Floating Point Numbers [Andy Chong Sam]

The IEEE 754 standard states that floating point numbers are stored with 32 bits:
1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa.

As an example let's see how 320.625 is represented in binary.

(1) Translate to Binary

We first represent the whole part of the number in binary so for 320 that is 101000000.

To represent the fractional part in binary, we'll continuously multiply the fractional part by 2 until we get a result of 1:

$$0.625 \times 2 = 1.25 \text{ save the } 1$$
$$0.25 \times 2 = 0.5 \text{ save the } 0$$
$$0.5 \times 2 = 1.0 \text{ save the } 1$$

The last step produces a 1, so we stop.

0.625 is 0.101 in binary.

We now have a representation for 320.625 in binary:

$$101000000.101$$

(2) Write in Scientific Notation

The result from part 1 becomes:

$$1.01000000101 \times 10^{8}$$

(3) Add Bias

127 is added to the exponent. The result from part 2 becomes:

$$1.01000000101 \times 10^{135}$$

(4) Determine the Sign

320.65 is positive so the sign bit will be 0.

(5) Translate Exponent to Binary

From part 3, 135 in binary is:

$$10000111$$

(6) Extract the Mantissa

The mantissa is everything to the right of the decimal point from part 3:

$$01000000101$$

(7) Collect Result

Combine the results from steps 4, 5, and 6. Pad zero's to the mantissa as needed. The binary representation of 320.65 is:

$$0\ 10000111\ 01000000101000000000000$$

We can always take a binary floating point number and determine its decimal equivalent. The formula used is:

$$d = (-1)^s(1 + m)\ 2^e$$

In this expression, s is the sign bit, m is the decimal representation of the mantissa, and e is the decimal representation of the exponent. With our problem, $s = 0$. We now translate the mantissa, 01000000101000000000000, back into decimal: $m = 2^{-2} + 2^{-9} + 2^{-11}$. For the exponent, 10000111 is the binary representation of 135, if we subtract the bias, 127, we get $e = 8$

So the decimal is: $(-1)^0(\ \ 1 + (2^{-2} + 2^{-9} + 2^{-11})\ \ )2^8 = 320.625$

An additional example is provided here:

Represent -23.15625 in binary using the IEEE 754 standard.

23 in binary is 10111. Let's figure out the fractional part:

$$\begin{aligned} 0.15625 \times 2 &= 0.3125 \quad \text{save the 0} \\ 0.3125 \times 2 &= 0.625 \quad \text{save the 0} \\ 0.625 \times 2 &= 1.25 \quad \text{save the 1} \\ 0.25 \times 2 &= 0.50 \quad \text{save the 0} \\ 0.50 \times 2 &= 1.0 \quad \text{save the 1} \\ &\text{We have a 1, so we stop} \end{aligned}$$

So 23.15625 is 10111.00101.

Rewrite in scientific notation: 10111.00101 becomes 1.011100101×10^4

Add 127 bias: 1.011100101×10^4 becomes $1.011100101 \times 10^{131}$

The exponent, 131, in binary is 10000011.

We're dealing with a negative number so the sign bit will be 1.

The mantissa is 011100101.

We know the sign bit, exponent, and mantissa, so the binary representation will be:

$$11000001101110010100000000000000$$

As with any number system some fractional amounts cannot be perfectly represented. This is not unusual even using day to day decimal math. Consider the fraction $\frac{1}{3}$. There is no way to represent this perfectly in decimal, we'd simply get an infinite string of threes after the decimal point.

Try and convert the decimal 0.1 into a fractional binary decimal in the manner we discussed, you will get an unending sequence of bits. In this case some form of rounding system is typically used.