Notes on this document:

Regrettably ***page 5*** is *missing*. Should I ever find it, I'll promptly update this PDF document. Sorry !

Valentin Albillo, 05-11-2021

HIDDEN POWER: The Productivity Pac's Binary Keywords - by Don Person

Now that the HP Productivity Pac has become such a popular item, a lot of HP owners have at their disposal an added bonus that they may not be aware of. This bonus is a set of very useful new binary programs. The 'BIG THREE' among these programs are "FILE/80BIN", "STEVIE" and "BIN24", which are found on the "FILE/80" disk.

The thrust of this article is a discussion of a number of the keywords that drive the "FILE/80" program, and the major program examined is "FILE/80BIN". The new Basic keywords supplied by this program fall into several categories, but the most interesting ones are implementations of new disk operating routines, some new array sorting routines and general system enhancements.

Here is the entire list of new FILE/80BIN keywords:

KEY SEARCH	RECORD SEARCH	SET BUFFCOL	REMOVE BLANKS
OPEN FILE	CLOSE FILE	RSECTOR	WSECTOR
ICHR$	INUM	RCHR$	RNUM
NUMOK?	SORT BUFFER IS	SORT PARMS	REVERSE
ICH2$	INUM2	MASK	UNMASK
BLANK?	PTARRAY IS	SIFT	MAKEHEAP
SORTHEAP	POP	POPALL	MSUS$
DISK FREE	DISC FREE	CAT$	VOL$
PRTIS	DTYPE	WPROT?	GIVEBACK

Let's start with some general purpose language enhancements (examples follow some of the descriptions).

PRTIS - Returns the address of the current PRINTER IS declaration.

```
DISP PRTIS
701
```

DTYPE(msus$) -The argument for this function is a string value representing a mass storage address.
The function returns a numeric result indicating the type of drive present.

```
DTYPE (":D700")
0
```

MASK [string reference] - This command performs a function similar to HGL$, to highlight a particular string on the screen. It sets the most significant bit in each character of the specified string. It may not be combined with DISP statements. 'MASK A$' would be OK. 'DISP MASK A$' would NOT be OK, and will return an error. This command is of most significant use in file indexing.

UNMASK [string reference] - This command clears the MSB in each character specified in the string reference, and is similar to the UNHGL command found in the BIN24 binary.

```
UNMASK A$ or UNMASK HEADER$[31,32]
```

NUMOK?(string) - This function evaluates a string to determine if the string in question is actually a representation of a number or not. 1= NUMBER OK; 0=SORRY, NOT A NUMBER. This is quite useful when used with the LINPUT statement, or the FORM binary program available from the HP User's library.

BLANK?(string) - This function tests a string to determine if there are any characters present other than spaces. 1=YES, it is blank; 0=NO, there are characters other than blanks here.

```
BLANK?(" ")
1
BLANK?(" * ")
0
```

POP - This command is programmable, but may also be executed in calculator mode. When executed, it removes ONE address from the top of the RTN (Return) stack. This allows for recursive use of subroutine calls and is a useful debugging tool as well. Executing it with no addresses on the stack of course causes an error, as does executing it in calculator mode with a de-allocated program present.

POPALL - This command is similar to POP but instead clears the stack completely. It does not cause an error when executed in and of itself. Executing a RETURN subsequently can however. (POP may already be familiar to programmers who have used other than SERIES 80 computers, while POPALL is not that common.)

Now for some fabulous new file manipulation statements. All of these new statements can be executed in 'calculator' mode too! These are direct sector read/write operations:

OPEN FILE [filename],buffer# - This statement opens a buffer, declared by buffer#, and assigns it to the named entry in the disk directory. The [filename] may declare a DATA file, a PROGRAM file or a BINARY PROGRAM file; in short any catalog entry. This opens the door to a way of examining Binary programs and compiled BASIC programs while stored in disk files (See our SNOOP'R program in this issue - Editor). It can also be applied for file recovery or examination when the programmer or user is uncertain of the contents. It allows for very efficient usage of disk space as well. The burden of management falls on the programmer.

```
OPEN FILE "NAME$",1 ! (Valid buffer numbers are 1 to 12)
```

CLOSE FILE buffer# - Closes the referenced file buffer.

RSECTOR [string],sector#,file buffer # - When executed, the indicated sector is read from mass storage to the specified string. The string MUST be dimensioned to 256 characters, or an error will result. The first sector in any file is sector zero. The file buffer number referenced must have been previously declared with an OPEN FILE statement. The entire sector will be read without regard to any existing file markers, including the EOF marker.

```
DIM A$[256]
RSECTOR A$,0,1 ! Reads the first sector of the file.
```

WSECTOR [string],sector#,file buffer# - This is the complement of RSECTOR. The entire sector will be re-written using the named string. The rules are the same as RSECTOR.

```
WSECTOR A$,0,1 ! Writes to the first record of A$.
```

DISK FREE A,B,MSUS$ or DISC FREE A,B,MSUS$ - This returns two values to the specified variables. The first variable receives the total number of available sectors on the specified disk. The second variable receives the largest contiguous block of sectors from the indicated MSUS. (A and B are variables used for example only). If there are no null files then the total number of sectors equals the largest block. This is the first instance I have encountered where duplicate syntax is supplied to cover both the accepted and "corrupt" spelling of the word "disk".

```
DISK FREE TOTAL_SECTORS, LARGEST_BLOCK, ":D700"
DISP TOTAL_SECTORS
298
DISP LARGEST_BLOCK
277
```

MSUS$(".VOLUME") - Returns the address of the specified volume.

```
MSUS$(".VOL")
:D700
```

VOL$(msus) - This keyword returns the name of the volume present in the referenced drive.

CAT$(numeric reference) - This keyword allows the programmer to access the directory of the disk in the current drive and to treat the entries as elements in a string array. It returns a string with the first ten characters indicating the name of the entry, then four characters for file type, then four more for 'file sector length', and then four more indicating the total number of records. eg:

```
CAT$(3)
FILENAME DATA02560012
```

WPROT?(MSUS) - When executed, the disk in the currently declared drive is checked for the presence of a write protect tab. 1=WRITE PROTECTED, 0=NO TAB PRESENT. This is a much simpler method for checking if a disk is write protected than using ON ERROR declarations to trap write protect errors.

Here is a short program that illustrates the direct sector read command. Of course, all the statements may be executed in 'calculator' mode too.

```
10 DIM A$[256] ! FOR OUR SECTORS
20 OPEN FILE CAT$(1),1
30 RSECTOR A$,0,1
40 DISP A$
50 END
```

With these keywords under our belt, a brief digression seems in order. It may be useful to know how the HP BASIC system stores and retrieves mass storage data. Here then is a brief explanation.

When an ordinary file buffer writes to a file under BASIC system control, it must always mark the contents of a file with markers to enable the CPU to determine how to treat retrieved data. A numeric value in mass storage is in coded binary form, and always occupies 8 bytes. The first character is a CONTROL 'C' ,which you may recognize as a lowercase n with a bar over it. The seven bytes which make up the number follow, with the most significant byte to the left.

When writing strings to a file, the BASIC system marks strings with a specific marker format. The string is preceded by CHR$(223) and is followed by two bytes, least significant byte first. These, when decoded, are the length of the string.

The end-of-file marker is CHR$(239), and is easily recognized visually in a file,thanks to the 86/87's nice crt display set. The system makes no distinction in the file between individual data and data arrays.

Because FILE/80BIN stores numbers, it too must code numbers for inclusion in files. It uses a format which is similar to the one that the BASIC system uses, but which is incompatible without additional bit manipulation. Here is the keyword pair which accomplishes this.

RCHR$("string representation of a real number") - This returns a 9 byte string. The first byte is CHR$(21). The second byte tells the system where to put the radix. The next six bytes are the real number in Binary coded Decimal (BCD) format, where each byte conveys two 'nibbles' of numeric BCD data. The last character is always a '1' or '0'. A '1' indicates that the originally referenced string was a number. Zero = no number here.

```
RCHR$ ("1234.5678")
```

RNUM("RCHR$") - When executed with the first 8 characters of a string generated by RCHR$, it returns a real number.

The FILE/80BIN program has a similar capability for strings. In this case, the task is performed by an INTEGER character string conversion. Here's how:

ICH2$(INTEGER) - The integer value must be a number from 0 to 65534. A string is returned, most significant bit first, whose bits represent the original number. Numbers greater than 65534 will generate two bytes of CHR$(255).

```
ICH2$ (18512)
HP
```

INUM2 (string) - Complement of ICH2$. This returns an integer from the 2 bytes. If more than 2 bytes are contained in the string, only the first two will be decoded. If only one character is present, a null CHR$(0) is assumed to precede it.

```
INUM2 ("HP")
18512
```

ICHR$ (string expression of a number,length of output string) - This returns a string representation of length specified by the second number. The bits of the string expression represent the binary value of the numeric reference, and only integers are allowed. Note that the last character of the string will be '1' or '0' for "number true or false", and that the first bit of the first byte is the sign bit (set = positive number, 0 = negative number).

```
ICHR$ ("79999999999999",6) ! Largest number that can be converted
```

INUM("string reference") - This returns a numeric value of the string. The conversion algorithm allows negative as well as positive values. If the first character is MASKed, then the natural binary value of the bits of the string will be converted to a positive number. If the mask bit is not set, the number is interpreted as a negative integer. In either case, the first bit is not part of the actual conversion process, but the other 7 bits of the leading character are. Maximum string length for conversion is 7 bytes.

```
INUM("HP")
-14256
H$="HP" @ MASK H$
INUM (H$)
18640
```

Unfortunately, though by design, HP did not choose to duplicate functions embodied in their BASIC . This makes the commands slightly more difficult to to use. To read the length of the first string in a file, you might key in the following :

```
10 pS=POS(SECTOR$,CHR$(223))
20 STRING_LENGTH=INUM2(REV$(SECTOR$[pS+1,pS+2]))
```

I have not touched on the following set of keywords.

SORT BUFFER IS	SORT PARMS	PTRARRAY IS	REMOVE BLANKS
MAKEHEAP	SORTHEAP	SIFT	GIVEBACK
SET BUFFCOL	REVERSE		

These are the implementation of a combined array/bubble sorter.

SORT BUFFER IS [string reference] - Use the largest string you can handle. DIMensioning the string reference at 65534 is desirable. This is the bubble column.

PTRARRAY IS [a one or 2 dimensional array] - This is the POINTER ARRAY.

I have not had the time to investigate thoroughly this group of keywords, so for now I can only suggest that the hardy among you investigate further, and share your results. Perhaps in a subsequent issue, I will be able to complete the guide started here.

Briefly, here are other keywords supplied by some of the short binaries in the Personal Productivity Pac:

BIN15 : CHAINA LOADA

BIN25 : DISK FREE (DISC FREE) CAT$() MSUS$ VOL$() PRTIS DTYPE

This group of keywords are the same as those supplied by FILE/80BIN. In addition, the following are added.

EXTEND# (BUFF#,NUMRECS) - This keyword allows you to add to the number of records in the LAST file on the present disk. If buffer# 1 is ASSIGNed to "TEST" then to add 9 more records to the TEST file, execute:

```
EXTEND# 1,9
```

EXTEND LAST FILE - Similar to EXTEND#.

MAY I HAVE (BUFFNO,DESIRED # OF RECS) - Returns a one (yes) or zero (no), depending on the number of free records.

```
MAY I HAVE (1,8)
1
```

The BIN43 program supplies some extra string manipulation tools:

REV$ - This allows you to reverse the order of the characters in the referenced string.

```
A$="ABCD"
DISP REP$ (A$)
DCBA
```

REP$ (string$,#reps) -This is the same as the keyword RPT$ supplied by by UTIL/1 and the AP ROM. For the record:

```
REP$ ("X",4)
XXXX
```

TRIM$ (STRING$) - This function allows the removal of leading and trailing blanks in the string.

```
B$=TRIM$(A$)
```

SCOPY (string$) - Screen copy function. AREAD revisited, but improved.

If you happen to have a copy of the new MIKSAM ROM manual in your possession (HP manual 00087-90614), you have a pretty reasonable syntax guide to the keywords supplied by "STEVIE". STEVIE, by the way, is a product of the PEACHTREE group produced for HP. The ROM implementation is almost the same as the STEVIE binary.

BIN24 is the driver for the Basic word processor program WORD/80. This binary was designed to be a little tricky to use, as it is initiated by the command "BIN24START". When executed, the program takes control of all keys including RESET. If you are simply "hacking" this can make it a difficult program to adopt. Once mastered, it offers very flexible control of the screen and keyboard. The exit command is "BIN24EXIT". A simple method for testing various keywords is to program the entry command and then an ON TIMER# branch to a line to execute the exit statement. This program is illustrated throughout the FILE/80 program listing.