Floating Point

Instructor: Andrew Case

Slides adapted from Randy Bryant & Dave O’Hallaron
Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Floating point in C
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: Examples

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 3/4</td>
<td>101.11&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.111&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>63/64</td>
</tr>
<tr>
<td></td>
<td>1.0111&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

### Observations

- Divide by 2 by shifting right
- Multiply by 2 by shifting left
- Numbers of form 0.111111...<sub>2</sub> are just below 1.0
  - \( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^i} + \ldots \rightarrow 1.0 \)
  - Use notation 1.0 – \( \varepsilon \)
Representable Numbers

- **Limitation**
  - Can only exactly represent numbers of the form \( x/2^k \)
  - Other rational numbers have repeating bit representations

- **Value**
  - **Representation**
  - 1/3 \( 0.0101010101[01]..._2 \)
  - 1/5 \( 0.001100110011[0011]..._2 \)
  - 1/10 \( 0.0001100110011[0011]..._2 \)
Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Floating point in C
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 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 \(M\))
Precisions

- Single precision: 32 bits
  1 8-bits  23-bits

- Double precision: 64 bits
  1 11-bits  52-bits
Normalized Encoding Example

- **Value:** Float \( F = 15213.0 \);
  - \( 15213_{10} = 11101101101101_2 \)
    - \( = 1.1101101101101_2 \times 2^{13} \)

- **Significand**
  - \( M = 1.1101101101101_2 \)
  - \( frac = 11011011011010000000000000_2 \)

- **Exponent**
  - \( E = 13 \)
  - \( Bias = 127 \)
  - \( Exp = 140 = 10001100_2 \)

- **Result:**
  - \( \begin{array}{c|cc|c}
  \hline
  & s & \text{exp} & \text{frac} \\
  \hline
  & 0 & 10001100 & 11011011011010000000000000 \\
  \hline
  \end{array} \)
Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Floating point in C
Floating Point in C

- C Guarantees Two Levels
  - `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: 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
Summary

• Don’t compare floating point numbers just for equality:
  if (fabs(result - expectedResult) < 0.0001)