UNM SCHOOL *of* ENGINEERING
*Department of Computer Science*

# CS341
# CDR(5): Floating Point rounding, Operations, properties

## Lecture #6 – part 1

Prof. Soraya Abad-Mota, PhD

# Recall How to represent 0.25?

- In the tiny 8-bit IEEE (4 bits exp, 3 bits frac)

  Normalized or denormalized?

  Normalized
- E = -2 = exp - Bias
- Bias = 2^3 - 1 = 7
- Exp = E + bias = -2 + 7 = 5

  1.00 x 2^-2
- Final 8-bit IEEE representation

  0 0101 000

# Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example
- Distribution of values and properties
- Rounding, addition, multiplication
- Floating point in C
- Summary

# Learning objectives

After this portion students should be able to:

1. List and describe the different kinds of *rounding*.
2. Define the *properties of floating point numbers represented in C*.
3. *Compare* the mathematical *properties of unsigned, two's complement and floating point* numbers.

# Plan for Week 4

- Finish FP with:
  - Rounding (sec. 2.4.4)
  - Properties of FP operations in IEEE repr. (2.4.5)
  - Comparison with integer arithmetic properties
  - FP in C (2.4.6)
- Finish Chapter 2
  - 2.1.3 Byte ordering
  - 2.1.4, 2.1.5
- Begin Ch. 3

# Floating Point Operations: Basic Idea

Can only approximate real arithmetic (Sec. 2.4.4)

- $\mathbf{x} +_f \mathbf{y} = \mathrm{Round}(\mathbf{x} + \mathbf{y})$

- $\mathbf{x} \times_f \mathbf{y} = \mathrm{Round}(\mathbf{x} \times \mathbf{y})$

- Basic idea
  - First compute exact result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into **`frac`**

# Rounding

- Rounding Modes (illustrate with $ rounding): default finds closest match

| | \$1.40 | \$1.60 | \$1.50 | \$2.50 | –\$1.50 |
|---|---|---|---|---|---|
| ◦ To-Even (default) | \$1 | \$2 | \$2 | \$2 | –\$2 |
| ◦ Toward-zero | \$1 | \$1 | \$1 | \$2 | –\$1 |
| ◦ Round down ($-\infty$) | \$1 | \$1 | \$1 | \$2 | –\$2 |
| ◦ Round up ($+\infty$) | \$2 | \$2 | \$2 | \$3 | –\$1 |

- All others (outside the default) are used to compute upper and lower bounds

# Closer Look at Round-To-Even

- Default Rounding Mode (closest match)
  - Hard to get any other kind without dropping into assembly
  - All others are statistically biased
    - Sum of set of positive numbers will consistently be over- or under- estimated

- Applying to Other Decimal Places / Bit Positions
  - When exactly halfway between two possible values
    - Round so that least significant digit is even
  - E.g., round to nearest hundredth

| | | |
|---|---|---|
| 7.8949999 | 7.89 | (Less than half way) |
| 7.8950001 | 7.90 | (Greater than half way) |
| 7.8950000 | 7.90 | (Half way—round up) |
| 7.8850000 | 7.88 | (Half way—round down) |

# Rounding Binary Numbers

- Binary Fractional Numbers
  - "Even" when least significant bit is 0
  - "Half way" when bits to right of rounding position = $100\ldots_2$

- Examples
  - Round to nearest 1/4 (2 bits right of binary point)

| Value | Binary | Rounded | Action | Rounded |
|---|---|---|---|---|
| 2 3/32 | $10.00011_2$ | $10.00_2$ | (<1/2—down) | 2 |
| 2 3/16 | $10.00110_2$ | $10.01_2$ | (>1/2—up) | 2 1/4 |
| 2 7/8 | $10.11100_2$ | $11.00_2$ | ( 1/2—up) | 3 |
| 2 5/8 | $10.10100_2$ | $10.10_2$ | ( 1/2—down) | 2 1/2 |

# Practice 1 (now and on your own)

- Practice problems 2.50, 2.51, and 2.52 on pages 121–122

Show binary fractional values rounded to the nearest half, according to the round-to-even rule. Show numeric value before and after.

A. 10.010
B. 10.011
C. 10.110
D. 11.001

# Practice 2 (Problem 2.52) (home)[1]

- Format A: k=3 exponent bits, exp bias = 3. n=4 fraction bits
- Format B: k=4 exponent bits, exp bias = 7. n=3 fraction bits

| Format A | | Format B | |
|---|---|---|---|
| Bits | Value | Bits | Value |
| 011 0000 | 1 | 0111 000 | 1 |
| 101 1110 | | | |
| 010 1001 | | | |
| 110 1111 | | | |

[1]Section 002 must do this Tuesday

# Integer Arithmetic: Basic Rules

- Addition:
  - Unsigned/signed: Normal addition followed by truncate, same operation on bit level
  - Unsigned: addition mod $2^w$
    - Mathematical addition + possible subtraction of $2^w$
  - Signed: modified addition mod $2^w$ (result in proper range)
    - Mathematical addition + possible addition or subtraction of $2^w$
- Multiplication:
  - Unsigned/signed: Normal multiplication followed by truncate, same operation on bit level
  - Unsigned: multiplication mod $2^w$
  - Signed: modified multiplication mod $2^w$ (result in proper range)

# Integer Arithmetic: Basic Rules (2)

- Unsigned ints, 2's complement ints are isomorphic rings: isomorphism = casting
- Left shift
  - Unsigned/signed: multiplication by $2^k$
  - Always logical shift
- Right shift
  - Unsigned: logical shift, div (division + round to zero) by $2^k$
  - Signed: arithmetic shift
    - Positive numbers: div (division + round to zero) by $2^k$
    - Negative numbers: div (division + round away from zero) by $2^k$
      Use biasing to fix

# Properties of Unsigned Arithmetic

- Unsigned Mult. with Addition Forms Commutative Ring
  - Addition is commutative group
  - Closed under multiplication: $0 \le \text{UMult}_w(u\,,\,v) \le 2^w - 1$
  - Multiplication Commutative: $\text{UMult}_w(u\,,\,v) = \text{UMult}_w(v\,,\,u)$
  - Multiplication is Associative

    $\text{UMult}_w(t, \text{UMult}_w(u\,,\,v)) = \text{UMult}_w(\text{UMult}_w(t, u\,), v)$
  - 1 is multiplicative identity: $\text{UMult}_w(u\,,\,1) = u$
  - Multiplication distributes over addition

    $\text{UMult}_w(t, \text{UAdd}_w(u\,,\,v)) = \text{UAdd}_w(\text{UMult}_w(t, u\,), \text{UMult}_w(t, v))$

# Properties of Two's Comp. Arithmetic

- Isomorphic Algebras
  - Unsigned multiplication and addition: Truncating to *w* bits
  - Two's complement mult. and addition: Truncating to *w* bits
- Both Form Rings
  - Isomorphic to ring of integers mod $2^w$
- Comparison to (Mathematical) Integer Arithmetic
  - Both are rings
  - Integers obey ordering properties, e.g.,

    $u > 0 \quad \Rightarrow \quad u + v > v$

    $u > 0, v > 0 \quad \Rightarrow \quad u \cdot v > 0$
  - These properties are not obeyed by two's comp. arithmetic

    $TMax + 1$ == $TMin$

    15213 * 30426 == -10030 (16-bit words)

# FP Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: $s1$ ^ $s2$
  - Significand M: $M1 \times M2$
  - Exponent E: $E1 + E2$

(^ is exclusive or)

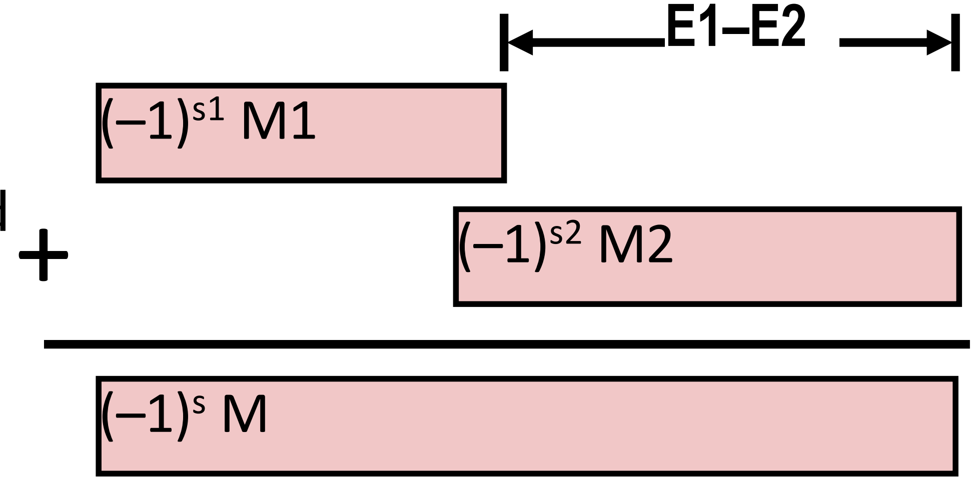
(tricks are used to avoid the exact computations)

- Fixing
  - If M $\geq$ 2, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit **`frac`** precision
- Implementation
  - Biggest chore is multiplying significands

# Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1} \ \ + \ \ (-1)^{s2}\ M2\ \ 2^{E2}$
  - Assume E1 > E2

Get binary points lined up




- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s, significand M:
    - Result of signed align & add
  - Exponent E: E1
- Fixing
  - If M ≥ 2, shift M right, increment E
  - if M < 1, shift M left k positions, decrement E by k
  - Overflow if E out of range
  - Round M to fit **`frac`** precision

# Mathematical Properties of FP Add

- Compare to those of Abelian Group
  - Closed under addition? Yes
    - But may generate infinity or NaN
  - Commutative? Yes
  - Associative? No (most important deficiency)
    - Overflow and inexactness of rounding
    - (3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14
  - 0 is additive identity? Yes
  - Every element has additive inverse? Almost
    - Yes, except for infinities & NaNs
- Monotonicity (not included in the group properties)
  - $a \geq b \Rightarrow a+c \geq b+c$? Almost
    - Except for infinities & NaNs

# Mathematical Properties of FP Mult

- Compare to Commutative Ring
  - Closed under multiplication? Yes
    - But may generate infinity or NaN
  - Multiplication Commutative? Yes
  - Multiplication is Associative? No
    - Possibility of overflow, inexactness of rounding
    - Ex: (1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20
  - 1 is multiplicative identity? Yes
  - Multiplication distributes over addition? No
    - Possibility of overflow, inexactness of rounding
    - 1e20*(1e20-1e20)= 0.0, 1e20*1e20 – 1e20*1e20 = NaN
- Monotonicity
  - $a \geq b$ & $c \geq 0 \Rightarrow a * c \geq b * c$? Almost (not in ints)
    - Except for infinities & NaNs

# Why should I care? (some conclusions)

- Rounding and overflow makes a mess of things – math doesn't work like in school OR with computer ints
- Addition and Multiplication not Associative or Distributive!
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
  - `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
  - `1e20*(1e20-1e20)= 0.0, 1e20*1e20 - 1e20*1e20 = NaN`
- Some things aren't exact (what is 0.1 in FP?):

```
for (double i = 0.0; i < 1.0; i += 0.1)
    printf("%.19f ", i);
```

0.0000000000000000000 0.1000000000000000056 0.2000000000000000111
0.3000000000000000444 0.4000000000000000222 0.5000000000000000000
0.5999999999999999778 0.6999999999999999556 0.7999999999999999334
0.8999999999999999112 0.9999999999999998890

- Every number has an additive and multiplicative inverse, unlike integer arithmetic (which has no mult. inverse)!
- So in some cases, you have to be really careful about the order in which you do things

# Stop here part 1

- Go to the next set of slides