How does the computer represent real numbers?

Part I: Converting a real number into the single-precision floating-point representation.

1. Convert the number into binary.
2. Write this binary number in scientific notation. Move the radix point so that only a single “1” is to its left. The number of digits the radix point had to move becomes the exponent on 2. If you had to move the point left, the exponent is positive. If the point moved right, the exponent is negative.
3. Decompose this scientific notation into its three parts: sign (1), exponent (8), mantissa (23).
   1. Negative number means the sign bit is 1. Positive number means the sign bit is 0.
   2. The exponent value needs to be converted into an 8-bit biased-127 number. Simply add 127 to the exponent and treat this value as unsigned.
   3. The “1” to the left of the radix point is understood, not actually represented in the computer. For clarity, we can write it in parentheses so that we don’t forget about it. Write out the remaining digits of the mantissa, those digits that were to the right of the radix point. The mantissa has 23 bits, so often we have a lot of trailing zeros.

Example: Let’s look at the number –4.25.

1. This number in binary is –100.01
2. To convert this number to binary scientific notation, we need to move the “.” 2 places to the left. This means the mantissa becomes 1.0001 and the exponent on 2 is 2. Our number now looks like –1.0001 \* 22.
3. To write out the floating-point representation, we consider the sign, exponent and mantissa separately.
   1. The number is negative, so the sign bit is 1.
   2. The exponent is 2. Add 127, and we get 129. How do we write 129 as an 8-bit unsigned number? We go to the binary store with $129, and we can buy the following: 128+1. So, the exponent looks like 10000001.
   3. The mantissa is literally 1.0001. But the computer only stores the fractional part of the mantissa. The initial “1.” Is understood. We can write this as (1) 0001. So, the computer only stores the 0001 part of the mantissa. But there should be a total of 23 mantissa bits, so there are 19 trailing zeros. We can write the mantissa as (1) 0001 019.
      * Putting it all together, we have 1 10000001 (1) 0001 019

Let’s try some more examples, such as 78.0 and –0.375.

Part II: Interpreting a floating-point representation.

A real number inside the computer is expressed in the floating-point notation. How do we figure out what it is? Basically we reverse the conversion steps.

1. Identify the 3 parts of the representation: the sign (1), exponent (8), and mantissa (23). Note that the hidden bit does not count as part of the 23 bit mantissa, because it is not really stored in the computer.
   1. Sign bit of 1 means the number is negative. Sign bit of 0 means the number is positive.
   2. The 8 bit exponent is expressed in biased-127. First, interpret this number as if it were unsigned. Then subtract 127 from your answer. This will tell you the true value of the exponent.
   3. The mantissa is the hidden (1) bit plus the 23 stored mantissa bits. Write the “(1)” as “1.” And put the other mantissa bits after the radix point. Interpret this number as binary, and convert to base 10.
2. Multiply the power of 2 with the mantissa value, and attach the + or – sign to the front.

Example: What does this floating-point pattern mean? 0 10000010 (1) 011

1. We first identify the sign, exponent and mantissa parts of the number.
   1. The sign bit is zero, so this means the entire number is positive.
   2. The exponent is 10000010. This looks like 128+2 = 130 as an unsigned number. But it’s really written in biased-127, so we need to subtract the bias. 130 – 127 = 3. This means the exponent is 3, or in other words that the mantissa is multiplied by 23.
   3. The mantissa is (1) 011. (Note that the 20 trailing zeros are not shown). This is another way of saying the mantissa is the binary number 1.011. This means 20 + 2–2 + 2–3
2. Let’s multiply the mantissa by the exponent of two. And don’t forget the sign. But here the sign is positive. We obtain: + 23 \* (1.0112) = 23 \*(20 + 2–2 + 2–3) = 23 + 21 + 20 = 8 + 2 + 1 = 11

Try some more examples. We can do the conversion in both directions to check our answers.