Aritmética Computacional

UFOP - ICEA - DECSI - CSI 203 - Prof. Dr. Eduardo Ribeiro
OAC I - Computer Organization and Architecture I

Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$

normalized

not normalized

- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types float and double in C

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

IEEE Floating-Point Format

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

IEEE Floating-Point Format

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit
  - 0 ⇒ non-negative; 1 ⇒ negative
- Normalize significand:
  - 1.0 ≤ |significand| < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation:
  - actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1203

Single-Precision Range

- **Exponents `00000000` and `11111111` reserved**
- **Smallest value**
  - **Exponent: `00000001`**
    ⇒ actual exponent = `1 – 127 = –126`
  - **Fraction: `000…00` ⇒ significand = `1.0`**
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- **Largest value**
  - **exponent: `11111110`**
    ⇒ actual exponent = `254 – 127 = +127`
  - **Fraction: `111…11` ⇒ significand ≈ `2.0`**
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

- **Exponents `0000…00` and `1111…11` reserved**
- **Smallest value**
  - **Exponent: `00000000001`**
    ⇒ actual exponent = `1 – 1023 = –1022`
  - **Fraction: `000…00` ⇒ significand = `1.0`**
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- **Largest value**
  - **Exponent: `11111111110`**
    ⇒ actual exponent = `2046 – 1023 = +1023`
  - **Fraction: `111…11` ⇒ significand ≈ `2.0`**
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Precision

- Relative precision
  - all fraction bits are significant
  - Single: approx 2^{-23}
    - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 6$ decimal digits of precision
- Examples of basis conversions
  - $2^{11} = 10$^(11 * 0.30103); $\log_{10}2 = 0.30103$
  - $10^{3} = 2$^(3 * 3.322); $\log_{2}10 = 3.322$

Floating-Point Precision

- Relative precision
  - Double: approx 2^{-52}
    - Equivalent to $52 \times \log_{10} 2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Floating-Point Example

- Represent -0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - $s = 1$
  - Fraction = $1000\ldots00_2$
  - Exponent = $-1 + \text{Bias}$
    - Single: $-1 + 127 = 126 = 01111110_2$
    - Double: $-1 + 1023 = 1022 = 01111111110_2$
- Single: 101111110 1000…00
- Double: 101111111110 1000…00

Floating-Point Example

- What number is represented by the single-precision float

  11000000101000...00

  - S = 1
  - Fraction = $01000...00_2$
  - Fxponent = 10000001_2 = 129

- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

Denormal Numbers

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-\text{Bias}}$$

- Smaller than normal numbers
  - allow for gradual underflow, with diminishing precision
- Denormal with fraction = 000...0
- Exponent = 000...0 ⇒ hidden bit is 0

$$x = (-1)^S \times (0 + 0) \times 2^{-\text{Bias}} = \pm 0.0$$

Two representations of 0.0!

Infinities and NaNs

- Exponent = `111...1`, Fraction = `000...0`
  - `±Infinity`
  - Can be used in subsequent calculations, avoiding need for overflow check
- Exponent = `111...1`, Fraction ≠ `000...0`
  - Not-a-Number (`NaN`)
  - Indicates illegal or undefined result
    - e.g., `0.0 / 0.0`
  - Can be used in subsequent calculations