Conversion from Decimal to Floating Point Representation

Written by: Thomas Finley, Source: https://www.cs.cornell.edu/~tomf/notes/cps104/floating.html

Say we have the decimal number 329.390625 and we want to represent it using floating point numbers. The method is to first convert it to binary scientific notation, and then use what we know about the representation of floating point numbers to show the 32 bits that will represent it.

If you know how to put a decimal into binary scientific notation, you'll get no benefit from reading this. If you don't, read this.

The first step is to convert what there is to the left of the decimal point to binary. 329 is equivalent to the binary 101001001. Then, leave yourself with what is to the right of the decimal point, in our example 0.390625.

Yes, I deliberately chose that number to be so convoluted that it wasn't perfectly obvious what the binary representation would be. There is an algorithm to convert to different bases that is simple, straightforward, and largely foolproof. I'll illustrate it for base two. Our base is 2, so we multiply this number times 2. We then record whatever is to the left of the decimal place after this operation. We then take this number and discard whatever is to the left of the decimal place, and continue with this progress on the resulting number. This is how it would be done with 0.390625.

```
0.390625 * 2 = 0.78125      0
0.78125  * 2 = 1.5625       1
0.5625   * 2 = 1.125        1
0.125    * 2 = 0.25         0
0.25     * 2 = 0.5          0
0.5      * 2 = 1            1
0
```

Since we've reached zero, we're done with that. The binary representation of the number beyond the decimal point can be read from the right column, from the top number downward. This is 0.011001.

As an aside, it is important to note that not all numbers are resolved so conveniently or quickly as sums of lower and lower powers of two (a number as simple as 0.2 is an example). If they are not so easily resolved, we merely continue on this process until we have however many bits we need to fill up the mantissa.

As another aside, to the more ambitious among you that don't know already, since this algorithm works similarly for all bases you could just as well use this for any other conversion you have to attempt. This can be used to your advantage in this process by converting using base 16.

```
0.390625 * 16 = 6.25        6
0.25     * 16 = 4           4
0
```

If we convert simply from hex to binary, 0x64 is 0110 0100, which is the same result as the 011001 yielded above. This method is much faster.

Anyway! We take those numbers that we got, and represent them as .011001, placing them in the order we acquired them. Put in sequence with our binary representation of 329, we get 101001001.011001. In our binary scientific notation, this is $1.01001001011001 * 2^8$. We then use what we know about how single precision numbers are represented to complete this process.

The sign is positive, so the sign field is 0.

The exponent is 8. 8 + 127 = 135, so the exponent field is 10000111.

The mantissa is merely 01001001011001 (remember the implied 1 of the mantissa means we don't include the leading 1) plus however many 0s we have to add to the right side to make that binary number 23 bits long.

Since one of the homework problems involves representing this as hex, I will finish with a hex number.

0	10000111	01001001011001000000000

Then we break it into four bit pieces (since each hexadecimal digit is the equivalent of 4 bits) and then convert each four bit quantity into the corresponding hexadecimal digit.

Binary	0100	0011	1010	0100	1011	0010	0000	0000
Hex	4	3	A	4	B	2	0	0

So, in hexadecimal, this number is 0x43A4B200.