commodore

The Transactor

comments and bulletins concerning your COMMODORE PET™

VOL 2

BULLETIN # 10

REMAINDER$

One little known use of the MID$ function is "remainder string". If the third parameter of the MID$ function is omitted the resulting string will be every character to the right of the specified start position for the string being operated on. For example:

```
1. A$ = "123456789"
2. B$ = MID$ ( A$, 2, 4 )    ;equals "2345"
3. B$ = MID$ ( A$, 2 )       ;equals "23456789"
```

This is not the same as RIGHT$ as this function returns an absolute number of characters starting from the rightmost position. This application works best when the right-hand portion of a string is wanted and the string length is not known.

BASIC 4.0 Preliminary Note

BASIC 4.0 ROMs for the 40 column PET are on their way! The main differences are:

1. Faster garbage collection
2. Disk commands included in BASIC

Of course most SYStem calls to ROM will require modification but PEEKs and POKEs should remain valid except for some locations that may have been labelled unused in BASIC 2.0. More on BASIC 4.0 in a later issue. Also see Jim Butterfield's new BASIC 4.0 memory maps, this issue.

All BASIC 2.0 programs will run on BASIC 4.0 except for one minor gotcha. BASIC 4.0 has reserved two more variables for it's own use; DS and DS$. When called, DS will contain the error number from the disk and DS$ will return the error number, description, track and sector much like hitting ">" and return with DOS Support. The same rule applies to DS and DS$ as ST, TI and TI$; they must not appear on the left of an "=" sign. If they do a ?SYNTAX ERROR will result. So if your programs use either of these two new reserved variables, it would be a good idea to change them before RUNning on BASIC 4.0. This could be easily done by running your programs through Jim Butterfield's Cross-Ref program from Transactor #9, Vol 2.

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

ID Changer

COMPUTE magazine, issue #5, published an article that allows the user to change the ID of a diskette. This can cause irreparable damage to your disks! The program changes only the the ID that gets printed with the directory. However, the ID precedes every sector on the disk and these do not get changed. An update will be published in the next COMPUTE but this early warning will be appreciated by some I'm sure.

Printer ROMs

Recent deliveries of Commodore printers have been released with the 04 ROM. Though this ROM fixes existing 03 ROM bugs, it has a tendency to lock into lower case, inhibiting upper case character printing. This happens after sending to secondary address 2 (receive data for format). Commodore has discontinued the 04 printer ROM and until the 08 ROM is released (sometime in the fall) the following software fix will prevent this bug from appearing. Lines 30 and 40 insert a 25 jiffy delay prior to OPENing the format channel:

```
10 OPEN 4, 4, 0
20 PRINT#4, "HELLO"
30 T = TI
40 IF TI - T < 25 THEN 40
50 OPEN 5, 4, 2
60 PRINT#5, "  AAA 999     ...etc.
```

This bug can also be used to your advantage i.e. for LISTing to the printer in lower case which was, in most cases, impossible on printers containing an 03 ROM. There is, however, an easier way of implementing it:

```
100 OPEN 7, 4, 7 : PRINT#7 : CLOSE 7
```

...puts the printer in lower case mode. Power down and up gets you back to upper case and graphics.

PRINT Speed - Up

In Transactor #2, Vol 2, a POKE was published that made PRINT to the screen much faster than normal. On recent machines this POKE can not only cause the machine to crash but may also result in internal damage! Avoid including this in your programs...especially those that you may want to RUN on other peoples machines. Software portability is very important, particularly business software. If your package crashes your clients machine, you may find yourself in a very embarassing situation.

Verbatim MD 577 Super Minidisk

In the past Commodore has frowned on the use of Verbatim diskettes for the 2040 floppy disk, particularly the MD

525-16. Verbatim recognized the problems with their disks and have improved the quality substantially. Result: The MD 577 Super Mini.

First, the thickness of the jacket PVC material has been increased from 7.5 to 8 mils giving the disks greater rigidity.

Secondly, the lamination pattern, which secures the inner lining to the jacket, was redesigned to eliminate potential "pillowing" problems. "Pillows" are minute raised areas on the lining surface which can interfere with the sideways movement of the disk.

Most importantly though, the new Verbatim MD 577s are provided with a factory installed "hard hole" or hub reinforcement ring, thus creating better centering ability and reducing the possibility of hub damage. Coincidentally, the performance of almost any diskette can be substantially improved by adding a hub ring prior to formatting.

Part of the problem was also the boxes they were packaged in, which put creases in the front two or three disks. These are no longer used.

We have tested the Verbatim 577s and found them to be of quite high quality. We've also decided to use them for distributing Commodore software which should appear on the market this fall.

Controlling Garbage Collections

Henry Troop,
Toronto, Ont.

We all know that the PET garbage collection can take an annoyingly long time. One highly frustrating time for a garbage collection to happen is while you are executing a GET loop input from the keyboard. There you are, typing away, and suddenly the cursor is still flashing at you, but no inputs are accepted.

To avoid this, we'd like to force an early garbage collection, at the start of the input, but only if it would have happened anyway.

First things first. A GET loop is very productive of garbage collections because it uses lots of memory. The typical form of this loop is:

```
10 GET A$: IF A$ = "" THEN 10
20 B$=B$+A$
```

What this does is create a set of partial strings. If the input is 'Mary had a little lamb', then the strings are:

```
M
Ma
Mar
Mary
and so on to
Mary had a little lam
Mary had a little lamb
```

That's a lot. Exactly how much ? We could count the number of characters and sum the numbers from 1 to n, but a rule of thumb is n squared over 2. (A more exact figure is (n squared + n)/2) For 22 characters, the memory used is 242 bytes. For 80 characters, it's around 3240 bytes.

So, what can we do about it. Well, we need some way of determining the free memory space. FRE(0) will do this - but it will cause a garbage collection, and we don't really want one yet. Let's define a function, FNFR(X):

```
1 DEF FNFR(X) = PEEK(48) + 256*PEEK(49) - (PEEK(46) +
256*PEEK(47))
```

That's simply the distance between the beginning of strings and the end of arrays. The argument is a dummy, just like FRE(X).

Our test then is:

```
5 IF FNFR(X) < (L*L)/2 THEN Q = FRE(0)
```

where L is the anticipated maximum string length.

One peculiarity of FNFR is that the statement:

PRINT FNFR(0)-FRE(0) is almost never the same as:

PRINT FRE(0)-FNFR(0) which is always 0.

EASTERN HOUSE SOFTWARE Assembler Language Macro Packages

- Graphics Drawing Compiler
- PET Music and Sound Composer

The two macro packages have similar requirements and content as follows:

Requirements

- ASSM/TED, Eastern House Software's assembler and text editor. This in turn requires:
- 16K RAM
- BASIC 2.0 ROMs

Content

- a library of macros. This includes several general purpose macros as well as those specific to the topic.
- complete and useful documentation
- patches to enhance ASSM/TED

Recommendation

If you are really into assembler language, and have ASSM/TED, you should buy the Graphics Compiler. This will give you a 'cookbook' on how to write macros, some very handy enhancements to the assembler, and the macros themselves.

Given the similarities, it would seem appropriate to obtain the package which is of greater interest to you - music or graphics. However, I believe that the Music Composer has a significant limitation: it supports only CB2 sound. Since CB2 can be driven entirely adequately from BASIC, do it in BASIC. The raw speed and exactness of timing which are the main benefits of using assembler are not required, so simplicity should be most important. Besides, if you want real music, you should be using the entire parallel port and a digital-to-analog converter. This is not really difficult to do, and the results are worth it. Keep in mind that the above comments reflect the attitudes and prejudices of one person. Just do not expect more than CB2 control from the Music Composer.

The remainder of this review concentrates on the Graphics Compiler.

What You Get

The graphics package includes a cassette with the macro library and an example program, and some excellent documentation. This includes:

- Genreral introduction
- Instruction Set - description of the 39 macros
- Enhancements - description of the additions to ASSM/TED
- Operation - how to use the package (mechanics)

- Useful details - some programming suggestions
- Adding your own macros - with a non-trivial example
- Instruction set summary
- Combining machine-language and BASIC programs
- Graphics Compiler source listing
- patches for the assembler

Instruction Set

```
ADD         - single-byte add
SUB         - single-byte subtract
BEGIN       - initialization and all subroutines
BELL        - make a beep on CB2
CLEAR       - clear screen from current cursor position to end
HOME        - home the cursor
DO          - loop the number of times specified in a variable
END         - terminate a DO loop
DEFINE      - set a variable to a specified value
DRAWD       - draw a line - down
DRAWL                     - left
DRAWR                     - right
DRAWU                     - up
GRAPHN      - set upper/lower case mode
GRAPHY      - set graphics mode
INPUTB      - input a byte as two hexadecimal digits
INPUTC      - input one ASCII character
JUMP        - jump - unconditional
JUMPE                - if a (one byte) variable contains zero
JUMPG                - if a variable is positive
JUMPGE               - if a variable is zero or positive
JUMPL                - if a variable is negative
JUMPLE               - if a variable is negative or zero
JUMPN                - if a variable is not zero
OUTPUTB     - output a byte as two hexadecimal digits
OUTPUTC     - output one ASCII character
PRINT       - print a text string
POSABS      - position the cursor to a specified location
POSREL      - move the cursor a specified number of rows and
              columns
REVRSY      - set reverse video on
REVRSN      - set reverse video off
SETA        - set the predefined variable A to a literal value
SETAB       - set A and B (note: A, B, C, and D are defined in
              BEGIN)
SETABC      - set A, B, and C
SETABCD     - set A, B, C, and D
VECTUR      - draw a diagonal line - up and right
VECTUL                             - up and left
VECTLR                             - lower right
VECTLL                             - lower left
```

There are also two undocumented macros, WAIT and SCROLL, as well as a number of internal macros used by begin.

The commands do provide some capabilities which are not found in BASIC (as single statements), although there is nothing radical. I would appreciate some form of ARC function, even if it meant invoking a subroutine in ROM to do the trigonometry, and the compatibility problems which

result. As well, graphs made with the quarter-squares (considering the screen to be an area of 80 by 50 positions) are very nice (and slow in BASIC) but not supported.

Assembler Enhancements

There are three additions to the assembler itself. These are:

- BUILD - a new command with three operands:
  - MACROS - to seal off your macros so that they are unaffected by NUMBER, EDIT, PRINT, etc.
  - LIBRARY - to seal off a library of external definitions (eg. page zero locations)
  - CLEAR - to allow you to modify macros or library

- FORMAT (CLEAR or SET) n - where 'n' is the maximum label length from 1 to 31 (default 10). This will clean up the listing if you have a narrow printer, and on the screen.

- allowing you to return to BASIC from ASSM/TED. Since the assembler uses the first 64 bytes of page zero, it is normally not possible to return to BASIC. The patch maintains a page-zero swap area, allowing you to go back and forth from one to the other. It can be extremely useful for testing a routine which is to be called from BASIC.

The enhancements come in hard copy form. You are required to enter a 240-byte routine using the machine language monitor, then single instructions at seven locations within the assembler itself. Memory required by the assembler thus goes up by two pages.

Summary

As stated earlier, I recommend the Graphics Compiler if you are a serious user of ASSM/TED, whether you are interested in graphics or not. The examples of macro definitions, several of which have nothing to do with graphics, and the additions to the assembler have significant value. The fine documentation and actual graphics support could well be treated as a bonus.

More On Screen Print

John McDonald
Wawa, Ont.

I found Jim Butterfield's machine language Screen Print Routine (Transactor #5) very useful in a program I am developing. But in order to stretch the forty columns on the screen to eighty columns on the printer I have added an enhancement.

The change is quite easy.

Method #1 using Supermon1.0

1. load the screen print routine code,

2. use command '.T 0359 03B3 035E' to open up 5 bytes in the code at $0359,

3. use command '.M 0359 035E' and change
   '.: 0359 A9 11 AE 4C E8 A9 11 AE' to
   '.: 0359 A9 01 20 D2 FF A9 11 AE',

4. use command '.M 03B0 03B7' and change 'A6' at $03B0 to 'A1',

5. use command '.S "dn:name",dv,033A,03B9'.

Method #2 using the Basic Loader for the code

1. load the screen print routine basic loader,

2. change 947 in line 100 to 952,

3. add ',1,32,210,255,169' to the end of the DATA statement at line 230,

4. change 166 at the end of line 330 to 161,

5. save the modified program.

This modification sends a control character (CHR$(1) as per the above modification) to the printer after every carriage return.

To use the screen print routine simply use 'SYS826' in your code. To change or ensure the mode of the routine just use 'POKE858,1 or 129' before the SYS826 command. For 'enhanced' mode, use '1': for 'unenhanced' mode, use '129'.

True ASCII Output

Henry Troup,
Toronto, Ont.

We are all aware that the PET does not use true ASCII coding internallly. However, many of us have printers that do use real ASCII. In order to get upper and lower case operation, some code conversion is needed.

In this article, I shall present two ways of doing the conversion: one in BASIC, and one machine language. Both operate by a table lookup. This has the advantage that any other code conversion (to screen poke, Baudot or teletype code, for example, or ISO, or EIA, or what have you) can be had simply by changing the table. Or, a simple conversion to lower case can be had by ANDing each byte with 127.

I personally keep the conversion table in a disk file. It is appended at the end of this article.

First, the BASIC method. We dimension an integer array, M%(255), and use it as the table. Then we assign the string to be converted to S$.

```
1000 REM CONVERSION ROUTINE
1010 M$="" : IF S$= ""  THEN 1050
1020 FOR I = 1 TO LEN(S$)
1030 M$ = M$ + CHR$ (M%(ASC(MID$(S$,I))))
1040 NEXT I
1050 RETURN
```

This is slow, but tolerable if you're not doing too much conversion. It uses 519 bytes for storage of the table, and needs an available space of about five times the length of the string for working storage (it will work with less, but garbage collections will cause delays).

Now, the machine language method. This is faster and uses less storage. Here is the assembler listing. This program operates on the variable after the SYS. You must set up the table (anywhere you can get 256 bytes of free memory), and move the BASIC pointers. Then you can call the program.

```
                        ;convert2.src
                        ;convert petascii to true
                        ;ascii by lookup
            sl = $dd    ;a convenient place to put
                        ;the pointer (used in tape i/o)
            ts = $7f00  ;start of table
            va = $44
.skip
             * = 826
.skip
            lda sl
            pha
            lda sl+1
            pha
            jsr $cdf8   ;check comma
            jsr $cf6d   ;find variable
            lda $07     ;check type
            bne start
            jmp $cc9a   ;type mismatch error if numeric
.skip
 start      cpx #$00    ;check for null string
                        ;or undefined variable
            beq null
            ldy #$02
            lda (va),y  ;ptr lo
            sta sl+1
            dey
            lda (va),y  ;ptr hi
            sta sl
            dey
            lda (va),y  ;length
            tay
            beq null
            dey
 loop2      lda (sl),y
                        ;any character handling routine
                        ;can be substituted for the
                        ;next lines
            tax
            lda ts,x    ;do table lookup
            sta (sl),y  ;put back in string
            dey
            cpy #$ff    ;test for end
            bne loop2
 null       pla         ;restore zero page
            sta sl+1
            pla
            sta sl
            rts
.end
 ;to use this routine:
 ;sys 826,(string variable)
 ;the converted string ]Lis returned into the original
space
 ;note: if the variable is defined in text, it will be
changed in text !
 ;string array variables work, except for the 0th element
 ;undefined variables are taken as nulls.
 ;undimmed arrays will be created
```

And as a basic loader: (which locates the table from the top of memory pointer)

```
10 DATA 165, 221,  72, 165, 222,  72
15 DATA  32, 248, 205,  32, 109, 207
20 DATA 165,   7, 208,,   3,  76, 154
25 DATA 204, 224,   0, 240,  31, 160
30 DATA   2, 177,  68, 133, 222, 136
35 DATA 177,  68, 133, 221, 136, 177
40 DATA  68, 168, 240,  14, 136, 177
45 DATA 221, 170, 189,  -1,  -2, 145
50 DATA 221, 136, 192, 255, 208, 243
60 DATA 104, 133, 222, 104

1000 FOR X = 826 TO 914:READ P
1010 IFP = -1 THEN P = PEEK(54):REM RELOCATE TABLE
1020 IFP = -2 THEN P = PEEK(53)
1030 POKE X,P :NEXTX
```

A Sample Initialization:

```
10 POKE53,PEEK(53-1):CLR:REM MOVE TOP OF MEMORY
20 OPEN4,4:GOSUB1000:REM GET PROGRAM
40 OPEN5,8,5,"CONVERT,S,R":REMM GET TABLE FROM DISK
50 FORX=0TO255:INPUT#5,M%:POKEPEEK(53)+X,M%:NEXTX:CLOSE5:REM
PUT TABLE IN
60  S$="THIS  IS  A  TEST":SYS826,S$:PRINT#4,S$:REM  ACTUAL
CONVERSION
```

This is much faster, and needs only the 256 bytes to store the table. The conversion table follows:

```
1000 data   0,   1,   2,   3,   4,   5,   6,   7,   8,   9
1010 data  10,  11,  12,  13,  14,  15
1020 data  16,  17,  18,  19,  20,  21,  22,  23,  24,  25
1030 data  26,  27,  28,  29,  30,  31,  32,  33,  34,  35
1040 data  36,  37,  38,  39,  40,  41,  42,  43,  44,  45
1050 data  46,  47,  48,  49,  50,  51,  52,  53,  54,  55
1060 data  56,  57,  58,  59,  60,  61,  62,  63,  64,  97
1070 data  98 , 99, 100, 101, 102, 103, 104, 105, 106, 107
1080 data 108, 109, 110, 111, 112, 113, 114, 115, 116, 117
1090 data 118, 119, 120, 121, 122,  91,  92,  93,  94,  95
1100 data  96,  97,  98,  99, 100, 101, 102, 103, 104, 105
1110 data 106, 107, 108, 109, 110, 111, 112, 113, 114, 115
1120 data 116, 117, 118, 119, 120, 121, 122, 123, 124, 125
1130 data 126, 127, 128, 129, 130, 131, 132, 133, 134, 135
1140 data 136, 137, 138, 139, 140, 141, 142, 143, 144, 145
1150 data 146, 147, 148, 149, 150, 151, 152, 153, 154, 155
1160 data 156, 157, 158, 159, 160, 161, 162, 163, 164, 165
1170 data 166, 167, 168, 169, 170, 171, 172, 173, 174, 175
1180 data 176, 177, 178, 179, 180, 181, 182, 183, 184, 185
1190 data 186, 187, 188, 189, 190, 191, 192,  65,  66,  67
1200 data  68,  69,  70,  71,  72,  73,  74,  75,  76,  77
1210 data  78,  79,  80,  81,  82,  83,  84,  85,  86,  87
1220 data  88,  89,  90, 219, 220, 221, 222, 223, 224, 225
1230 data 226, 227, 228, 229, 230, 231, 232, 233, 234, 235
1240 data 236, 237, 238, 239, 240, 241, 242, 243, 244, 245
1250 data 246, 247, 248, 249, 250, 251, 252, 253, 254, 255
```

PET 2040 DISK BUFFER I/O ROUTINE

J.HOOGSTRAAT,
BOX 20 SITE 7 SS 1
CALGARY. ALTA.

The major difficulty in programming direct access routines for the PET 2040 disk drives is the computation of the exact location of the recorded information on a disk sector, for the reason that the PET prints its data to the disk rather than transferring it byte for byte.

This results in variable length records on each disk write, unless the programmer takes special care converting each variable to a fixed length string variable before writing it to the disk. This is not too bad for string variables, but other variables could be ranging in length from one to more than ten characters after conversion to an equivalent string variable.

Suppose we want to program a direct access file consisting of records made up of an ITEM-NO, DESCRIPTION and COST.

The ITEM-NO ranges from 1 to 9999
The DESCRIPTION is 12 bytes long
The COST ranges from .00 to 9999999.00

We need 4 characters for the ITEM-NO, 12 for the DESCRIPTION and 10 for the COST. This would total up to 26 characters per record, but in order to be able to read it back we have to add at least one carriage return character after the COST string. After reading we can de-compose the information with MID$ calls. Or, if we wish to be able to update each field individually, a carriage return character must be added after each field, which ups our total record length to 29 characters

I personally found this method rather wasteful and cumbersome to program with all the STR$ calls and BLANK padding. No other software seemed to be available, except for Bill Macleans Block Get Routine published in the Commodore Transactor Vol 2, Dec 31, 1979. An excellent routine, but it can only read from the disk buffers with special care to be taken for the allocation of the input string variable.

So, what I needed was a routine with the following characteristics:

.. Be able to read the disk block buffers.

.. Be able to write the disk block buffers.

.. No need for blank padding of any variables or the need of adding carriage return characters.

.. Record and read numeric variables as 5 binary characters, as stored in PET's memory. This allows records of up to 51 numeric variables on a disk sector.

.. Be able to read single character string variables with an

ASC value of zero, in stead of getting a NULL string.

.. Exercise full control over the Block Buffer Pointers.

.. Perform like a basic WRITE or READ statement.

.. No need for special declarations or dummy manipulations of input variables.

.. Be able to output any kind of proper expressions.

.. Be totally relocatable.

Aided with Jim Butterfields excellent PET maps and the Macro-Tea assembler of Skyles Electric Works, I succesfully coded the needed routine.

I'll explain how to use it with some basic coding examples.

The basic format for the call to the PET 2040 disk buffer I/O routine is:

SYS XX, IO, CH, (BP ,VA ,(LN))

XX = Address were the routine is loaded.

IO = Input / Output key value.

CH = Disk direct access channel no.

BP = Buffer pointer value.

VA = Variable name.

LN = No of characters.

For single BP control the IO values are:

0 For normal reading.
1 For normal writing.
2 For special reading.
3 Same as 1.

For multiple BP control the IO values are:

4 For normal reading.
5 For normal writing.
6 For special reading.
7 Same as 5

BASIC NUMERIC VARIABLE EXAMPLES

```
10 DK = 1: CE = 15: CH = 2: XX = 634
20 OPEN CE,8,CE
30 OPEN CH,8,CH,"#"

40 T = 2: S = 5: BP = 13

50 REM WRITE 3 VARIABLES  TO  DISK

60 SYS XX, 1, CH, BP, A, B, C :REM OUTPUT

70 PRINT#CE, "U2:"CH;DK;T;S

880 REM READ 3 VARIABLES FROM DISK

90 PRINT#CE, "U2:"CH;DK;T;S

100 SYS XX, 0, CH, BP, X, Y, Z :REM INPUT
```

In this example we are writing the 3 numeric variables (A,B,C) to the disk buffer starting at character position 13. The result is then written to disk drive 1 at Track 2, Sector 5. The buffer pointer is automatically incremented by 5 for each variable and the variables are recorded in internal PET format. Note no padding or carriage returns needed. After the write, the variables are read back into X, Y and Z.

For numeric variables the parameter LN is implied and must not be coded.

If the PRINT#CE calls were omitted, no actual disk writing or reading would take place, but merely a transfer to and from the disk buffer allocated to channel CH, which maybe useful in passing parameters between overlays.

Statement 60 could be something like

```
60 SYS XX, 1, CH, BP,     1.,        A, A+B*C :REM OUTPUT
```

or

```
60 SYS XX, 1, CH, BP, SQR(A), SIN(A+B),   A/B :REM OUTPUT
```

or

```
60 SYS XX, 1, CH, BP,   1.+C,       -A, -55.5 :REM OUTPUT
```

The number of concatenated variables is only limited by the maximum length of a BASIC line. But at least one must be specified. We could also replace statement 60 by the following lines:

```
60 SYS XX, 1, CH, BP   , A   :REM OUTPUT
61 SYS XX, 1, CH, BP+ 5, B   :REM OUTPUT
62 SYS XX, 1, CH, BP+10, C   :REM OUTPUT
```

Which have the same effect as the original line 60.

Statement 100 could also be replaced by the following lines, which would read back the exact same information in

the variables X, Y and Z.

```
100 SYS XX, 0, CH, BP+ 5, Y, Z :REM INPUT
101 SYS XX, 0, CH, BP   , Y    :REM INPUT
```

If we want more control over the buffer pointer on the write, the value for IO must be 4 for reading and 5 for writing.

Statements 60 and 100 which were:

```
 60 SYS XX, 1, CH, BP, A, B, C :REM OUTPUT

100 SYS XX, 0, CH, BP, X, Y,  Z :REM INPUT
```

can now be coded as:

```
 60 SYS XX, 5, CH, BP, A, BP+ 5, B, BP+10, C :REM OUTPUT

100 SYS XX, 4, CH, BP, X, BP+ 5, Y, BP+10, Z :REM INPUT
```

The difference is that each variable now has a buffer pointer value preceeding it. The statements can now also be:

```
 60 SYS XX, 5, CH, BP+ 5, B, BP, A, BP+10, C :REM OUTPUT

100 SYS XX, 4, CH, BP+10, Z, BP, X, BP+ 5, Y :REM INPUT
```

Since we now have full buffer pointer control.

BASIC STRING VARIABLES EXAMPLES

```
10 DK = 1: CE = 15: CH = 2: XX = 634
20 OPEN CE,8,CE
30 OPEN CH,8,CH,"#"

40 T = 2: S = 5: BP = 13

50 REM WRITE 3 STRING VARIABLES  TO  DISK
   ---------------------------------------
60 SYS XX, 1, CH, BP, A$,5, B$,6, C$,10 :REM OUTPUT
70 PRINT#CE, "U2:"CH;DK;T;S

80 REM READ  3 STRING VARIABLES FROM DISK
   ---------------------------------------
90 PRINT#CE, "U2:"CH;DK;T;S

100 SYS XX, 0, CH, BP, X$,5, Y$,6, Z$,10 :REM INPUT
```

In this example we are writing the 3 STRING variables (A$,B$,C$) to the disk buffer starting at character position 13. The result is then written to disk drive 1 at Track 2, Sector 5.

The difference between a numeric variable and a string variable is that the string variable is followed by LN, its length or number of characters. The specied length does not have to be the actual length of the string variable. In our example the first 5 characters of X$ are transferred,

followed by the first 6 characters of Y$ and then the first 10 characters of Z$.

The buffer pointer is automatically incremented by 5,6 and 10. Note no padding or carriage returns needed. After the write, the variables are read back into the string$ X$, Y$ and Z$

Lets now examine what happens if we have the following statements:

```
55 Z$ = "HANS"+"MARGARET"

60 SYS XX, 1, CH, BP, Z$,LEN(Z$) :REM OUTPUT
```

The disk buffer (CH) will now contain starting at character position 13 the text "HANSMARGARET". The same results of the next statement:

```
60 SYS XX, 1, CH, BP, "HANS"+"MARGARET",12 :REM OUTPUT
```

And the statement:

```
100 SYS XX, 0, CH, BP, Z$,12 :REM INPUT
```

Will input and create a string variable with a length of 12 characters and containing the text "HANSMARGARET". However the statement:

```
100 SYS XX, 0, CH, BP, Z$,10 :REM INPUT
```

Will input and create a string variable with a length of 10 characters and containing the text "HANSMARGAR". Or the statements:

```
100 SYS XX, 0, CH, BP   , X$,6 :REM INPUT
101 SYS XX, 0, CH, BP+7, Z$,5 :REM INPUT
```

Will input and create two string variables X$ and Z$, containing "HANSMA" AND "GARET"

Note that no extra linefeeds or carriage return characters are written and that the record space needed for the original ITEM-NO, DESCRIPTION and COST example is now 5+12+5 or 22 characters instead of the 29 needed without this buffer I/O routine.

If the PRINT#CE calls were omitted no actual disk writing or reading would take place, but merely a transfer to and from the disk buffer allocated to channel CH, which again maybe useful in passing parameters between overlays, or to do some fancy string manipulations.

P.E.:

```
10 A$ = "XXXXXXXXXX"
11 B$ = "YYYYY"
12 SYS XX, 1, CH, 2, A$, LEN(A$) :REM OUTPUT
13 SYS XX, 1, CH, 5, B$, LEN(B$) :REM OUTPUT
14 SYSS XX, 0, CH, 2, A$, 10      :REM INPUT
```

First writes the string variables A$ and B$ overlaying the A$ information and then inputs and creates a string variable A$ containing "XXXYYYYYXX".

Statement 60 could be something like

```
60 SYS XX, 1, CH, BP, A$+"X",5, A$+B$,6, A$+"Z"+C$,10
:REM OUTPUT
```

The number of concatenated string variables is only limited by the maximum length of a BASIC line. But at least one must be specified. We could also replace statement 60 by the following lines:

```
60 SYS XX, 1, CH, BP   , A$,5  :REM OUTPUT
61 SYS XX, 1, CH, BP+ 5, B$,6  :REM OUTPUT
62 SYS XX, 1, CH, BP+11, C$,10 :REM OUTPUT
```

Which have the same effect as the original line 60.

Statement 100 could also be replaced by the following lines, which would read back the exact same information in the string variables X$, Y$ and Z$

```
100 SYS XX, 0, CH, BP+5, Y$,6, Z$,10 :REM INPUT
101 SYS XX, 0, CH, BP  , X$,5        :REM INPUT
```

If we want more control over the buffer pointer on the write, the value for IO must be 4 for reading and 5 for writing.

Statements 60 and 100 which were:

```
 60 SYS XX, 1, CH, BP, A$,5, B$,6, C$,10 :REM OUTPUT
100 SYS XX, 0, CH, BP, X$,5, Y$,6, Z$,10 :REM INPUT
```

can now be coded as:

```
 60 SYS XX, 5, CH, BP,A$,5, BP+5,B$,6, BP+11,C$,10 :REM
OUTPUT

100 SYS XX, 4, CH, BP,X$,5, BP+5,Y$,6, BP+11,Z$,10 :REM
INPUT
```

The difference is that each string variable now has a buffer pointer value preceeding it and still its length following it. The statements can now also be:

```
 60 SYS XX, 5, CH, BP+5,B$,6, BP+11,C$,10, BP,A$,5 :REM
OUTPUT

100 SYS XX, 4, CH, BP+11,Z$,10, BP,A$,5, BP+5,Y$,6 :REM
INPUT
```

Since we now have full buffer pointer control.

So far I only discussed write and reads of string variables of the same length on the writing and reading.

Now suppose we have the following statements:

```
55 A$ = "HANS"
60 SYS XX, 5, CH, 10,A$,10 , 20,A$+A$,10 :REM OUTPUT
```

This transfers to the buffer, starting at character location 10, the characters "hans*****hanshans**", where the "*" stands for an automatic padded carriage return character with an ASC value of 13. In other words the routine will always write the number of characters requested but if the output string expression is too short, the output will be padded with carriage return characters. This has a nice effect when we read the same data back with the following statement:

```
100 SYS XX, 4, CH, 10,A$,10 , 20,B$,10 :REM INPUT
```

This call will input and create the two string variables A$ and B$, but their contents will be "HANS" AND "HANSHANS", since the input quits on the first encountered carriage return characters for each variable and their length will be 4 and 8. However an otherwise null character string will always be returned as a character string of ASC value zero with a length of one.

Sometimes this technique is undesirable and we want to get back every character, no matter what their ASC values are. Now the special read I/O values 2 or 6 are to be used. The statement:

```
100 SYS XX, 6, CH, 10,A$,10 , 20,B$,10 :REM INPUT
```

Will now input and create an A$ and B$ variable containing "hans******" and "hanshans**".

Note, the length limit of a string variable is 255 bytes, allowing us to read or write entire disk buffer blocks at once.

By no means do we have to write separate statements for numeric or string variables, we can mix them up. The following statements are quite legal:

```
 51 IT  = 5469
 52 SS$ = "PET COMPUTER"
 53 CO  = 1365.25
 60 SYS XX, 1, CH, 2, IT, SS$,12, CO  :REM OUTPUT
100 SYS XX, 6, CH, 7,A$,12 ,2,A, 19,B :REM INPUT
```

Again the read call for I/O = 6 will properly return:

A$ = "PET COMPUTER", A = 5469, B = 1365.25

Still confused, please contact me !

```
                0010; ROUTINE TO TRANSFER FLOATING POINT VARIABLES AND STRING
                0020; VARIABLES BETWEEN PET'S MEMORY AND A D/A DISK BUFFER.
                0030; ------------------------------------------------------------
                0040;
                0050; WRITTEN BY J.HOOGSTRAAT
                0060;
                0070;            BOX-20,  SITE 7,  SS1
                0080;            CALGARY, T2M-4N3, ALTA
                0090;            PHONE    (403)239-0900
                0100;
                0110; ------------------------------------------------------------
                0120;
                0130; THIS ROUTINE IS TOTAL RELOCATABLE AND CAN BE LOADED ANYWHERE.
                0140;
                0150; FLOATING POINT VARIABLES ARE TRANSFERRED AS 5 BYTES ONLY.
                0160;
                0170; STRING VARIABLES ARE TRANSFERRED WITHOUT LINEFEEDS
                0180; OR CARRIAGE RETURNS.
                0190;
                0200; THIS ROUTINE IS IDEALLY SUITABLE FOR DIRECT DISK ACCESSING,
                0210; SINCE ALL BUFFER POINTERS CAN BE CALCULATED EXACTLY.
                0220;
                0230; ------------------------------------------------------------
                0240;
                0250;
                0260            .OS
                0270            .BA 634       ;FIRST CASSETTE BUFFER FOR NOW.
                0280;
                0290; LOCAL VARIABLES
                0300;
                0310STADR       .DI $1        ;SAVED ROUTINE START ADDRESS.
                0320SYSXX       .DI $11       ;BASIC ROUTINE START ADDRESS AS SYS XX.
                0330;
                0340IO          .DI $B1       ;SAVED IO.
                0350DCH         .DI $B2       ;SAVED DCH.
                0360LNG         .DI $B7       ;SAVED REQ. LENGTH.
                0370STP          .DI $B8       ;SAVED DATA TYPE.
                0380;
                0390; LOCAL VALUES
                0400;
                0410DCE         .DI $F        ;DISK COMMAND CHANNEL.
                0420CRT         .DI $D        ;CARRIAGE RETURN.
                0430FLN         .DI $5        ;FLT PNT WORD LENGTH.
                0440;
                0450; BASIC AREAS USED
                0460;
                0470DTP         .DI $07       ;DATA TYPE.
                0480SLN         .DI $16        ;STRING LENGTH.
                0490SAD         .DI $17       ;STRING ADDRESS.
                0500CAD         .DI $44       ;CURRENT VARIABLE ADDR.
                0510ACC         .DI $5E       ;ACCUMULATOR.
                0520NCH         .DI $77       ;NEXT INPUT FIELD CHAR.
                0530ASB         .DI $100      ;ASC BUFFER.
                0540;
027A-A511       0550START       LDA *SYSXX    ;START START ADDR
027C-8501       0560            STA *STADR    ;FOR SELF RELOCATION.
027E-A512       0570            LDA *SYSXX+1
0280-8502       0580            STA *STADR+1
                0590;
0282-20F8CD     0600            JSR CHKCOM    ;UPTO NEXT FIELD.
```

```
0285-209FCC 0610            JSR EVAEXP    ;EVALUATE EXPRESSION.
0288-20D2D6 0620            JSR FLTFIX    ;CONVERT TO INTEGER.
028B-84B1   0630            STY *IO       ;SAVE IO.
            0640;
028D-20F8CD 0650            JSR CHKCOM    ;UPTO NEXT FIELD.
0290-209FCC 0660            JSR EVAEXP    ;EVALUATE EXPRESSION.
0293-20D2D6 0670            JSR FLTFIX    ;CONVERT TO INTEGER.
0296-84B2   0680            STY *DCH      ;SAVE DCH.
            0690;
0298-20F8CD 0700AGAIN       JSR CHKCOM    ;UPTO NEXT FIELD.
029B-209FCC 0710            JSR EVAEXP    ;EVALUATE EXPRESSSION.
029E-20E9DC 0720            JSR BINASC    ;CVT BPT TO ASC.
            0730;
            0740; ISSUE PRINT#CE, "B-P:"CH;BP
            0750; -------------------------
            0760;
02A1-A20F   0770            LDX #DCE                ;OPEN CHANNEL 'CE'.
02A3-20C9FF 0780            JSR STODEV
            0790;
02A6-A0C4   0800            LDY #BPDCH-START        ;SET RELOCATION.
            0810;
02A8-A5B2   0820            LDA *DCH                ;STOW ASC OF DCH
02AA-0930   0830            ORA #$30                ;IN THE TEXT.
02AC-9101   0840            STA (STADDR),Y
            0850;
02AE-A0C1   0860            LDY #BPTXT-START        ;SET RELOCATION.
02B0-B101   0870OUTBP       LDA (STADR),Y           ;OUTPUT "B-P:"CH.
02B2-20D2FF 0880            JSR OUTCHR
02B5-C8     0890            INY
02B6-C0C6   0900            CPY #BPTXE-START        ;END OF TEXT ?
02B8-D0F6   0910            BNE OUTBP               ;NO, CONTINUE.
            0920;
02BA-A201   0930            LDX #1
02BC-BD0001 0940BPOUT       LDA ASB,X               ;OUTPUT ASC OF BP
02BF-F00A   0950            BEQ BPDON               ;END OF ASC.
02C1-20D2FF 0960            JSR OUTCHR
02C4-E8     0970            INX
02C5-D0F5   0980            BNE BPOUT               ;CONTINUE TILL END.
02C7-F002   0990            BEQ BPDON
            1000;
02C9-D0CD   1010AGAJJ       BNE AGAIN
            1020;
02CB-20CCFF 1030BPDON       JSR RESTIO
            1040;
            1050; ISSUE PRINT#CH FOR INPUT OR OUTPUT
            1060; ----------------------------------
            1070;
02CE-A6B2   1080            LDX *DCH
            1090;
02D0-A5B1   1100            LDA *IO       ;CHECK IO.
02D2-2901   1110            AND #1
02D4-F005   1120            BEQ OPINP     ;INPUT.
            1130;
02D6-20C9FF 1140OPOUT       JSR STODEV    ;OPEN OUTPUT CH.
02D9-D003   1150            BNE TRFER
            1160;
02DB-20C6FF 1170OPINP       JSR STIDEV    ;OPEN INPUT CH.
            1180;
02DE-20F8CD 1190TRFER       JSR CHKCOM    ;UPTO NEXT FIELD.
            1200;
```

```
02E1-A905   1210            LDA #FLN      ;DEFAULT LENGTH
02E3-85B7   1220            STA *LNG      ;TO FLT PNT LENGTH.
02E5-8516   1230            STA *SLN
            1240;
02E7-A5B1   1250            LDA *IO       ;CHECK IO.
02E9-2901   1260            AND #1
02EB-F053   1270            BEQ RINPT     ;READ INPUT.
            1280;
            1290; WRITE OUTPUT DATA
            1300; -----------------
            1310;
02ED-209FCC 1320WOUTP       JSR EVAEXP    ;EVALUATE EXPRESSION.
02F0-08     1330            PHP           ;SAVE STATUS
            1340;
02F1-A507   1350            LDA *DTP      ;CHARACTER STRING ?
02F3-F01D   1360             BEQ FLTDT     ;NO FLT PNT VARIABLE.
            1370;
            1380; OUTPUT STRING EXPRESSION
            1390;
02F5-207DD5 1400            JSR DSCSTR    ;DISCARD TEMP STRING
            1410;
02F8-28     1420            PLP           ;GET STATUS
02F9-100A   1430            BPL WOUTS     ;NOT A CONTANT STRING
            1440;
02FB-A002   1450WOUTC       LDY #2        ;SAVE STRING ADDRESS
02FD-B144   1460STRAD       LDA (CAD),Y
02FF-991600 1470            STA SLN,Y
0302-88     1480            DEY
0303-10F8   1490            BPL STRAD
            1500;
0305-20F8CD 1510WOUTS       JSR CHKCOM    ;UPTO NEXT FIELD.
0308-209FCC 1520            JSR EVAEXP    ;EVALUATE EXPRESSION.
030B-20D2D6 1530            JSR FLTFIX    ;CONVERT TO INTEGER.
030E-84B7   1540            STY *LNG      ;SAVE REQ. LENGTH.
0310-D011   1550            BNE WRITE     ;READY FOR OUTPUT.
            1560;
            1570; OUTPUT FLT PNT DATA IN ACCUMULATOR
            1580;
0312-28     1590FLTDT       PLP           ;CLEAR STACK
            1600;
0313-A563   1610            LDA *ACC+5    ;CORRECT SIGN ?
0315-3004   1620            BMI FLTCR     ;NO.
            1630;
0317-065F   1640            ASL *ACC+1    ;REMOVE SIGN BIT
0319-465F   1650            LSR *ACC+1    ;FROM ACCUMULATOR.
            1660;
031B-A95E   1670FLTCR       LDA #L,ACC    ;SET OUTPUT
031D-A000   1680            LDY #H,ACC    ;ADDRESS TO THE
031F-8517   1690            STA *SAD      ;ACCUMLATOR.
0321-8418   1700            STY *SAD+1
            1710;
            1720; OUTPUT CHARACTER LOOP
            1730;
0323-A000   1740WRITE       LDY #0        ;SET CHAR POINTER.
            1750;
0325-A90D   1760WRIT1       LDA #CRT      ;DEFAULT TO CR.
0327-C416   1770            CPY *SLN      ;MORE THAN ACTUAL LENGTH ?
0329-B002   1780            BCS WRIT2     ;YES, USE CR.
032B-B117   1790            LDA (SAD),Y   ;USE INPUT CHAR.
032D-20D2FF 1800WRIT2       JSR OUTCHR    ;OUTPUT THIS CHAR.
```

```
                1810;
0330-C8         1820            INY
0331-C4B7       1830            CPY *LNG        ;ALL DONE ?
0333-D0F0       1840            BNE WRIT1       ;NO.
0335-F061       1850            BEQ FIELD       ;YES.
                1860;
                1870; INBETWEEN JUMP AND CONSTANTS
                1880; ----------------------------
                1890;
0337-D090       1900AGAIJ       BNE AGAJJ
0339-F0A3       1910TRFEJ       BEQ TRFER
                1920;
033B-422D50     1930BPTXT       .BY 'B-P'
033E-5820       1940BPDCH       .BY 'X '
                1950BPTXE       .DI =
                1960;
                1970; READ INPUT DATA
                1980; ---------------
                1990;
0340-206DCF     2000RINPT       JSR GETVAR      ;GET VARIABLE ADDR.
                2010;
0343-8517       2020            STA *SAD        ;DEFAULT INPUT ADDRESS.
0345-8418       2030            STY *SAD+1      ;TO FLT PNT VARIABLE
                2040;
0347-A507       2050            LDA *DTP        ;SAVE AND CHECK DATA TYPE.
0349-85B8       2060            STA *STP
                2070;
                2080; INPUT FLT PNT VARIABLE
                2090;
034B-F020       2100            BEQ READI       ;FLT PNT INPUT VARIABLE.
                2110;
                2120; INPUT STRING VARIABLE
                2130;
034D-20F8CD     2140            JSR CHKCOM      ;UPTO NEXT FIELD.
0350-209FCC     2150            JSR EVAEXP      ;EVALUATE EXPRESSION.
0353-20D2D6     2160            JSR FLTFIX      ;CONVERT TO INTEGER.
                2170;
0356-98         2180            TYA
0357-A000       2190            LDY #0
0359-8516       2200            STA *SLN        ;SAVE REQ. LLENGTH.
035B-9117       2210            STA (SAD),Y     ;SAVE IN STRING INDEX.
                2220;
035D-20D0D3     2230            JSR GETSPC      ;GET SPACE FOR STRING.
                2240;
0360-98         2250            TYA
0361-A002       2260            LDY #2          ;SAVE ADDRESS OF SPACE
0363-9117       2270            STA (SAD),Y     ;IN STRING INDEX
0365-8545       2280            STA *CAD+1      ;AND CURRENT VARIABLE ADDRESS.
0367-8A         2290            TXA
0368-88         2300            DEY
0369-9117       2310            STA (SAD),Y
036B-8544       2320            STA *CAD
                2330;
036D-A000       2340READI       LDY #0          ;SET CHAR POINTER.
                2350;
036F-A5B1       2360            LDA *IO         ;CHECK IO.
0371-2902       2370            AND #2          ;SPECIAL STRING READ ?
0373-F002       2380            BEQ READ1       ;NO.
                2390;
0375-84B8       2400            STY *STP        ;CHANGE FROM 'FF' TO '00'.
```

```
                2410;
0377-20CFFF     2420READ1       JSR INPCHR      ;INPUT A CHAR.
                2430;
037A-C90D       2440            CMP #CRT        ;CARRIAGE RETURN ?
037C-D004       2450            BNE READ2       ;NO.
                2460;
037E-A6B8       2470            LDX *STP        ;YES. STRING ?
0380-D009       2480            BNE READ4       ;YES. TERMINATE STRING.
                2490;
0382-9144       2500READ2       STA (CAD),Y     ;STOW CHAR INTO INPUT.
                2510;
0384-C8         2520READ3       INY
0385-C416       2530            CPY *SLN        ;ALL DONE ?
0387-D0EE       2540            BNE READ1       ;NO.
0389-F00D       2550            BEQ FIELD       ;YES.
                2560;
038B-98         2570READ4       TYA
038C-F0F4       2580            BEQ READ2       ;INTERCEPT NULL STRINGS
                2590;
038E-A000       2600            LDY #0          ;SET RECORDED STRING LENGTH.
0390-84B8       2610            STY *STP        ;RESET DATA TYPE.
0392-9117       2620            STA (SAD),Y     ;TRUNCATE STRING IN INDEX.
0394-A8         2630            TAY
0395-18         2640            CLC
0396-90EC       2650            BCC READ3       ;CONTINUE READING.
                2660;
                2670; CHECK FOR MORE FIELDS
                2680; ---------------------
                2690;
0398-A000       2700FIELD       LDY #0          ;MORE FIELDS ARE PRESENT
039A-B177       2710            LDA (NCH),Y     ;IF THERE IS A COMMA IN
039C-C92C       2720            CMP #',         ;BASIC'S INPUT BUFFER.
039E-D008       2730            BNE ADONE       ;NO, WE QUIT.
03A0-A5B1       2740            LDA *IO         ;WHAT KIND
03A2-290C       2750            AND #12
03A4-F093       2760            BEQ TRFEJ       ;GO AGAIN, NO BP
03A6-D08F       2770            BNE AGAIJ       ;GO AGAIN, BP SET
                2780;
                2790; TERMINATE ROUTINE
                2800; -----------------
                2810;
03A8-20CCFF     2820ADONE       JSR RESTIO      ;RESTORE I/O DEVICE.
03AB-60         2830            RTS
                2840;
                2850; BASIC ROUTINES USED
                2860;
                2870EVAEXP      .DE $CC9F       ;EVALUATE EXPRESSION.
                2880CHKCOM      .DE $CDF8       ;CHECK FOR COMMA.
                2890GETVAR      .DE $CF6D       ;GET BASIC VARIABLE.
                2900GETSPC      .DE $D3D0       ;GET STRING SPACE.
                2910DSCSTR      .DE $D57D       ;DISCARD TEMP STRING.
                2920FLTFIX      .DE $D6D2       ;FLOAT TO INTEGER. CONVERSION
                2930BINASC      .DE $DCE9       ;CONVERT FLT TO ASC.
                2940RESTIO      .DE $FFCC       ;RESTORE DEFAULT I/O ADDRESSES.
                2950STIDEV      .DE $FFC6       ;SET INPUT DEVICE.
                2960STODEV      .DE $FFC9       ;SET OUTPUTT DEVICE.
                2970INPCHR      .DE $FFCF       ;INPUT CHARACTER.
                2980OUTCHR      .DE $FFD2       ;OUTPUT CHARACTER.
                2990            .EN
```

LABELS

STADR	= 0001	SYSXX	= 0011
IO	= 00B1	DCH	= 00B2
LNG	= 00B7	STP	= 00B8
DCE	= 000F	CRT	= 000D
FLN	= 0005	DTP	= 0007
SLN	= 0016	SAD	= 0017
CAD	= 0044	ACC	= 005E
NCH	= 0077	ASB	= 0100
START	= 027A	AGAIN	= 0298
OUTBP	= 02B0	BPOUT	= 02BC
AGAJJ	= 02C9	BPDON	= 02CB
OPOUT	= 02D6	OPINP	= 02DB
TRFER	= 02DE	WOUTP	= 02ED
WOUTC	= 02FB	STRAD	= 02FD
WOUTS	= 0305	FLTDT	= 0312
FLTCR	= 031B	WRITE	= 0323
WRIT1	= 0325	WRIT2	= 032D
AGAIJ	= 0337	TRFEJ	= 0339
BPTXT	= 033B	BPDCH	= 033E
BPTXE	= 0340	RINPT	= 0340
READI	= 036D	READ1	= 0377
READ2	= 0382	READ3	= 0384
READ4	= 038B	FIELD	= 0398
ADONE	= 03A8	/EVAEXP	= CC9F
/CHKCOM	= CDF8	/GETVAR	= CF6D
/GETSPC	= D3D0	/DSCSTR	= D57D
/FLTFIX	= D6D2	/BINASC	= DCE9
/RESTIO	= FFCC	/STIDEV	= FFC6
/STODEV	= FFC9	/INPCHR	= FFCF
/OUTCHR	= FFD2		

HEXADECIMAL DUMP

```
027A A5 11 85 01 A5 12 85 02
0282 20 F8 CD 20 9F CC 20 D2
028A D6 84 B1 20 F8 CD 20 9F
0292 CC 20 D2 D6 84 B2 20 F8
029A CD 20 9F CC 20 E9 DC A2
02A2 0F 20 C9 FF A0 C4 A5 B2
02AA 09 30 91 01 A0 C1 B1 01
02B2 20 D2 FF C8 C0 C6 D0 F6
02BA A2 01 BD 00 01 F0 0A 20
02C2 D2 FF E8 D0 F5 F0 02 D0
02CA CD 20 CC FF A6 B2 A5 B1
02D2 29 01 F0 05 20 C9 FF D0
02DA 03 20 C6 FF 20 F8 CD A9
02E2 05 85 B7 85 16 A5 B1 29
02EA 01 F0 53 20 9F CC 08 A5
02F2 07 F0 1D 20 7D D5 28 10
02FA 0A A0 02 B1 44 99 16 00
0302 88 10 F8 20 F8 CD 20 9F
030A CC 20 D2 D6 84 B7 D0 11
0312 28 A5 63 30 04 06 5F 46
031A 5F A9 5E A0 00 85 17 84
0322 18 A0 00 A9 0D C4 16 B0
032A 02 B1 17 20 D2 FF C8 C4
0332 B7 D0 F0 F0 61 D0 90 F0
033A A3 42 2D 50 33 20 20 6D
0342 CF 85 17 84 18 A5 07 85
034A B8 F0 20 20 F8 CD 20 9F
0352 CC 20 D2 D6 98 A0 00 85
035A 16 91 17 20 D0 D3 98 A0
0362 02 91 17 85 45 8A 88 91
036A 17 85 44 A0 00 A5 B1 29
0372 02 F0 02 84 B8 20 CF FF
037A C9 0D D0 04 A6 B8 D0 09
0382 91 44 C8 C4 16 D0 EE F0
038A 0D 98 F0 F4 A0 00 84 B8
0392 91 17 A8 18 90 EC A0 00
039A B1 77 C9 2C D0 08 A5 B1
03A2 29 0C F0 93 D0 8F 20 CC
03AA FF 60
```

```
60000 REM DATA STATEMENTS FOR D/A BUFFER ROUTINE
60001 REM
60002 REM TOTAL LENGTH 306 BYTES
60003 REM
60004 DATA 165,  17, 133,   1, 165,  18, 133,   2
60005 DATA  32, 248, 205,  32, 159, 204,  32, 210
60006 DATA 214, 132, 177,  32, 248, 205,  32, 159
60007 DATA 204,  32, 210, 214, 132, 178,  32, 248
60008 DATA 205,  32, 159, 204,  32, 233, 220, 162
60009 DATA  15,  32, 201, 255, 160, 196, 165, 178
60010 DATA   9,  48, 145,   1, 160, 193, 177,   1
60011 DATA  32, 210, 255, 200, 192, 198, 208, 246
60012 DATA 162,   1, 189,   0,   1, 240,  10,  32
60013 DATA 210, 255, 232, 208, 245, 240,   2, 208
60014 DATA 205,  32, 204, 255, 166, 178, 165, 177
60015 DATA  41,   1, 240,   5,  32, 201, 255, 208
60016 DATA   3,  32, 198, 255,  32, 248, 205, 169
60017 DATA   5, 133, 183, 133,  22, 165, 177,  41
60018 DATA   1, 240,  83,  32, 159, 204,   8, 165
60019 DATA   7, 240,  29,  32, 125, 213,  40,  16
60020 DATA  10, 160,   2, 177,  68, 153,  22,   0
60021 DATA 136,  16, 248,  32, 248, 205,  32, 159
60022 DATA 204,  32, 210, 214, 132, 183, 208,  17
60023 DATA  40, 165,  99,  48,   4,   6,  95,  70
60024 DATA  95, 169,  94, 160,   0, 133,  23, 132
60025 DATA  24, 160,   0, 169,  13, 196,  22, 176
60026 DATA   2, 177,  23,  32, 210, 255, 200, 196
60027 DATA 183, 208, 240, 240,  97, 208, 144, 240
60028 DATA 163,  66,  45,  80,  88,  32,  32, 109
60029 DATA 207, 133,  23, 132,  24, 165,   7, 133
60030 DATA 184, 240,  32,  32, 248, 205,  32, 159
60031 DATA 204,  32, 210, 214, 152, 160,   0, 133
60032 DATA  22, 145,  23,  32, 208, 211, 152, 160
60033 DATA   2, 145,  23, 133,  69, 138, 136, 145
60034 DATA  23, 133,  68, 160,   0, 165, 177,  41
60035 DATA   2, 240,   2, 132, 184,  32, 207, 255
60036 DATA 201,  13, 208,   4, 166, 184, 208,   9
60037 DATA 145,  68, 200, 196,  22, 208, 238, 240
60038 DATA  13, 152, 240, 244, 160,   0, 132, 184
60039 DATA 145,  23, 168,  24, 144, 236, 160,   0
60040 DATA 177, 119, 201,  44, 208,   8, 165, 177
60041 DATA  41,  12, 240, 147, 208, 143,  32, 204
60042 DATA 255,  96
60043 END
```

```
100 REM A RANDOM FILE DEMONSTRATION
110 REM WHICH NEEDS NO BLOCK-ALLOCATE
120  REM BY USING THE SPACE ALLLOCATED
130 REM OF ANY PREVIOUS CREATED FILE.
140 REM
150 REM THE RANDOM UPDATES CAN BE BITS
160 REM OF INFORMATION OF UPTO 254
170 REM BYTES OF STRING INFORMATION.
180 REM
190 REM FLOATING POINT VARIABLES ALWAYS
200 REM ARE ONLY 5 BYTES LONG. THE FIVE
210 REM BYTES PET USES.
220 REM
230 REM THIS DEMONSTRATION NEEDS THE
240 REM D/A BUFFER ROUTINE LOADED AT
250 REM XX=634.
260 REM
 270 REM TESTING DONE ON DISK DRIVE 1
280 REM
290 REM ========================
300 REM       J.HOOGSTRAAT
310 REM
320 REM BOX 20, SITE 7, SS 1
330 REM CALGARY, ALTA.    T2M-4N3
340 REM           PH(403) 239-0900
350 REM ========================
360 REM
370 REM
380 REM CREATE A SEQUENTIAL TEST FILE
390 REM ----------------------------
400 REM
410 F$="TESTING-TESTING"
420 XX=634:GOSUB1120
430 DK=1:CE=15:CS=2:CR=3:NN=200
440 DIMT(40),S(40)
450 A$="I"+CHR$(48+DK):OPENCE,8,CE,A$
460 A$="@"+CHR$(48+DK)+":"+F$+",U,W"
470 OPENCS,8,CS,A$
480 A$="...":FORI=1TO3:A$=A$+A$:NEXT
490 FORI=1TO27:PRINT#CS,A$:NEXT
500 CLOSECS
510 REM
520 REM FIND TRACK AND SECTOR EXTENTS
530 REM FOR CREATED TEST FILE
540 REM ----------------------------
550 REM
560 L=LEN(F$)
570 A$=CHR$(48+DK)+":"+F$+",U,R"
580 OPENCS,8,CS,A$
590 T=18:S=1:N=0
600 PRINT#CE,"U1:"CS;DK;T;S
610 SYSXX,0,CS,1,S$,1:S=ASC(S$)
620 FORI=2TO255STEP32
630 SYSXX,0,CS,I,A$,2,T$,1,S$,1,N$,L
640 IFASC(A$)>128ANDF$=N$THEN670
650 NEXT:IFS<255THEN600
660 PRINT"FILE "F$" NOT FOUND":END
670 N=N+1
```

```
680 T(N)=ASC(T$):S(N)=ASC(S$)
690 PRINT#CE,"U1:"CS;DK;T(N);S(N)
700 GET#CS,T$,T$,S$:IFT$<>""THEN670
710 CLOSECS
720 REM
730 REM OPEN RANDOM FILE WITH THE TEST
740 REM FILE EXTENTS. FILL IT ALL UP
750 REM ------------------------------
760 REM
770 PRINT"[cs]"
780 OPENCR,8,CR,"#"
790 FORI=1TON:A$=CHR$(I+48)
800 FORL=1TO5:A$=A$+A$+A$:NEXT
810 PRINT#CE,"U1:"CR;DK;T(I);S(I)
820 SYSXX,1,CR,2,I,-1,A$,NN
830 SYSXX,0,CR,2,S,U,A$,NN
840 PRINT"[dn]BLOCK";S:PRINTA$;
850 PRINT#CE,"U2:"CR;DK;T(I);S(I)
860 NEXT
870 REM
880 REM UPDATE SOME TEXT IN A BLOCK
890 REM ---------------------------
900 REM
910 REM
920 INPUT"[dn]BLOCK,POS,TEXT";B,P,B$
930 PRINT"[cs]"
940 FORI=1TON
950 PRINT#CE,"U1:"CR;DK;T(I);S(I)
960 IFI<>BTHEN990
970 SYSXX,1,CR,7,P
980 SYSXX,1,CR,11+P,B$,LEN(B$)
990 SYSXX,0,CR,2,S,U,A$,NN
1000 PRINT"[dn]BLOCK"S;
1010 PRINT" LAST UPDATE AT POS";U
1020 PRINTA$;
1030 PRINT#CE,"U2:"CR;DK;T(I);S(I)
1040 NEXT
1050 GOTO920
1060 REM
1070 REM LOAD UP THE D/A BUFFER ROUTINE
1080 REM AT LOCATION XX. THIS ROUTINE
1090 REM A TOTAL RELOCATABLE.
1100 REM ----------------------------
1110 REM
1120 FORI=1TO306:READA:POKEXX-1+I,A:NEXT
1130 RETURN
1140 REM
1150 REM INSERT DATA STATEMENTS
1160 REM FOR D/A BUFFER ROUTINE HERE
1170 REM TOTAL LENGTH 306 BYTES
1180 REM
```

The following article was submitted by Sheldon H. Dean of Calgary, Alberta. Mr. Dean's interface is a modified version of a serial interface by Harvey B. Herman and Charles Pate that appeared in the March/April, 1980 issue of COMPUTE Magazine. Unlike the circuit in COMPUTE (and also one in an earlier Transactor), Sheldon's frees up the parallel user port by including an onboard oscillator for the UART clock....

> This interface provides a 300 baud (bps) asynchronous communication interface between the PET implemented, IEEE-488 bus and the Heathkit H14 line printer using the 20mA current loop standard. The interface is consructed using standard TTL devices. It provides true upper and lower case ASCII without the necessity of a handshake between the PET and the H14 printer. The interface plugs directly into the IEEE port on the PET and a simple wire pair connects the interface to the H14 printer by a standard DB-25S cable connector.
>
> Sheldon H. Dean

Although Sheldon has designed the interface for his Heathkit printer, it could undoubtedly be connected to other equipment that uses the 20mA current loop convention.

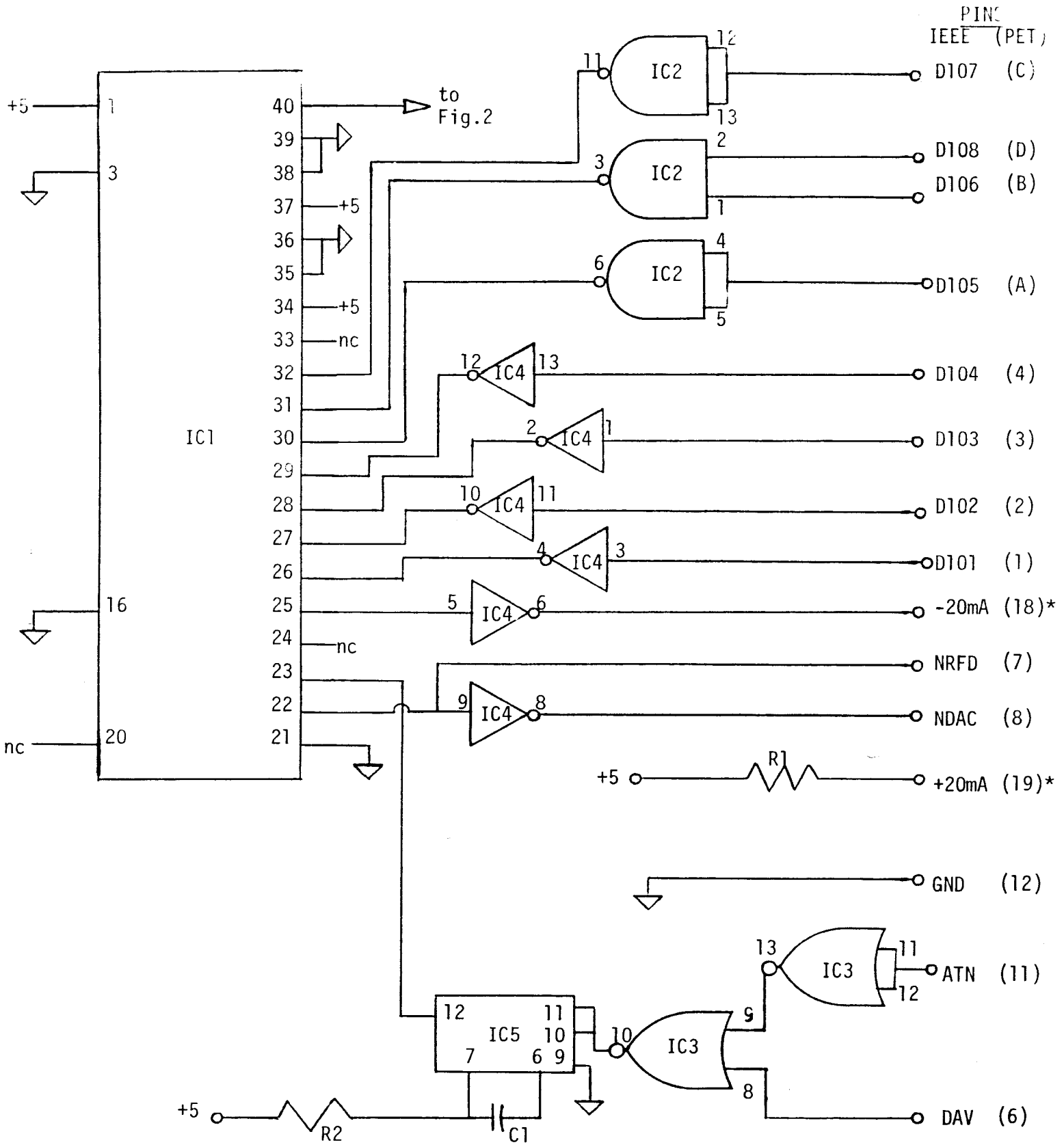

Figure One

* see note 4

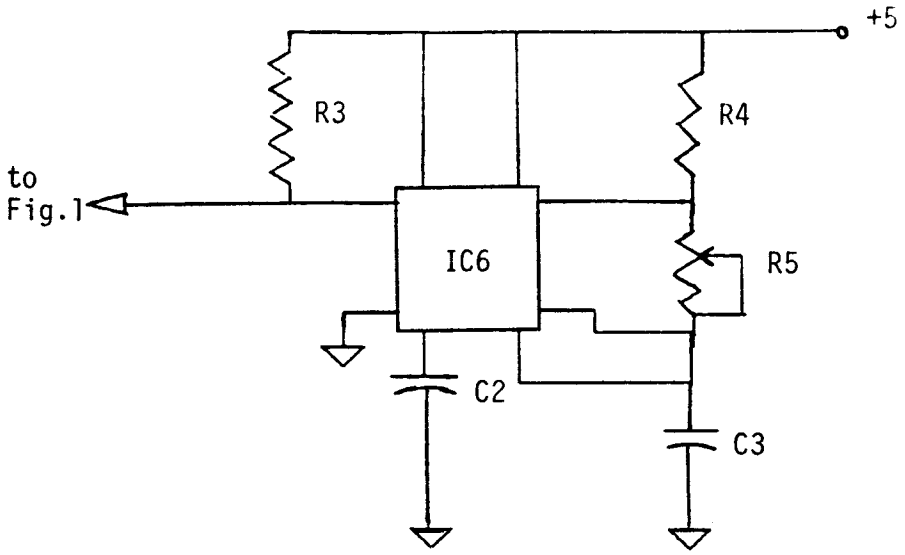

Figure Two

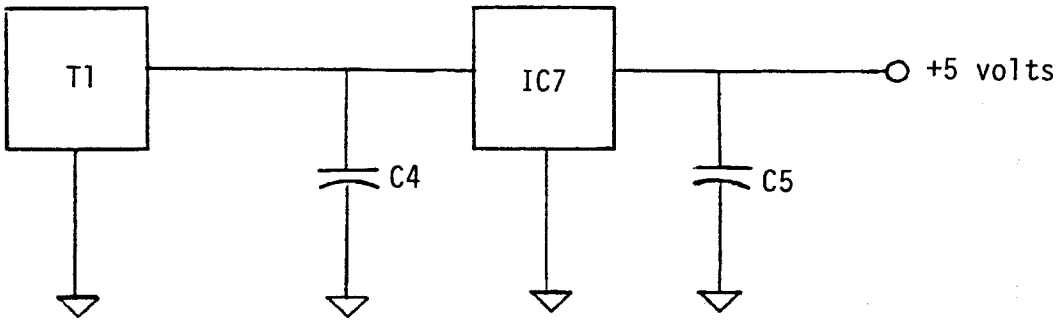

Figure Three

IC#	+5 volts	ground
1	1,34,37	3,16,21,35,36,38,39
2	14	7
3	14	7
4	14	7
5	16	8,9
6	4,8	1

TABLE ONE

This table lists the power requirements of the various ICs.

PARTS LIST

C1	.001 uF mylar
C2	.1 uF mylar
C3	.005 uF mylar
C4	100 uF, 6V electrolytic
C5	33 uF, 10V tantalum electrolytic
R1	220 ohm, 1/4 watt
R2	10000 ohm, 1/4 watt
R3	2200 ohm, 1/4 watt
R4	1000 ohm, 1/4 watt
R5	50000 ohm, 10 turn precision potentiometer (see note 1)
IC1	Intersil IM6402 UART or equivalent (see note 2)
IC2	7400 Quad 2 input NAND gate
IC3	7402 Quad 2 input NOR gate
IC4	7404 Hex Inverter
IC5	74123 Monostable multivibrator
IC6	555 Timer
IC7	7805 Five volt regulator
T1	9 volt DC 300 mA calculator power supply transformer
S1	40 pin IC socket
S2, S3, S4	14 pin IC socket
S5	16 pin IC socket
S6	8 pin IC socket

Cinch 251-12-13-160 edge connector or equivalent
Amphenol DB-25S cable connector or equivalent
Radio Shack 276-153 printed circuit board or equivalent

NOTES:

1. The output of figure two is adjusted to 4800 Hz using R5. This provides a data transfer rate of 300 bps. A frequency counter is necessary to accurately accomplish this.

2. Any pin compatible UART may be used such as a S1883 or AY-3-1015.

3. The circuit may be implemented using MOS devices for most gates. This will result in reduced power consumption.

4. Pin 18 and pin 19 are pins of the DB-25S connector and provide the -20 mA and +20 mA signals for the printer. All other pin assignments are on the PET IEEE edge connector.

There's been quite a lot written about disk files, and tape files, but very little about the PET's logical files. Here are some suggestions and a routine which may have some utility.

When you OPEN a file, you specify a logical file number, a device number, and (optionally) a secondary address, and filename. Then the PET does what is necessary. This information is saved, the number of files open is incremented and checked, and action is taken to open the file.

The file data is stored in three tables - logical files, devices, and secondary addresses. The tables start at $0251 ($0242 old ROM), $025B ($024C), and $0265 ($0256) respectively. The count of number of files is at $00AE ($0262). The filename is not saved - it's sent to the device.

The secondary address is OR'd with $60, and then stored. If no SA is specified, a value of $FF will be found in the table.

When a file is closed, the file last opened is swapped into its place. So if you open files 1, 3, and 5; and then close 1, the file table contains entries for 5 and 3 (plus a dummy copy of 5).

Now, we can write a routine to check on file status. Here it is:

```
10 REM FIND FILE STATUS
15 INPUT"LOGICAL FILE NUMBER ";LF
20  NF  =  PEEK(174):IF  NF  =  0  THEN  PRINT  "NO  FILES
OPEN":END
30 PF = 0:FOR X=1 TO NF:IF PEEK(592+X) = LF THEN PF = X
40 NEXTX:IF PF = 0 THEN PRINT "FILE" LF "NOT OPEN":END
50 PRINT "LOGICAL FILE";LF "OPEN"
52 PRINT "ON DEVICE";PEEK(602+PF)
55 P = PEEK(612+PF) AND 159 :IF P = 159 THEN P = 0
60 PRINT "WITH SECONDARY ADDRESS";P
```

To use this, just open the files, and GOTO10. If you RUN the program, you'll abort all files.

You could use a version of this routine if you're doing dynamic LOADs - files are not affected by the LOAD, and you can find them.

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	235-236	(80-column) Output vector
00F9-00FA	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
0353-0380	851-896	DOS command string buffer
03EE-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	(80-column) CRT Controller
F000-FFFF	61440-65535	Reset, I/O handlers, Tape routines

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:
B3FF-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-BA8D	Perform PRINT#
BA8E-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-D35B 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-F19D 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-F767 Perform SAVE
F768-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-FA9B 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
```

Index Transactor #10