CENG 3420
Computer Organization & Design

Lecture 08: Floating Numbers

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Textbook: Chapter 3.5)

Spring 2022

Floating Point Number

Scientific notation: 6.6254×10^{-27}

- A normalized number of certain accuracy (e.g. 6.6254 is called the mantissa)
- Scale factors to determine the position of the decimal point (e.g. 10^{-27} indicates position of decimal point and is called the exponent; the **base** is implied)
- Sign bit

Normalized Form

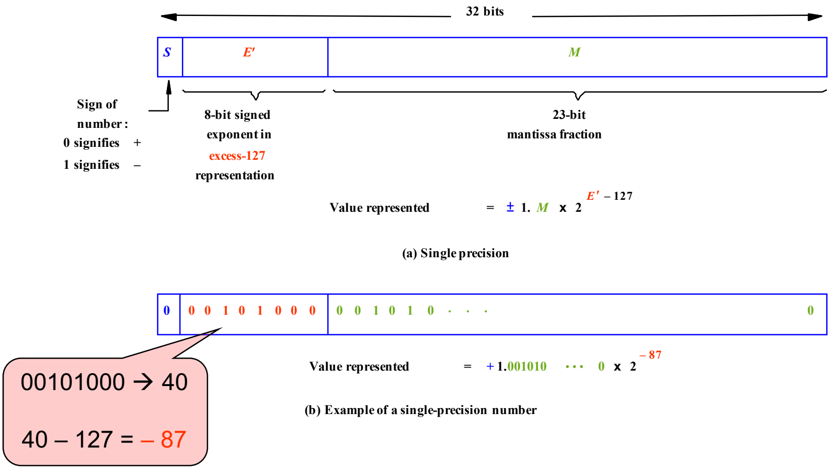
- Floating Point Numbers can have multiple forms, e.g.

$$
\begin{aligned}
0.232 \times 10^4 &= 2.32 \times 10^3 \\
&= 23.2 \times 10^2 \\
&= 2320. \times 10^0 \\
&= 232000. \times 10^{-2}
\end{aligned}
$$

- It is desirable for each number to have a unique representation => Normalized Form
- We normalize Mantissa's in the Range $[1..R)$, where R is the Base, e.g.:
  - $[1..2)$ for BINARY
  - $[1..10)$ for DECIMAL

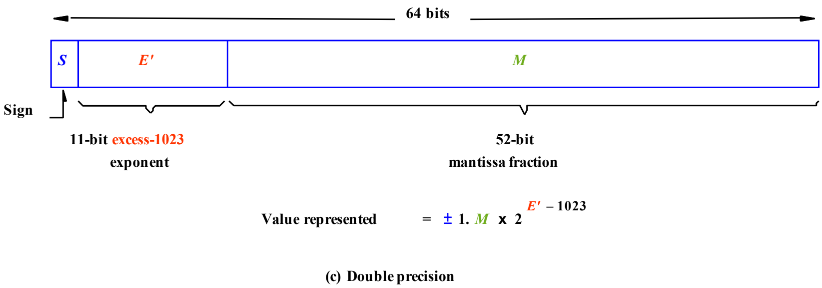
IEEE Standard 754 Single Precision

32-bit, float in C / C++ / Java



IEEE Standard 754 Double Precision

64-bit, float in C / C++ / Java



Question:

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?

Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

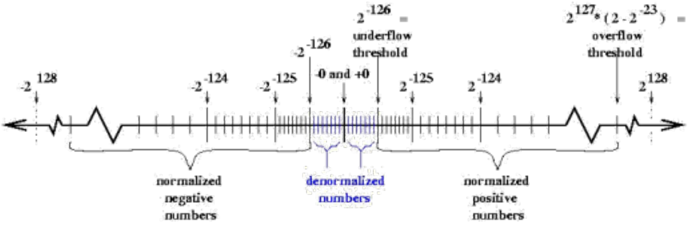
Ref: IEEE Standard 754 Numbers

- Normalized +/– 1.d…d x 2^{exp}
- ***Denormalized*** +/– 0.d…d x 2^{min_exp} → to represent *near-zero* numbers
  e.g. + 0.0000…0000001 x 2^{-126} for Single Precision

Format	# bits	# significant bits	macheps	# exponent bits	exponent range
Single	32	1+23	2^{-24} (~10^{-7})	8	2^{-126} – 2^{+127} (~$10^{\pm 38}$)
Double	64	1+52	2^{-53} (~10^{-16})	11	2^{-1022} – 2^{+1023} (~$10^{\pm 308}$)
Double Extended	>=80	>=64	<=2^{-64}(~10^{-19})	>=15	2^{-16382} – 2^{+16383} (~$10^{\pm 4932}$)

(Double Extended is *80 bits on all Intel machines*)

macheps = Machine Epsilon = ε_{mach} = $2^{-(\#\text{ significand bits})}$



Special Values

Exponents of all 0's and all 1's have special meaning

- E=0, M=0 represents 0 (sign bit still used so there is ± 0)
- E=0, $M \neq 0$ is a denormalized number $\pm 0.M \times 2^{-127}$ (smaller than the smallest normalized number)
- E=All 1's, M=0 represents $\pm$`Infinity`, depending on Sign
- E=All 1's, $M \neq 0$ represents `NaN`

More digits than in the representation: 3 extra bits of less significance

- Guard bits, Round bit, and Sticky bit (GRS)

```
mantissa format plus extra bits:

   1.XXXXXXXXXX  0  0  0

   ^    ^        ^  ^  ^
   |    |        |  |  |
   |    |        |  |  -  sticky bit (s)
   |    |        |  -  round bit (r)
   |    |        -  guard bit (g)
   |    -  10 bit mantissa from a representation
   -  hidden bit
```

(a) guard and round bits are just 2 extra bits of precision that are used in calculations.

```
mantissa from representation,  1100000100
must be shifted by 8 places (to align radix points)
                                          g r s
Before first shift:  1.1100000100 0 0 0
After 1 shift:       0.1110000010 0 0 0
After 2 shifts:      0.0111000001 0 0 0
After 3 shifts:      0.0011100000 1 0 0
After 4 shifts:      0.0001110000 0 1 0
After 5 shifts:      0.0000111000 0 0 1
After 6 shifts:      0.0000011100 0 0 1
After 7 shifts:      0.0000001110 0 0 1
After 8 shifts:      0.0000000111 0 0 1
```

(b) The sticky bit is an indication of what is in lesser significant bits, starts with 0, remains 1 if ever shifted into.

Rounding

Rounding to nearest even

- Round to the nearest representable number
  - 1x...x = More than half way (round up)
  - 0x...x = Less than half way (round down)

1.001100110×2^4

Additional bits: 110

↓

Round up (fraction + 1)

0	10011	001101

1.111111101×2^4

Additional bits: 101

↓

Round up (fraction + 1)

$$\begin{array}{r} 1.111111 \times 2^4 \\ + \; 0.000001 \times 2^4 \\ \hline 10.000000 \times 2^4 \\ 1.000000 \times 2^5 \end{array}$$

Requires renormalization

0	10100	000000

1.001101001×2^4

Additional bits: 001

↓

Leave fraction

0	10011	001101

Rounding to nearest even

- If exactly halfway between (100....), round to representable value with 0 in last significant bit

Example 1

1.001100100×2^4

Additional bits: 100

⇓

Rounding options are: 1.001100 or 1.001101

In this case, round down

0	10011	001100

Example 2

1.111111100×2^4

Additional bits: 100

⇓

Rounding options are: 1.111111 or 10.000000

In this case, round up

$$
\begin{array}{rl}
 & 1.111111 \times 2^4 \\
+ & 0.000001 \times 2^4 \\
\hline
 & 10.000000 \times 2^4 \\
 & 1.000000 \times 2^5
\end{array}
$$

Requires renormalization

0	10100	000000

Example 3

1.001101100×2^4

Additional bits: 100

⇓

Rounding options are: 1.001101 or 1.001110

In this case, round up

0	10011	001110

Other Features

+, -, x, /, sqrt, remainder, as well as conversion to and from integer are correctly rounded

- As if computed with infinite precision and then rounded
- Transcendental functions (that cannot be computed in a finite number of steps e.g., sine, cosine, logarithmic, , e, etc.) may not be correctly rounded

Exceptions and Status Flags

- Invalid Operation, Overflow, Division by zero, Underflow, Inexact

Floating point numbers can be treated as "integer bit-patterns" for comparisons

- If Exponent is all zeroes, it represents a denormalized, very small and near (or equal to) zero number
- If Exponent is all ones, it represents a very large number and is considered infinity (see next slide.)

Dual Zeroes: +0 (0x00000000) and -0 (0x80000000): they are treated as the same

Other Features

Infinity is like the mathematical one

- Finite / Infinity $\rightarrow$ 0
- Infinity $\times$ Infinity $\rightarrow$ Infinity
- Non-zero / 0 $\rightarrow$ Infinity
- Infinity$^{\{\text{Finite or Infinity}\}}$ $\rightarrow$ Infinity

NaN (Not-a-Number) is produced whenever a limiting value cannot be determined:

- Infinity - Infinity $\rightarrow$ NaN
- Infinity / Infinity $\rightarrow$ NaN
- 0 / 0 $\rightarrow$ NaN
- Infinity $\times$ 0 $\rightarrow$ NaN
- If x is a NaN, $x \neq x$
- Many systems just store the result quietly as a NaN (all 1's in exponent), some systems will signal or raise an exception

Inaccurate Floating Point Operations

- E.g. Find 1st root of a quadratic equation
  - r = (–b + sqrt(b*b – 4*a*c)) / (2*a)

Sparc processor, Solaris, gcc 3.3 (ANSI C),

```
Expected Answer   0.00023025562642476431
double            0.00023025562638524986
float             0.00024670246057212353
```

- Problem is that if c is near zero,

$$\text{sqrt}(b*b - 4*a*c) \approx b$$

- Rule of thumb: use the highest precision which does not give up too much speed

Catastrophic Cancellation

- (a – b) is inaccurate when a ≈ b
- Decimal Examples
  - Using 2 significant digits to compute mean of 5.1 and 5.2 using the formula (a+b) / 2:

    a + b = 10 (with 2 sig. digits, 10.3 can only be stored as 10)
    10 / 2 = 5.0 (the computed mean is less than both numbers!!!)
  - Using 8 significant digits to compute sum of three numbers:

    $(11111113 + (-11111111)) + 7.5111111 = 9.5111111$

    $11111113 + ((-11111111) + 7.5111111) = 10.000000$
- Catastrophic cancellation occurs when

$$\left| \frac{[round(x)"\bullet"round(y)] - round(x \bullet y)}{round(x \bullet y)} \right| >> \varepsilon_{mach}$$