UNM SCHOOL of ENGINEERING
Department of Computer Science

CDR(3): Floating Point representation Lecture #4 – part 2

(Section 2.4 textbook)

Prof. Soraya Abad-Mota, PhD

Learning objectives

After the Floating point sessions students should be able to:

1. Describe how are fractional binary numbers interpreted.
2. Describe the IEEE floating point standard representation to approximate real numbers.
3. Perform conversions between IEEE fp and real numbers in decimal.
4. Define and perform the operations of rounding, addition, and multiplication of floating point.
5. Define how are floating point numbers represented in C.

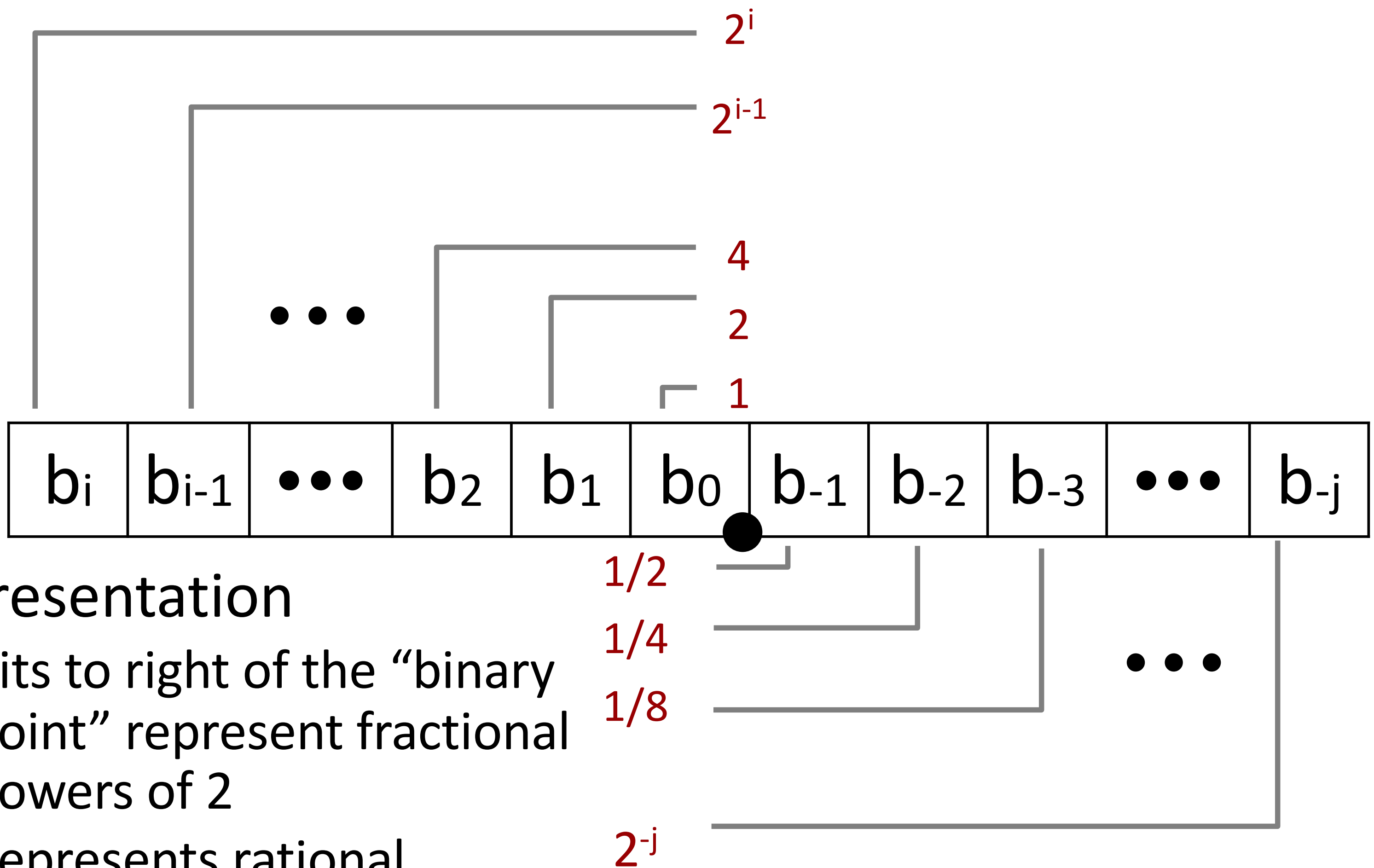
Floating Point coverage

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in C
- Properties of the computer representation of FP and integers
- Summary

Fractional binary numbers

- What is 1011.101_2?
- After the *"binary point"* → negative powers of 2

Fractional Binary Numbers



- Representation
  - Bits to right of the "binary point" represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Fractional binary numbers

- What is 1011.101_2?
  - $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}$ (left of bp)
    - $1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = ½ + 1/8 = 5/8$ (right of bp)

  - 11 5/8 in decimal

Fractional Binary Numbers: Examples

Value	Representation	Right of the binary point
5 3/4	**101.11_2**	3/4 = 1/2 + 1/4
2 7/8	**10.111_2**	7/8 = ½ + ¼ + 1/8
1 7/16	**1.0111_2**	7/16 = ¼ + 1/8 + 1/16

- Observations
  - Divide by 2 by shifting right (unsigned) (moving binary point to the left)
  - Multiply by 2 by shifting left (moving binary point to the right)
  - Numbers of form $0.111111\ldots_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$ to represent them

Representable Numbers

- Limitation #1
  - Can only exactly represent numbers of the form $x/2^k$
    - Other rational numbers have repeating bit representations and their value is approximated ("with increasing accuracy by lengthening the binary representation.").
  - Value Representation
    - 1/3 **0.0101010101[01]…$_2$**
    - 1/5 **0.001100110011[0011]…$_2$**
    - 1/10 **0.0001100110011[0011]…$_2$**

Representable Numbers (2)

- Limitation #2

  Just one setting of binary point within the *w* bits (positional notation)

  - Limited range of numbers (very small values? very large?)
  - not very efficient for very large numbers bit pattern with a large number of zeroes
  - Instead, we want to represent numbers of the form
    x * 2^y and just give x and y

Floating Point

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

IEEE Floating Point

- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats, each computer manufacturer had their own
  - Supported by all major CPUs
- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining the standard

Floating Point Representation

- Numerical Form: (interpretation of bit pattern)

$$(-1)^s \, M \times 2^E$$

  - Sign bit s determines whether number is negative or positive
  - Significand M normally a fractional value in range [1.0,2.0) or [0,1)
  - Exponent E weights value by power of two
- Encoding (how we represent it in the computer)
  - MSB s is sign bit s (1 bit)
  - exp field encodes E (but is not equal to E) (k bits)
  - frac field encodes M (but is not equal to M) (n bits)

s	exp	frac

Precision options

- Single precision: 32 bits

s	exp	frac
1	8-bits	23-bits

- Double precision: 64 bits

s	exp	frac
1	11-bits	52-bits

- Extended precision: 80 bits (Intel only)

s	exp	frac
1	15-bits	63 or 64-bits

Possible values encoded

The bit representation encodes a value in one of the 3 cases below, determined by *exp* value.

- Case 1: “Normalized” values
- Case 2: “Denormalized” values, to represent +0, -0, and numbers that are very close to 0.0
- Case 3: Special values, for infinity and NaN

1. “Normalized” Values

$v = (-1)^s M 2^E$

- When: exp ≠ 000…0 and exp ≠ 111…1
- Exponent coded as a biased value: E = Exp – Bias
  - Exp: unsigned value of exp field
  - Bias = 2^{k-1} - 1, where k is number of exponent bits
    - Single precision: 127 (Exp: 1…254, E: -126…127)
    - Double precision: 1023 (Exp: 1…2046, E: -1022…1023)
- Significand coded with implied leading 1: M = $1.xxx…x_2$ (1.0 >= M < 2.0)
  - xxx…x: bits of frac field
  - Minimum when frac = 000…0 (M = 1.0)
  - Maximum when frac = 111…1 (M = 2.0 – ε)
  - Get extra leading bit (= 1) for “free”

First Normalized Encoding Example

- Value: `float F = 13.0;`
  - $13_{10} = 1101_2$
    $= 1.101_2 \times 2^3$

$v = (-1)^s \, M \, 2^E$
$E = \text{Exp} - \text{Bias}$

- Significand

  M = $\mathtt{1.\underline{101}_2}$

  `frac=` $\mathtt{10100000000000000000000_2}$

  23 bits for frac

- Exponent

  E = 3

  $Bias$ = 127 (k = 8, $2^{k-1} - 1 = 128 - 1$)

  Exp = 130 = $\mathtt{10000010_2}$

- Result:

s	exp	frac
0	10000010	10100000000000000000000

Another Normalized Encoding Example

- Value: `float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$

$$v = (-1)^s M\, 2^E$$
$$E = Exp - Bias$$

- Significand
  M = $1.\underline{1101101101101}_2$
  frac= $\underline{1101101101101}0000000000_2$

23 bits for frac

- Exponent
  E = 13
  Bias = 127 (k = 8, $2^{k-1} - 1 = 128 - 1$)
  Exp = 140 = 10001100_2

- Result:

s	exp	frac
0	10001100	11011011011010000000000

2. Denormalized Values

$v = (-1)^s \, M \, 2^E$
$E = 1 - \text{Bias}$

- Condition: exp = 000…0
- Exponent value: E = 1 – Bias (instead of E = 0 – Bias)
- Significand coded with implied leading 0: $M = 0.xxx...x_2$
  - xxx…x: bits of frac
- Cases
  - exp = **000…0**, frac = **000…0**
    - Represents zero value
    - Note distinct values: +0 and –0 (why?)
  - exp = **000…0**, frac ≠ **000…0**
    - Numbers closest to 0.0
    - Equispaced

3. Special Values

Condition: **`exp`** = **`111…1`**

- Case: **`exp`** = **`111…1`**, **`frac`** = **`000…0`**
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$

- Case: **`exp`** = **`111…1`**, **`frac`** $\neq$ **`000…0`**
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., sqrt(–1), $\infty - \infty$, $\infty \times 0$

Visualization: Floating Point Encodings