Common Trace Format - Bug #888

BE vs LE bitfield packing is not easy to understand + undefined behaviour for mixed BE/LE

05/21/2015 03:05 PM - Philippe Proulx

Status:	New	**Start date:**	05/21/2015
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

In section 4.1.5 (Integers), examples should be written to explain how integers are packed one after the other depending on the byte order, because it is not obvious with only the current text.

Here's an example for both big-endian and little-endian situations.

Big-endian

Big-endian words are packed from left to right within a sequence of bytes.

Let's pack the following words, in this order, each one having a 1-bit alignment (each individual letter represents a bit, the first one being the most significant):

- ABCDEFGHIJ (10 bits)
- klmno (5 bits)
- PQRS (4 bits)
- tuvwxyz (7 bits)

Starting at a byte offset, here are the steps (# are eventual bits of padding, or bits that belong to other, eventual fields):

```
ABCDEFGH IJ######
ABCDEFGH IJklmno#
ABCDEFGH IJklmnoP QRS#####
ABCDEFGH IJklmnoP QRStuvwx yz######
```

Little-endian

Little-endian words have less straightforward packing rules.

Always start with the least significant bits of the word to pack, and, from right to left within a given byte (LSBs to MSBs), put as many bits as you can of said word's least significant bits, in the usual order (MSBs of the word's substring are still on the right in relation to the LSBs).

Let's take the same words as the big-endian example above. Starting at a byte offset, here are the steps:

```
CDEFGHIJ ######AB
CDEFGHIJ #klmnoAB
CDEFGHIJ SklmnoAB #####PQR
CDEFGHIJ SklmnoAB vwxyzPQR ######tu
```

Bit position

In both examples, at the end, the current bit position is 26. Only, in big-endian we are travelling from left to right within the sequence of bytes, whereas in little-endian, we are travelling from right to left within each byte of the sequence:

Big-endian:

```
########
^
0

ABCDEFGH IJ######
```

```
-------> ->^
          10

ABCDEFGH IJklmno#
          ---->^
               15

ABCDEFGH IJklmnoP QRS#####
               > -->^
                    19

ABCDEFGH IJklmnoP QRStuvwx yz######
                    ----> ->^
                            26
```

Little-endian:

```
########
       ^
       0

CDEFGHIJ ######AB
<-------      ^<-
             10

CDEFGHIJ #klmnoAB
         ^<----
         15

CDEFGHIJ SklmnoAB #####PQR
         <            ^<--
                      19

CDEFGHIJ SklmnoAB vwxyzPQR ######tu
                  <----         ^<-
                                26
```

Aligning fields within bytes is easy when we understand this bit position direction. For example, if we want to align the second word (klmno) at multiples of 4, here's the result in both byte orders:

Big-endian:

```
ABCDEFGH IJ##klmn oPQRStuv wxyz####
                              ^
                              28
```

Little-endian:

```
CDEFGHIJ lmno##AB xyzPQRSk ####tuvw
                             ^
                             28
```

This is also why packing words which have different byte orders within one byte is **undefined**, and the specification should explicitly tell so as well.

Floating point numbers

Floating point numbers obey to the same rules. A CTF floating point number, in its binary form, is a word with a length equal to the sum of its mantissa and exponent lengths.

Consider the 1-bit aligned integer 347 (word 101011011), followed by -3.1415927 represented on an 8-bit exponent, 24-bit mantissa, 4-bit aligned floating point number (word 11000000010010010000111111011011):

Big-endian:

```
########
^
```

```
0

10101101 1#######
          ^
          9

10101101 1###1100 00000100 10010000 11111101 1011####
                                                 ^
                                                 44
```

Little-endian:

```
########
       ^
       0

01011011 #######1
               ^
               9

01011011 1011###1 11111101 10010000 00000100 ####1100
                                                ^
                                                44
```

Strings

The packing rules do not apply to CTF strings since their characters (always 8-bit words) must be aligned on multiples of 8.

History

#1 - 05/21/2015 03:09 PM - Philippe Proulx

Examples with appropriate Unicode characters: https://upl.io/aw7p9g.

#2 - 05/21/2015 03:32 PM - Mathieu Desnoyers

- Subject changed from BE vs LE bitfield packing is not easy to undestand + undefined behaviour for mixed BE/LE to BE vs LE bitfield packing is not easy to understand + undefined behaviour for mixed BE/LE

#3 - 05/22/2015 07:15 PM - Philippe Proulx

- Description updated