Data Representation: Floating Point Numbers

Csci 2021 - Machine Architecture and Organization
Professor Pen-Chung Yew

With sides from Randy Bryant and Dave O'Hallaron

Floating Point Numbers

- 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 1011.101₂?

Fractional Binary Numbers

- Representation
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number: \( \sum_{i=-\infty}^{\infty} b_i \times 2^i \)

Fractional Binary Numbers: Examples

- Value | Representation
  - 5 3/4 | 101.11₂
  - 2 7/8 | 10.111₂
  - 1 7/16 | 1.0111₂

- Observations
  - Divide by 2 by shifting right (unsigned)
  - Multiply by 2 by shifting left
  - Numbers of form 0.11111₁₂ are just below 1.0
  - \( 1/2 + 1/4 + 1/8 + ... + 1/2^n + ... \rightarrow 1.0 \)
  - Use notation 1.0 – \( \varepsilon \)

Representable Numbers

- Limitation #1
  - Can only exactly represent numbers of the form \( x/2^w \)
  - Other rational numbers may have repeating bit representations
    - Value | Representation
      - 1/3 | 0.0101010101[01]…₂
      - 1/5 | 0.001100110011[0011]…₂
      - 1/10 | 0.0001101100110011[0011]…₂

- Limitation #2
  - Just one setting of binary point within the \( w \) bits
  - Limited range of numbers (very small values? very large?)
Floating Point Numbers

- 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 CPU vendors
- 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

Announcement 2/3/2016

- Prof. Yew’s office hour on Friday 2/5/206 has been extended by ½ hour, i.e. from 10am-11:30am
- Data Lab due 11:55pm, next Monday 2/8/2016
  - Check Data Lab Forum for common Q&As, or post your problems there.
- Homework Assignment #1 has been issued on Monday 2/1/2016
  - Download from Moodle class webpage
  - Due date before class Wednesday 2/10/2016

Review: Representable Numbers

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

Floating Point Representation

- Numerical Form: \((-1)^s M \times 2^e\)
  - Sign bit \(s\) determines whether number is negative or positive
  - Significant \(M\) normally a fractional value in range \((1,0,2,0)\).
  - Exponent \(E\) weights value by power of two
- Encoding
  - Most significant bit (MSB) is sign bit \(s\)
  - exp field encodes \(E\) (but is not equal to \(E\))
  - frac field encodes \(M\) (but is not equal to \(M\))
  - Three different encoding schemes: (1) Normalized (exp ≠ 000…000, exp ≠ 111…111), (2) Denormalized (exp=000…000), (3) Not-a-Number (NaN) (exp = 1111…111)

Precision options

- Single precision: 32 bits
  - exp frac 23-bits
  - 1 8-bits
- Double precision: 64 bits
  - exp frac 52-bits
  - 1 11-bits
- Extended precision: 80 bits (Intel only, internal to hardware)
  - exp frac 63 or 64-bits
  - 1 15-bits
(1) “Normalized” Values

- When: \( \exp \neq 000...0 \) and \( \exp \neq 111...1 \)

- Exponent coded as a biased value: \( \text{Exp} = \exp + \text{Bias} \)
  - \( \exp \): unsigned value of \( \exp \) field
  - \( \text{Bias} = 2^{k-1} - 1 \), where \( k \) is the 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: \( \text{M} = 1.xxx…x \)
  - \( xxx…x \): bits of \( \text{frac} \) field
  - Minimum when \( \text{frac} = 000…0 \) (M = 1.0)
  - Maximum when \( \text{frac} = 111…1 \) (M = 2.0 − ε)

(2) Denormalized Values

- Condition: \( \exp = 000...0 \)

- Exponent value: \( E = 1 – \text{Bias} \) (instead of \( E = 0 – \text{Bias} \))

- Significand coded with implied leading 0: \( M = 0.xxx…x \)
  - \( xxx.x\): bits of \( \text{frac} \)
  - Allows gradual underflow

- Cases
  - \( \exp = 000..0, \text{frac} = 000..0 \)
    - Represents zero value
    - Note distinct values: +0 and −0 (why?)
  - \( \exp = 000.0, \text{frac} = 000.0 \)
    - Numbers closest to 0.0
    - Equi-spaced

(3) Special Values

- Condition: \( \exp = 111..1 \)

- Case: \( \exp = 111..1, \text{frac} = 000..0 \)
  - Represents value \( \infty \) (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., 1.0/0.0 = −\( \infty \)/−\( \infty \) = \( \infty \)/\( \infty \) = \( \infty \)

- Case: \( \exp = 111..1, \text{frac} \neq 000..0 \)
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., \( \sqrt{-1} = \infty = +0 \)

Visualization: Floating Point Encodings

Floating Point Numbers

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

### 8-bit Floating Point Representation
- Sign bit is in the most significant bit
- Next four bits are the exponent, with a bias of 7
- Last three bits are the \( \text{frac} \)

### Same general form as IEEE Format
- Normalized, denormalized
- Representation of 0, NaN, infinity

#### Dynamic Range (Positive Only)

| e | exp | frac | Value
|---|-----|-----|-----------------|
| 0 0000 000 | -6 | 0 | closest to zero
| 0 0000 001 | -6 | 1/8*1/64 = 1/512 | 0.001953125 |
| 0 0000 110 | -6 | 6/8*1/64 = 6/512 | 0.09765625 |
| 0 0000 111 | -6 | 7/8*1/64 = 7/512 | 0.125 |

#### Dynamic Range (Positive Only)

| e | exp | frac | Value
|---|-----|-----|-----------------|
| 0 0001 000 | -6 | 8/8*1/64 = 8/512 | closest to 1 below
| 0 0001 001 | -6 | 9/8*1/64 = 9/512 | 0.1875 |

#### Dynamic Range (Positive Only)

| e | exp | frac | Value
|---|-----|-----|-----------------|
| 0 0110 110 | -1 | 16/8*1/2 = 16/16 | closest to 1 above
| 0 0110 111 | -1 | 15/8*1/2 = 15/16 | 1 |

#### Dynamic Range (Positive Only)

| e | exp | frac | Value
|---|-----|-----|-----------------|
| 0 1110 110 | 7 | 14/8*128 = 224 | largest norm
| 0 1110 111 | 7 | 15/8*128 = 240 | 2 |

#### Dynamic Range (Positive Only)

| e | exp | frac | Value
|---|-----|-----|-----------------|
| 0 1111 000 | n/a | inf | infinity

### Distribution of Values

- 6-bit IEEE-like format
  - \( e = 3 \) exponent bits
  - \( f = 2 \) fraction bits
  - Bias is \( 2^{3-1} = 3 \)

- Notice how the distribution gets denser toward zero.

### Special Properties of the IEEE Encoding

- FP Zero Same as Integer Zero
  - All bits = 0

- Can (Almost) Use Unsigned Integer Comparison
  - Must first compare sign bits
  - Must consider -0 = 0
  - NaNs problematic
    - Will be greater than any other values
    - What should comparison yield?
  - Otherwise OK
    - Denormal vs. normalized
    - Normalized vs. infinity
    - Negative is smaller than positive (not so in 2’s complement)

### Floating Point Numbers

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

- \( x + y = \text{Round}(x + y) \)
- \( x \times y = \text{Round}(x \times y) \)

Basic idea
- First compute exact result
- Make it fit into desired precision
  - Possibly overflow if exponent too large
  - Possibly round to fit into \( \text{frac} \)

Closer Look at Round-To-Even

- Default Rounding Mode – Round to Even
  - Need to use assembly programming to get other rounding modes
  - 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
    - \( 7.8949999 \) \( \approx \) 7.89 (Less than half way)
    - \( 7.8950001 \) \( \approx \) 7.90 (Greater than half way)
    - \( 7.8950000 \) \( \approx \) 7.90 (Half way—round up)
    - \( 7.8850000 \) \( \approx \) 7.88 (Half way—round down)

Rounding

- Rounding Modes (Illustrate with 5 rounding)
  - Towards zero
  - Round down (\(-\))
  - Round up (\(+\))
  - Nearest Even (default)

    | Value | Binary | Rounded | Action | Rounded Value |
    |-------|--------|---------|--------|---------------|
    | 1.4   | 1.00   | 1       |        | 1             |
    | 1.6   | 1.00   | 1       |        | 1             |
    | 1.5   | 1.00   | 1       |        | 1             |
    | 2.5   | 1.01   | 1       |        | 2             |
    | -1.5  | 1.01   | 1       |        | 1             |

- Fixing
  - Exponent: \( E_1 \) (the larger of \( E_x \) and \( E_y \))
  - Sign: \( +_{E_1} \) if \( s_{E_1} \) \( = \) \(+\)
  - Sign of signed align & add
  - Round M to fit \( \text{frac} \) precision

Floating Point Addition

\[ (-1)^{s_{E_1}} M_1 \times 2^{E_1} + (-1)^{s_{E_2}} M_2 \times 2^{E_2} \]

- Exact Result: \((-1)^{s_{E_1}} M_1 \times 2^{E_1} \)
  - \( E_1 > E_2 \)
  - \( E_1 = E_2 \)
  - \( E_1 < E_2 \)

- Fixing
  - If \( M \geq 2 \), shift \( M \) right, increment \( E \)
  - If \( E \) out of range, overflow
  - Round \( M \) to fit \( \text{frac} \) precision

FP Multiplication

\[ (-1)^{s_1} M_1 \times 2^{E_1} \times (-1)^{s_2} M_2 \times 2^{E_2} \]

- Exact Result: \((-1)^{s_1} M_1 \times 2^{E_1} \)
  - Assume \( E_1 > E_2 \)

- Exponent \( E_{\text{result}} \)
  - Result of signed align & add

- Fixing
  - If \( M \geq 2 \), 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 \( \text{frac} \) precision
Review: Floating Point Encodings

- Normalized: $\pm 1 \times \text{fraction} \times 2^{\text{exponent}}$
- Denormalized: $0 \times \text{fraction} \times 2^{\text{exponent} - \text{bias}}$
- Inf: $\pm \infty$
- NaN: Not a Number

Review: Distribution of Values

- 6-bit IEEE-like format
  - $e = 3$ exponent bits
  - $f = 2$ fraction bits
  - Bias is $2^{3-1} - 1 = 3$

  Notice how the distribution gets denser toward zero.

  8 values

Review: Positive Only

- $s \times 2^E$ = Value
- $s = 0$ or 1
- $E = \text{Exp} - \text{Bias}$
- $E$ is in the range $-6$ to $6$

Review: A Close-up View

6-bit IEEE-like format
- $e = 3$ exponent bits
- $f = 2$ fraction bits
- Bias is 3

Review: Floating Point Addition

- $(-1)^{s_1} M_1 \times 2^{E_1} + (-1)^{s_2} M_2 \times 2^{E_2}$
- Assume $E_1 > E_2$

  Exact Result: $(-1)^s M \times 2^E$
  - Sign $s$, significand $M$:
  - Result of signed align & add
  - Exponent $E$: $E_1$ (the larger of $E_1$ and $E_2$)

  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 $\text{frac}$ precision

Review: Posi+ve Only

- Get binary points lined up
- Assume $E_1 > E_2$
Mathematical Properties of FP Add

- Mathematical Properties
  - Commutative? Yes
  - Associative? No
  - 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? Yes, except for infinities & NaNs

- Monotonicity
  - \(a \leq b \Rightarrow a + c \leq b + c\)
  - Except for infinities & NaNs

Floating Point Numbers

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

Summary

- IEEE Floating Point has clear mathematical properties
- Represents numbers of form \(M \times 2^e\)
- One can reason about operations independent of implementation
  - As if computed with perfect precision and then rounded
  - Not the same as real arithmetic
  - Violates associativity/distributivity
  - Makes life difficult for compilers & serious numerical applications programmers

Mathematical Properties of FP Mult

- Mathematical Properties
  - Multiplication Commutative? Yes
  - Multiplication is Associative? No
  - Possibility of overflow, inexactness of rounding
    - Ex: \((1e20 \times 1e20) \times 1e-20 = 1, 1e20 \times (1e20 \times 1e-20) = 1e20\)
  - 1 is multiplicative identity? Yes
  - Multiplication distributes over addition? No
  - Possibility of overflow, inexactness of rounding
    - \(1e20 \times (1e20 - 1e20) = 0.5, 1e20 \times 1e20 - 1e20 \times 1e20 = NaN\)

- Monotonicity
  - \(a \leq b \& c \geq 0 \Rightarrow a \times c \leq b \times c\)
  - Except for infinities & NaNs

Floating Point in C

- C Guarantees Two Levels
  - `float` single precision
  - `double` double precision

- Conversions/Casting
  - Type casting between int, float, and double changes bit representation
  - `double/float` → int
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - `int` → double
    - Exact conversion, as long as int has ≤ 53 bit word size
  - `int` → float
    - Will round according to rounding mode

Interesting Numbers

<table>
<thead>
<tr>
<th>Description</th>
<th>exp</th>
<th>frac</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>00...00</td>
<td>00...00</td>
<td>0.0</td>
</tr>
<tr>
<td>Smallest Pos. Denorm.</td>
<td>00...00</td>
<td>00...00</td>
<td>(2^{-126} \times 2^{-53} = 1.18 \times 10^{-38})</td>
</tr>
<tr>
<td>Smallest Pos. Denorm.</td>
<td>00...00</td>
<td>00...00</td>
<td>(2^{-126} \times 2^{-53} = 1.18 \times 10^{-38})</td>
</tr>
<tr>
<td>Largest Denormalized</td>
<td>01...11</td>
<td>11...11</td>
<td>(1.0 - \varepsilon \times 2^{-126} = 2^{-32})</td>
</tr>
<tr>
<td>Largest Normalized</td>
<td>11...11</td>
<td>11...11</td>
<td>(1.0 \times 2^{-32} = 2^{-32})</td>
</tr>
<tr>
<td>One</td>
<td>01...11</td>
<td>00...00</td>
<td>1.0</td>
</tr>
<tr>
<td>Largest Normalized</td>
<td>11...11</td>
<td>11...11</td>
<td>(2.0 - \varepsilon \times 2^{32} = 2^{32})</td>
</tr>
<tr>
<td>Single</td>
<td>1.0 \times 10^20</td>
<td>1.0 \times 10^20</td>
<td>(2^{32} = 2^{32})</td>
</tr>
<tr>
<td>Double</td>
<td>1.8 \times 10^{38}</td>
<td>1.8 \times 10^{38}</td>
<td>(2^{32} = 2^{32})</td>
</tr>
</tbody>
</table>
Machine-Level Data Representation (Done) to Machine-Level Program Representation (Next)