Commodore

The Transactor

comments and bulletins concerning your COMMODORE PET™

VOL 2
BULLETIN 12

Transactor Article Contest Winners

In Transactor #8, we promised awards for the best articles published in Volume 2. We also promised free subscriptions to The Transactor Volume 3 for any article published. Here are the winners:

Best Article goes to J. Hoogstraat of Calgary, Alberta, for his BASIC Labelling Routine published this issue and also for his 2040 Disk I/O Routine in bulletin #10. Mr. Hoogstraat gets a free Visicalc package.

Runner up award goes to F. Van Duinen of Toronto, Ontario, for ?LOAD ERROR, D.R.I.P and Program Plus. Mr. Van Duinen receives a Commodore calculator, model # SR9190.

Free Volume 3 subscriptions are going to:

J. Hoogstraat	F. Van Duinen	*Jim Russo
*Kevin Erler	John A. Cooke	Rick Ellis
*James Yost	Chuan Chee	Jim Hindson
Michael Casey	G. Hathaway	W.T. Garbutt
*B. Brown	*L.D. Gardner	Tom Wojdylo
Dave Hook	Paul Barnes	*S. Donald
Henry Troup	Gord Campbell	*John Macdonald
*Sheldon H. Dean	Don White	Dave Berezowski
Brad Templeton		*Robert Oei

* Please call or send in your address.

This contest will be held again for The Transactor Volume 3, prizes may differ.

If you're asking "What about Jim Butterfield?", don't worry, he's been well taken care of.

As a Commodore dealer, Bill MacLean of BMB CompuScience was not eligible for a prize, but we'll figure something out.

I'd like to thank all who contributed to Volume 2 and special thanks to Jim and Bill for some really excellent stuff! Special thanks also to Terry Garbutt for his truly genuine help and support. Hoping to hear from all of you in Volume 3, I remain,

Karl J. Hildon
Editor, The Transactor

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

Exclusive OR on Your PET

In boolean algebra there are three main operators: AND, OR and NOT. All three of these are included in PET BASIC. However, one sometimes very useful boolean function was not included in BASIC. This is the EXclusive OR function. EXOR is a function of AND, OR and NOT:

```
(a)EXOR(b) = ((a) AND (NOT(b)) ) OR ((b) AND (NOT(a)) )
```

Of course the above would result in ?SYNTAX ERRORS if coded literally. The following will accomplish a% EXclusive OR'd with b% in BASIC.

```
ex% = ((a%)and(not(b%)))or((b%)and(not(a%)))
```

An Extra Note on Logical Operators

Try RUNning this short program: (enter exactly as shown)

```
10 xt=5 : xf=6 : rem just random values
20 print xtandxf
```

Now replace line 20 with each of the following line 20's and RUN each again.

```
20 print xtorxf
20 print xforxt
```

Each of the three will result in ?SYNTAX ERROR IN LINE 20. But why? When you hit return on a line of BASIC, the PET procedes to "tokenize" the line by parsing the characters from left to right.

Line:	Would be tokenized as:
20 print xtandxf	print x tan dxf
20 print xtorxf	print x to rxf
20 print xforxt	print x for xt

A general rule: When preceding logical operators with floating point variables, insert a space or enclose the variable in brackets. Integer type variables will not be succeptable to this problem because the "%" sign will act as a delimiter. Brackets are still necessary for hierarchy of operations.

This gotcha surfaces in one other command in BASIC 4.0:

```
header"diskname",d0,ifx
```

The breakdown of this would be format a diskettes on drive 0 with "diskname" as the title and "fx" as the disk id. But on hitting return, a ?SYNTAX ERROR is printed because ifx is tokenized as if x .

The BASIC Difference Is:

	BASIC 1.0	BASIC 2.0	BASIC 4.0
PEEK(50003) =	0	1	160
Disable STOP POKE	537,136	144, 49	144, 88
Enable STOP POKE	537,133	144, 46	144, 85

New BASIC 4.0 machines reportedly crash on some old programs. The culprit is most likely a disable STOP key POKE. Also check for POKE59458,62, the screen speed-up POKE. As mentioned before, this can also crash machines. See article this issue on BASIC 2.0 - BASIC 4.0 Conversions for more info.

Screen Loading

All you need is a "screen-set-up" routine to "draw" your screen out, and this program will store it on disk:

```
100 REM SCREEN SAVER
110 OPEN 8, 8, 8, "0:SCREEN NAME,P,W"
120 PRINT#8, CHR$(0)CHR$(128);
130 EN=33767 : IF PEEK(50003)=160 THEN EN=34767
140 FOR J=32768 TO EN
150 PRINT#8, PEEK(J);
160 NEXT
170 CLOSE 8
180 END
```

Line 130 sets end screen (EN) to 33767 for 40 columns, 34767 for 80 columns.

SAVE this program and do a NEW. Now enter:

```
 10 ON X GOTO 120
100 PRINT "[clrscrn]";
110 X=1 : LOAD "0:SCREEN NAME",8
120 END
```

RUN this and the old screen should pop back on the screen as fast a loading 1k from disk. The cursor will remain in the home position since nothing is actually printed. No pointers or variables are changed since it was a "dynamic load". But the loader program would RUN from the beginning, hence the ON X GOTO statement. This could be expanded to accommodate more screen loads simply by adding more GOTO data to line 10 and setting X appropriately prior to the load. The SCREEN SAVER program could also be modified to store only a portion of the screen. But don't forget to change the load address in line 120, else the files will always load back to screen starting at HOME.

More On The NEC Spinwriter

In Transactor #11, the internal switch positions for the NEC Spinwriter were published to get proper operation with WordPro. Dr. G. Piasecki of Oakville, Ontario, tells me that one of the switches should be left unchanged.

back board : SW1 = 0 0 0 0 1 X 1 1

The switch at position 6 (labelled "X") will be set ON or OFF (1 or 0) depending on your particular Spinwriter. This is done at the factory. Set the other switches as shown but do not change position 6. See page 1 of Transactor #11 for more info. In summary:

back board : SW1 = 0 0 0 0 1 X 1 1

2nd from back board : SW1 = 0 0 0 0 0 0 0 0

SW2 = 1 0 1 0 0 0 0 0

SW3 = 1 0 1 0 0 0 0 0

Another Contest!

This one's a quickie and will be decided before publication of Bulletin #1, Volume 3.

Objective: The shortest macro routine to push the stack pointer onto the stack. Inotherwords, a PHS instruction.

1. All internal registers (.A, .X, .Y and .P) must have their original contents once the stack pointer is on the stack.

2. Macro only; no using RAM (excluding the stack) to store internal registers

The person with the shortest routine will receive a free subscription to The Transactor Volume 3. Submit entries to:

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

Oops!

Dave Hooks Card Print Utility was published in the last issue without a listing of the program. The listing and the cross reference follow Bits and Pieces.

Also, our last bulletin was actually #11. It was labelled "10" by mistake. The real bulletin 10 is labelled "# 10".

Subscription Renewals

Don't forget, this is the last issue of The Transactor, Volume 2. A Volume 3 subscription form has been included in this issue. The Transactor Volume 3 will be $10.00 for 6 issues, back issues included.

Transactor #7 Insert

The center page of this issue contains more helpful reference lists and tables. It has been designed so that, if you wish, it can be removed and inserted as the center page of Transactor #7.

Flash!

The POP SYS for BASIC 2.0 also has an equivalent BASIC 4.0 entry point:

```
BASIC 2.0:   SYS 50583
BASIC 4.0:   SYS 46610
```

This SYS "POPS" or cleans the stack of all GOSUBs and NEXT loops. Watch for this when converting BASIC 2.0 to BASIC 4.0 too!

CROSS REFERENCE - PROGRAM CARD UTILITY

```
A%      42020 42030 43020 43140
C%      42000 42010 43100 43130
D%      40000 40010 40020 41000 41010 42000 43040
D%(     40020 41000 41010 42010 43130
E9      40000
F$(     40080 42500 42510 42520 43500 43510 43520
I       40000 40020 40030 40050 40060 40070 40080 41000 41010
I$(     40030 42050 42750 43160 43750
J       40000 40020 40070 40080 42050 42070 42500 42510 42520 42750 43160
        43250 43500 43510 43520 43750
J%      40000 41000 41010
K%      40000 41000 41010
L       42000 42030 42800 43100 43140 43800
L%      42020 42030 43010 43140
M%      43040 43100
P%      43000 43100
S$(     40050 40060 42070 43250
S%      42010 42070 42500 42510 43130 43250 43500 43510
S%(     40070 42070 43250
S1$     40040 40050
S2$     40040 40060
T%      42030 42040 42050 42070 42500 42510 42520 42750 43140 43150 43160
        43250 43500 43510 43520 43750
TB%     42020 42030 43030 43140
TI      40000
V%      42010 42050 42060 42070 42500 42510 42520 42750 43130 43160 43170
        43250 43500 43510 43520 43750
Z         130   140   150 15010 43000 43010 43020 43030 43040
Z$        120   130   160 15000 15010
```

```
10 DATA A,2,3,4,5,6,7,8,9,10,J,Q,K
20 DATA,,,1,,,,1,,,,,,,1,1,,,1,,,1,2,,,,,,,2,2,,,1,,,2
30 DATA2,,,2,,,2,2,,1,,2,,2,2,,2,,2,,2,2,,2,1,2,,2,2,1,2,,2,1,2
40 DATA" ▀▀ ▞▀  "," ▀▝▞ ⊔ ▞  "," ▌▀▞▀▀▚▞  "
50 DATA" ▒▀✕▀▞▒ ","  ▀▀▞▀▀ ▞ "
60 DATA"   ⊓ ▙ ▞ ","  ▀ ▚"," ┏┓  "," ▀▞│ │ "," ▀    ▙ "," ▒○○▒ "
70 DATA" ▀    ▞ ","  │ ▌▀ ","  ▀ ▀▀","▀▀  ▞▀ "," │▒│ "," ▀    ▝▞ "
80 DATA" ▒▚▚▚▒ "," ▙   ▞▀ "," │▒ │  "," ▀   ▚"
90 GOSUB40000
100 PRINT"▚"TAB(10)"▞CARD UTILITY":PRINT"▚1. DISPLAY CARDS":PRINT"▚2. SHUFFLE
110 PRINT"▚3. SUBROUTINE FOR GAMES":PRINT"▚4. QUIT":PRINT"▚▚▞SELECTION  ?";
120 GETZ$:IFZ$=""THEN120
130 Z=VAL(Z$):PRINTZ:IFZ<1ORZ>4THEN100
140 IFZ=4THENEND
150 ONZGOSUB42000,41000,43000:PRINT"▚▞DONE--HIT A KEY
160 GETZ$:IFZ$=""THEN160
170 GOTO100
14998 END
15000 INPUT"   ▒▌▌▌▌";Z$:IFZ$="▒"THEN15000        :REM INPUT SBR.
15010 Z=VAL(Z$):RETURN
40000 I=RND(-TI*1E9):J=0:D%=0:J%=0:K%=0        :REM INITIALIZATION
40010 INPUT"NUMBER OF DECKS    1▌▌▌▌";D%
40020 DIMD%(D%*52):FORI=1TOD%:FORJ=0TO51:D%(52*(I-1)+J)=J:NEXTJ,I:D%=D%*52-1
40030 DIMI$(13):FORI=1TO13:READI$(I):NEXTI
40040 S1$="  ♣   ♦   ♥   ♠   ":S2$=" ♣   ♣ ♦    ♦ ♥    ♥ ♠    ♠ "
40050 DIMS$(2,3):FORI=0TO03:S$(0,I)="       ":S$(1,I)=MID$(S1$,I*4+1,7)
40060 S$(2,I)=MID$(S2$,I*6+1,7):NEXTI
40070 DIMS%(10,7):FORI=1TO10:FORJ=1TO7:READS%(I,J):NEXTJ,I
40080 DIMF$(3,7):FORI=1TO3:FORJ=1TO7:READF$(I,J):NEXTJ,I
40090 RETURN
40999 REM SHUFFLE
41000 FORI=0TOD%:J%=(D%+1-I)*RND(1):K%=D%(J%)
41010 D%(J%)=D%(D%-I):D%(D%-I)=K%:NEXTI:RETURN
41999 REM DISPLAY ALL CARDS
42000 PRINT"▚":C%=0:FORL=0TOD%:C%=C%+1:
42010 S%=D%(C%-1)/13:V%=D%(C%-1)-13*S%+1
42020 L%=7:A%=5:TB%=0
42030 IFL/A%=INT(L/A%)THENT%=TB%:PRINTLEFT$("▚QQQQQQQQQQQQQQQQQQ",L%):GOTO42050
42040 T%=T%+8:PRINT"▔▔▔▔▔▔▔▔▔";
42050 PRINTTAB(T%)"▞"LEFT$(I$(V%)+"       ",7):FORJ=1TO7
42060 IFV%>10THEN42500
42070 PRINTTAB(T%)"▞"S$(S%(V%,J),S%):GOTO42750
42500 IFJ=1THENPRINTTAB(T%)"▞ "MID$("♣♦♥♠",S%+1,1)F$(V%-10,J):GOTO42750
42510 IFJ=7THENPRINTTAB(T%)"▞"F$(V%-10,J)"▞"MID$("♣♦♥♠",S%+1,1)" ":GOTO42750
42520 PRINTTAB(T%)"▞"F$(V%-10,J)
42750 NEXTJ:PRINTTAB(T%)"▞"RIGHT$("       "+I$(V%),7)
42800 NEXTL:RETURN
42999 REM GAME-TYPE SUBROUTINE
43000 PRINT"▚HOW MANY CARDS TO PRINT";:GOSUB15000:P%=Z
43010 PRINT"START ON LINE (1-16)";:GOSUB15000:L%=Z
43020 PRINT"HOW MANY ACROSS (1-5)";:GOSUB15000:A%=Z
43030 PRINT"START AT TAB (0-32)";:GOSUB15000:TB%=Z
43040 M%=D%+1:PRINT"SHUFFLE AFTER (1-"M%")";:GOSUB15000:M%=Z
43100 PRINT"▚":C%=0:FORL=0TOP%-1:C%=C%+1:IFC%=M%+1THENC%=1:GOSUB41000
43130 S%=D%(C%-1)/13:V%=D%(C%-1)-13*S%+1
43140 IFL/A%=INT(L/A%)THENT%=TB%:PRINTLEFT$("▚QQQQQQQQQQQQQQQQQQ",L%):GOTO43160
43150 T%=T%+8:PRINT"▔▔▔▔▔▔▔▔▔";
43160 PRINTTAB(T%)"▞"LEFT$(I$(V%)+"       ",7):FORJ=1TO7
43170 IFV%>10THEN43500
43250 PRINTTAB(T%)"▞"S$(S%(V%,J),S%):GOTO43750
43500 IFJ=1THENPRINTTAB(T%)"▞ "MID$("♣♦♥♠",S%+1,1)F$(V%-10,J):GOTO43750
43510 IFJ=7THENPRINTTAB(T%)"▞"F$(V%-10,J)"▞"MID$("♣♦♥♠",S%+1,1)" ":GOTO43750
43520 PRINTTAB(T%)"▞"F$(V%-10,J)
43750 NEXTJ:PRINTTAB(T%)"▞"RIGHT$("       "+I$(V%),7)
43800 NEXTL:RETURN
```

J. Hoogstraat
Calgary, Alta

This amazing routine resides in the second cassete buffer and allows the use of labels in basic and has no effect on the speed of basic.

A label starts with a # character and is retricted in length to the basic line length.

EXAMPLE NO LABELS

```
100 FOR I = 1 TO 3
110 ON I GOSUB 500, 550, 600
120 NEXT
130 GOTO 800
140 :
500 PRINT "SUBROUTINE";I :RETURN
510 :
550 PRINT "SUBROUTINE";I :RETURN
560 :
600 PRINT "SUBROUTINE";I :RETURN
610 :
800 PRINT "END OF TEST":END
```

EXAMPLE WITH LABELS

```
10  SYS826
20  :
100 FOR I = 1 TO 3
110 ON I GOSUB #SUB1, #SUB2, #SUB3
120 NEXT
130 GOTO #ALLDONE
140 :
500 #SUB1:PRINT "SUBROUTINE";I :RETURN
510 :
550 #SUB2
555 PRINT"SUBROUTINE";I :RETURN
560 :
600 #SUB3:PRINT "SUBROUTINE";I :RETURN
610 :
800 #ALLDONE:PRINT "END OF TEST":END
```

The #labels can be mixed up with basic statement numbers.

```
110 ON I GOSUB #SUB1, 550, #SUB3
```

Since the routine resides in the second cassette buffer and modifies the basic GET character routine, it prohibits the use of any other routines in the second cassette buffer or the use of the DOS support program. However it can be made part of the DOS support program.

I do have available a modified DOS support program which includes the following:

1. Regular DOS support.

2. The BASIC label support interface.

3. An excellent repeat key function.

4. A basic disk append command. no messing around with tapes

Just send me $20.00 and a floppy and I will return a copy of the above including all the assembly source on your floppy, or for $27.00 I'll send you a floppy with the same.

By the way, the # label prefix is my choice and can be altered to any other special character.

Have a lot of Basic fun !!!

Editor's Note:

Mr. Hoogstraat's routine works on BASIC 2.0 only. To convert to BASIC 4.0, some JSRs would need changing. Also, the program could no longer reside in the second cassette buffer. This space is used by some new BASIC 4.0 commands.

```
800 FOR J=826 TO 1008 : READ X :POKE J, X : NEXT
826 DATA 169,  71, 133, 113, 169,   3
832 DATA 133, 114, 169,  76, 133, 112
838 DATA  96, 230, 119, 208,   2, 230
844 DATA 120, 164,  55, 200, 208,   3
850 DATA  76, 118,   0, 160,   0, 177
856 DATA 119, 201,  35, 208, 245, 186
862 DATA 189,   1,   1, 201,  62, 240
868 DATA  24, 201, 172, 240,  20, 201
874 DATA 143, 240,  16, 201, 105, 208
880 DATA 107,  32, 112,   0, 201,  44
886 DATA 208, 249, 104, 104,  76,  95
892 DATA 200, 200, 166,  40, 165,  41
898 DATA 208,   8, 160,   0, 177,  92
904 DATA 170, 200, 177,  92, 134,  92
910 DATA 133,  93, 133,  91, 177,  92
916 DATA 208,   3,  76, 235, 199,  24
922 DATA 165,  92, 105,   4, 133,  90
928 DATA 144,   2, 230,  91, 136, 177
934 DATA  90,  32, 226,   3, 133,  89
940 DATA 177, 119, 200,  32, 226,   3
946 DATA 197,  89, 208, 206, 201,   0
952 DATA 208, 235, 104, 104, 186, 189
958 DATA 255,   0, 201, 143, 208,  21
964 DATA 165, 120,  72, 165, 119,  72
970 DATA 165,  55,  72, 165,  54,  72
976 DATA 169, 141,  72, 169, 198,  72
982 DATA 169, 195,  72,  32, 205, 199
988 DATA  32,   0, 200,  76, 118,   0
994 DATA 201,  32, 240,   8, 201,  58
996 DATA 240,   4, 201,  44, 208,   2
998 DATA 169,   0,  96
```

```
.M 033A 03F0
.
.:  033A A9 47 85 71 A9 03 85 72
.:  0342 A9 4C 85 70 60 E6 77 D0
.:  034A 02 E6 78 A4 37 C8 D0 03
.:  0352 4C 76 00 A0 00 B1 77 C9
.:  035A 23 D0 F5 BA BD 01 01 C9
.:  0362 3E F0 18 C9 AC F0 14 C9
.:  036A 8F F0 10 C9 69 D0 6B 20
.:  0372 70 00 C9 2C D0 F9 68 68
.:  037A 4C 5F C8 C8 A6 28 A5 29
.:  0382 D0 08 A0 00 B1 5C AA C8
.:  038A B1 5C 86 5C 85 5D 85 5B
.:  0392 B1 5C D0 03 4C EB C7 18
.:  039A A5 5C 69 04 85 5A 90 02
.:  03A2 E6 5B 88 B1 5A 20 E2 03
.:  03AA 85 59 B1 77 C8 20 E2 03
.:  03B2 C5 59 D0 CE C9 00 D0 EB
.:  03BA 68 68 BA BD FF 00 C9 8F
.:  03C2 D0 15 A5 78 48 A5 77 48
.:  03CA A5 37 48 A5 36 48 A9 8D
.:  03D2 48 A9 C6 48 A9 C3 48 20
.:  03DA CD C7 20 00 C8 4C 76 00
.:  03E2 C9 20 F0 08 C9 3A F0 04
.:  03EA C9 2C D0 02 A9 00 60 00
```

```
                0010            .OS
                0020            .BA $33A
                0030;
                0040; ---------------------------------
                0050; - BASIC LABEL SUPPORT INTERFACE -
                0060; ---------------------------------
                0070;
                0080; SYS826 ACTIVATES THE BASIC LABEL
                0090; SUPPORT INTERFACE AND ALLOWS THE
                0100; USE OF LABELS IN BASIC FOR 'GOTO'
                0110; 'THEN' AND 'GOSUB' STATEMENTS.
                0120;
                0130; A LABEL IS PREFIXED WITH A
                0140; # CHARACTER AND TERMINATES
                0150; WITH A BLANK, COMMA OR COLON.
                0160;
                0170; BY J.HOOGSTRAAT
                0180;
                0190; BOX 20, SITE 7, SS 1
                0200; CALGARY, T2M-4N3
                0210; ALBERTA. 403-239-0900
                0220;
                0230; ---------------------------------
                0240;
                0250; HOOK UP THE BASIC LABEL INTERFACE
                0260;
033A-A947       0270HOOKUP      LDA #L,LABELS
033C-8571       0280            STA *GETCHR+1
033E-A903       0290            LDA #H,LABELS
0340-8572       0300            STA *GETCHR+2
0342-A94C       0310            LDA #$4C
0344-8570       0320            STA *GETCHR
0346-60         0330            RTS
                0340;
                0350; BASIC LABELS SUPPORT INTERFACE
                0360;
0347-E677       0370LABELS      INC *CHAD       ;DO MISSING PART
0349-D002       0380            BNE =+3         ;OF GETCHR.
034B-E678       0390            INC *CHAD+1
                0400;
034D-A437       0410            LDY *CLIN+1     ;IMMEDIAT MODE ?
034F-C8         0420            INY
0350-D003       0430            BNE LABEL1      ;NOT IMMEDIAT.
                0440;
0352-4C7600     0450NLABEL      JMP GOTCHR      ;NORMAL CONTINUE.
                0460;
0355-A000       0470LABEL1      LDY #0          ;# PREFIX ?
0357-B177       0480            LDA (CHAD),Y
0359-C923       0490            CMP #'#
035B-D0F5       0500            BNE NLABEL      ;NO PREFIX, EXIT.
                0510;
                0520; DECIDE ON WHAT ACTION TO TAKE
                0530;
035D-BA         0540CHKLAB      TSX
035E-BD0101     0550            LDA $101,X      ;GET STACK VALUE.
                0560;
```

```
0361-C93E   0570            CMP #S.THEN   ;BASIC THEN ?
0363-F018   0580            BEQ FLABEL    ;YES, FIND LABEL.
            0590;
0365-C9AC   0600            CMP #S.GOTO   ;BASIC GOTO ?
0367-F014   0610            BEQ FLABEL    ;YES, FIND LABEL.
            0620;
0369-C98F   0630            CMP #S.GSUB   ;BASIC GOSUB ?
036B-F010   0640            BEQ FLABEL    ;YES, FIND LABEL.
            0650;
036D-C969   0660            CMP #S.ONDO   ;BASIC ON.DO ?
036F-D06B   0670            BNE SKPLAB    ;NO, IT'S A LABEL.
            0680;
            0690; ON.DO ACTION
            0700;
0371-207000 0710SCOMMA      JSR GETCHR    ;FOR ON.DO
0374-C92C   0720            CMP #',       ;STATEMENT GET PAST
0376-D0F9   0730            BNE SCOMMA    ;THE COMMA.
0378-68     0740            PLA
0379-68     0750            PLA
037A-4C5FC8 0760            JMP ON.RET    ;RETURN TO ON.DO
STUFF.
            0770;
            0780; GOTO, THEN OR GOSUB ACTION
            0790;
037D-C8     0800FLABEL      INY
037E-A628   0810            LDX *BSTR     ;COPY START ADDRESS
0380-A529   0820            LDA *BSTR+1   ;OF BASIC.
0382-D008   0830            BNE CKSTAT    ;GO CHECK FIRST STAT.
            0840;
0384-A000   0850NXSTAT      LDY #0        ;SET ADDRESS OF NEXT
0386-B15C   0860            LDA (CLAD),Y          ;BASIC
STATEMENT.
0388-AA     0870            TAX
0389-C8     0880            INY
038A-B15C   0890            LDA (CLAD),Y
            0900;
038C-865C   0910CKSTAT      STX *CLAD     ;SETUP CURRENT
038E-855D   0920            STA *CLAD+1   ;BASIC LINE ADDRESS.
0390-855B   0930            STA *TMP2+1
            0940;
0392-B15C   0950            LDA (CLAD),Y          ;END OF BASIC
?
0394-D003   0960            BNE CKSTAT1   ;NO, CONTINUE.
            0970;
0396-4CEBC7 0980            JMP UNDEFD    ;UNDEF'D STATEMENT.
            0990;
0399-18     1000CKSTAT1     CLC  ;GET PAST NEXT BASIC
039A-A55C   1010            LDA *CLAD     ;LINE ADDRESS AND
BASIC
039C-6904   1020            ADC #4        ;STATEMENT NUMBER.
            1030;
039E-855A   1040            STA *TMP2     ;SAVE THE ADDRESS.
03A0-9002   1050            BCC =+3
03A2-E65B   1060            INC *TMP2+1
            1070;
03A4-88     1080            DEY
```

```
                1090;
                1100; SEARCH BASIC FOR MATCHING LABEL
                1110;
03A5-B15A       1120MATCH       LDA (TMP2),Y            ;CHECK IF THE
03A7-20E203     1130            JSR CORRECT   ;LABEL MATCHES THE
03AA-8559       1140            STA *TMP1     ;SPECIFIED LABEL.
03AC-B177       1150            LDA (CHAD),Y
03AE-C8         1160            INY
03AF-20E203     1170            JSR CORRECT
03B2-C559       1180            CMP *TMP1
03B4-D0CE       1190            BNE NXSTAT    ;NO MATCH FOUND.
                1200;
03B6-C900       1210            CMP #0        ;END OF LABEL ?
03B8-D0EB       1220            BNE MATCH     ;NO, CONTINUE
MATCHING.
                1230;
03BA-68         1240            PLA
03BB-68         1250            PLA
                1260;
03BC-BA         1270            TSX
03BD-BDFF00     1280            LDA $FF,X     ;MATCHING LABEL
FOUND.
03C0-C98F       1290            CMP #S.GSUB   ;GOSUB ACTION ?
03C2-D015       1300            BNE NOSUB     ;NO, THEN OR GOTO.
                1310;
                1320; STACK CORRECTION FOR GOSUB
                1330;
03C4-A578       1340            LDA *CHAD+1
03C6-48         1350            PHA
03C7-A577       1360            LDA *CHAD
03C9-48         1370            PHA
03CA-A537       1380            LDA *CLIN+1
03CC-48         1390            PHA
03CD-A536       1400            LDA *CLIN
03CF-48         1410            PHA
03D0-A98D       1420            LDA #$8D
03D2-48         1430            PHA
03D3-A9C6       1440            LDA #H,SUBRET
03D5-48         1450            PHA
03D6-A9C3       1460            LDA #L,SUBRET
03D8-48         1470            PHA
                1480;
03D9-20CDC7     1490NOSUB       JSR SETLAD    ;SET LINE ADD.
                1500;
03DC-2000C8     1510SKPLAB      JSR SKPSTT    ;SKIP STATEMENT.
                1520;
03DF-4C7600     1530NOPREFIX    JMP GOTCHR    ;BACK TO BASIC.
                1540;
                1550; LABEL CHARACTER CORRECTIONS
                1560;
03E2-C920       1570CORRECT     CMP #'
03E4-F008       1580            BEQ CORRECT1
03E6-C93A       1590            CMP #':
03E8-F004       1600            BEQ CORRECT1
03EA-C92C       1610            CMP #',
03EC-D002       1620            BNE CORRECT2
```

```
03EE-A900   1630CORRECT1   LDA #0
03F0-60     1640CORRECT2   RTS
            1641;
            1642; SYSTEM ADDRESS EQUATIONS
            1650;
            1660CLIN       .DI $36      ;BASIC CURR LINE NO
            1670BSTR       .DI $28      ;BASIC START ADD
            1680CHAD       .DI $77      ;BASIC CURR CHAR ADD
            1690CLAD       .DI $5C      ;BASIC CURR LINE ADD
            1700;
            1710GETCHR     .DI $70      ;GET NEXT CHAR ROUT
            1720GOTCHR     .DI $76      ;GET CURR CHAR ROUT
            1730;
            1740S.THEN     .DI $3E      ;STACK KEY 'THEN'
            1750S.GOTO     .DI $AC      ;STACK KEY 'GOTO'
            1760S.GSUB     .DI $8F      ;STACK KEY 'GOSUB'
            1770S.ONDO     .DI $69      ;STACK KEY 'ON.DO'
            1780;
            1790UNDEFD     .DI $C7EB    ;UNDEF'D STAT ERR
            1800SETLAD     .DI $C7CD    ;SET NEW LINE ADD
            1810SKPSTT     .DI $C800    ;SKIP REST OF STAT
            1820;
            1830ON.RET     .DI $C85F    ;ON.DO RETURN ADD
            1840SUBRET     .DI $C6C3    ;GOSUB RETURN ADD
            1850;
            1860TMP1       .DI $59      ;WORK SPACE
            1870TMP2       .DI $5A      ;WORK SPACE
            1880           .EN

HOOKUP   = 033A    LABELS   = 0347
NLABEL   = 0352    LABEL1   = 0355
CHKLAB   = 035D    SCOMMA   = 0371
FLABEL   = 037D    NXSTAT   = 0384
CKSTAT   = 038C    CKSTAT1  = 0399
MATCH    = 03A5    NOSUB    = 03D9
SKPLAB   = 03DC    NOPREFIX = 03DF
CORRECT  = 03E2    CORRECT1 = 03EE
CORRECT2 = 03F0    CLIN     = 0036
BSTR     = 0028    CHAD     = 0077
CLAD     = 005C    GETCHR   = 0070
GOTCHR   = 0076    S.THEN   = 003E
S.GOTO   = 00AC    S.GSUB   = 008F
S.ONDO   = 0069    UNDEFD   = C7EB
SETLAD   = C7CD    SKPSTT   = C800
ON.RET   = C85F    SUBRET   = C6C3
TMP1     = 0059    TMP2     = 005A
```

Commodore is now distributing computers and disks with new operating systems. These are, of course, BASIC 4.0 and DOS 2.0. But many users that have BASIC 2.0 and DOS 1.0 are asking themselves, "Should I upgrade ?".

The new operating systems offer many advantages over the old, but there are cases where upgrading may hurt more than help. This would refer to those who 1) have a working system performing without mishap, and 2) don't do any programming of their own. More specifically, this would be businesses that have aquired equipment and a custom program(s) to perform special tasks. There are suttle differences in the new systems that may cause discrepencies once upgraded. However, this does not rule out the possibility of upgrading. Higher capacity may be necessary to maintain your systems efficiency. This would mean a "forced" upgrade to the 8050 disk, which contains the new DOS, and program modification may be required.

Serious programmers, on the other hand, should consider upgrading as seriously as their programs. Some new features are:

BASIC 4.0

1. Garbage collection time has been reduced to negligible.

2. Shifted RUN/STOP loads and runs first disk file.

3. Disk error channel read automatically into DS and DS$, same as TI and TI$ read the clock. These new variables are one reason programs may require mods. See article this issue on converting.

4. PRINT# command omits line feed after carriage return on files OPENed with a logical file number less than 128; 128 or greater still sends CRLF.

5. Disk commands now included in the BASIC. Although BASIC 2.0 could handle the disk, PRINT#ing to the command channel was somewhat clumsy.

BASIC 2.0	BASIC 4.0	
LOAD"prog",8	DLOAD"prog"	
SAVE"1:prog",8	DSAVE"prog",d1	;defaults to d0
VERIFY"1:prog",8	VERIFY"1:prog",8	;no change
OPEN 2,8,6,"1:file,s,w"	DOPEN#2,"file",u8,d1,w	;defaults unit 8, omit w for read no change for USR files
CLOSE 2	DCLOSE#2,d1 ON u8	;omit "#2" and "d1" and close all files ON u8
LOAD"$1",8:LIST	DIRECTORY d1 or CATALOG d1	
PRINT#15,"N1:title,xx"	HEADER"title",d1,ixx	
'' "S1:prog"	SCRATCH"prog",d1	
'' "V1"	COLLECT d1	
'' "D1=0"	BACKUP d0 TO d1	
'' "R1:file=1:prog"	RENAME "prog",d1 TO "file",d1	
'' "C1:prog=0:prog"	COPY "prog",d0 TO "prog",d1	

Direct access disk commands do not change in BASIC 4.0 (i.e. format is still PRINT#15,"ul", b-a, b-p, etc.) but do change in DOS 2.0. (see DOS 2.0 below). Also note that the INITIALIZE command does not get keyword priveledges in BASIC 4.0. BASIC 4.0 was designed to work best with DOS 2.0 which does automatic initializes. BASIC 4.0 also has other commands that work only with DOS 2.0:

```
APPEND#2,"file",dl
CONCAT "more data",d0 TO "existing data",dl
RECORD#2, 3000, 5
```

The APPEND# command OPENs an existing file for writing. DOS 2 positions to the end of that file such that data can be "appended".

The CONCAT command concatenates one file "TO" another existing file (SEQ type files only). Concatenating was possible with the DOS 1.0 'C'opy command, but an extra sequence of scratch and rename commands would be necessary to accomplish the above:

```
DOS2    CONCAT "more data",d0 TO "existing data",dl
DOS1    PRINT#15,"C1:temporary=1:existing data,0:more data"
        PRINT#15,"S1:existing data"
        PRINT#15,"R1:existing data=1:temporary"
        PRINT#15,"S0:temporary"
```

Thanks to DOS 2.0, a single BASIC 4.0 command does it all! But remember, DOS 2.0 does the work; BASIC 4 only sends the command string to the disk command channel.

RECORD# works the DOS 2 Relative Record System. This feature of the new DOS makes it virtually indispensable!

Although the above three commands belong to BASIC 4.0, they can be simulated with BASIC 2.0, however, DOS 2.0 must be in the disk for them to work. (See article on DOS 2.0 commands from BASIC 4.0)

DOS 2.0

1. Automatic initializing.

2. "@" SAVE with replace fixed.

3. Formatting and Duplicating approximately 5 times faster.

4. Directory track and 6 other tracks have 1 less sector for 144 directory entries max and 664 blocks free max. It was felt that the recording density for DOS 1.0 diskette middle tracks was too high for reliability. DOS 1.0 diskettes will require converting to work on DOS 2.0 (see COPY command below). Although both diskette types can be read on either DOS, writing DOS 2 diskettes with DOS 1 is fatal. DOS 2 doesn't allow writing to DOS 1 disks.

5. RENAME command fixed.

6. COPY command now allows default characters. (e.g. COPY "fi*",d0 to "*",d1 would copy all files starting with "fi" on d0 to the same name on d1. Also COPY d0 TO d1 copies all files over... good for converting DOS 1.0 diskettes to DOS 2.0 diskettes)

7. "B-W" direct access commands removed; use "U2" instead. All others remain the same.

8. Sector byte zero now accessible from B-P command.

9. Error channel cleared on receiving correct command syntax. DOS 1 left the error light on until completion of a successful command (excluding LOAD"$0",8).

The Relative Record File System

Built in to the new DOS 2.0 is a filing system known as The Relative Record System. It's called Relative Record because each record is relative to another.

When a relative file (type REL on directory) is created, each record will have the same byte length. The length of the records are chosen by the user and can be any length between 1 and 254. No bytes are wasted which means, in most cases, records will span sector boundaries.

Essentially, a REL file is like an SEQ file with entry points. These entry points are stored in "side sectors" which take up space on the disk, but are transparent to the user. Each side sector can handle up to 30K with a maximum of 6 side sectors. This limits REL files to 180K, but since 2040 diskettes are 170K, a REL file could use up the whole disk. The 180K limit also applies to the 8050.

The speed of the system is incredible; maximum 3 block reads to access any record, regardless of file size.

A maximum of three REL files can be open on the disk simultaneously provided no other files are open.

The command set associated with REL files is:

DOPEN#
RECORD#
INPUT#
GET#
DCLOSE#

REL files can be COPYd, SCRATCHed, RENAMEd, etc., just like any other file. Treat them no differently than any other file, but with the same amount of respect. REL files must be DOPENd and DCLOSEd properly, using ST and DS/DS$ for file status interrogation.

First you must decide how many bytes maximum your information will need. This will be the number of bytes maximum per field plus one byte for a carriage return at the end of each field. You could save on bytes by not using carriage returns but then you must know how to split up the record into fields using MID$ upon retrieval. Once again, no more than 80 characters without a carriage return.

Once you've chosen a length or Record Size, put it in a variable, say RS. Choose a logical file number, a filename and a drive and:

```
DOPEN#6, "FILENAME",D0,L(RS)
```

You can write or read a REL file once opened. When DOPENing for the first time, the record size (RS) must be specified. After that the length need not be given. If it is, it must be the same as before else a disk error will occur and the disk will abort the open attempt.

On creating the file, the disk procedes to build records in disk RAM. These will be empty until you fill them with data. An empty record starts with CHR$(255) followed by RS-1 CHR$(0)'s. (see note 1 below)

You are now ready to store data. The DOPEN automatically positions to record number 1. After a PRINT#, the DOS will position to record 2. This means that placing multiple strings into a single record must be done using one PRINT# statement, else the strings will go into successive record numbers. Assuming R$=CHR$(13)...

```
DO        100 PRINT#6,"HELLO"R$;A$;R$;B$;R$;X%;R$;

DON'T!    100 PRINT#6,"HELLO"R$;
          110 PRINT#6,A$;R$;
          120 PRINT#6,B$;R$;
          130 PRINT#6,X%;R$;
```

This would put "HELLO" in record #1, A$ in record 2, B$ in record 3 and X% in record #4.

This could be a drawback, especially if your variables are in an array and you wish to use a loop to output all to the same record #. This brings us to the RECORD# command.

```
RECORD#LF,(RR),(PN)
```

RECORD# tells the file (LF) to position to record number RR at byte position PN within the record. The variable PN can be from 1 to 254. Variables in the RECORD# command must be enclosed in brackets. Output using a loop might look like:

```
100 PN=1
110 FOR J=1 TO NF          ;NF=number of fields
120 RECORD#6,(RR),(PN)
130 PRINT#6, FL$(J);R$;
140 PN=PN+LEN(FL$(J))+1 ;+1 for carriage rtn
150 NEXT
```

The ";R$;" in line 130 could be left off since this would be handled by BASIC.

Another method would be to concatenate the fields into one string and output:

```
100 FL$=""
110 FOR J=1 TO NF
110 FL$ = FL$+FL$(J)+R$
120 NEXT
130 PRINT#6,FL$
```

Remember... strings in memory can be length 255 max. Max REL record length is 254. If you print a string to a REL record that is longer than the record length, an OVERFLOW IN RECORD error will occur in the error channel. BUT, the first RS characters of the string will make it into the record; the rest will be lost. Should this happen, there probably won't be a carriage return at the end of the record. That doesn't matter. You will still be able to retrieve this data. As a matter of fact, carriage returns are not necessary at the end of a record, even if the data doesn't fill the record! "But why?", you ask....

REL Record Retrieval

As mentioned earlier, an empty record starts with CHR$(255) followed by RS-1 CHR$(0)'s. This is done by the DOS.

Let's say our record size is 50. If we take the characters H, E, L, L, and O, and send them into REL REC #1 starting at position 1 without a carriage return, (i.e. PRINT#6,"HELLO";) the DOS would do as it's told and put "HELLO" into REL REC #1 with no carriage return. Not too surprising, eh. However, once that's done, the DOS procedes to "pad" the remainder of the record with CHR$(0)'s; in this case 45 of 'em. The DOS is now positioned at REL REC #2.

Now let's say we position back to REL REC #1 with a RECORD#6,1 command.

The INPUT# command stops on carriage return or EOI. ST is set to 64 on EOI, otherwise ST = 0. (see note 2 for details)

If we now execute an INPUT#, the DOS sends the H, E, L, L, and O. But when the DOS sees the CHR$(0) it also sends EOI which is just as good as a carriage return. ST is set to 64 and the DOS positions automatically to the next record; REL REC #2.

The DOS would also send EOI if the character being sent was from the last position in the record. In this case the record is not full, but this means that the character in the last position doesn't have to be a CHR$(13). You can save 1 byte per record this way. For 2500 records that's almost 10 full blocks!

Back to our example, INPUT# terminated when the DOS saw CHR$(0) and sent EOI. This has further ramifications. Suppose you were to execute something like:

```
100 RECORD#6, 1, 1
110 PRINT#6,"HELLO";     ;or "HELLO";R$;
120 RECORD#6, 1, 20
130 PRINT#6,"JIM";
```

there would be CHR$(0)'s left in between "HELLO" and "JIM". "JIM" would be lost forever to INPUT#, unless you position back to it using RECORD# before INPUT#ing. Otherwise, only GET# could get it back. The DOS does not send EOI with CHR$(0) when using GET#.

Therefore, if you're anticipating blanks between data, or blank fields representing no data, it's best to construct the record in RAM first using spaces as field padding. Remember though, leading spaces will PRINT# to the disk, but INPUT# (as with INPUT) ignores them. Leading spaces include spaces at the beginning of a record and spaces immediately following a carriage return within a record.

Printover

Recall that the PRINT# command sends the characters into the record and then pads to the end of the record with CHR$(0)'s. This can be hazardous, especially if valid data exists beyond the data being sent into the record. This data would be wiped out with zeros. One more reason why you should construct the record in RAM first. You could get around this by putting the new data into the disk buffer with a "Memory-Write" routine, but that's fairly advanced and we won't cover that here.

End Of File Detection

The following routine could be used to read the entire contents of a REL file:

```
10 DOPEN#8,"FILE NAME"
20 INPUT#8,A$
30 PRINT A$
40 IF DS=50 THEN DCLOSE#8 : END
50 GOTO 20
```

On DOPENing, the file positions to record 1 and automatically positions to successive records after INPUT#ing each records' valid data. This would continue until reaching a record that hasn't yet been formatted. DS/DS$ would read 50, RECORD NOT

PRESENT. But the last record used isn't necessarily the last record formatted. (see note 1.) Storing the number of the last record used would take care of that. Give it a SEQ file of it's own and update it every time it changes using "@" write with replace.

Empty files start with CHR$(255). This gets done by the DOS initially, but if a record DELETE is done, this "empty" flag should be replaced (i.e. PRINT#1f,CHR$(255)). This available file space can then be detected for future use.

One Minor Gotcha

When a REL file is DOPENed for the first time, only one sector is allocated for data. If the file is aborted (i.e. no DCLOSE, DIRECTORY display, reset, etc.) before the DOS allocates a second data sector, the side sector information doesn't get written to the disk. That second data sector allocation forces the side sector onto the disk, but DCLOSing properly will always prevent this.

To be absolutely sure, although probably unecessary, the following routine could be used:

```
50000 DOPEN#1f,"FILE NAME",D0,L(RS)
50010 RECORD#1f,(INT(254/RS)+1)
50020 PRINT#1f,CHR$(255);
50030 DCLOSE#1f
50040 RETURN
```

The fix actually defeats its own purpose as the file is properly DCLOSEd in line 50030!

This would only have to be done once and your file is ready for I/O. Once againg, the record size (RS) need only be given in the very first DOPEN.

NOTE 1

When a REL file is created, the DOS goes looking for some RAM to use inside the disk unit; a 256 byte buffer. The first two bytes are used to store the track and sector numbers of the next sector in the REL file just like SEQ files. The remaining 254 bytes are for record space, hence the 254 byte maximum record size.

At this point the DOS fills the record space with CHR$(0)'s and puts a CHR$(255) "marker" in the first byte of each record. This byte would be a multiple of the record size. If the record size were 50, there would be CHR$(255) at bytes 2, 52, 102, 152, 202, and 252 (offset by 2 due to track & sector bytes at 0 and 1).

If REL REC #1 were currently being written to or read from, you could procede to read or write REL RECs 2, 3, 4, and 5 without any mechanical disk activity. Requesting record #6 (i.e. RECORD#1f,6,1) would return an error #50,

RECORD NOT PRESENT because disk space for a 6th record hasn't yet been formatted. But 5 records don't fill the buffer completely; there are still 4 bytes left (252-255). These belong to record #6. The next PRINT# would start putting characters into these 4 bytes, at which point the DOS would find another available scetor, stick it's co-ordinates into bytes 0 and 1, and write the buffer contents onto the diskette. Now the buffer is re-formatted with the first 46 bytes of the record space belonging to record #6. A DCLOSE would write the rest of the data to disk. Requesting record #3000 would force the DOS to format all records inbetween before allowing access to the record.

NOTE 2

1. INPUT# continues to input characters from the disk until it sees a carriage return (, comma or a colon but we'll ignore these here). The next line of your program should be a check of ST. If there is more data, ST will be 0; if not, ST will be 64. (see ST table, center page)

2. INPUT# also terminates on receiving EOI (End Or Identify). EOI has a line of it's own on the IEEE bus. INPUT# checks this line. If it turns on, then no matter what character INPUT# has just received, inputting stops and ST is set to 64.

That all sounds like a lot but it really isn't. The Relative Record System is really quite easy to work. Being new, it'll take some getting used to. Once you're storing data in REL RECS, you'll hate to think how you did it any other way!

Paul Higginbottom,
Commodore U.K. Software Department

The best way I found to convert programs, was to divide all of the programs into four catagories. These are as follows:

1. Programs written entirely in BASIC, with no PEEK, POKE, USR, WAIT or SYS statements.

2. Programs written entirely in BASIC, with PEEK, POKE, USR, WAIT and/or SYS statements.

3. Programs written partly in BASIC and in machine code, with PEEK, POKE, USR, WAIT or SYS statements.

4. Programs written entirely in machine code.

First, I would like to discuss the utilities I use when converting programs. I use BASIC AID for the BASIC conversion. This has FIND, CHANGE (something the TOOLKIT lacks), NUMBER (renumber), KILL (to exit), DELETE, and BREAK (drops you into the monitor). This is a BUTTERFIELD abbreviation of our own BASIC AID (MP096, now on sale for 10 pounds! and has 16 commands - I think), but for BASIC 4.0. Also I use SUPERMON4.REL (by BUTTERFIELD/WOZNIAK/SEILER/QUITEAFEWOTHERS) which is an add-on to the monitor commands for 4.0, allowing you to hunt for code or text, disassemble, assemble, list memory in ASCII as well as hex, step through programs with trace or step, etc. I use a disk unit for conversion, but I should think a tape user could do the same sort of thing, only slower. The memory maps mentioned below have been published and are avaialable in any one of a number of current publications.

Now I will go through each catagory, one at a time.

1. This catagory shouldn't need any conversion.

2. Let's take the POKE statements first. Apart from those used to alter the screen RAM (which stay the same), usually the corresponding locations from machine to machine can be found by looking at Jim Butterfield's memory maps, which are public domain documents. The only other problems that seem to arise, are when a location has been POKEd with a certain value to make the PET function in a different way. A good example of this is the well known one that will disable the RUN/STOP key. If you understand why it works, then conversion to BASIC 4.0 is easy. All that is necessary, is to add three to the current contents of 144. On a 2.0 PET, POKE144,49 will disable the stop key. This is three more than its normal contents (46). Therefore POKE144,PEEK(144)+3 would work on either machine. Just to save you the bother, it is in fact POKE144,88 (to disable), and POKE144,85 (to enable), on BASIC 4.0 machines.

BASIC 4.0 MEMORY MAP

Compiled by Jim Butterfield

There are some differences between usage between the 40- and 80-column machines.

Hex	Decimal	Description
0000-0002	0-2	USR jump
0003	3	Search character
0004	4	Scan-between-quotes flag
0005	5	Input buffer pointer; # of subscripts
0006	6	Default DIM flag
0007	7	Type: FF=string, 00=numeric
0008	8	Type: 80=integer, 00=floating point
0009	9	Flag: DATA scan; LIST quote; memory
000A	10	Subscript flag; FNX flag
000B	11	0=INPUT; $40=GET; $98=READ
000C	12	ATN sign/Comparison Evaluation flag
000D-000F	13-15	Disk status DS$ descriptor
0010	16	Current I/O device for prompt-suppress
0011-0012	17-18	Integer value (for SYS, GOTO etc)
0013-0015	19-21	Pointers for descriptor stack
0016-001E	22-30	Descriptor stack(temp strings)
001F-0022	31-34	Utility pointer area
0023-0027	35-39	Product area for multiplication
0028-0029	40-41	Pointer: Start-of-Basic
002A-002B	42-43	Pointer: Start-of-Variables
002C-002D	44-45	Pointer: Start-of-Arrays
002E-002F	46-47	Pointer: End-of-Arrays
0030-0031	48-49	Pointer: String-storage(moving down)
0032-0033	50-51	Utility string pointer
0034-0035	52-53	Pointer: Limit-of-memory
0036-0037	54-55	Current Basic line number
0038-0039	56-57	Previous Basic line number
003A-003B	58-59	Pointer: Basic statement for CONT
003C-003D	60-61	Current DATA line number
003E-003F	62-63	Current DATA address
0040-0041	64-65	Input vector
0042-0043	66-67	Current variable name
0044-0045	68-69	Current variable address
0046-0047	70-71	Variable pointer for FOR/NEXT
0048-0049	72-73	Y-save; op-save; Basic pointer save
004A	74	Comparison symbol accumulator
004B-0050	75-80	Misc work area, pointers, etc
0051-0053	81-83	Jump vector for functions
0054-005D	84-93	Misc numeric work area
005E	94	Accum#1: Exponent
005F-0062	95-98	Accum#1: Mantissa
0063	99	Accum#1: Sign
0064	100	Series evaluation constant pointer
0065	101	Accum#1 hi-order (overflow)
0066-006B	102-107	Accum#2: Exponent, etc.
006C	108	Sign comparison, Acc#1 vs #2
006D	106	Accum#1 lo-order (rounding)
006E-006F	110-111	Cassette buff len/Series pointer
0070-0087	112-135	CHRGET subroutine; get Basic char
0077-0078	119-120	Basic pointer (within subrtn)
0088-008C	136-140	Random number seed.
008D-008F	141-143	Jiffy clock for TI and TI$
0090-0091	144-145	Hardware interrupt vector
0092-0093	146-147	BRK interrupt vector
0094-0095	148-149	NMI interrupt vector
0096	150	Status word ST
0097	151	Which key down; 255=no key
0098	152	Shift key: 1 if depressed
0099-009A	153-154	Correction clock
009B	155	Keyswitch PIA: STOP and RVS flags
009C	156	Timing constant for tape
009D	157	Load=0, Verify=1
009E	158	Number of characters in keybd buffer
009F	159	Screen reverse flag
00A0	160	IEEE output; 255=character pending
00A1	161	End-of-line-for-input pointer
00A3-00A4	163-164	Cursor log (row, column)
00A5	165	IEEE output buffer
00A6	166	Key image
00A7	167	0=flash cursor
00A8	168	Cursor timing countdown
00A9	169	Character under cursor
00AA	170	Cursor in blink phase
00AB	171	EOT received from tape
00AC	172	Input from screen/from keyboard
00AD	173	X save
00AE	174	How many open files
00AF	175	Input device, normally 0
00B0	176	Output CMD device, normally 3
00B1	177	Tape character parity
00B2	178	Byte received flag
00B3	179	Logical Address temporary save
00B4	180	Tape buffer character; MLM command
00B5	181	File name pointer; MLM flag, counter
00B7	183	Serial bit count
00B9	185	Cycle counter
00BA	186	Tape writer countdown
00BB-00BC	187-188	Tape buffer pointers, #1 and #2
00BD	189	Write leader count; read pass1/2
00BE	190	Write new byte; read error flag
00BF	191	Write start bit; read bit seq error
00C0-00C1	192-193	Error log pointers, pass1/2
00C2	194	0=Scan/1-15=Count/$40=Load/$80=End
00C3	195	Write leader length; read checksum
00C4-00C5	196-197	Pointer to screen line
00C6	198	Position of cursor on above line
00C7-00C8	199-200	Utility pointer: tape, scroll
00C9-00CA	201-202	Tape end addrs/End of current program
00CB-00CC	203-204	Tape timing constants
00CD	205	0=direct cursor, else programmed
00CE	206	Tape read timer 1 enabled
00CF	207	EOT received from tape
00D0	208	Read character error
00D1	209	# characters in file name
00D2	210	Current file logical address
00D3	211	Current file secondary addrs
00D4	212	Current file device number
00D5	213	Right-hand window or line margin
00D6-00D7	214-215	Pointer: Start of tape buffer
00D8	216	Line where cursor lives
00D9	217	Last key/checksum/misc.
00DA-00DB	218-219	File name pointer
00DC	220	Number of INSERTs outstanding
00DD	221	Write shift word/read character in
00DE	222	Tape blocks remaining to write/read
00DF	223	Serial word buffer
00E0-00F8	224-248	(40-column) Screen line wrap table
00E0-00E1	224-225	(80-column) Top, bottom of window
00E2	226	(80-column) Left window margin
00E3	227	(80-column) Limit of keybd buffer
00E4	228	(80-column) Key repeat flag
00E5	229	(80-column) Repeat countdown
00E6	230	(80-column) New key marker
00E7	231	(80-column) Chime time
00E8	232	(80-column) HOME count
00E9-00EA	233-234	(80-column) Input vector
00EB-00EC	235-216	(80-column) Output vector
00F8-00FA	249-250	Cassette status, #1 and #2
00FB-00FC	251-252	MLM pointer/Tape start address
00FD-00FE	253-254	MLM, DOS pointer, misc.
0100-010A	256-266	STR$ work area, MLM work
0100-013E	256-318	Tape read error log
0100-01FF	256-511	Processor stack
0200-0250	512-592	MLM work area: Input buffer
0251-025A	593-602	File logical address table
025B-0264	603-612	File device number table
0265-026E	613-622	File secondary adds table
026F-0278	623-632	Keyboard input buffer
027A-0339	634-825	Tape#1 input buffer
033A-03F9	826-1017	Tape#2 input buffer
033A	826	DOS character pointer
033B	827	DOS drive 1 flag
033C	828	DOS drive 2 flag
033D	829	DOS length/write flag
033E	830	DOS syntax flags
033F-0340	831-832	DOS disk ID
0341	833	DOS command string count
0342-0352	834-850	DOS file name buffer
0353-03B0	851-4096	DOS compile string buffer
03E8-03F7	1006-1015	(80-column) Tab stop table
03FA-03FB	1018-1019	Monitor extension vector
03FC	1020	IEEE timeout defeat
0400-7FFF	1024-32767	Available RAM including expansion
8000-83FF	32768-33791	(40-column) Video RAM
8000-87FF	32768-34815	(80-column) Video RAM
9000-AFFF	36864-45055	Available ROM expansion area
B000-DFFF	45056-57343	Basic, DOS, Machine Lang Monitor
E000-E7FF	57344-59391	Screen, Keyboard, Interrupt programs
E810-E813	59408-59411	PIA 1 - Keyboard I/O
E820-E823	59424-59427	PIA 2 - IEEE-488 I/O
E840-E84F	59456-59471	VIA - I/O and timers
E880-E881	59520-59521	CRT Controller
F000-FFFF	61440-65535	Reset, I/O handlers, Tape routines

PET 4.0 ROM Routines — Jim Butterfield, Toronto

The 40-character and 80-character machines are the same except for addresses $E000-$E7FF.

This map shows where various routines lie. The first address is not necessarily the proper entry point for the routine. Similarly, many routines require register setup or data preparation before calling.

	Description
B000-B065	Action addresses for primary keywords
B066-B093	Action addresses for functions
B094-B0B1	Hierarchy and action addresses for operators
B0B2-B20C	Table of Basic keywords
B20D-B321	Basic messages, mostly error messages
B322-B34F	Search the stack for FOR or GOSUB activity
B350-B392	Open up space in memory
B393-B39F	Test: stack too deep?
B3A0-B3CC	Check available memory
B3CD	Send canned error message, then:
B39F-B41E	Warm start; wait for Basic command
B41F-B4B5	Handle new Basic line input
B4B6-B4E1	Rebuild chaining of Basic lines
B4E2-B4FA	Receive line from keyboard
B4FB-B5A2	Crunch keywords into Basic tokens
B5A3-B5D1	Search Basic for given line number
B5D2	Perform NEW, and;
B5EC-B621	Perform CLR
B622-B62F	Reset Basic execution to start
B630-B6DD	Perform LIST
B6DE-B784	Perform FOR
B785-B7B6	Execute Basic statement
B7B7-B7C5	Perform RESTORE
B7C6-B7ED	Perform STOP or END
B7EE-B807	Perform CONT
B808-B812	Perform RUN
B813-B82F	Perform GOSUB
B830-B85C	Perform GOTO
B85D	Perform RETURN, then:
B883-B890	Perform DATA: skip statement
B891	Scan for next Basic statement
B894-B8B2	Scan for next Basic line
B8B3	Perform IF, and perhaps:
B8C6-B8D5	Perform REM: skip line
B8D6-B8F5	Perform ON
B8F6-B92F	Accept fixed-point number
B930-BA87	Perform LET
BA88-BAAD	Perform PRINT#
BAAE-BAA1	Perform CMD
BAA2-BB1C	Perform PRINT
BB1D-BB39	Print string from memory
BB3A-BB4B	Print single format character
BB4C-BB79	Handle bad input data
BB7A-BBA3	Perform GET
BBA4-BBBD	Perform INPUT#
BBBE-BBF4	Perform INPUT
BBF5-BC01	Prompt and receive input
BC02-BCF6	Perform READ
BCF7-BD18	Canned Input error messages
BD19-BD71	Perform NEXT
BD72-BD97	Check type mismatch
BD98	Evaluate expression
BEE9	Evaluate expression within parentheses
BEEF	Check parenthesis, comma
BF00-BF0B	Syntax error exit
BF8C-C046	Variable name setup
C047-C085	Set up function references
C086-C0B5	Perform OR, AND
C0B6-C11D	Perform comparisons
C11E-C12A	Perform DIM
C12B-C1BF	Search for variable
C1C0-C2C7	Create new variable
C2C8-C2D8	Setup array pointer
C2D9-C2DC	32768 in floating binary
C2DD-C2FB	Evaluate integer expression
C2FC-C4A7	Find or make array
C4A8	Perform FRE, and:
C4BC-C4C8	Convert fixed-to-floating
C4C9-C4CE	Perform POS
C4CF-C4DB	Check not Direct
C4DC-C509	Perform DEF
C50A-C51C	Check FNx syntax
C51D-C58D	Evaluate FNx
C58E-C59D	Perform STR$
C59E-C5AF	Do string vector
C5B0-C61C	Scan, set up string
C61D-C669	Allocate space for string
C66A-C74E	Garbage collection
C74F-C78B	Concatenate
C78C-C7B4	Store string
C7B5-C810	Discard unwanted string
C811-C821	Clean descriptor stack
C822-C835	Perform CHR$
C836-C861	Perform LEFT$
C862-C86C	Perform RIGHT$
C86D-C896	Perform MID$
C897-C8B1	Pull string data
C8B2-C8B7	Perform LEN
C8B8-C8C0	Switch string to numeric
C8C1-C8D0	Perform ASC
C8D1-C8E2	Get byte parameter
C8E3-C920	Perform VAL
C921-C92C	Get two parameters for POKE or WAIT
C92D-C942	Convert floating-to-fixed
C943-C959	Perform PEEK
C95A-C962	Perform POKE
C963-C97E	Perform WAIT
C97F-C985	Add 0.5
C986	Perform subtraction
C998-CA7C	Perform addition
CA7D-CAB3	Complement accum#1
CAB4-CAB8	Overflow exit
CAB9-CAF1	Multiply-a-byte
CAF2-CB1F	Constants
CB20	Perform LOG
CB5E-CBC1	Perform multiplication
CBC2-CBEC	Unpack memory into accum#2
CBED-CC09	Test & adjust accumulators
CC0A-CC17	Handle overflow and underflow
CC18-CC2E	Multiply by 10
CC2F-CC33	10 in floating binary
CC34	Divide by 10
CC3D	Perform divide-by
CC45-CCD7	Perform divide-into
CCD8-CCFC	Unpack memory into accum#1
CCFD-CD31	Pack accum#1 into memory
CD32-CD41	Move accum#2 to #1
CD42-CD50	Move accum#1 to #2
CD51-CD60	Round accum#1
CD61-CD6E	Get accum#1 sign
CD6F-CD8D	Perform SGN
CD8E-CD90	Perform ABS
CD91-CDD0	Compare accum#1 to memory
CDD1-CE01	Floating-to-fixed
CE02-CE28	Perform INT
CE29-CEB3	Convert string to floating-point
CEB4-CEE8	Get new ASCII digit
CEE9-CEF8	Constants
CF78	Print IN, then:
CF7F-CF92	Print Basic line #
CF93-D0C6	Convert floating-point to ASCII
D0C7-D107	Constants
D108	Perform SQR
D112	Perform power function
D14B-D155	Perform negation
D156-D183	Constants
D184-D1D6	Perform EXP
D1D7-D220	Series evaluation
D221-D228	RND constants
D229-D281	Perform RND
D282	Perform COS
D289-D2D1	Perform SIN
D2D2-D2FD	Perform TAN
D2FE-D32B	Constants
D32C-D358	Perform ATN
D35C-D398	Constants
D399-D3B5	CHRGET sub for zero page
D3B6-D471	Basic cold start
D472-D716	Machine Language Monitor
D717-D7AB	MLM subroutines
D7AC-D802	Perform RECORD
D803-D837	Disk parameter checks
D838-D872	Dummy disk control messages
D873-D919	Perform CATALOG or DIRECTORY
D91A-D92E	Output
D92F-D941	Find spare secondary address
D942-D976	Perform DOPEN
D977-D990	Perform APPEND
D991-D9D1	Get disk status
D9D2-DA06	Perform HEADER
DA07-DA30	Perform DCLOSE
DA31-DA64	Set up disk record
DA65-DA7D	Perform COLLECT
DA7E-DAA6	Perform BACKUP
DAA7-DAC6	Perform COPY
DAC7-DAD3	Perform CONCAT
DAD4-DB0C	Insert command string values
DB0D-DB39	Perform DSAVE
DB3A-DB65	Perform DLOAD
DB66-DB98	Perform SCRATCH
DB99-DB9D	Check Direct command
DB9E-DBD6	Query ARE YOU SURE?
DBD7-DBE0	Print BAD DISK
DBE1-DBF9	Clear DS$ and ST
DBFA-DC67	Assemble disk command string
DC68-DE29	Parse Basic DOS command
DE2C-DE48	Get Device number
DE49-DE86	Get file name
DE87-DE9C	Get small variable parameter
	** Entry points only for E000-E7FF **
E000	Register/screen initialization
E0A7	Input from keyboard
E116	Input from screen
E202	Output character
E442	Main Interrupt entry
E455	Interrupt: clock, cursor, keyboard
E600	Exit from Interrupt
**	**
F000-F0D1	File messages
F0D2	Send 'Talk'
F0D5	Send 'Listen'
F0D7	Send IEEE command character
F109-F142	Send byte to IEEE
F143-F150	Send byte and clear ATN
F151-F16B	Option: timeout or wait
F16C-F16F	DEVICE NOT PRESENT
F170-F184	Timeout on read, clear control lines
F185-F192	Send canned file message
F193-F1D0	Send byte, clear control lines
F19E-F1AD	Send normal (deferred) IEEE char
F1AE-F1BF	Drop IEEE device
F1C0-F204	Input byte from IEEE
F205-F214	GET a byte
F215-F265	INPUT a byte
F266-F2A1	Output a byte
F2A2	Abort files
F2A6-F2C0	Restore default I/O devices
F2C1-F2DC	Find/setup file data
F2DD-F334	Perform CLOSE
F335-F342	Test STOP key
F343-F348	Action STOP key
F349-F350	Send message if Direct mode
F351-F355	Test if Direct mode
F356-F400	Program load subroutine
F401-F448	Perform LOAD
F449-F46C	Print SEARCHING
F46D-F47C	Print LOADING or VERIFYING
F47D-F4A4	Get Load/Save parameters
F4A5-F4D2	Send name to IEEE
F4D3-F4F5	Find specific tape header
F4F6-F50C	Perform VERIFY
F50D-F55F	Get Open/Close parameters
F560-F5E4	Perform OPEN
F5E5-F618	Find any tape header
F619-F67A	Write tape header
F67B-F694	Get start/end addrs from header
F695-F6AA	Set buffer address
F6AB-F6C2	Set buffer start & end addrs
F6C3-F6CB	Perform SYS
F6CC-F6DC	Set tape write start & end
F6DD-F747	Perform SAVE
F748-F7AE	Update clock
F7AF-F7FD	Connect input device
F7FE-F84A	Connect output device
F84B-F856	Bump tape buffer pointer
F857-F879	Wait for PLAY
F87A-F88B	Test cassette switch
F88C-F899	Wait for RECORD
F89A	Initiate tape read
F8CB	Initiate tape write
F8E0-F92A	Common tape I/O
F92B-F934	Test I/O complete
F935-F944	Test STOP key
F945-F975	Tape bit timing adjust
F976-FA8B	Read tape bits
FA9C-FBBA	Read tape characters
FBBB-FBC3	Reset tape read address
FBC4-FBC8	Flag error into ST
FBC9-FBD7	Reset counters for new byte
FBD8-FBF3	Write a bit to tape
FBF4-FC85	Tape write
FC86-FCBF	Write tape leader
FCC0-FCDA	Terminate tape; restore interrupt
FCDB-FCEA	Set interrupt vector
FCEB-FCF8	Turn off tape motor
FCF9-FD0A	Checksum calculation
FD0B-FD15	Advance load/save pointer
FD16-FD4B	Power-on Reset
FD4C-FD5C	Table of interrupt vectors
**	**
** Jump table:	
FF93-FF9E	CONCAT,DOPEN,DCLOSE,RECORD
FF9F-FFAA	HEADER,COLLECT,BACKUP,COPY
FFAB-FFB6	APPEND,DSAVE,DLOAD,CATALOG
FFB7-FFBC	RENAME,SCRATCH
FFBD	Get disk status
FFC0	OPEN
FFC3	CLOSE
FFC6	Set input device
FFC9	Set output device
FFCC	Restore default I/O devices
FFCF	INPUT a byte
FFD2	Output a byte
FFD5	LOAD
FFD8	SAVE
FFDB	VERIFY
FFDE	SYS
FFE1	Test stop key
FFE4	GET byte
FFE7	Abort all files
FFEA	Update clock
FFFA-FFFF	Hard vectors: NMI, Reset, INT

A Few Entry Points, 1.0 / 2.0 / 4.0 ROMs Jim Butterfield

Entry points seen in various programmer's machine language programs. The user is cautioned to check out the various routines carefully for proper setup before calling, registers used, etc.

ORIG	UPGR	4.0	DESCRIPTION
	C2D8	B350	Open space in BASIC text
	C328	B3A0	Check available memory
C357	C355	B3CD	?OUT OF MEMORY
C359	C357	B3CF	Send Basic error message
C38B	C389	B3FF	Warm start, Basic
	C39B	B40F	Main CHRGET entry
C3AC	C3AB	B41F	Crunch & insert line
C430	C439	B4AD	Fix chaining & READY.
C433	C442	B4B6	Fix chaining
	C46F	B4E2	Receive line from keyboard
C48D	C495	B4FB	Crunch tokens
C522	C52C	B5A3	Find line in Basic
C553	C55D	B5D4	Do NEW
C567	C572	B5E9	Reset Basic and do CLR
C56A	C575	B5EC	Do CLR
C59A	C5A7	B622	Reset Basic to start
C6B5	C6C4	B74A	Continue Basic execution
C863	C873	B8F6	Get fixed-point number from Basic.
C9CE	C9DE	BADB	Send Return,LF if in screen mode
C9D2	C9E2	BADF	Send Return, Linefeed
CA27	CA1C	BB1D	Print string
CA2D	CA22	BB23	Print precomputed string
CA47	CA43	BB44	Print "?"
CA49	CA45	BB46	Print character (output .A to device)
CE11	CDF8	BEF5	Check for comma
CE13	CDFA	BEF7	Check for specific character
CE1C	CE03	BF00	'SYNTAX ERROR'
CFD7	CFC9	C187	Find fl-pt variable, given name
D079	D069	C2B9	Bump Variable Address by 2
D0A7	D09A	C2EA	Float to Fixed conversion
D278	D26D	C4BC	Fixed to Float conversion
na	D472	FD11	Entry to m.l.m. (dec. 54386 & 64785 resp.)
D679	D67B	C8D7	Get byte to X reg
D68D	D68F	C8EB	Evaluate String
D6C4	D6C6	C921	Get two parameters
D73C	D773	C99D	Add (from memory)
D8FD	D934	CB5E	Multiply by memory location
D9B4	D9EE	CC18	Multiply by ten
DA74	DAAE	CCD8	Unpack memory variable to Accum #1
DAA9	DAE3	CD0D	Copy Acc #1 to (X,Y) location
DB1B	DB55	CD7F	Completion of Fixed to Float conversion
DC9F	DCD9	CF83	Print fixed-point value
DCA9	DCE3	CF8D	Print floating-point value
DCAF	DCE9	CF93	Convert number to ASCII string
E3EA	E3D8	E202	Print a character
na	E76A	D717	Output 4 ASCII hex chars from $FB,FC
na	E775	D722	Output .A as 2 hex digits
na	E7A7	D754	Input 2 hex digits to A
na	E7E0	D78D	Transfer 1 ASCII hex digit to A in binary
na	E7B6	D763	Input 1 hex digit to A
E7DE	F156	F185	Print system message
F0B6	F0B6	F0D2	Send 'talk' to IEEE
F0BA	F0BA	F0D5	Send 'listen' to IEEE
F12C	F128	F143	Send Secondary Address
E7DE	F156	F185	Send canned message
F167	F16F	F19E	Send character to IEEE
F17A	F17F	F1B6	Send 'untalk'
F17E	F183	F1B9	Send 'unlisten'
F187	F18C	F1C0	Input from IEEE
F2C8	F2A9	F2DD	Close logical file
F2CD	F2AE	F2E2	Close logical file in A
F32A	F301	F335	Check for Stop key
F33F	F315	F349	Send message if Direct mode
na	F322	F356	LOAD subroutine
F3DB	F3E6	F425	?LOAD ERROR
F3E5	F3EF	F42E	Print READY & reset Basic to start
F3FF	F40A	F449	Print SEARCHING...
F411	F41D	F45C	Print file name
F43F	F447	F486	Get LOAD/SAVE type parameters
F462	F466	F4A5	Open IEEE channel for output.
F495	F494	F4D3	Find specific tape header block
F504	F4FD	F53C	Get string
F52A	F521	F560	Open logical file from input parameters
F52D	F524	F563	Open logical file
F579	F56E	F5AD	?FILE NOT FOUND, clear I/O
F57B	F570	F5AF	Send error message
F5AE	F5A6	F5E5	Find any tape header block
F64D	F63C	F67B	Get pointers for tape LOAD
F667	F656	F695	Set tape buffer start address
F67D	F66C	F6AB	Set cassette buffer pointers
F6E6	F6F0	F72F	Close IEEE channel
F78B	F770	F7AF	Set input device from logical file number
F7DC	F7BC	F7DF	Set output device from LFN.
F83B	F812	F857	PRESS PLAY..; wait
F85E	F835	F87A	Sense tape switch
F87F	F855	F89A	Read tape to buffer
F88A	F85E	F8A3	Read tape
F8B9	F886	F8CB	Write tape from buffer
F8C1	F88E	F8D3	Write tape, leader length in A
F913	F8E6	F92B	Wait for I/O complete or Stop key
FBDC	FB76	FBBB	Reset tape I/O pointer
FD1B	FC9B	FCE0	Set interrupt vector
FFC6	FFC6	FFC6	Set input device
FFC9	FFC9	FFC9	Set output device
FFCC	FFCC	FFCC	Restore default I/O devices
FFCF	FFCF	FFCF	Input character
FFD2	FFD2	FFD2	Output character
FFE4	FFE4	FFE4	Get character

MNE.	IMMED	ZPAGE	ZPG,X	ZPG,Y	(I,X)	(I),Y	ABSOL	ABS,X	ABS,Y	INDAB	RELAT	IMPL.
ORA	0 9	0 5	1 5		0 1	1 1	0 D	1 D	1 9			
AND	2 9	2 5	3 5		2 1	3 1	2 D	3 D	3 9			
EOR	4 9	4 5	5 5		4 1	5 1	4 D	5 D	5 9			
ADC	6 9	6 5	7 5		6 1	7 1	6 D	7 D	7 9			
STA		8 5	9 5		8 1	9 1	8 D	9 D	9 9			
LDA	A 9	A 5	B 5		A 1	B 1	A D	B D	B 9			
CMP	C 9	C 5	D 5		C 1	D 1	C D	D D	D 9			
SBC	E 9	E 5	F 5		E 1	F 1	E D	F D	F 9			
ASL		0 6	1 6				0 E	1 E				
ROL		2 6	3 6				2 E	3 E				
LSR		4 6	5 6				4 E	5 E				
ROR		6 6	7 6				6 E	7 E				
STX		8 6		9 6			8 E					
LDX	A 2	A 6		B 6			A E		B E			
DEC		C 6	D 6				C E	D E				
INC		E 6	F 6				E E	F E				
BIT		2 4					2 C					
STY		8 4					8 C					
LDY	A 0	A 4	B 4				A C					
CPY	C 0	C 4					C C					
CPX	E 0	E 4					E C					
BPL											1 0	
BVC											5 0	
BCC											9 0	
BNE											D 0	
BMI											3 0	
BVS											7 0	
BCS											B 0	
BEQ											F 0	
JSR							2 0					
JMP							4 C			6 C		
BRK												0 0
RTI												4 0
RTS												6 0
PHP												0 8
CLC												1 8
PLP												2 8
SEC												3 8
PHA												4 8
CLI												5 8
PLA												6 8
SEI												7 8
DEY												8 8
TYA												9 8
TAY												A 8
CLV												B 8
INY												C 8
CLD												D 8
INX												E 8
SED												F 8
ASL												0 A
ROL												2 A
LSR												4 A
ROR												6 A
TXA												8 A
TXS												9 A
TAX												A A
TSX												B A
DEX												C A
NOP												E A

DS & DS$: Disk Status Variables

DS returns the CBM disk error number & DS$ returns a string consisting of the error number, error description and track & sector, if applicable.

DS	Error Description
0	OK, no error exists
2-19	Unused error messages: can occur, **should be ignored**
20	read error; block header not found
21	read error; sync character not found
22	read error; data block not present
23	read error; checksum error in data
24	read error; byte decoding error
25	write error; write verify error
26	write protect on
27	read error; checksum error in header
28	write error; data extends into next block
29	disk id mismatch
30	syntax error; general syntax
31	syntax error; invalid command
32	syntax error; command line greater than 58 chars
33	syntax error; invalid filename
34	syntax error; no filename given
39	syntax error; command file not given
50	record not present
51	overflow in record
52	file too large
60	file open for write
61	file not open
62	file not found
63	file exists
64	file type mismatch
65	no block; t,s is next available block
66	illegal track or sector
67	illegal system track or sector
70	no channels (available)
71	dir error (directory error)
72	disk full (could indicate directory full)
73	cdm dos v2 (or v2.5 for 8050); power up message, also indicates write attempt with DOS mismatch, DOS 2.0 & 2.5 only
74	drive not ready (8050 only)

Note: After files are SCRATCHed, the number of files scratched will be returned with a "files scratched" error message. This is not an error condition.

ST: The Status Word

ST returns the CBM status corresponding to the last I/O operation, whether over cassette, screen, keyboard or IEEE.

ST bit	ST value	Cassette Read	IEEE	Tape Verify and Load
na	0	OK	OK	OK
0	1		time out on write	
1	2		time out on read	
2	4	Short block		Short block
3	8	Long block		Long block
4	16	Unrecoverable read error		Any mismatch
5	32	Checksum error		Checksum error
6	64	End of file	EOI	
7	-128	End of tape	Device not present	End of tape

80 Column Screen, Line Start Addresses

Ln#	Dec	Hex	Notes
0	32768	$8000	
1	32848	8050	
2	32928	80A0	
3	33008	80F0	
4	33088	8140	
5	33168	8190	
6	33248	81E0	
7	33328	8230	
8	33408	8280	
9	33488	82D0	
10	33568	8320	
11	33648	8370	
12	33728	83C0	
13	33808	8410	
14	33888	8460	
15	33968	84B0	
16	34048	8500	
17	34128	8550	
18	34208	85A0	
19	34288	85F0	
20	34368	8640	
21	34448	8690	
22	34528	86E0	
23	34608	8730	
24	34688	8780	

8032 Control Characters

This table is a summary of the 8032 screen control functions. The ESC/RVS characters will display as lower/upper case or upper case/graphics, depending on which mode you're in. POKE59468,X (where X=12 for graphics, 14 for lower case) still changes modes without changing the gap between the lines. Notice that complimentary functions differ by 128 using CHR$. See the Commodore BASIC 4.0 manual for details on functions.

Function	CHR$(value)	ESC/RVS char.	Keyboard Combination
BELL	7	g	
GRAPHICS	142	shift n	
TEXT	14	n	BOTHShifts / "
SCROLL DOWN	153	shift y	LeftShift / TAB / I
SCROLL UP	25	y	
SET BOTTOM	143	shift o	Shift / Z / A / L
SET TOP	15	o	Z / A / L
INSERT LINE	149	shift u	Shift / RVS / A / L
DELETE LINE	21	u	RVS / A / L
ERASE BEGIN	150	shift v	Shift / TAB / ← / DEL
ERASE END	22	v	/ TAB / ← / DEL
SET/CLR TAB	137	shift i	Shift / TAB
TAB	9	i	TAB

← is the leftarrow key, not cursor right.

Window POKEs

```
Screen TOP:   224,T  where T=0 to 24
    BOTTOM:   225,B  where B=T to 24
      LEFT:   226,L  where L=0 to 79
     RIGHT:   213,R  where R=L to 79
```

If the program is entirely BASIC, then the USR and SYS commands will not be used (unless routines from the ROMs are being used). If ROM routines are being used, again memory maps are necessary.

The WAIT command is generally only used for keyboard activity: WAIT152,1 (wait for shift key), and WAIT158,1 (wait until bit 0 of the number of keypresses in the buffer is a 1; i.e wait until an odd number of keypresses > 0). The two just mentioned would be the same on 2.0 and 4.0.

The USR command would only be used if machine code was also used, but that is not covered in this catagory.

3. All hints made in catagory 2 should be observed for this catagory as well. The USR command uses bytes 1 and 2 as an indirect address to a machine code routine. The parameter in the USR command is 'floated' and put into the first accumulator. The address POKEd into the bytes 1 and 2 will obviously not need to be changed, but the actual machine code routines, will more than likely need to be changed. The routines most commonly used by USR routines are FLPINT (floating point to integer conversion for accumulator #1, and of course INTFLP (the other way round!). The corresponding locations can again be found in the Butterfield memory maps. Use FIND/POKE1/ to find the USR command set-up statements, and work out the hex address. Use SUPERMON to disassemble the USR code, and make any changes on the screen (JMP's into ROM usually). You should also know where your program starts in memory. To find this out off of a disk unit on a BASIC 4.0 machine, the following program will do:

```
10 INPUT"FILENAME";F$:INPUT"DRIVE";DR
20 DOPEN#1,(F$),D(DR):IF DS THEN PRINTDS$:GOTO60
30 GET#1,A$,B$:N$=CHR$(0)
40 AD=ASC(A$+N$)+ASC(B$+N$)*256
50 PRINT"PROGRAM STARTS AT"AD
60 DCLOSE#1
```

You may want to add a little hex converter into the program.

To resave programs that do not start at $0401/1025, you would need to drop into the monitor (SYS4 for example). Then you would need to see where your program ends by typing in .M 002A 002A <RETURN>. The contents of 002A,002B are the end of your program (LOW, HIGH). Let us say for example that .: 002A 40 1B 40 1B 40 1B 00 00 appears. To save your program onto drive 0 on disk, you would need to type:-

```
.S "0:FILENAME",08,033A,1B41
                  !      !
       Start address     1 More than necessary,
 ($033A for example)     because the monitor
                         doesn't save the last byte!
```

4. Programs written entirely in machine code usually fall into three catagories.

(i) Those that use ROM entry points, and system variables all over the place.
(ii) Those that only use system variables (keyboard usually).
(iii) Those that manage everything by themselves.

As before, I will handle each case separately.

(i) Tiresome, because usually the whole program will have to be disassembled onto paper, and the listing gone through with a pen, whilst clutching memory maps!

(ii) Shouldn't be too much trouble, since most system variables are the same.
NOTE: $97 (151) = Keyboard Matrix coordinate on graphics keyboards,
= Unshifted ASCII on business keyboards.

(iii) Will almost certainly work. Only keyboard type may cause problems.

Editor's Note:

SUPERMON4.REL and AID4 are available from all Canadian Commodore dealers as part of the Commodore Assembler Developement Pak.

Most programs will probably fall into category 1 and won't need too much conversion at all. If a program run turns suddenly quite, check for the obvious first (i.e. STOP key disable and don't forget that nasty screen POKE).

Also remember that BASIC 4.0 has reserved two more variables besides TI, TI$ and ST. These are DS and DS$; the Disk Status. Any of these on the left of an "=" sign will cause ?SYNTAX ERROR, however, they are allowed on the right. If your date or something appears as "00, ok, 00, 00" or if a variable starts acting weird then you've probably missed one.

Programs using PRINT# should also take note. The PRINT# command no longer outputs a LINE FEED after the carriage return unless the logical file # is 128 or greater. This won't need too much attention since most programmers inhibit line feeds in their PRINT# statements by following with CHR$(13); . However, if for some reason the program depends on that line feed, simply change the file numbers to 128 or greater.

One last point to bear in mind (although chances of this one surfacing are slim to nil) is the fact that strings stored in RAM now require two more bytes of overhead. This gets you the faster garbage collection. However, if your 2.0 system packs PET's RAM to capacity with a lot of good strings

(i.e. large string arrays with considerable length strings) then on 4.0 these two extra bytes per string can add up and possibly cause ?OUT OF MEMORY ERROR. Once again, highly doubtlful.

Although converting programs can be a pain, the advantages of BASIC 4.0 make it all worth it.

I really shouldn't be telling you this because Commodore does not reccommend this combination of equipment. However, there are still owners of the original 8k PETs that have upgraded to BASIC 2.0 to work disk, but can't upgrade to BASIC 4.0 because there simply aren't enough sockets on the board. BASIC 4.0 requires one ROM installed in the $B000 socket which does not exist on original machine boards.

If you have a PET/CBM that came with BASIC 2.0 (three empty sockets), I strongly reccommend that you upgrade to BASIC 4. If you bought the machine after July 1st, 1980, then the upgrade is free, so why not! The advantages of BASIC 4.0 are listed in another article in this issue.

For those of you who don't upgrade your BASIC but do upgrade your DOS, you'll have to use the PRINT#15," command to access some of the new DOS 2.0 features. Of course all of the old DOS 1.0 commands remain the same except for "B-W"; use "U2" instead.

APPEND#

This BASIC 4 command OPENs a SEQ file for appending:

```
BASIC4:   APPEND#6, "FILENAME"          ;defaults to D0,U8
BASIC2:   OPEN 6,8,4,"0:FILENAME,A"     ;,A for append
```

CONCAT

This one's quite simply a variation of the DOS1.0 Copy command. However, if sent to DOS1.0, a dos syntax error would be placed in the error channel.

```
BASIC4:   CONCAT "FILE 2",D1 TO "FILE 1",D0
BASIC2:   PRINT#15,"C0:FILE 1=0:FILE 1,1:FILE 2"
```

RECORD#

Two commands are affected here. First you need to DOPEN a relative file, specifying the length of each relative record; 50 in the following example:

```
BASIC4:   DOPEN#6,"REL FILE NAME",L50
BASIC2:   OPEN 6,8,SA, "0:REL FILE NAME,L"+CHR$(50)
```

(See BASIC 4.0 and DOS 2.0 for more on The Relative Record system, this issue).

The RECORD# command uses the logical file number, but the BASIC 2.0 artificial RECORD# command uses the secondary address (SA) that you chose in the OPEN command. In BASIC 4.0 the DOPEN command choses an SA for you.

```
BASIC4:   RECORD#6, (RR), 2          ;RR is rel rec #
BASIC2:   HI = INT(RR/256) : LO = RR-HI*256
          PRINT#15,"P"CHR$(SA+96)CHR$(LO)CHR$(HI)CHR$(2)
```

The "P" stands for Position. The command tells the DOS to position to relative record number RR. The "2" tells the DOS to position to the second character of the record before reading or writing. 96 is added to SA because that's how RECORD# does it.

This program demonstrates how to use the artificial relative record commands. BASIC 4.0 users should be able to replace them with the high level syntax.


```
1000 OPEN1,8,15:REM OPEN I/O CHAN
1100 INPUT"[CS]FILENAME ";F$
1110 CLOSE2:OPEN2,8,2,F$:REM OPEN IT
1120 GOSUB9000:REM ANY ERROR ?
1130 IFEN=0THEN1200:REM NO - GO ON
1140 IFEN<>62THENGOSUB9100:END
1150 INPUT"RECORD SIZE ";RS
1160 F$=F$+",L"+CHR$(RS):GOTO1110
1200 INPUT"READ,WRITE,END  ";A$
1220 A$=MID$(A$,1,1)
1230 IFA$="R"THEN2000
1240 IFA$="W"THEN3000
1250 IFA$="E"THEN4000
1260 PRINT"[CU]";:GOTO1200
2000 :
2005 :
2010 :REM ** READ A RECORD **
2020 :
2030 INPUT"RELATIVE RECORD NUMBER ";RR
2040 INPUT"RECORD POSITION ";PN
2050 GOSUB9200:REM POSITION DISK
2060 GOSUB9000:REM CHECK THE DISK
2070 IFEN<>0THENGOSUB9100:GOTO1200
2080 INPUT#2,A$:PRINTA$:GOTO1200
3000 :
3005 :
3010 :REM ** WRITE A RECORD **
3020 :
3030 INPUT"RELATIVE RECORD NUMBER ";RR
3040 PN=1:INPUT"DATA";A$
3050 GOSUB9200:REM POSITION DISK
3060 GOSUB9000:REM CHECK THE DISK
3070 IFEN<>0THENGOSUB9100
3080 PRINT#2,A$:GOTO1200
4000 CLOSE2:CLOSE1:END
9000 :
9001 :
9002 :REM ** READ DISK MESSAGE **
9003 :
9005 INPUT#1,EN$,EM$,ET$,ES$
9010 EN=VAL(EN$):RETURN
9100 :
9101 :
9102 :REM ** PRINT DISK MESSAGE **
9103 :
9105 PRINTEN$","EM$","ET$","ES$:RETURN
9200 :
9201 :
9202 :REM ** DOES RECORD#2,(RR),(PN)
9203 :
9205 RH=INT(RR/256):RL=RR-RH*256
9210 C$="P"+CHR$(2+96)+CHR$(RL)+CHR$(RH)
9220 C$=C$+CHR$(PN)
9230 PRINT#1,C$:RETURN
```

This is the complete list of DOS bugs compiled by Mark Clarke and Paul Higginbottom of Commodore U.K. Most aren't too rotten and we've added fixes where possible, but the ultimate fix is upgrading to DOS 2.X.

DOS Bugs Date: 25th June 1980 Revs: Jan/81

The list below is a guide to the errors that we are aware of through reports, and internal research.

Below is a list of all the release versions of the DOS and their ROM numbers.

DOS Description	ROMs
1.0 standard 2040/3040	901468-06/07 & 901466-02
2.0 upgrade 2040/3040	901468-11/12/13 & 901468-04
2.5 standard 8050	901462-01/02 & 901465-02

The DOS 2 series is designed to work with pets that have BASIC 4.0. All PETs (apart from BASIC 1.0) can run with all versions of the DOS. However certain facilities (e.g relative record capability) available in BASIC 4.0, will not work on DOS 1.0. See article on BASIC 4.0 and DOS 2.0 for details.

Dos 1.0

1. Save with replace

There are a few problems with the bam here. We suspect that the problems are side effects of other commands. Solution: scratch first, then SAVE or for those who want to be extra cautious, SAVE under a temporary file name, scratch old file then rename temporary to old file name and scratch temporary file.

2. Rename

This command fails occasionally, although disk unit gives 'ok' error message. We suspect this is caused when (i) the last directory entry has been scratched, (ii) the number of entries in the directory (including scratched entries) is a multiple of eight (i.e the directory fills an exact number of blocks), and (iii) there are other scratched files on the directory elsewhere. Solution: execute RENAME, then OPEN the file using appropriate file type parameters. Check error channel; if file exists then COPY on same drive to new file name and scratch old file name. This requires enough free blocks to accomodate both files.

3. Duplicate

If an error happens during duplicate, the disk will hang trying the problem again and again. If this duplicate command string is in an OPEN statement, then the PET also crashes, as the PET will not be operational until the file has been OPENed.

4. Sequential files

If a sequential file of 254 characters (or any multiple), is written to the disk, then an extra carriage return is added to the end of the file. This is because the disk unit starts a new block (with a CR), when the current block is full, before checking to see if end of file is reached. PET will hang if INPUT# reads to the end of the file.

5. Write protect problem

Any attempt to write data to a write protected diskette, will cause subsequent reading of this and other diskettes (with or without write protect tabs on), to do at least one write to be made to the diskette. Solution: power down disk unit or send system reset to the command channel i.e. PRINT#15,"u:" or "uj".

6. Block Allocate & Block Free

There appears to be a problem when using the commands with numeric parameters. If the parameters are in a string then command is ok.

7. Illegal track & sectors

If illegal track or sector parameters are given to the block commands, then partial overlaying of error messages results. Especially as a result of bug number 6 above.

8. Block Free

If an unallocated block is freed, the block count is automatically incremented by one, and thus an erroneous number of blocks free can be generated (more than 670 !). Solution: Validate will restore order on the diskette.

9. Validate I

If an error occurs while validation of a diskette is taking place, then the bam will be left in an intermediary state. Solution: re-initialisation of the diskette is necessary in order to restore it to a safe state.

10. Validate II

The validate command frees any sectors allocated for random access.

11. SAVE & OPEN without drive number

This causes partial updating on both drives, thus corrupting both bams.

12. DOS handling of the IEEE bus

Occasionally during multiple 'GET#'s, the disk unit transmits a data byte onto the bus, even when the PET has hold of attention. This gives the appearance that a command is being sent to the peripherals.

13. Extensive copying

We have found that after long periods of usage of the copy disk files program (for software production) that if the original diskette becomes corrupted, then the drive refuses to initialise any good copies (checked on other drives), and has to be reset to resume normal operation.

14. Memory read

The byte returned from a memory-read operation is not accompanied by EOI on the bus. This causes INPUT# to crash the pet. Solution: GET# seems safer.

15. Buffer pointer at 0

A B-P command of zero sets the buffer pointer to 1.

Dos 2.0

1. Disk full message too late

If a file is being written to the disk (see Note 1) and a disk full message occurs, then because there is no space left, it will be impossible to close the file, and recover the data. Solution: If in doubt, check error channel. If Disk Full, *COLLECT disk, format new disk and repeat.

* Diskettes, from any DOS, containing improperly closed files should be COLLECTED or Validated only! Do not SCRATCH files that show a "*" beside the file type on the directory. These files may point into good files. If it does, the scratch command will free all the blocks in the chain which might include blocks that belong to the good file. These blocks would be overwritten in subsequent operations, corrupting your good file.

2. Relative Record create without BAM update.

When a relative file is DOPENed for the first time, one sector is allocated for data and one for side sector data. If the file is aborted (i.e. no DCLOSE, DIRECTORY display, reset, etc.) before the DOS allocates a second data sector, the side sector information doesn't get written to the disk. Solution: DCLOSing properly will always prevent this. (see Relative file article under BASIC 4.0 and DOS 2.0, this issue.)

Note 1. SAVE and OPEN with replace now work. Not that write/replace writes the new file first, then scratches the old file. If this order were reversed, the old file would be scratched and the new file might give Disk Full error. Both files would be lost. This way, if Disk Full occurs, the old file remains in untouched. The remaining available blocks that were used in the write attempt can be freed with a COLLECT.

Dos 2.5

1. Disk full message too late

Same as DOS 2.0. This is not really a bug but rather something the user or programmer should be checking for. Keep an eye on the Blocks Free too.

2. Copy dX to dY abort

The 8050 aborts a drive to drive Copy (i.e. complete copy with no explicit pattern matching) with a DISK ID MISMATCH error. This occurs after the first 8 directory entries are copied over. Solution: Use Duplicate instead.

NMI is the Non Maskable Interrupt. An interrupt is a way of telling the processor that its attention is needed for something else - right now! The regular PET interrupts are generated every 1/60th second, and are used to process the clock, keyboard, stop key and so on. These interrupts can be 'shut off' by setting the interrupt mask. There is, however, another interrupt, NMI. NMI cannot be masked - that means that it is always active.

On the old PET, the NMI line is held high (off) by the hardware. If you have an old PET, there's nothing you can do. The 6502 NMI vector is at $FFFA-$FFFB. This vector is in ROM. It points to a routine in ROM at $FCFE. This routine does a jump indirect through location $94-95 in zero page. On power-up, these locations are set to point at $C389, the BASIC warm start.

So, what can we do with NMI ? Well, it can get us out of a few sticky situations with the disk. The NMI line is available on the expansion port. The port is two connectors of 50 pins each. NMI is on the front connector, on the inside. Count forwards from the break between the two connectors. NMI is the second pin. RESET is the fourth pin. If you have a RESET button which uses an alligator clip to connect to the RESET line, just move it to this pin. Otherwise, get a mini or micro size clip and connect it to NMI. Now get another lead to ground (any of the outer pins on the connector), and connect a switch between the two. Are we ready ?

Now, when you push the RESET button, you ground the NMI line, and the 6502 jumps to the BASIC warm-start. Try it - nothing spectacular, the machine just prints READY and the cursor. OK, now let's do something silly. Try WAIT32768,1,1: Normally, that's a crash. Push NMI - READY. Neat, isn't it.

At this point, we can see that NMI can recover from some crashes - but for others (processor crashes, not infinite loops) we'll still need RESET.

But now comes the interesting stuff. We can change the NMI vector at $94,95 to anything we want. If we point it at $FD17, we can use NMI to jump to the monitor at any time. Useful for machine language programs - and all you need is an RTI instruction to get back to where you were. (You could use it to try and examine BASIC while it runs, too.)

But, that's pretty tame. OK, how about having two BASIC programs available alternately. Here's how it can be done. Set up the first BASIC program in the usual place. Set its end-of-memory pointer to 1K short of half of your memory. That is, in a 32K machine, set eom to $3C00, in 8k, to $0C00. Then copy all of zero-page to the 256 bytes just after the eom pointer of this program, and the stack to the next 256. Now, set the start of BASIC to after this stuff. For 32k, that's

$3E00. Set the eom pointer to 512 bytes short of the real end-of-memory. That would be at $7E00. Now save all of 0-page into this space, and follow it with the stack.

Now, we can write a routine (in the cassette buffer) to swap the two copies of 0-page and the stack around. You'll also have to juggle the top of the stack somewhat. When you push NMI, the PC and the stack pointer go on the stack. You'll need to push the X,Y, and accumulator, too. Then do the swap, and restore X, Y, A. Then an RTI should get things rolling. Point the NMI vector (and the copies of the NMI vector) to this routine. Once all of this is debugged, we can start one of the programs running. Then push NMI, and we swap to the other program. Push the button again, and back to the other program.

I haven't done this, so I can't promise that I didn't miss something out. If anyone does implement it (and finds a use for it!), I'd like to hear.

You can also use NMI to handle some outside device. Good luck!

Editor's Note:

Henry's concept is sound. It would require some careful thought, although not much programming to accomplish. An article on this would be a likely candidate for Best Apllication award of Volume 3.

Entry points seen in various programmer's machine language programs. The user is cautioned to check out the various routines carefully for proper setup before calling, registers used, etc. This list is an update to one published in Compute.

ORIG	UPGR	4.0	DESCRIPTION
	C2D8	B350	Open space in BASIC text
	C328	B3A0	Check available memory
C357	C355	B3CD	?OUT OF MEMORY
C359	C357	B3CF	Send Basic error message
C38B	C389	B3FF	Warm start, Basic
	C39B	B40F	Main CHRGET entry
C3AC	C3AB	B41F	Crunch & insert line
C430	C439	B4AD	Fix chaining & READY.
C433	C442	B4B6	Fix chaining
	C46F	B4E2	Receive line from keyboard
C48D	C495	B4FB	Crunch tokens
C522	C52C	B5A3	Find line in Basic
C553	C55D	B5D4	Do NEW
C567	C572	B5E9	Reset Basic and do CLR
C56A	C575	B5EC	Do CLR
C59A	C5A7	B622	Reset Basic to start
C6B5	C6C4	B74A	Continue Basic execution
C863	C873	B8F6	Get fixed-point number from Basic.
C9CE	C9DE	BADB	Send Return,LF if in screen mode
C9D2	C9E2	BADF	Send Return, Linefeed
CA27	CA1C	BB1D	Print string
CA2D	CA22	BB23	Print precomputed string
CA47	CA43	BB44	Print "?"
CA49	CA45	BB46	Print character (output .A to device)
CE11	CDF8	BEF5	Check for comma
CE13	CDFA	BEF7	Check for specific character
CE1C	CE03	BF00	'SYNTAX ERROR'
CFD7	CFC9	C187	Find fl-pt variable, given name
D079	D069	C2B9	Bump Variable Address by 2
D0A7	D09A	C2EA	Float to Fixed conversion
D278	D26D	C4BC	Fixed to Float conversion
na	D472	FD11	Entry to m.l.m. (dec. 54386 & 64785 resp.)
D679	D67B	C8D7	Get byte to X reg
D68D	D68F	C8EB	Evaluate String
D6C4	D6C6	C921	Get two parameters
D73C	D773	C99D	Add (from memory)
D8FD	D934	CB5E	Multiply by memory location
D9B4	D9EE	CC18	Multiply by ten
DA74	DAAE	CCD8	Unpack memory variable to Accum #1
DAA9	DAE3	CD0D	Copy Acc #1 to (X,Y) location
DB1B	DB55	CD7F	Completion of Fixed to Float conversion
DC9F	DCD9	CF83	Print fixed-point value
DCA9	DCE3	CF8D	Print floating-point value
DCAF	DCE9	CF93	Convert number to ASCII string
E3EA	E3D8	E202	Print a character
na	E76A	D717	Output 4 ASCII hex chars from $FB,FC
na	E775	D722	Output .A as 2 hex digits
na	E7A7	D754	Input 2 hex digits to A
na	E7E0	D78D	Transfer 1 ASCII hex digit to A in binary
na	E7B6	D763	Input 1 hex digit to A
E7DE	F156	F185	Print system message

```
F0B6 F0B6 F0D2 Send 'talk' to IEEE
F0BA F0BA F0D5 Send 'listen' to IEEE
F12C F128 F143 Send Secondary Address
E7DE F156 F185 Send canned message
F167 F16F F19E Send character to IEEE
F17A F17F F1B6 Send 'untalk'
F17E F183 F1B9 Send 'unlisten'
F187 F18C F1C0 Input from IEEE
F2C8 F2A9 F2DD Close logical file
F2CD F2AE F2E2 Close logical file in A
F32A F301 F335 Check for Stop key
F33F F315 F349 Send message if Direct mode
 na  F322 F356 LOAD subroutine
F3DB F3E6 F425 ?LOAD ERROR
F3E5 F3EF F42E Print READY & reset Basic to start
F3FF F40A F449 Print SEARCHING...
F411 F41D F45C Print file name
F43F F447 F486 Get LOAD/SAVE type parameters
F462 F466 F4A5 Open IEEE channel for output.
F495 F494 F4D3 Find specific tape header block
F504 F4FD F53C Get  string
F52A F521 F560 Open logical file from input parameters
F52D F524 F563 Open logical file
F579 F56E F5AD ?FILE NOT FOUND, clear I/O
F57B F570 F5AF Send error message
F5AE F5A6 F5E5 Find any tape header block
F64D F63C F67B Get pointers for tape LOAD
F667 F656 F695 Set tape buffer start address
F67D F66C F6AB Set cassette buffer pointers
F6E6 F6F0 F72F Close IEEE channel
F78B F770 F7AF Set input device from logical file number
F7DC F7BC F7DF Set output device from LFN.
F83B F812 F857 PRESS PLAY..; wait
F85E F835 F87A Sense tape switch
F87F F855 F89A Read tape to buffer
F88A F85E F8A3 Read tape
F8B9 F886 F8CB Write tape from buffer
F8C1 F88E F8D3 Write tape, leader length in A
F913 F8E6 F92B Wait for I/O complete or Stop key
FBDC FB76 FBBB Reset tape I/O pointer
FD1B FC9B FCE0 Set interrupt vector
FFC6 FFC6 FFC6 Set input device
FFC9 FFC9 FFC9 Set output device
FFCC FFCC FFCC Restore default I/O devices
FFCF FFCF FFCF Input character
FFD2 FFD2 FFD2 Output character
FFE4 FFE4 FFE4 Get character
```

Most of us find that the WAIT statement is of limited use. Until recently, the only use I had ever found was:

```
WAIT 59411, 8, 8
```

to wait for the cassette recorder play switch. But I did find some amusing and useful applications for WAIT.

First, a quick review.

The statement WAIT I, J, K causes the value of location I to be exclusive-OR'ed with K, and AND'ed with J. If the result is 0, the process repeats until a non-zero result is obtained. Most often, only tangible results are obtained when values of J and K are powers of 2 (1, 2, 4, 8, 16, etc.) since WAIT is a bit testing function. However testing for combinations of bits can also be useful. Be very careful though... during WAIT, the STOP is not tested. If a WAIT command is in entered, be certain a non-zero will occur or else!

Obviously, most memory locations will be of very little interest with respect to WAIT. The only locations which are of interest, in fact, are those which are affected by external events. There are two sets of these: the keyboard/ cassette/ user port/ IEEE locations in E-page, and a few in zero page. It's the zero page locations I want to talk about.

GET Loops

The classic get loop is:

```
1000 GET A$: IF A$ = "" GOTO 1000
```

which loops until a non-null input is received. The same effect can be obtained by WAITing for the keyboard buffer pointer:

```
1000 WAIT 158, 127: GET A$
```

This waits until the keyboard buffer count (decimal 158 for new ROM, 525 for old) is non-zero. It's a little harder to understand, but shorter and probably slightly faster. For experimentation, try replacing the GET command with INPUT and the 127 with 2, 4 and 8.

WAITing for a key

Very often, a GET loop is used on a "Push Any Key To Continue" basis. One interesting alternative is to use:

```
WAIT 152, 1
```

This waits for the shift key to be pushed (old ROM, 516). The advantage is that nothing is put in the keyboard buffer, so that you need not clear the buffer.

Or, if you want to have fun, try experimenting with WAITing for location 151 - key held down (515, old ROM). WAIT 151, 127, 255 will wait for any key. Specific keys are harder to WAIT for, since WAIT will only wait on one bit at a time. Remember that we're talking about un-decoded keyboard values here.

WAITing for the Clock

The real time clock occupies locations 141-143 in zero page. WAITing for one particular bit in the clock to change state will give an interesting delay effect. For example, WAIT 142, 1, 1 will wait for the rightmost bit of the second byte. This bit changes state every 256 jiffies, or 4 and a fraction seconds. WAIT 143, 1, 1 will wait till the start of the next jiffy.

While some of these are not particularly useful, playing with the WAIT statement is quite a bit of fun. If anyone finds any more useful or interesting locations, I'll be WAITing to hear from you.

8032 Control Characters

This table is a summary of the 8032 screen control functions. The ESC/RVS characters will display as lower/upper case or upper case/graphics, depending on which mode you're in. POKE59468,X (where X=12 for graphics, 14 for lower case) still changes modes without changing the gap between the lines. Notice that complimentary functions differ by 128 using CHR$(. See the Commodore BASIC 4.0 manual for details on functions.

Control Function	CHR$(value)	ESC/RVS char.
BELL	7	g
GRAPHICS	142	shift n
TEXT	14	n
SCROLL DOWN	153	shift y
SCROLL UP	25	y
SET BOTTOM	143	shift o
SET TOP	15	o
INSERT LINE	149	shift u
DELETE LINE	21	u
ERASE BEGIN	150	shift v
ERASE END	22	v
SET/CLR TAB	137	shift i
TAB	9	i

The above describes the special 80 column screen control functions. The functions can be activated two ways; by using CHR$(and the appropriate value or, preferably, by placing the appropriate character in reverse field within quotes. This is done by entering quote mode, hitting 'ESC', then 'RVS' and the character. For example, to do a Scroll Down enter quote mode and type 'ESC', 'RVS', shift & 'Y' and RETURN. 'ESC' takes you out of quote mode. If you wish to continue with more characters following the Scroll Down you'll have to do an OFF/RVS, another quote and DELete the quote. This is comparable to the cursor control characters but not quite so automatic.

Although you could use the CHR$(values, the ESC/RVS method saves bytes and will eventually become much more legible. After all, when was the last time you used a CHR$(17) to do a cursor right. (or is it a cursor up?... or is 17 delete?... no, I think it's a cursor down... I'd better check... hmm)

There is still another way to activate these functions without using PRINT. This is directly from the keyboard. But you say "There is no key on the keyboard assigned to do a scroll down or set top...". By pressing certain key combinations simultaneously, the keyboard value that is passed to the operating system will be the CHR$ value that activates the function. This information was published by Roy Busdiecker in Compute #7, but Roy found many combinations that do the same functions. I've listed only the easiest ones to remember.

Control Function	Key Combination
TEXT	BOTHShifts / "
GRAPHICS	
SCROLL DOWN	LeftShift / TAB / I
SCROLL UP	
SET BOTTOM	Shift / Z / A / L
SET TOP	Z / A / L
INSERT LINE	Shift / RVS / A / L
DELETE LINE	RVS / A / L
ERASE BEGIN	Shift / TAB / leftarrow / DEL
ERASE END	/ TAB / leftarrow / DEL
SET/CLR TAB	Shift / TAB
TAB	TAB

The two empty spaces beside TEXT and SCROLL UP are empty because they haven't been found yet. If anyone does, please let me know.

The window can also be POKEd to size. The pokes are:

```
Screen TOP:   224,T  where T=0 to 24
    BOTTOM:   225,B  where B=T to 24
      LEFT:   226,L  where L=0 to 79
     RIGHT:   213,R  where R=L to 79
```

I'm not sure what weird or interesting effects you can get by making TOP less than BOTTOM or LEFT greater than RIGHT. This is handled by the 6845 Screen Controller chip. The 6845 does all kinds of neat things which we'll cover in a future Vol 3 Transactor.

A halt-scroll key has been added to the 8032. LIST a fairly long program and touch the ":" key. To restart scrolling, hit the left arrow key which is also the slow-scroll key.

ESCape quite simply escapes you from quote mode or insert mode (where cursor keys get displayed as reverse characters).

SYS 54386 is the command to Call the monitor rather than break to the monitor which can be done with SYS4.

POKE 144,88 disables the STOP and the clock. POKE 144,85 enables.

To clear the window hit or PRINT 2 HOMEs consecutively. If a "window reset disable" were desired, it would be easy enough to insert a pre-interrupt routine to zeroize the home count ($E8) so that the 8032 would never see 2 HOMEs in a row. The code would be LDA #0, STA $E8, JMP (the IRQ vector). Enter it fast with these steps:

1. Enter m.l.m. with SYS4
2. Type: m 027a 027a
3. .: 027a a9 00 85 e8 4c 55 e4 00
4. Now take the cursor up and change the IRQ vector to 027a <RETURN>
5. Exit the mlm with x <RETURN>
6. Set a window with the key combination (above)
7. Just try and clear it!

Best use for this would be for bulletproof INPUT. The program would set the window to one screen line with rightwindow - leftwindow = max input length. Then OPEN 1,0 (input file from the keyboard) and use INPUT#1,A$. This way, no question mark is printed and hitting RETURN with no data input doesn't break out of the program. The window could not be cleared by the user either thanks to the pre-interrupt. Wella!...failsafe keyboard input!

Over the past 18 months, the TPUG has been meeting at Sheridan College's Oakville Ontario campus. On behalf of Commodore, the club executive and the members, I'd like to thank Sheridan for providing an excellent facility. Special thanke to Frank Winter, Ted Bangay, Margo Martin and Dave Langden for making it all possible.

The TPUG has now moved its meeting location to Leaside High School (south side of Eglinton Ave E., just east of Bayview Ave.) Meeting dates are scheduled for February 11, March 11, April 8, May 13 and June 10.

The TPUG is also compiling a library of programs. These programs have been categorized into Machine Language and Utilities, Music, Games, Education, Math and Science, Medical, Business, Ham Radio, Telecommunications and Miscellaneous. The disks will be made available to club members only (regular or associate members) at $10.00 each. See Transactor #11 for membership particulars.

Two disks are available now. Their directory listings are shown here.

Should any other PET clubs wish to make announcements in The Transactor, please don't hesitate to send them in!

Disk Drive No. 0: best michigan 1

universal wedge
qubic.alt
keno
mousemaze
kingdom/pics
quandry
dragon.maze!
clouzot!
snake.alt
spade.instructs
magic.square
spades
anti-air/bus
battleship.alt2
billiards!
clue
dog.star.adven
dominoes
draw.poker
dungeon 1.4
dungeon.alt3
m.b.instructions
madman.race
mille bourne
dice.pig
startrek.alt4
find.color
craps.odds
tank.war.alt
horserace
snowflake
wumpus.alt
listener

Blocks Free

5

Disk Drive No. 1: tpug-m
tpug-m.l. 1

universal wedge
copy all
supermon 1.rel
supermon1 ins
supermon 2.rel
supermon 4.rel
supermon2/4 ins
extramon9g)$1000
extramon9b)$1000
extramon inst
append/renum.rel
rom test--btfld
trace.rel(basic)
ramtest)$500
screen print
un-new/sys826
keysort2$7454
keysort2-2demo
keysort2-1demo
keysort2$1c54
low case list
disk append
disk mod/v1
disk id corrctor
disk peek
view bam
block get 1.0
bl get)$033a
keyprint/826
disk name (r)
copyprog
keymake
copydisk/sys973
tape test No.
tape write (No.)
copycat!sys934
copycat'sys934
disk logger
catalog
search
utinsel.rel
aid4
compactor
cassette.to.disk
datamaker
keysort.exe16/32
keysort.demo1
keysort.demo2
keysort.exe8k
cross-ref
basic.aid.exe

Blocks Free

283

XDOS 2.2 is a "souped up" DOS Support (plus a number of other neat commands) for PETs with BASIC 2.0. It comes on an EPROM that can be installed in any of the three empty sockets, depending on which XDOS you buy:

XDOS 2.2-9	goes in socket	UD3
XDOS 2.2-A	goes in socket	UD4
XDOS 2.2-B	goes in socket	UD5

Easy-to-follow documentation included

XDOS initialization is done with a SYS. XDOS can also live with other interrupt driven software. As long as XDOS is started last, any previous IRQ vector tampering won't be disabled.

Command Summary

MENU

The MENU command displays a maximum of 36 directory entries (one drive at a time) starting with drive 0. Very pretty! Hitting SPACE continues to display directory entries from drive 0 to the end of the directory. SPACE again procedes to display drive 1. STOP clears the screen and returns to BASIC.

All MENU commands default to both drives, however drive 0 or 1 only can be specified by following the command with the drive number.

As directory entries are displayed, they are assigned a number from 0 to Z. Hitting the corresponding character hilights the filename, then LOADs and RUNs the program. Of this would not work for SEQ files, which brings us to...

DMENU

DMENU does a directory listing just like MENU, but selecting a SEQ file will print file content to the screen. Preceding with an asterisk (i.e. *DMENU) would send the file contents to the printer.

CMENU

Copies multiple files from on drive to another. Quick and easy to use. Simply select the corresponding characters and hit return.

SMENU

Works the same as CMENU, but for scratching files.

Printer Support

XDOS has some really friendly hard copy commands. Each is a variation using the "*".

Screen Print

Typing a ">*" <return> enables screen print. Following this, pressing in order and holding down the keys 'SHIFT' 'RVS' and '*', dumps the screen to the printer, even while a program is running!

Preceding any output type command with a "*" has the same effect as typing:

```
OPEN253,4:CMD253 : (your commands) : PRINT#253:CLOSE253
```

Some examples:

```
*LIST
*FOR J=32 TO 95 : ?J,CHR$(J) : NEXT
```

Machine Language Monitor Access

Typing a "." <return> drops you into the m.l.m. Following the "." with your mlm command will exucute the mlm command right from BASIC. No need for SYS4.

Disk Fix

XDOS will prevent some deadly DOS 1.0 bugs, namely SAVE with replace and SAVE or OPEN without a drive number. XDOS intercepts these commands before they are executed.

In addition, XDOS includes all the DOS Support commands; > or @, / (load) and the uparrow (load & run).

At present XDOS 2.2 works on BASIC 2.0 machines only. A new XDOS is in the works for BASIC 4.0 and should be available by March/81.

Cost is 97.50 + 2.50 for postage, etc. Contact:

Prominico Industrial Electronics
1921 Burrard St.
VANCOUVER, B.C.
V6C 2J3
604-738-7811

Index Transactor #12