Floating Point

These slides adapted from materials provided by the textbook

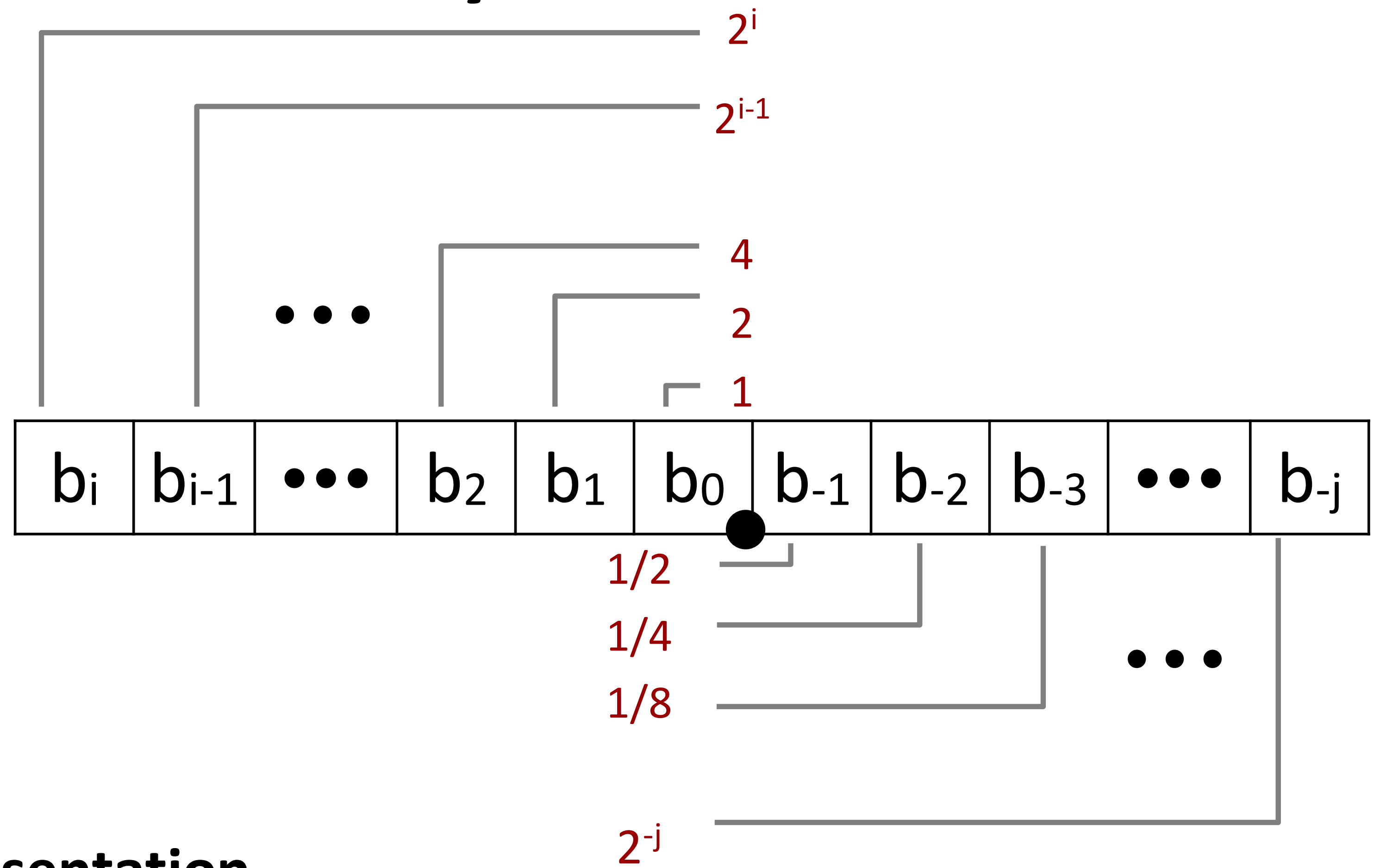
Floating Point

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

Fractional binary numbers

- **What is 1011.101_2?**

Fractional Binary Numbers



- **Representation**
  - Bits to right of "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?**

Fractional Binary Numbers: Examples

- **Value** **Representation**

5 3/4	101.11_2
2 7/8	10.111_2
1 7/16	1.0111_2

- **Observations**
  - Divide by 2 by shifting right (unsigned)
  - Multiply by 2 by shifting left
  - 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 $1.0 - \varepsilon$

Representable Numbers

- **Limitation #1**
  - Can only exactly represent numbers of the form $x/2^k$
    - Other rational numbers have repeating bit representations
  - Value Representation
    - 1/3 $0.0101010101[01]\ldots_2$
    - 1/5 $0.001100110011[0011]\ldots_2$
    - 1/10 $0.0001100110011[0011]\ldots_2$
- **Limitation #2**
  - Just one setting of binary point within the w bits
    - Limited range of numbers (very small values? very large?)

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
  - 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 standard

Floating Point Design Challenges

- Only so many bits. Need to be efficient
- Need to handle “special” numbers like Infinity
- Numbers near zero are “special” because 1/x is common
- Properly rounding numbers is important in many algorithms
- What to represent large numbers (1.23×10^{15}) and small (1.23×10^{-15})

Floating Point Representation

- **Numerical Form:**

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

  - Sign bit **s** determines whether number is negative or positive
  - Significand **M** normally a fractional value in range [1.0,2.0).
  - Exponent **E** weights value by power of two

- **Encoding**
  - MSB **s** is sign bit **s**
  - **exp** field encodes **E** (but is not equal to E)
  - **frac** field encodes **M** (but is not equal to M)

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

"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 = $1.xxx…x_2$**
  - 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 for "free"

Normalized Encoding Example

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

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

- **Significand**

  M = `1.`$\underline{\texttt{1101101101101}}_2$
  `frac`= $\underline{\texttt{1101101101101}}\texttt{0000000000}_2$

- **Exponent**

  E = 13
  $Bias$ = 127
  Exp = 140 = $\texttt{10001100}_2$

- **Result:**

s	exp	frac
0	10001100	11011011011010000000000

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…x2**
  - 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

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 ≠ 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