM |
| 9000-BFFF | 36864-49151 | Available ROM expansion area |
| C000-E077 | 49152-57463 | Microsoft Basic |
| E078-E7F8 | 57464-59384 | Keyboard/Screen/Interrupt monitor |
| E810 | 59408 | PIA1 - Keyboard A register; (Direction with CRA2=1) |
| E811 | 59409 | PIA1 - Keyboard A control |
| E812 | 59410 | PIA1 - Keyboard B register; (Direction with CRB2=1) |
| E813 | 59411 | PIA1 - Keyboard B control |
| E820 | 59424 | PIA2 - IEEE A register; (Direction with CRA2=1) |
| E821 | 59425 | PIA2 - IEEE A control |
| E822 | 59426 | PIA2 - IEEE B register; (Direction with CRB2=1) |
| E823 | 59427 | PIA2 - IEEE B control |
| E840 | 59456 | VIA I/O register B |
| E841 | 59457 | VIA I/O register A with handshake |
| E842-E843 | 59458-59459 | VIA Data Direction regs, A and B |
| E844-E845 | 59460-59461 | VIA Timer 1 |
| E846-E847 | 59462-59463 | VIA Timer 1 latch |
| E848-E849 | 59464-59465 | VIA Timer 2 |
| E84A | 59466 | VIA shift register |
| E84B | 59467 | ACS: T1.T1.T2.SR.SR.SR.PB.PA |
| E84C | 59468 | PCR: B2.B2.B2.B1.A2.A2.A2.A1 |
| E84D-E84E | 59469-59470 | IFR, IER: T1.T2.CB1.BC2.SR.CA1.CA2 |
| E84F | 59471 | I/O Register A without handshake |
| F000-FFFF | 61440-65535 | Reset/tape/diagnostic monitor |

| | | | | | | | | | |
|------|--------------------|-------------|--------------------------|-------------------|---------------------|--------------------------|--|----|-------|
| E810 | DIAGNOS. SENSE | IEEE EOI in | CASSETTE SENSE #2 | CASSETTE SENSE #1 | KEYBOARD ROW SELECT | | | PA | 59408 |
| E811 | TAPE#1 INPUT FLAG | ... | SCREEN BLANK OUTPUT | | DDR A ACCESS | CASSETTE #1 READ CONTROL | | | 59409 |
| E812 | KEYBOARD ROW INPUT | | | | | | | | 59410 |
| E813 | RETRACE I FLAG | ... | CASSETTE #1 MOTOR OUTPUT | | DDR B ACCESS | RETRACE INTERR. CONTROL | | | 59411 |

| | | | | | | | | | |
|------|-------------|-----|---------------|--|--------------|---------------------|--|--|-------|
| E820 | IEEE INPUT | | | | | | | | 59424 |
| E821 | ATN I FLAG | ... | IEEE NDAC out | | DDR A ACCESS | IEEE ATN in CONTROL | | | 59425 |
| E822 | IEEE OUTPUT | | | | | | | | 59426 |
| E823 | SRQ I FLAG | ... | IEEE DAV out | | DDR B ACCESS | IEEE SRQ in CONTROL | | | 59427 |

| | | | | | | | | | | |
|------|--|-------------|-------------|----------------------------|--------------------|------------------------------------|------------------|----------------------|-------|---------|
| E840 | DAV in | NRPD in | RETRACE IN | CASS #2 MOTOR | CASSETTE OUTPUT | ATN out | NRPD out | NDAC in | PB | 59456 |
| E841 | | | | | | | | | | 59457 |
| E842 | DIRECTION REGISTER B (FOR E84D) | | | | | | | | | 59458 |
| E843 | DIRECTION REGISTER A (FOR E84F) (P.U.P.) | | | | | | | | | 59459 |
| E844 | TIMER 1 | | | | | | | | | L 59460 |
| E845 | | | | | | | | | | H 59461 |
| E846 | TIMER 1 | | | | | | | | | L 59462 |
| E847 | LATCH | | | | | | | | | H 59463 |
| E848 | TIMER 2 | | | | | | | | | L 59464 |
| E849 | | | | | | | | | | H 59465 |
| E84A | SHIFT REGISTER | | | | | | | | | 59466 |
| E84B | T1 CONTROL | | T2 CONTR. | | SHIFT REG. CONTROL | | | PB, PA LATCH CONTROL | | 59467 |
| E84C | CB2 (P.U.P. PUM) CONTROL | | | CB1 in CASSETTE#2 POLARITY | | CA2 (Graphics, Lower Case) Control | | CA1 in POLARITY | | 59468 |
| E84D | IRQ STATUS | T1 INT | T2 INT | CB1 CASSETTE#2 INT | CB2 INT | SR INT | CA1 (P.U.P.) INT | CA2 INT | 59469 | |
| E84E | ENABLE CLEAR/SET | T1 INT ENAB | T2 INT ENAB | CB1 INT ENAB | CB2 INT ENAB | SR INT ENAB | CA1 INT ENAB | CA2 INT ENAB | 59470 | |
| E84F | PARALLEL USER PORT I/O (PA) | | | | | | | | | 59471 |

CDC1-CDE7 checks for special characters (+,-,",.) at start of expression
 CDE8-CDF6 performs NOT function
 CDF7-CE04 checks for various functions
 CE05 evaluates expression within parentheses (
 CE0B checks for right parenthesis)
 CE0E checks for left parenthesis (
 CE11-CE1B checks for comma
 CE1C-CE20 prints SYNTAX ERROR and exits
 CE21-CE27 sets up function for future evaluation
 CE28-CE39 set up a variable name search
 CE3B-CE96 checks for special variables TI, TI\$, and ST
 CE97-CED5 identifies and sets up function references
 CED6-CF05 perform the OR and AND functions
 CF06-CF6D performs comparisons
 CF6E-CF7A sets up DIM execution
 CF7B-DO0E searches for a Basic variable
 DO0F-DO78 creates a new Basic variable
 DO79-DO87 logs Basic variable location
 DO88-DO98 is array pointer subroutine
 DO99-DO9C is 32768 in floating binary
 DO9D-DOB8 is floating point-to-fixed conversion for signed values
 DOB9-D263 locates and/or creates arrays
 D264-D277 performs FRE function
 D278-D284 converts fixed point-to-floating
 D285-D28A performs POS function
 D28B-D294 checks direct/indirect command, gives 'ILLEGAL DIRECT'
 D295-D348 executes DEF statements and evaluation FNx
 D349-D36A performs STR\$ function
 D36B-D3E1 scans and sets up string elements
 D3D2-D403 builds string vectors
 D404-D503 does 'garbage collection' - discards unwanted strings
 D504-D527 performs CHR\$ function
 D5D8-D653 performs LEFT\$, RIGHT\$, MID\$ functions
 D654-D662 performs LEN, gets string length
 D663-D672 performs ASC function
 D673-D684 gets a single-byte value from Basic
 D685-D6C3 evaluates VAL function
 D6C4-D6CF gets two arguments (16-bit and 8-bit) from Basic
 E6E0-D6E5 checks argument is in range 0-65535
 D6E6-D701 performs PEEK and POKE
 D702-D71D executes WAIT statement
 D71E-D890 performs addition and subtraction
 D891-D8BE contains floating-point constants
 98BF-D8FC performs LOG function
 D8FD-D95D performs multiplication
 D95E-D988 loads secondary accumulator from memory (\$B8 to \$BD)
 D989-D9B3 test and adjust primary/secondary accumulators
 D9B4-D9E0 routines to multiply or divide by 10
 D9E1-DA73 performs division
 DA74-DA98 loads primary accumulator from memory (\$b0-\$B5)
 EA99-DACD transfers primary accumulator to memory
 DACE-DADD transfers secondary accumulator to primary
 DADE-DAEC transfers primary accumulator to secondary
 EAED-EAFC rounds the primary accumulator
 DAFD-DB29 extracts primary sign; performs SGN function
 DB2A-DB2C performs ABS
 DB2D-DB6C compares primary accumulator to memory

DB6D-DB9D Convert Floating point to fixed, unsigned
 DB9E-DBC4 perform INT function
 DBC5-DC4F convert ASCII string to floating point
 DC50-DC84 get new ASCII digit
 DC94-DCAE print Basic Line number
 DCAF-DDE2 convert floating point to ASCII string (at 0100 up)
 DDE3-DE23 conversion constants - decimal or clock
 DE24-DE2D evaluation SQR function
 DE2E-DE66 evaluation of power function
 DE67-DE71 negate (monadic -)
 DEAO-DEF2 perform EXP function
 DEF3-DF3C perform function series evaluation
 DF45-DF9D perform RND calculation
 DF9E evaluate COS function
 DFA5-DFED evaluate SIN function
 DFEE-E019 evaluate TAN function
 E048-E077 evaluate ATN function
 E0B5-E0CC Basic scan program, transferred to OOC2-00D9
 E0D2-E173 completion of power-on-reset; memory test, etc.
 E19B-E1BB partial test for TI and TI\$
 E1BC-E1E0 input/read/get director
 E1E1-E27C initialize I/O registers, clear screen, reset subroutine
 E27D-E3C3 receive input from keyboard/screen
 E3C4-E3E9 set up new screen line
 E3EA-E52F output character to screen
 E530-E5DA check for and perform screen scrolling
 E5DB-E66A start new screen line
 E66B-E67D interrupt entry
 E67E-E683 interrupt return
 E685-E73E hardware interrupt routine: cursor flash, tape motor, keyboard
 E73F-E7AB convert keyboard matrix to ASCII
 E7AC-E7B9 write-on-screen subroutine
 E7DE-E7EB print canned monitor message
 F0B6-F1CB IEEE-488 channel open, test, close
 F1CC-F22F get input character from keyboard, screen cassette, IEEE
 F230-F27C output character to screen, cassette, IEEE
 F27D-F2A3 restore normal I/O, clear IEEE channels
 F2A4-F2AA abort (not close!) all files
 F2AB-F2B7 locate logical file table entry
 F2B8-F2C7 transfer file table entries to Device, Command
 F2C8-F329 perform file CLOSE
 F32A-F33E test stop key
 F33F-F345 test if direct/indirect command for suppressing file advice
 F346-F3FE perform file LOAD
 F3FF-F421 print "SEARCHING .. "
 F422-F432 print "LOADING .. " or "VERIFYING"
 F433-F461 get parameters for LOAD and SAVE
 F462-F494 perform IEEE sequences for LOAD, SAVE, and OPEN
 F495-F4BA search for specific tape header
 F4BB-F4D3 perform VERIFY
 F4D4-F529 get parameters for OPEN and CLOSE
 F52A-F5AD perform OPEN
 F5AE-F5E2 search for any tape header
 F5E3-F5EC clear tape buffer
 F5ED-F64C write tape header
 F64D-F666 get start & end addresses from tape header

F667-F67C Set buffer start address
 F67D-F694 set tape buffer start and end pointers
 F695-F69D perform SYS command
 F69E-F71B perform SAVE
 F71C-F735 find unused secondary address
 F736-F78A update clock
 F78B-F7DB set input device
 F7DC-F82C set output device
 F82D-F83A bump tape buffer counter
 F83B-F85D wait for cassette PLAY switch
 F85E-F870 test cassette switch line
 F871-F87E wait for cassette RECORD and PLAY switches
 F87F-F8B8 read tape initiation routine
 F8B9-F8D1 write tape initiation routine
 F8D2-F912 complete tape read or write
 F913-F91D wait for I/O completion
 F91E-F92D test stop key and abort if necessary
 F92E-F95E subroutine to set tape read timing
 F95F-FBDB interrupt routine for tape read
 FBDC-FBE4 save memory pointer
 FBE5-FBEB set ST error flag
 FBEC-FBFF subroutine to count 8 serial bits per byte
 FCOO-FC1B subroutine to write a bit to tape
 FC1C-FCFA interrupt 1 for tape write - entry at FC21
 FCFB-FD15 terminate I/O and restore normal vectors
 FD16-FD37 subroutine to set interrupt vector
 FD38-FD47 power-on reset entry; test for diagnostic
 FD48-FD7B diagnostic routine
 FD7C-FD8F checksum routine
 FD90-FD9A pointer advance subroutine
 FD9B-FFB1 diagnostic routines
 JUMP TABLE:
 FFC0 OPEN
 FFC3 CLOSE
 FFC6 set input device
 FFC9 set output device
 FFCC restore normal I/O devices
 FFCF input character (from screen)
 FFD2 output character
 FFD5 LOAD
 FFD8 SAVE
 FFDB VERIFY
 FFDE SYS
 FFE1 test stop key
 FFE4 get character from keyboard buffer
 FFE7 abort all I/O channels
 FFEA update clock

 FFED-FFFA turn off cassette motors
 FFFA-FFFB NMI vector (mangled)
 FFFC-FFFD reset vector
 FFFE-FFFF interrupt vector

I N D E X

| <u>General</u> | Pg. |
|---|-------|
| ASCII & POKE Values Table | 8 |
| BASIC - FORTRAN Comparison | 17 |
| Bits and Pieces | 1,53 |
| Commercial Confusion | 84 |
| Computer Courses | 119 |
| Computer Mini-course (PET). | 66 |
| Data File Patches | 27,56 |
| Editing | 4 |
| Failsafing | 62 |
| Instruction Set, 6502 | 143 |
| Interesting Memory Locations | 5 |
| Interrupts on the Commodore PET | 103 |
| Interrupt Structure | 3 |
| Inverse Trig. Functions | 2 |
| Keyboard Matrix | 5 |
| Keyboard Values | 11 |
| Keyword Abbreviations | 50 |
| Keyword Values | 12 |
| Machine Language | 89 |
| Memory Map (J. Butterfield's) | 149 |
| Micro Magazine | 85 |
| Newsletter Addresses | 60 |
| ON - GOTO | 59 |
| Programme Overlays | 43 |
| ROM Comparison: 011 vs. 019 | 20 |
| Shifted Capitals | 94 |
| Standard Symbols | 54 |
| Time Delay Tips | 25 |
| Timing Tables | 6 |
| T.I.S. Workbooks | 85 |
| UNLIST | 57 |
| User Port Cookbook | 123 |
| User Port Information | 9,28 |
| <u>Software</u> | |
| Assembler, 6502 | 24 |
| Auto - Repeat | 92 |
| Business Software | 118 |
| Calc. Simulator | 51 |
| Decvert | 93 |
| Hexvert | 93 |
| LIFE For Your PET | 30,92 |
| LIFE, Snowless | 42 |
| Non - Stop | 91 |
| Plotting | 26 |
| Prime Number Test | 15 |
| Reflex Test | 24 |
| Squares, Cubes, Roots | 16 |

| | |
|---------------------------------------|-----|
| Squiggle Version 2.0 | 21 |
| Tax Ontario 1978 | 86 |
| Twenty Number Sort | 15 |
| Upper to Lower Case Convert | 121 |
| View | 91 |
| World Capitols | 19 |

Hardware

| | |
|-----------------------------------|---------|
| Available Hardware | 9,120 |
| Cassette Fix | 16 & 95 |
| D/A Converter | 58 |
| Hampet: RTTY Interface | 66 |
| Selectric Interface | 117 |
| Video Monitor Interface | 23 |
| X-Y Plotter | 9,28 |