"Floating Point Addition Example"

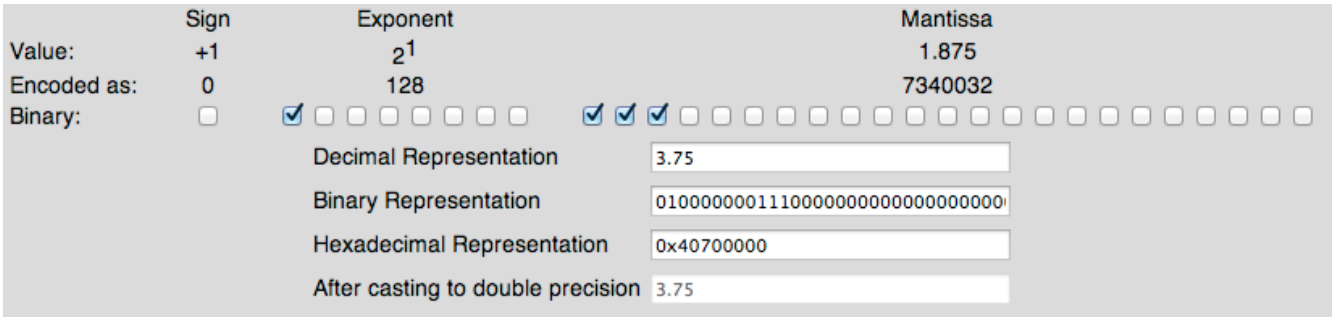
For posting on the resources page to help with the floating-point math assignments.

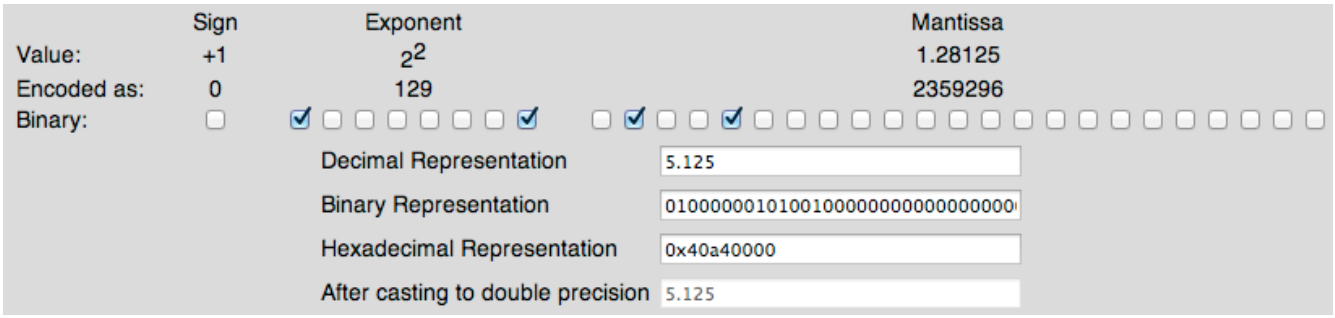
Problem

Add the floating point numbers 3.75 and 5.125 to get 8.875 by directly manipulating the numbers in IEEE format.

Step 1: Decompose Operands (and add implicit 1)

First extract the fields from each operand, as shown with the h-schmidt converter:





For 3.75, the sign bit is 0 (+), the exponent is 128 (1 unbiased), the mantissa (including the implicit 1 shown in bold) is:
0000 0000 **1**111 0000 0000 0000 0000 0000 = 0x00f00000

For 5.125, the sign bit is 0 (+), the exponent is 129 (2 unbiased), the mantissa (including the implicit 1 shown in bold) is:
0000 0000 **1**010 0100 0000 0000 0000 0000 = 0x00a40000

Step 2: Equalizing Operand Exponents

To equalize exponents we must shift one or the other of the mantissas and adjust the corresponding exponent. If the first exponent is smaller than the second, we shift the first mantissa to the right and add the absolute difference in exponents to the first exponent. If vice versa, we do the same to the second mantissa and exponent.

Note that we could end up shifting all the bits off the right side, leaving a zero? How do floating-point units handle this? How about operands with the value 0.0, is that a special case?

For this example the first exponent is 128, second exponent is 129, absolute difference is 1, so first exponent is smaller, so we must adjust the first mantissa and exponent, and leave the second mantissa and exponent unchanged.

- Shift first mantissa right by 1: 0x00f00000 >> 1 = 0x00780000
- Increase the first exponent by 1: 128 + 1 = 129

Step 3: Convert operands from signed magnitude to 2's complement

For each operand that is negative, convert the mantissa to 2's complement by inverting the bits and adding 1. Neither operand is negative in this example, so nothing needs to be done.

Step 4: Add Mantissas

Both operands have an exponent of 129, so we can just add mantissas to get a positive result with the same exponent. Note that this stop will work regardless of the sign of the exponents, because they are in 2's complement format.

0x00780000 + 0x00a40000 = 0x011c0000 = 0000 000**1** 0001 1100 0000 0000 0000 0000

Note that the leftmost 1 bit is no longer in bit 23, as required for the IEEE format!

Step 5: Convert result from 2's complement to signed magnitude

If the result is negative, convert the mantissa back to signed magnitude by inverting the bits and adding 1. The result is positive in this example, so nothing needs to be done.

Step 6: Normalize Result

Because the leftmost 1 bit is not in the right place, we must shift the mantissa right or left to put it back into the IEEE format, and adjust the exponent accordingly. If the leftmost 1 bit is left of bit 23, we must shift the mantissa to the right and increase the exponent. If the leftmost 1 bit is at bit 23, there is no normalization required. If the leftmost 1 bit is right of bit 23, we must shift the mantissa to the left and decrease the exponent. In the example, we see the first case:

BEFORE NORMALIZATION

Sign of result = 0
Exponent of result = 129
Mantissa of result = 0000 0001 0001 1100 0000 0000 0000 0000 = 0x011c0000

The leftmost 1 bit is bit 24, so we must shift the mantissa right by 1 and add 1 to the exponent:

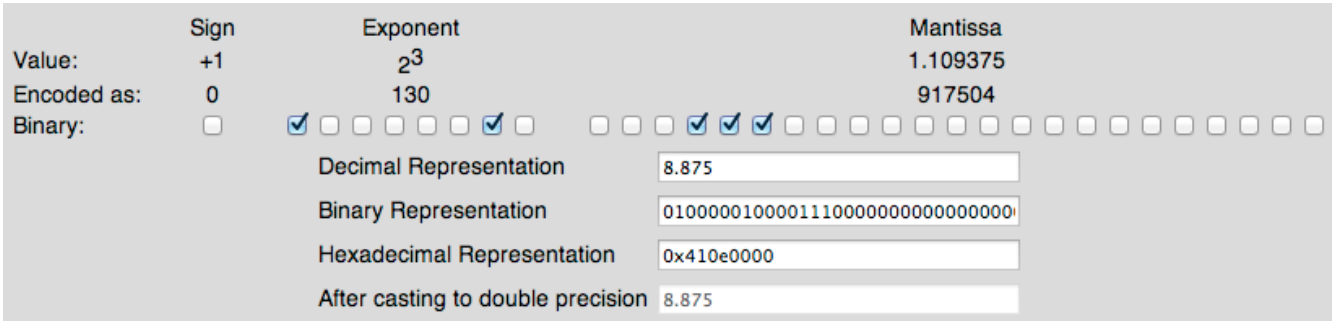
AFTER NORMALIZATION

Sign of result = 0
Exponent of result = 129 + 1 = 130 = 100000010
Mantissa of result = 0x011c0000 >> 1 = 0x008e0000 = 0000 0000 **1**000 1110 0000 0000 0000 0000

Step 7: Compose Result (and remove implicit 1)

Sign of result = 0
Exponent of result = 130 = 10000010
Mantissa of result (with implicit 1) = 0x008e0000 = 0000 0000 **1**000 1110 0000 0000 0000 0000
Mantissa of result (remove implicit 1) = 0x000e0000 = 0000 0000 0000 1110 0000 0000 0000 0000

So final result is: 0 10000010 000 1110 0000 0000 0000 0000 = 0x410e0000, as confirmed by the h-schmidt converter:



To summarize the result of the addition:

3.75 (0x40700000) = + 5.125 (0x40a40000) = 8.875 (0x410e0000)