PAK3LEX – Packing Long Integers

Abstract

PAK3LEX is a small assembly-language LEX file written in 1991 for the HP-71B pocket computer to pack long integers in the range from 0 to 16,777,215 into 3-character strings and, conversely, to unpack a 3-character string into the equivalent long integer value.

Keywords: *pack, unpack, long integers, LEX file, assembly language, Hewlett-Packard, HP-71B, pocket computer*

1. Introduction

PAK3LEX is a small *(90 bytes)* assembly-language *LEX (Language Extension*) file that I wrote in 1991 for the *HP-71B* pocket computer which adds two new *BASIC* keywords (both are functions) to pack long integers in the range from *0* to *16,777,215* (2^{24}-1) into 3-character strings and, conversely, to unpack a 3-character string into the original, equivalent long integer value. To be efficient, the pack/unpack procedure must run as fast as possible, as the arrays involved are usually quite big and thus many elements will have to be processed. This is accomplished by implementing it in assembly language for maximum speed, in the form of the two new keywords **`PAK`** and **`UNPAK`** provided by this *LEX* file.

The HP-*71B BASIC* language dialect allows the user to define **`INTEGER`** variables and arrays, but they limit the range to integer values from *-99,999* to *+99,999* and use 9.5 bytes for a simple variable and 3 bytes per element (plus 9.5 bytes overhead) for arrays. Using instead **`REAL`** variables/arrays expands the integer range from *-999,999,999,999 to +999,999,999,999* and the memory requirements are the same for a simple variable but 8 bytes per element (again, plus 9.5 bytes overhead) for arrays, which is much more expensive for large arrays.

If the user just needs to store many long integer values in the range from *0* to *16,777,215* this can be done much cheaper in terms of *RAM* by using the new keyword **`PAK`** to convert each integer value into a 3-character string which occupies just 3 bytes and can be stored as part of a long string (maximum savings) or as an element of a string array. This means savings of up to 5 bytes per element vs. using a **`REAL`** array or alternatively, using 3 bytes per element but with a maximum value up to *16,777,215* vs. 99,999 when using an **`INTEGER`** array. The element can be converted back from its packed 3-character string representation to the original long integer value by using the new keyword **`UNPAK`**.

2. Program Listing

This following source listing should be entered into a *TEXT* file, possibly using the *ASCII Text Editor* available in the *Hewlett-Packard*'s *HP-71B FORTH/Assembler ROM*. The text file should then be assembled using the *Assembler* in said *ROM*, which will generate the *PAK3LEX* file in memory. The resulting *LEX* file should be *90 bytes* long once assembled.

Alternatively, you may use any other suitable text editors or *HP-71B* cross-assemblers (such as those written for execution on a PC) and then send the resulting *LEX* file to the *HP-71B* via *HP-IL* or by reading it from mass storage (disk or tape). Consult your particular assembler's documentation for the required procedures.

Once *PAK3LEX* is in memory, its two keywords are available for inmediate use either from the command line (but not in *CALC* mode) or in programs. Don't forget to turn off and on the machine to register the new *LEX* file with the *HP-71B*'s operating system.

```
       LEX 'PAK3LEX'  name of the lex file: PAK3LEX
       ID #54         LEX id: 54
       MSG 0          no error messages
       POLL 0         no poll handling
EXPR   EQU #0F23C     resumes expression evaluation
FLOAT  EQU #1B322     converts a decimal integer to floating point
HXDCW  EQU #0ECB4     converts a hexadecimal integer to decimal
RJUST  EQU #12AE2     converts a floating point to decimal integer
DCHXW  EQU #0ECDC     converts a decimal integer to hexadecimal
FNRTN1 EQU #0F216     returns the value of the function to the system
POP1N  EQU #0BD1C     extracts a floating point from the stack
POP1S  EQU #0BD38     extracts a string from the stack
       ENTRY DTH      1st keyword: entry in DTH
       CHAR #F        1st keyword: is a function
       ENTRY HTD      2nd keyword: entry in HTD
       CHAR #F        2nd keyword: another function
       KEY 'PAK$'     1st keyword: PAK$(N)
       TOKEN 1        token 1 in this lexfile
       KEY 'UNPAK'    2nd keyword: UNPAK(S$)
       TOKEN 2        token 2 in this lexfile
       ENDTXT         end of tables

       NIBHEX 811     PAK$(N): string function, 1 numeric parameter
DTH    GOSBVL POP1N   get parameter (floating point) from the stack
       GOSBVL RJUST   convert it to a decimal integer
       C=A    W       copy it to C
       GOSBVL DCHXW   convert it to hexadecimal integer
       P=     5       point to the 6th hex digit
       D1=D1- 6       send digits 1st-6th to the stack
       DAT1=C WP
       C=0    W       preparing the header: 6-nibble string
       P=     0
       LCHEX  60F
       SETDEC
       D1=D1- 16      send header to the stack
       DAT1=C W
       GOVLNG EXPR    return to system

       NIBHEX 411     UNPAK(S$): numeric function, 1 string parameter
HTD    SETHEX
       GOSBVL POP1S   extract string header from the stack
       C=0    W       extract string (6 nibbles) from the stack
       P=     5
       C=DAT1 WP      its 6 nibbles are placed in C
       D1=D1+ 6
       GOSBVL HXDCW   convert those 6 hex digits to decimal
       GOSBVL FLOAT   convert this decimal integer to floating point
       C=A    W
       GOVLNG FNRTN1  return to system
       END
```

3. Usage Instructions

Once *PAK3LEX* is in memory its two *BASIC* keywords are available and can be used from the command line (but not in *CALC* mode) or in programs, using this syntax:

- Packing: **PAK$**(<*numeric expression*>) *string function with one numeric parameter*

  where the *numeric expression* must evaluate to an integer-valued real in the range *0* to *16,777,215*. The result will be a 3-character string. *See* ***Note 1***.

- Unpacking: **UNPAK**(<*string expression*>) *numeric function with one string parameter*

  where the *string expression* must evaluate to a 3-character string. The result will be an integer-valued real in the range *0* to *16,777,215. See* ***Note 1***.

4. Examples

The following example can be useful to check that the *LEX* file is correctly loaded in memory:

4.1 Example 1

From the command line, pack and then unpack the integer value *4276803*. (**>** is the command-line prompt):

```
>PAK$(4276803)    END LINE    →    ABC
>UNPAK("ABC")     END LINE    →    4276803
```

Notes

1. ***Very important:*** In order to be as short and fast as possible this *LEX* file does not do any kind of error checking,. Therefore it is *mandatory* and you must absolutely make sure that:

- for **PAK,** the numeric expression passed to it must evaluate to an integer-valued real in the range *0* to *16,777,215*.
- for **UNPAK,** the string expression passed to it must evaluate to a string *exactly* 3 characters long, no more, no less.

Failure to comply with these requirements risks unpredictable and possibly dangerous behaviour (i.e: corrupting memory and/or the file system in *RAM*, freezing and/or having to reset the machine).

2. This *LEX* file can be very easily modified to cater for other ranges. For instance, it's trivial to modify it to pack/unpack long integers in the range *0* to *4,294,967,295* (or alternatively *-2,147,483,647* to *+2,147,483,647)* in just 4 bytes using 4-character strings (thus saving 4 bytes [50%] over storing them in a **REAL** array).

3. It's also possible to modify it to cater for non-integer values. For instance, I once created a modified version of this *LEX* file to pack fixed-point values in the range *-82.00000* to *+84.77215*, i.e.: non-integer numbers with up to 5 decimal places, in just 3 bytes. This can be useful, e.g.: to store many readings from a digital voltmeter connected to the *HP-71B* via *HP-IL*, for later processing. Using this modified *LEX* file each reading takes just 3 bytes, even though it has 7-digit, 5-decimal accuracy, and the packing/unpacking procedure runs at the same speed as the all-integer case.

References

Hewlett-Packard HP 82441A *Forth/Assembler ROM Owner's Manual* for the HP-71

Copyrights