# Floating Point Representation

20 January 2018

# BACKGROUND

The usual representation of a fractional number as

$$b_{n-1} b_{n-2} \ldots b_1 b_0 . b_{-1} b_{-2} \ldots b_m$$

may not always serve our purpose.

This form

- is not suitable to represent a very big or a very small number.
  - Consider the number $5 \times 2^{40}$ and it would require the string 101 followed by 40 zeros. Clearly, a 32 bit representation would fail.
- it has the accuracy issues as well.

# IEEE 754

Though the typical notation of representing a big or a small number was known to us for long (the floating point notation; e.g., Avagadro's Number $= 6.022 \times 10^{23}$ or the charge of an electron $= -1.602 \times 10^{-19}$ C) the same was not formally used in computers until the widespread acceptance of the IEEE 754 floating point standard.

# Problem: No common standard

Prior to 1980 there was no common standard followed by the manufacturer (h/w and s/w) and emphasis was more on the ease of representation rather than accuracy; this leads to

- major portability and
- accuracy issues

for scientific packages involving floating point calculations.
Fortunately, IEEE 754 is now followed universally and the issues have been solved across all computing platforms.

# IEEE 754

INTEL in the 1980s appointed Prof. Kahan of University of California at Berkeley to propose a standard to be used for their 8087 floating point coprocessor; a supporting hardware for 8086/88 CPU to improve floating point performance in the PCs'. The proposal submitted by Prof. Kahan was ultimately used by the IEEE appointed floating point standard committee with minor modification.

# IEEE 754 notation

Typical floating point representations uses 32 bits (Single precision) and 64 bits (Double precision).

| | Sign bit (s) | Exponent (e) | Fraction (f) |
|---|---|---|---|
| Single Precision | 1-bit ($b_{31}$) | 8-bits ($b_{30-23}$) | 23-bits ($b_{22-0}$) |
| Double Precision | 1-bit ($b_{63}$) | 11-bits ($b_{62-52}$) | 52-bits ($b_{51-0}$) |

The numeric value of the floating point number is computed as
$V = (-1)^s M \times 2^E$ where

- s is the sign bit
- E is the biased exponent
- M is the significand

# IEEE 754 ... continued

- A bias (b) $2^{(no.\ of\ bits\ in\ e-1)} - 1$ (127 for single precision) is subtracted from the e-bits to form E (i.e., E = e - b).
  [Note : if e = 0 then E = 1 - b ]
- The fractional part $(f)$ is stored as a 23-bit string from the implied binary point to the left of the leftmost bit (MSbit) of $f$. For processing a Significand $M$ is computed as $M = 1.0 + f$ to form a normalised significand such that $1 \leq M < 2$. This virtual extra bit is used to expand the Significand by 1 bit; free of cost.
  [Note: $M$ is taken as $0.0 + f$ when e = 0]

# Normalised, De-normalised & NaN

Depending on the bit string in the exponent part (e) the real numbers stored using IEEE 754 are interpreted as i) Normalised numbers; the common case., ii) Denormalised number and iii) Sepcial values; like a) Infinity b) Not a Number (NaN).

| | | | |
|---|---|---|---|
| Normalised | s | $e = 1 \ldots 254$ | f |
| De-normalised | s | e = 0000 0000 | f |
| Infinity | s | e = 1111 1111 | f = 000 0000 0000 0000 0000 0000 |
| NaN | s | e = 1111 1111 | $f \neq 0$ |

- **Normalised numbers**
  - $e$ can be any 8-bit value except 0 and 255.
  - Consequently for a bias $b$ of 127 $E = e - b$ will have the range of $-126$ to 127.
- $f$ represents any 23 bit string and the significand is normalised by adding 1.0 with $f$ giving $1 \leq M < 2$ range to M.

# Normalised, De-normalised & NaN ... continued

Denormalised number is a requirement as through the normalised representation you

- Cannot represent zero; and
- small numbers close to zero may have less accuracy.

The *e* bits would be zero indicating a **De-normalised** representation for which

- $E = 1 - b$; and
- $M = 0.f$; i.e., 1.0 would not be added and M will have the range of $0 \leq M < 1$

*f* represents any 23 bit string for single precision.

# Normalised, De-normalised & NaN ... continued

**Special and NaN** is a clever way of representing non-real numbers or something which are exceptional cases.
Here, $e$ bit string consists of all 1s. And we have two different cases.

- If the $f$ part is zero then the representation indicates $+\infty$ or $-\infty$ depending on the sign bit $s$.
- If the $f$ part is non-zero then the representation indicates any NaN indicating something which is not a real number. Say, for example $\sqrt{-1}$ or a real variable not yet assigned any legitimate value or divide by zero error by using some agreed upon pattern of $f$ bit string.

# Examples of Floating Point Encoding : Normalized

As a practical example let us examine how an integer x = 12345 is encoded if we change it to FP (i.y., float y = (float) x;)
The binary encoding of $12345_{10}$ = 11 0000 0011 1001. We create a normaized representation of the above by

$$12345_{10} = 1.\mathbf{100\ 0000\ 1110\ 01} \times 2^{13}$$

[Note: The binray point is moved towards left by 13 position; so to keep the value same in normalized form we need to multiply it by $2^{13}$] So, the final form in single precision is

0 10001100 **100 0000 1110 01** 0000000000

[Note: MSB is s=0; e = 140; Therefore E = e - 127 (the bias) = 13; reflecting multiplication of the normalized M by $2^{13}$; Also see that the ten 0s' in the LSB position to make f to 23-bits. Finally, the leading 1 of M is not stored]

# Examples of Encoding : De-Normalized and NaN

| Value | 32-bit Hex (single precision) | Type |
|---|---|---|
| 0 | 00 00 00 00 | De-normalized |
| -0 | 00 00 00 00 | De-normalized |
| 0.0 | 00 00 00 00 | De-normalized |
| -0.0 | 80 00 00 00 | De-normalized |
| $\sqrt{-1}$ | FF C0 00 00 | NaN |
| $-\sqrt{-1}$ | 7F C0 00 00 | NaN |
| 1/(1.609e-39) | 7F 80 00 00 | Special ($+\infty$) |
| -1/(1.609e-39) | FF 80 00 00 | Special ($-\infty$) |

float variable x is assigned different values and the corresponding memory storage is displayed in Hex to see the encoding for comprehension.
[Ref: Computer Systems: A Programmer's Perspective by R E BRYANT and D R O'Hallaron, 3rd Edition, Pearson India]

# Examples of Floating Point Encoding: Continued

Let us see the encoding of a normalized negative real number, say x = -12345.75. The binary encoding is $-12345.75_{10}$ = -11 0000 0011 1001.11. Note, the presence of 11 after the binary point (.). The corresponding Floating point is given below:

1 10001100 **100 0000 1110 01** 11 00000000

[Note: MSB is s=1; e = 140; Therefore E = e - 127 (the bias) = 13; reflecting multiplication of the normalized M by $2^{13}$; Also see i) the presense of 11 (before the trailing eight zeroes that reflects ) $2^{-1}(= 0.5)$ + $2^{-2}(= 0.25)$ = 0.75 and; ii) the eight 0s' in the LSB position to make f to 23-bits. Finally, the leading 1 of M is not stored]

# Range

It may be noted that the difference between a 32-bit integer representation and the corresponding floating point representation is

- the larger dynamic range for the latter;
- all the integers are equally spaced in the number line;
- the real numbers are not equally spaced in the number line;
- Most of the real numbers cannot be exactly represented (we have infinite number of real numbers between any two real numbers however close); only approximated.

# Range ... contd

The following table shows some of the positive number representations in single precision.

| Number | exp | frac | Value | Decimal |
|---|---|---|---|---|
| Zero | $00\ldots00$ | $0\ldots00$ | $0$ | $0.0$ |
| Smallest (Denorm.) | $00\ldots00$ | $0\ldots01$ | $2^{-23} \times 2^{-126}$ | $1.4 \times 10^{-45}$ |
| Largest (Denorm.) | $00\ldots00$ | $1\ldots11$ | $(1-\epsilon) \times 2^{-126}$ | $1.2 \times 10^{38}$ |
| Smallest (Norm.) | $00\ldots01$ | $0\ldots00$ | $1 \times 2^{-126}$ | $1.2 \times 10^{-38}$ |
| One | $01\ldots11$ | $0\ldots00$ | $1 \times 2^{0}$ | $1.0$ |
| Largest (Norm.) | $11\ldots10$ | $1\ldots11$ | $(2-\epsilon) \times 2^{127}$ | $3.4 \times 10^{38}$ |

# Rounding

Rounding is done to reduce inaccuracy in representing a real number. Thus the goal to represent a number $x$ is to find out the closest match $x'$ in a systematic manner. Instead of a close match methods to set lower ($x^-$) and upper bounds ($x^+$) such that $x^- \leq x \leq x^+$ are also used. There are 4 rounding methods used in IEEE 754 standard. The first one is to find a close match. The next three are based on setting a lower and an upper boundary.

| | Values to be rounded | | | | |
|---|---|---|---|---|---|
| Mode | 1.40 | 1.60 | 1.50 | 2.50 | -1.50 |
| Round-to-even | 1 | 1 | 1 | 2 | -1 |
| Round-toward-zero | 1 | 2 | 2 | 2 | -2 |
| Round-down | 1 | 1 | 1 | 2 | -2 |
| Round-up | 2 | 2 | 2 | 3 | -1 |