# Chapter 5 Floating-Point Arithmetic.

(Read §§5.1, 5.2)

## §5.3 Floating-Point Representation.

Most computers store numbers in binary (base 2) format. Due to space limit, not all numbers are represented exactly, they are rounded to fit the word size. Thus, arithmetic operations are not performed exactly.

Small errors in data $\xrightarrow[\text{algorithm}]{\text{poor}}$ Big errors in solutions

## Binary or Base 2 System (briefly, §5.2)

$\{0,1\}$

A number in the system: ... ... $\underset{4s}{\uparrow}$ $\underset{2s}{\uparrow}$ $\underset{1s}{\uparrow}$

$(2^2 s)$ $(2^1 s)$ $(2^0 s)$

Ex: $1010_2 = 10$

8 4 2 1 — in base 10

0 1's
1 $2^1 = 2$
0 4's
1 $2^3 = 8$

(Compare to base 10:

... 100s 10s 1s

$(10^2 s)$ $(10^1 s)$ $(10^0 s)$)

$11101_2 = 29$ in base 10

$$\begin{matrix} 16 & 8 & 4 & 2 & 1 = 2^0 \\ \shortparallel & \shortparallel & \shortparallel & \shortparallel & \\ 2^4 & 2^3 & 2^2 & 2^1 & \end{matrix}$$

Binary arithmetic:

$$\begin{array}{r} {\scriptstyle 11} \\ 1010 \\ +\ 11011 \\ \hline 100101 \end{array} \quad \begin{array}{l} \to 10 \\ \to 27\ (?) \\ (=37) \end{array}$$

Decimals:

base 10: $1/10 = 0.1$

$1010 \overline{)1.0}$ base 2: $0.0\overline{0011}00_2$ (long division)

$\pi, e, \sqrt{2}$ can only be approximated by decimals.

$\boxed{Q:}$ How do we represent numbers on our computers?

- Bit: 1 or 0
- Word: a certain number of bits

Early computers used fixed-point representation:

one bit for a sign and a certain number of bits to store parts of a binary number to the left of the binary point, and the remaining bits — to store the part to the right of the point.

The system is limited: for example,

16 bits point 15 bits

can't store numbers $\geq 2^{16}$ | fixed | can't store positive numbers $< 2^{-15}$

More flexible: floating-point representation.

Adopted standard is the IEEE standard. (80's)

Institute of Electrical & Electronics Engineers

- Consistent representation of floating-point numbers across machines.
- Correctly rounded arithmetic.
- Consistent treatment of exceptional situation, e.g., division by 0.

There are three IEEE precisions:

- single-precision word has 32 bits
- double precision word has 64 bits
- extended-precision: 80 bits.

So, a binary number is written in the form $\pm m \times 2^{e}$, $1 \leq m < 2$

sign mantissa or significand exponent

- Single-precision word: 32 bits

| 1 | 8 | 23 |
|---|---|---|
| sign | exponent $e$ | mantissa $m$ |

$0 = $ "+"

$1 = $ "−"

Ex: $10 = 1010_2 = 1.010_2 \times 2^3 \longrightarrow$

| 1 | 8 | 23 |
|---|---|---|
| 0 | $e=3$ | 1.0100 .... 0 |

If a number can be stored exactly using this setting $\Rightarrow$ we call it a floating-point number. Otherwise, a number is rounded to a floating-point number.

e.g., $1/10 = 1.1\overline{0011}00_2 \times 2^{-4}$ must be rounded.

Improvement: $\pm m \times 2^e \, , \quad 1 \leq m < 2$

$m = \underbrace{b_0}_{\text{always } 1} . b_1 b_2 \ldots b_n \Rightarrow$ do not need to store!

We only store $b_1 \ldots b_n$ knowing that $b_0 = 1$

E.g., $10 = 1010_2 = 1.010_2 \times 2^3 \longrightarrow$ | 0 | e=3 | 0100 .... 0 |

"hidden-bit representation"

There are special numbers, like 0, that need a special way to be represented:

$0 \neq 1. b_1 b_2 \ldots \times 2^e$

- The gap between 1 & the next larger floating-point number is called the machine precision and is denoted by $\varepsilon$ (MATLAB: "eps").

Single precision: after 1, comes $1+2^{-23}$ :

$$1 = 2^0$$

$$\boxed{\overbrace{0}^{1}\;\Big|\;\overbrace{e=0}^{8}\;\Big|\;\overbrace{00\,\ldots\ldots\,001}^{23}}$$

So, $\varepsilon = 2^{-23} \approx 1.2 \times 10^{-7}$.

(Note: $1_{10} = (1.0)_2 \times 2^0 \longrightarrow \boxed{0 \;\big|\; e=0 \;\big|\; 00\,\ldots\,00}$ )

MATLAB: the default precision is double precision, i.e. 64 bit per word:

| 1 | 11 | 52 |
| --- | --- | --- |
| | | |
| sign | exponent | mantissa |

$1+2^{-52}$ is the closest to 1 larger number.

$\varepsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ (type "eps" in MATLAB).

<u>Note</u>: gap between 0 & the smallest positive number can be filled using subnormal numbers.

# §§ 5.4, 5.5. IEEE Floating-Point Arithmetic & Rounding.

Recall: - consistency across machines
- correct rounding
- consistent treatment of exceptional situations (NaN: 0/0)

IEEE standard.

Consider IEEE double precision:

| If: e field (11 bits) is: | then Number is: | Type |
|---|---|---|
| 00000000000 | $\pm(0.b_1b_2\dots b_{52})_2 \times 2^{-1022}$ | → 0 or subnormal |
| $(1_{10}=)$ 00 .... 01 | $\pm(1.b_1b_2\dots b_{52})_2 \times 2^{-1022}$ | normalized numbers: |
| $(2_{10}=)$ 00 ... 10 | $\pm(1.b_1b_2\dots b_{52})_2 \times 2^{-1021}$ | |
| ⋮ | ⋮ | |
| $(1023_{10}=)$ 011 ... 11 | $\pm(1.b_1b_2\dots b_{52})_2 \times 2^{0}$ | e field = actual exp. + 1023 |
| ⋮ | | |
| $(2046_{10}=)$ 11 .... 10 | $\pm(1.b_1\dots b_2)_2 \times 2^{1023}$ | |
| 11 ... 11 | $\pm\infty$ if $b_1 = \dots = b_2 = 0$; NaN otherwise | → Exception |

Smallest number ($>0$): $(1.0)_2 \times 2^{-1022} \approx 2.2 \times 10^{-308}$

Largest — " — ($>0$): $(1.1\dots1)_2 \times 2^{1023} \approx 1.8 \times 10^{308}$

Exponent field = actual e + 1023, in

Single precision: actual e + 127.

So, why is that? E.g., 8 bits in single precision for $e$ can only give values from 0 to 255. To cover $e<0$, the exponent is 127 greater than the real one, e.g. for $(1.01011101)_2 \times 2^5$, the eight-bit exp. field is $5+127=132=(10000100)_2$.

Special exponent fields: all 0's or 1's.
Number $(1.1)_2 \times 2^{-1024} = (0.011)_2 \times 2^{-1022}$:

$$\underbrace{\boxed{0}}_{1}\underbrace{\boxed{00\ldots0}}_{11}\underbrace{\boxed{0110\ldots0}}_{52} \quad \text{(Subnormal)}$$

less precision than normalized numbers.
0: all 0's in exp. field & mantissa.
All 1's: exceptions.

MATLAB: $0/0 \rightarrow NaN$
$a/0 \ (a \neq 0) \rightarrow Inf \ (\pm\infty)$

Previously: $1/0 - 2/0 = 0$
Now: $1/0 - 2/0 = NaN$
Also: $\infty + a = \infty$ $\qquad 2/Inf = 0$
$a - \infty = -\infty$ $\qquad Inf/2 = Inf$
$\infty \times 0 = NaN$, etc. $\qquad 2/0 = Inf$

Rounding: 4 modes

- Round down: $round(x) \leq x$
- Round up: $round(x) \geq x$

- Round towards 0 : round(x) is either round-down(x) or round-up(x), whichever lies between 0 & x. If $x>0 \Rightarrow$ round(x) = round-down(x), if $x<0 \Rightarrow$ round(x) = round-up(x).
- Round to the nearest : either round-down or round-up, whichever is closer. → Default.

Ex: is $1/10 = 1.100\overline{1100}_2 \times 2^{-4}$ in double-precision

| 0 | 01111111011 | $\overbrace{1001100 1100 \ldots 11001101 0}^{52}$ |
|---|---|---|

$+1023-4 = 1019_{10}$

repeats

11 0011 00

round-up (nearest)

- Absolute rounding error is

$$|\text{round}(x) - x|$$

In double precision, $|\text{round}(x) - x| < \underbrace{2^{-52}}_{\varepsilon} \times 2^e$

- Relative rounding error is

$$\frac{|\text{round}(x) - x|}{|x|} < \frac{\varepsilon \times 2^e}{m \times 2^e} < \varepsilon$$

We can write: $\text{round}(x) = x(1+\delta)$ w/ $|\delta| < \varepsilon$

IEEE standard: if $a$ & $b$ are fl.-point numbers then

$$a \oplus b = \text{round}(a+b) = (a+b)(1+\delta_1)$$
$$a \ominus b = \text{round}(a-b) = (a-b)(1+\delta_2)$$
$$a \otimes b = \text{round}(ab) = (ab)(1+\delta_3)$$
$$a \oslash b = \text{round}(a/b) = (a/b)(1+\delta_4)$$

w/ $|\delta_i| < \varepsilon$, $i = 1,2,3,4$.

→ Read §5.6: examples on rounding.

§5.7: Exceptions.

$\pm\infty$, NaN

Other:

1) overflow: true result is greater than the largest fl.-pt. number
$\left((1.1\ldots1)_2 \times 2^{1023} \approx 1.8 \times 10^{308}\right.$ for double precis.$\left.\right)$

Can be set to $\infty$ or the largest number.

2) Underflow: true result is less than the smallest fl.-pt. number.
Stored as a subnormal number or set to 0.