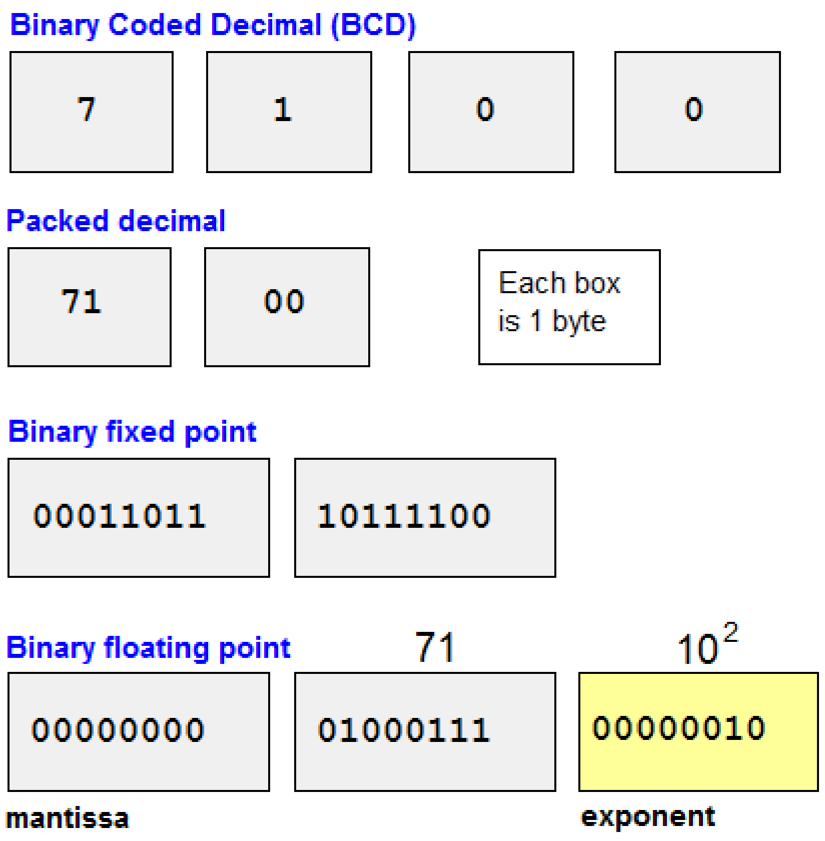
Floating point numbers and operations

There are posts on representation of floating point format. The objective of this article is to provide a brief introduction to floating point format.

The following description explains terminology and primary details of IEEE 754 binary floating-point representation. The discussion confines to single and double precision formats.



Usually, a real number in binary will be represented in the following format,

$$I_m I_{m-1} \ldots I_2 I_1 I_0 . F_1 F_2 \ldots F_n F_{n-1}$$

Where I_m and F_n will be either 0 or 1 of integer and fraction parts respectively.

A finite number can also represented by four integers components, a sign (s), a base (b), a significant (m), and an exponent (e). Then the numerical value of the number is evaluated as

$(-1)^s \times m \times b^e$ ——— Where $m < |b|$

Depending on base and the number of bits used to encode various components, the IEEE 754 standard defines five basic formats. Among the five formats, the binary32 and the binary64 formats are single precision and double precision formats respectively in which the base is 2.

Table – 1 Precision Representation

Precision	**Base**	**Sign**	**Exponent**	**Significant**
Single precision	2	1	8	23+1
Double precision	2	1	11	52+1

Single Precision Format:

As mentioned in Table 1 the single precision format has 23 bits for significant (1 represents implied bit, details below), 8 bits for exponent and 1 bit for sign.

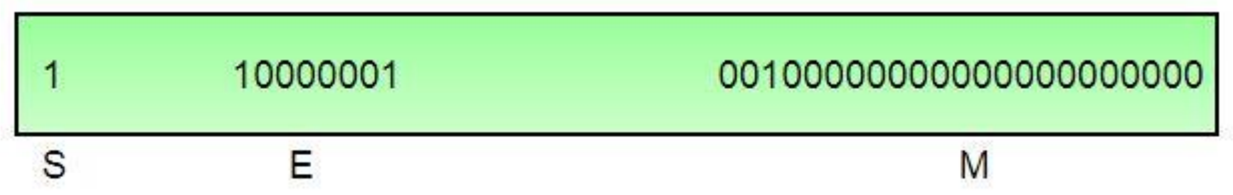
For example, the rational number 9÷2 can be converted to single precision float format as following,

$$9_{(10)} \div 2_{(10)} = 4.5_{(10)} = 100.1_{(2)}$$

The result said to be ***normalized***, if it is represented with leading 1 bit, i.e. $1.001_{(2)} \times 2^2$. (Similarly when the number $0.000000001101_{(2)} \times 2^3$ is normalized, it appears as $1.101_{(2)} \times 2^{-6}$). Omitting this implied 1 on left extreme gives us the ***mantissa*** of float number. A normalized number provides more accuracy than corresponding ***de-normalized*** number. The implied most significant bit can be used to represent even more accurate significant (23 + 1 = 24 bits) which is called ***subnormal*** representation. *The floating point numbers are to be represented in normalized form.*

The subnormal numbers fall into the category of de-normalized numbers. The subnormal representation slightly reduces the exponent range and can't be normalized since that would result in an exponent which doesn't fit in the field. Subnormal numbers are less accurate, i.e. they have less room for nonzero bits in the fraction field, than normalized numbers. Indeed, the accuracy drops as the size of the subnormal number decreases. However, the subnormal representation is useful in filing gaps of floating point scale near zero.

In other words, the above result can be written as $(-1)^0 \times 1.001_{(2)} \times 2^2$ which yields the integer components as s = 0, b = 2, significant (m) = 1.001, mantissa = 001 and e = 2. The corresponding single precision floating number can be represented in binary as shown below,



Where the exponent field is supposed to be 2, yet encoded as 129 (127+2) called ***biased exponent***. The exponent field is in plain binary format which also represents negative exponents with an encoding (like sign magnitude, 1’s complement, 2’s complement, etc.). The biased exponent is used for the representation of negative exponents. The biased exponent has advantages over other negative representations in performing bitwise comparing of two floating point numbers for equality.

A ***bias*** of (2^{n-1} – 1), where **n** is # of bits used in exponent, is added to the exponent (e) to get biased exponent (*E*). So, the biased exponent (*E*) of *single precision* number can be obtained as

$$E = e + 127$$