Using floating point to represent numbers

Expressing very large and very small numbers
Consider the speed of light expressed in m/s, it is 300000000 m/s
Consider the voltage measured in an electrocardiogram, expressed in volts: 0.0002 V

These are examples of very small and very big numbers that we need to process. To ease arithmetic operations involving such large/small numbers, we express them in **scientific notation**, which involves a sign and a real number, M that is multiplied by 10 raised to a number, E:

$$+/- M * 10^{E}$$

M is called the **mantissa** and E is called the **exponent**.

If the mantissa is chosen so that it is greater than 1 and smaller than 10, then the number represented is **normalised**.

Binary representation of scientific notation: floating point representation

When a normalised number represented by using scientific notation, is expressed using binary numbers, then it is represented by **floating point** representation. The MSB of the binary number is used to express the sign of the represented number. A number of bits is used to represent the exponent and the remaining bits are used to represent the mantissa.

For a 32 bit number, the MSB bit represents the sign of the number, the following 8 bits represent the exponent and the remaining 23 bits represent the mantissa. The number is represented as:

$$\text{sign}* (1 + \text{mantissa}) * 2^{(\text{exponent} - 127)}$$

Excess notation is used to describe the exponent, thus permitting the representation of both positive and negative powers, leading to the representation of both very large and very small numbers.

Because the number is normalised (eg 1.2345), we store its fractional value in the mantissa and assume that its integral part is equal to 1. Therefore, the integral part does not need to be stored and a 1 is added to the number when the value of the number is calculated.

To convert a number represented by scientific notation to a floating point number:

1. determine the sign of the number. Set the MSB if negative, clear it if positive
2. convert the number to its binary equivalent
3. normalise it
4. determine the exponent by considering how many times the radix point needs to be shifted in order to obtain the original number

Example:
Consider the number:

$$6.625_{10}$$

Its binary equivalent is:

$$110.101$$

Another way to represent it is:

$$110.101 * 2^0$$

To normalise it, the radix point is shifted two times to the left. It is now represented as:

$$1.10101 * 2^2$$

The number stored to represent the mantissa is: 10101. Because 23 bits are allocated to represent it, the number is right adjusted to: **000 0000 0000 0000 0001 0101**

To calculate the number stored to represent the exponent, the offset of 127 is added to the above power of 2. The number stored to represent the exponent is: 127 + 2 = 129, which when converted to binary, is: **1000 0001**

Because we have a positive number, the sign bit is **0**.

Putting back all the above into a single, 32 bit number, we get:

0 1000 0001 000 0000 0000 0000 0001 0101

To convert a 32 bit floating point number back into decimal, the reverse procedure is followed:

1. the floating point number is broken into its constituent parts describing the **sign**, **exponent** and **mantissa**
2. the resulting numbers are applied to the formula used earlier to define the structure of the floating point number:

$$\textbf{sign} * (1 + \textbf{mantissa}) * 2^{(\textbf{exponent} - 127)}$$