Floating Point Numbers Representation, Operations, and Accuracy

Floating Point

- Representation for non-integer numbers
 - Including very small and very large numbers
- Like scientific notation

$$-2.34 \times 10^{56}$$

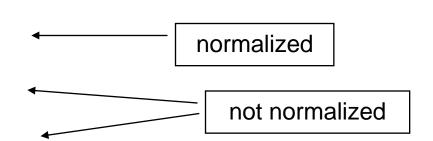
$$-+0.002 \times 10^{-4}$$

$$_{\rm 0}$$
 +987.02 × 10⁹

In binary

$$\bullet$$
 ±1. $xxxxxxxx_2 \times 2^{yyyy}$

□ 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 Precision)

single: 8 bits single: 23 bits double: 11 bits double: 52 bits

S Exponent Fraction

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

- \square S: sign bit (0 \Rightarrow non-negative, 1 \Rightarrow negative)
- □ Normalize significand: 1.0 ≤ |significand| < 2.0</p>
 - 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 = 1023

Single-Precision Range

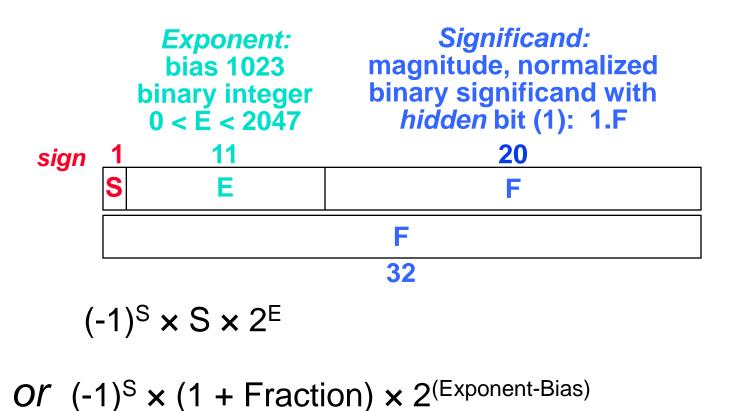
- Exponents 00000000 and 11111111 reserved
- Smallest value
 - Exponent: 00000001 \Rightarrow actual exponent = 1 - 127 = -126
 - Fraction: $000...00 \Rightarrow significand = 1.0$
 - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$

Largest value

- exponent: 111111110 \Rightarrow actual exponent = 254 - 127 = +127
- Fraction: 111...11 ⇒ significand ≈ 2.0
- $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

IEEE 754 Double Precision

Double precision number represented in 64 bits



Double-Precision Range

- Exponents 0000...00 and 1111...11 reserved
- Smallest value
 - Exponent: 0000000001 \Rightarrow actual exponent = 1 - 1023 = -1022
 - Fraction: $000...00 \Rightarrow significand = 1.0$
 - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$

Largest value

- Fraction: 111...11 ⇒ significand ≈ 2.0
- $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

IEEE 754 FP Standard Encoding

- Special encodings are used to represent unusual events
 - ± infinity for division by zero
 - NAN (not a number) for the results of invalid operations such as 0/0
 - True zero is the bit string all zero

Single Precision		Double Precision		Object
E (8)	F (23)	E (11)	F (52)	Represented
0000 0000	0	00000000	0	true zero (0)
0000 0000	nonzero	00000000	nonzero	± denormalized number
0000 0001 to 1111 1110	anything	00000001 to 11111110	anything	± floating point number
1111 1111	0	1111 1111	0	± infinity
1111 1111	nonzero	1111 1111	nonzero	not a number (NaN)

Floating-Point Precision

- Relative precision
 - all fraction bits are significant
 - Single: approx 2⁻²³
 - Equivalent to 23 × log₁₀2 ≈ 23 × 0.3 ≈ 7 decimal digits of precision
 - Double: approx 2⁻⁵²
 - Equivalent to 52 x log₁₀2 ≈ 52 x 0.3 ≈ 16 decimal digits of precision

Floating-Point Example

■ What number is represented by the single-precision float:

11000000101000...00

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

Floating-Point Addition

- Consider a 4-digit decimal example
 - $> 9.999 \times 10^{1} + 1.610 \times 10^{-1}$
- 1. Align decimal points
 - Shift number with smaller exponent
 - $> 9.999 \times 10^1 + 0.016 \times 10^1$
- 2. Add significands
 - $> 9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$
- □ 3. Normalize result & check for over/underflow
 - \rightarrow 1.0015 × 10²
- 4. Round and renormalize if necessary
 - \rightarrow 1.002 × 10²

Floating-Point Addition

- Now consider a 4-digit binary example
 - $\rightarrow 1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2} (0.5 + -0.4375)$
- 1. Align binary points
 - Compare exponents, shift number with smaller one

$$\rightarrow 1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$$

2. Add significands

$$> 1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$$

- 3. Normalize result & check for over/underflow
 - \rightarrow 1.000₂ × 2⁻⁴, with no over/underflow
- 4. Round and renormalize if necessary
 - \rightarrow 1.000₂ × 2⁻⁴ (no change) = 0.0625

Floating-Point Multiplication

- Consider a 4-digit decimal example
 - \rightarrow 1.110 × 10¹⁰ × 9.200 × 10⁻⁵
- 1. Add exponents
 - For biased exponents, subtract bias from sum
 - New exponent = 10 + -5 = 5
- 2. Multiply significands
 - \rightarrow 1.110 × 9.200 = 10.212 \Rightarrow 10.212 × 10⁵
- 3. Normalize result & check for over/underflow
 - \rightarrow 1.0212 × 10⁶
- 4. Round and renormalize if necessary
 - > 1.021 × 10⁶
- 5. Determine sign of result from signs of operands
 - \rightarrow +1.021 × 10⁶

Floating-Point Multiplication

- Now consider a 4-digit binary example
 - $\sim 1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2} (0.5 \times -0.4375)$
- 1. Add exponents
 - \rightarrow Unbiased: -1 + -2 = -3
 - ▶ Biased: (-1 + 127) + (-2 + 127) = -3 + 254 127 = -3 + 127
- 2. Multiply significands
 - \rightarrow 1.000₂ × 1.110₂ = 1.1102 \Rightarrow 1.110₂ × 2⁻³
- 3. Normalize result & check for over/underflow
 - \rightarrow 1.110₂ × 2⁻³ (no change) with no over/underflow
- 4. Round and renormalize if necessary
 - \rightarrow 1.110₂ × 2⁻³ (no change)
- 5. Determine sign: if same, +; else, -
 - $-1.110_2 \times 2^{-3} = -0.21875$

Accurate Arithmetic

- IEEE Std 754 specifies additional rounding control
 - Extra bits of precision (Guard, Round, Sticky)
 - Choice of rounding modes
 - Allows programmer to fine-tune numerical behavior of a computation
- Not all FP units implement all options
 - Most programming languages and FP libraries just use defaults
- Trade-off between hardware complexity, performance, and market requirements

Support for Accurate Arithmetic

- IEEE 754 FP rounding modes
 - Always round up (toward +∞)
 - Always round down (toward -∞)
 - Truncate (toward 0)
 - Round to nearest even (when the Guard || Round || Sticky are 100)
 always creates a 0 in the least significant (kept) bit of F
- Rounding (except for truncation) requires the hardware to include extra F bits during calculations
 - Guard bit used to provide one F bit when shifting left to normalize a result (e.g., when normalizing F after division or subtraction) G
 - Round bit used to improve rounding accuracy R
 - Sticky bit used to support Round to nearest even; is set to a 1 whenever a 1 bit shifts (right) through it (e.g., when aligning F during addition/subtraction)

Associativity

Parallel programs may interleave operations in unexpected orders

		(x+y)+z	x+(y+z)
X	-1.50E+38		-1.50E+38
У	1.50E+38	0.00E+00	
Z	1.0	1.0	1.50E+38
		1.00E+00	0.00E+00

- Assumptions of associativity may fail, since FP operations are not associative!
- Need to validate parallel programs under varying degrees of parallelism