CSCI-UA.0201-003

Computer Systems Organization

Lecture 7: Floating Points

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Background: Fractional binary numbers

• What is \(1011.101_2\)?
Background: Fractional Binary Numbers

- Value:

\[
\sum_{k=-j}^{i} b_k \times 2^k
\]
## Fractional Binary Numbers: Examples

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 3/4</td>
<td>101.11₂</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.111₂</td>
</tr>
</tbody>
</table>

### Observations
- Divide by 2 by shifting right
- Multiply by 2 by shifting left
- 0.111111...₂ is just below 1.0
  - \( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^i} + \ldots \rightarrow 1.0 \)
Why not fractional binary numbers?

- Not efficient
  - $3 \times 2^{100} \rightarrow 10100000000 \ldots \ 0$
    - 100 zeros

- Given a finite length (e.g. 32-bits), cannot represent very large nor very small numbers ($\varepsilon \rightarrow 0$)
IEEE Floating Point

- IEEE Standard 754
  - Supported by all major CPUs

- Driven by numerical concerns
  - Standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard
Floating Point Representation

• Numerical Form:
  \((-1)^s \ M \ 2^E\)
  
  – Sign bit \(s\) determines whether number is negative or positive
  
  – Significand \(M\) a fractional value in range \([1.0,2.0)\) or \([0,1.0)\)
  
  – 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 \(M\))

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
</table>
Precisions

- **Single precision: 32 bits**

  - 1 8-bits
  - 23-bits

- **Double precision: 64 bits**

  - 1 11-bits
  - 52-bits

- **Extended precision: 80 bits (Intel only)**

  - 1 15-bits
  - 63 or 64-bits
1. Normalized Encoding

- **Condition**: $\exp \neq 000\ldots0$ and $\exp \neq 111\ldots1$

  referred to as Bias

- **Exponent** is: $E = \text{Exp} - (2^{k-1} - 1)$, $k$ is the # of exponent bits
  
  - Single precision: $E = \exp - 127$  \hspace{1cm} Range(E)=??
  
  - Double precision: $E = \exp - 1023$  \hspace{1cm} Range(E)=??

- **Significand** is: $M = 1.xxx\ldots x_2$
  
  - Range($M$) = $[1.0, 2.0 - \varepsilon)$
  
  - Get extra leading bit for free
Normalized Encoding Example

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

- **Significand**
  - $M = 1.\overline{1101101101101}_{2}$
  - $\frac{\text{frac}}{} = \overline{1101101101101000000000000}_{2}$

- **Exponent**
  - $E = \exp - \text{Bias} = \exp - 127 = 13$
  - $\Rightarrow \exp = 140 = \overline{10001100}_{2}$

- **Result:**
  - $\overline{0 \text{ 10001100 11011011101101000000000000}}$
  - $\text{s exp frac}$
2. Denormalized Encoding

- **Condition**: $\exp = 000\ldots0$

- **Exponent value**: $E = 1 - Bias$ (instead of $E = 0 - Bias$)

- **Significand is**: $M = 0.xxx\ldots x_2$ (instead of $M=1.xxx_2$)

- **Cases**
  - $\exp = 000\ldots0$, $\frac{\text{frac}}{} = 000\ldots0$
    - Represents zero
    - Note distinct values: $+0$ and $-0$
  - $\exp = 000\ldots0$, $\frac{\text{frac}}{} \neq 000\ldots0$
    - Numbers very close to $0.0$
3. Special Values Encoding

- **Condition:** \( \text{exp} = 111\ldots1 \)

- **Case:** \( \text{exp} = 111\ldots1, \text{frac} = 000\ldots0 \)
  - Represents value \( \infty \) (infinity)
  - Operation that overflows
  - E.g., \( 1.0/0.0 = -1.0/-0.0 = +\infty, \ 1.0/-0.0 = -\infty \)

- **Case:** \( \text{exp} = 111\ldots1, \text{frac} \neq 000\ldots0 \)
  - 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

-∞ - Normalized - Denorm + Denorm + Normalized +∞

NaN 0 0 NaN
Tiny Floating Point Example

- Toy example: 6-bit Floating Point Representation
  - Bias?

Normalized E = \exp - (2^{3-1}-1) = \exp - 3
Denormalized E = 1 - 3 = -2
Distribution of Values

8 values

Denormalized
Normalized
Infinity

Denormalized
Normalized
Infinity

Denormalized
Normalized
Infinity
Special Properties of 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, greater than any other values
  – Otherwise OK
    • Denorm vs. normalized
    • Normalized vs. infinity
# Floating Point Operations

- $x +_f y = \text{Round}(x + y)$
- $x \times_f y = \text{Round}(x \times y)$
- **Basic idea:** compute exact result, round to fit (possibly overflow)

## Rounding Modes

<table>
<thead>
<tr>
<th></th>
<th>$1.40$</th>
<th>$1.60$</th>
<th>$1.50$</th>
<th>$2.50$</th>
<th>$-1.50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Towards zero</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$2$</td>
<td>$-1$</td>
</tr>
<tr>
<td>Round down ($-\infty$)</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$2$</td>
<td>$-2$</td>
</tr>
<tr>
<td>Round up ($+\infty$)</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$3$</td>
<td>$-1$</td>
</tr>
<tr>
<td>Nearest Even (default)</td>
<td>$1$</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$-2$</td>
</tr>
</tbody>
</table>
Round to nearest even

• If the fraction is “half-way” then round upward/downward such that the resulting most significant bit is even.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Rounded</th>
<th>Action</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3/32</td>
<td>10.00011₂</td>
<td>10.00₂</td>
<td>(&lt;1/2—down)</td>
<td>2</td>
</tr>
<tr>
<td>2 3/16</td>
<td>10.00110₂</td>
<td>10.01₂</td>
<td>(&gt;1/2—up)</td>
<td>2 1/4</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.11100₂</td>
<td>11.00₂</td>
<td>(1/2—up)</td>
<td>3</td>
</tr>
<tr>
<td>2 5/8</td>
<td>10.10100₂</td>
<td>10.10₂</td>
<td>(1/2—down)</td>
<td>2 1/2</td>
</tr>
</tbody>
</table>

We consider the least significant bit value 0 to be even and 1 to be odd.
Floating Point in C

- **C**:  
  - `float` single precision  
  - `double` double precision

- **Conversions/Casting**  
  - 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
  - `int → double`
    - Exact conversion, as long as `int` has ≤ 53 bit word size
  - `int → float`
    - Will round according to rounding mode
Conclusions

• IEEE Floating Point has clear mathematical properties
• Represents numbers as: \((-1)^S \times M \times 2^E\)
• One can reason about operations independent of implementation
  – As if computed with perfect precision and then rounded