Computer Arithmetic

Ned Nedialkov

McMaster University

19 January 2023

Outline

Floating-point number system

A floating-point (FP) system is characterized by four integers (β, t, L, U), where

- β base of the system or radix
- t number of digits or precision
- $[L, U]$ exponent range

A number x in the system is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{t-1}}{\beta^{t-1}} \right) \times \beta^e$$

where

- $0 \le d_i \le \beta - 1$, $i = 0, \ldots, t-1$
- $e \in [L, U]$

Floating-point number system cont.

- The string of base β digits $d_0 d_1 \cdots d_{t-1}$ is called mantissa or significand
- $d_1 d_2 \cdots d_{t-1}$ is called fraction
- A common way of expressing x is

$$\pm\, d_0.d_1 \cdots d_{t-1} \times \beta^e$$

- A FP number is normalized if d_0 is nonzero
  denormalized otherwise

Floating-point number system cont.

Example 1. Consider the FP $(10, 3, -2, 2)$.

- Numbers are of the form

$$d_0.d_1d_2 \times 10^e, \quad d_0 \neq 0,\ e \in [-2, 2]$$

- largest positive number 9.99×10^2
- smallest positive normalized number 1.00×10^{-2}
- smallest positive denormalized number 0.01×10^{-2}
- denormalized numbers are e.g. 0.23×10^{-2}, 0.11×10^{-2}
- 0 is represented as 0.00×10^0

Rounding

How to store a real number

$$x = \pm\, d_0.d_1 \cdots d_{t-1} d_t d_{t+1} \cdots \times \beta^e$$

in t digits?

Denote by $\mathsf{fl}(x)$ the FP representation of x

- Rounding by chopping (also called rounding towards zero)
- Rounding to nearest. $\mathsf{fl}(x)$ is the nearest FP to x
  If a tie, round to the even FP
- Rounding towards $+\infty$. $\mathsf{fl}(x)$ is the smallest FP $\geq x$
- Rounding towards $-\infty$. $\mathsf{fl}(x)$ is the largest FP $\leq x$

Rounding cont.

Example 2. Consider the FP $(10, 3, -2, 2)$.

Let $x = 1.2789 \times 10^1$

- chopping: $\mathrm{fl}(x) = 1.27 \times 10^1$
- nearest: $\mathrm{fl}(x) = 1.28 \times 10^1$
- $+\infty$: $\mathrm{fl}(x) = 1.28 \times 10^1$
- $-\infty$: $\mathrm{fl}(x) = 1.27 \times 10^1$

Let $x = 1.275000 \times 10^1$. It is in the middle between 1.27 and 1.28.
When a tie, typically round to the even, the number with even last digit

- nearest: $\mathrm{fl}(x) = 1.28 \times 10^1$

Machine epsilon

- Machine epsilon: the distance from 1 to the next larger FP number
  E.g. in $t = 5$ decimal digits, $\epsilon_{\text{mach}} = 1.0001 - 1.0000 = 10^{-4}$
  Note: 1.00005 is not representable in this FP system, just denotes the middle
  Another definition: smallest $\epsilon > 0$ such that $\text{fl}(1 + \epsilon) > 1$
  These two definitions are not equivalent. We shall use the first one.
  Unit roundoff: $u = \epsilon_{\text{mach}}/2$

Machine epsilon cont.

When rounding to the nearest

$$\mathsf{fl}(x) = x(1+\epsilon), \quad \text{where } |\epsilon| \le u$$

i.e.

$$\frac{\mathsf{fl}(x) - x}{x} = \epsilon$$
$$\left|\frac{\mathsf{fl}(x) - x}{x}\right| = |\epsilon| \le u$$

IEEE 754

- IEEE 754 standard for FP arithmetic (1985)
- IEEE 754-2008, IEEE 754-2019
- Most common (binary) single and double precision since 2008 half precision

	bits	t	L	U	ϵ_{mach}
single	32	24	-126	127	$\approx 1.2 \times 10^{-7}$
double	64	53	-1022	1023	$\approx 2.2 \times 10^{-16}$

	range	smallest normalized	smallest denormalized
single	$\pm 3.4 \times 10^{38}$	$\pm 1.2 \times 10^{-38}$	$\pm 1.4 \times 10^{-45}$
double	$\pm 1.8 \times 10^{308}$	$\pm 2.2 \times 10^{-308}$	$\pm 4.9 \times 10^{-324}$

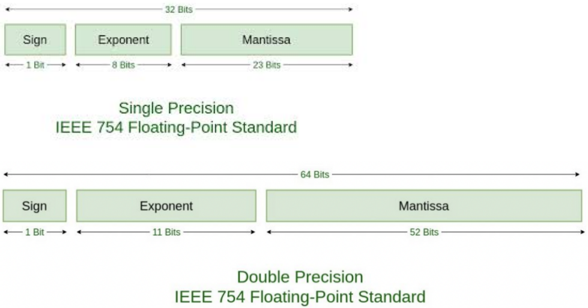
(These are $\approx$ values)

IEEE 754 cont.

Exceptional values

- `Inf`, `-Inf` when the result overflows, e.g. `1/0.0`
- `NaN` "Not a Number" results from undefined operations e.g. `0/0`, `0*Inf`, `Inf/Inf`
  `NaN`s propagate through computations

IEEE 754 cont.



(From
`https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/`)

- sign 0 positive, 1 negative
- exponent is biased
- first bit of mantissa is not stored, sticky bit, assumed 1

IEEE 754 cont.

Single precision

- FP numbers
  - biased exponent: from 1 to 254, bias: 127
  - actual exponent: $1-127=-126$ to $254-127=127$
- `Inf`
  - sign: 0 for `+Inf`, 1 for `-Inf`
  - biased exponent: all 1's, 255
  - fraction: all 0's
- `NaN`
  - sign: 0 or 1
  - biased exponent: all 1's, 255
  - fraction: at least one 1
- `0`
  - sign: 0 for $+0$, 1 for -0
  - biased exponent: all 0's
  - mantissa: all 0's

IEEE 754 cont.

Double precision

- bias 1023
- biased exponent: from 1 to 2046
- actual exponent: from -1022 to 1023
- rest similar to single

IEEE 754 cont.

Why biased exponent?

- Consider an IEEE FP number as a signed integer, that is, the sequence of bits as a signed integer
  Easy to compare FP numbers by comparing them as integers
- What if the exponent is stored as a signed number in 2's complement representation?
- Consider single precision, and assume the exponent is stored as a signed integer. Assume we have two positive numbers $x > y$ with exponents 5 and -5

IEEE 754 cont.

- 5 in 8 bits is 00000101
- -5 in 2's complement is 11111011
- Then x and y are of the form

$$x = \underbrace{0}_{+}\underbrace{00000101}_{5}\underbrace{\cdots}_{\text{23 bits}}$$
$$y = \underbrace{0}_{+}\underbrace{11111011}_{-5}\underbrace{\cdots}_{\text{23 bits}}$$

If we compare them bit by bit, $x < y$, which is not the case

IEEE 754 cont.

FP arithmetic

For a real x and rounding to nearest

$$\mathsf{fl}(x) = x(1+\epsilon), \quad |\epsilon| \le u$$

u is the unit roundoff of the precision

The arithmetic operations are correctly rounded, i.e. for x and y IEEE numbers and rounding to the nearest

$$\mathsf{fl}(x \circ y) = (x \circ y)(1+\epsilon), \quad \circ \in \{+,-,*,/\}, \quad |\epsilon| \le u$$

Also correctly rounded are

- conversions between formats and to and from strings
- square root
- fused multiply and add, FMA, computes $a * x + b$ with single rounding

Precisions

	number of bits			
precision	significand	exponent	range	unit roundoff
bfloat16	8	8	$10^{\pm 38}$	4×10^{-3}
half	11	5	$10^{\pm 5}$	5×10^{-4}
single	24	8	$10^{\pm 38}$	6×10^{-8}
double	53	11	$10^{\pm 308}$	1×10^{-16}
quad	113	15	$10^{\pm 4932}$	1×10^{-34}

- `half`: NVIDIA, AMD GPUs, Fujitsu A64FX ARMs
- `bfloat16`: Google TPUs, nvidia GPUs, ARM, and Intel CPUs
- quad typically in software; hardware implementations IBM System/390 G5, IBM Power9 CPU

- Modern GPUs
  - `half`, `bfloat16` much faster than `single`
  - `single` 2x faster than `double`
  - e.g. nvidia a100 half:single 4x, single:double 2x
- Modern CPUs
  - `single` can be 2x faster than `double`
- `quadruple` in software
  a good implementation ≈ 10x slower than `double`
- Exploiting lower precision(s):
  - faster floating-point computations
  - less: storage, data movement, communications, energy consumption

Useful links

- ▸ IEEE 754 double precision visualization
- ▸ C. Moler. Floating Point Numbers
- ▸ IEEE 754
- ▸ N. Higham. Half Precision Arithmetic: fp16 Versus bfloat16
- ▸ GNU Multiple Precision Arithmetic Library
- ▸ Quadruple-precision floating-point format