Many of the slides in this lecture are either from or adapted from slides provided by the authors of the textbook “Computer Systems: A Programmer’s Perspective,” 2nd Edition and are provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O’Hallaron in Fall 2010. These slides are indicated “Supplied by CMU” in the notes section of the slides.
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: \[ \sum_{k=-j}^{i} b_k \times 2^k \]

Supplied by CMU.
Representable Numbers

- **Limitation #1**
  - can exactly represent only numbers of the form \( n/2^k \)
    - other rational numbers have repeating bit representations
  - value representation
    - \( 1/3 \) \(0.0101010101[01]_{-2}\)
    - \( 1/5 \) \(0.001100110011[0011]_{-2}\)
    - \( 1/10 \) \(0.0001100110011[0011]_{-2}\)

- **Limitation #2**
  - just one setting of decimal point within the \( w \) bits
    - limited range of numbers (very small values? very large?)
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

Supplied by CMU.
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 \( s \)
  - \( \text{exp} \) field encodes \( E \) (but is not equal to \( E \))
  - \( \text{frac} \) field encodes \( M \) (but is not equal to \( M \))

Supplied by CMU.
Supplied by CMU.

On x86 hardware, all floating-point arithmetic is done with 80 bits, then reduced to either 32 or 64 as required.
“Normalized” Values

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

- Exponent coded as biased value: \( E = \text{Exp} - \text{Bias} \)
  - \( \text{exp} \): unsigned value \( \text{exp} \)
  - \( \text{bias} = 2^{k-1} - 1 \), 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 \( \text{frac} \)
  - minimum when \( \text{frac}=000...0 \) (\( M = 1.0 \))
  - maximum when \( \text{frac}=111...1 \) (\( M = 2.0 - \epsilon \))
  - get extra leading bit for “free”

Supplied by CMU.
Normalized Encoding Example

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

- **Significand**
  - \( M = 1.1101101101101_2 \)
  - \( frac = 1101101101101000000000000_2 \)

- **Exponent**
  - \( E = 13 \)
  - \( bias = 127 \)
  - \( exp = 140 = 1001100_2 \)

- **Result:**

  ![Normalized Encoding Example](image)

Supplied by CMU.
Denormalized Values

- **Condition:** exp = 000...0
- **Exponent value:** E = –Bias + 1 (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

Supplied by CMU.
Special Values

- **Condition:** $\exp = 111...1$

- **Case:** $\exp = 111...1, \frac{a}{b} = 000...0$
  - represents value $\infty$ (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:** $\exp = 111...1, \frac{a}{b} \neq 000...0$
  - not-a-number (NaN)
  - represents case when no numeric value can be determined
  - e.g., $\sqrt{-1}$, $\infty - \infty$, $\infty \times 0$
Supplied by CMU.
Tiny Floating-Point Example

```

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4-bits</td>
<td>3-bits</td>
</tr>
</tbody>
</table>

- **8-bit Floating Point Representation**
  - the sign bit is in the most significant bit
  - the next four bits are the exponent, with a bias of 7
  - the last three bits are the frac

- **Same general form as IEEE Format**
  - normalized, denormalized
  - representation of 0, NaN, infinity
```
# Dynamic Range (Positive Only)

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
<th>E</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>000</td>
<td>-6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>001</td>
<td>-6</td>
<td>1/8*1/64 = 1/512</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>010</td>
<td>-6</td>
<td>2/8*1/64 = 2/512</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>110</td>
<td>-6</td>
<td>6/8*1/64 = 6/512</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>111</td>
<td>-6</td>
<td>7/8*1/64 = 7/512</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>000</td>
<td>-6</td>
<td>8/8*1/64 = 8/512</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>001</td>
<td>-6</td>
<td>9/8*1/64 = 9/512</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>111</td>
<td>-1</td>
<td>14/8*1/2 = 14/16</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>111</td>
<td>-1</td>
<td>15/8*1/2 = 15/16</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>000</td>
<td>0</td>
<td>8/8*1 = 1</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>001</td>
<td>0</td>
<td>9/8*1 = 9/8</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>010</td>
<td>0</td>
<td>10/8*1 = 10/8</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>110</td>
<td>7</td>
<td>14/8*128 = 224</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>111</td>
<td>7</td>
<td>15/8*128 = 240</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
<td>000</td>
<td>n/a</td>
<td>inf</td>
</tr>
</tbody>
</table>

- Denormalized numbers: Closest to zero.
- Normalized numbers: Closest to 1 below.
- Normalized numbers: Closest to 1 above.
- Normalized numbers: Largest norm.

Supplied by CMU.
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.**

Supplied by CMU.
Distribution of Values (close-up view)

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

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3-bits</td>
<td>2-bits</td>
</tr>
</tbody>
</table>

-1 \[\rightarrow\] -0.5 \[\rightarrow\] 0 \[\rightarrow\] 0.5 \[\rightarrow\] 1

- Denormalized
- Normalized
- Infinity

Supplied by CMU.
Quiz 1

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

What number is represented by 0 011 10?

a) 12  
b) 1.5  
c) .5  
d) none of the above
Floating-Point Operations: Basic Idea

• $x +_e y = \text{Round}(x + y)$

• $x \times_e y = \text{Round}(x \times y)$

• Basic idea
  – first compute exact result
  – make it fit into desired precision
    » possibly overflow if exponent too large
    » possibly round to fit into frac

Supplied by CMU.
## Rounding

- **Rounding modes (illustrated with $ rounding)**

<table>
<thead>
<tr>
<th>Mode</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>

Supplied by CMU.
Floating-Point Multiplication

- \((-1)^{s_1} M_1 \ 2^{E_1}\) \(-\times\) \((-1)^{s_2} M_2 \ 2^{E_2}\)
- **Exact result:** \((-1)^s M \ 2^E\)
  - sign \(s\): \(s_1 \wedge s_2\)
  - significand \(M\): \(M_1 \times M_2\)
  - exponent \(E\): \(E_1 + E_2\)

- **Fixing**
  - if \(M \geq 2\), shift \(M\) right, increment \(E\)
  - if \(E\) out of range, overflow (or underflow)
  - round \(M\) to fit \(\frac{\text{precision}}{\text{fraction}}\)

- **Implementation**
  - biggest chore is multiplying significands

Supplied by CMU.

Note that to compute \(E\), one must first convert \(\exp_1\) and \(\exp_2\) to \(E_1\) and \(E_2\), then add them togethet and check for underflow or overflow (corresponding to \(-\infty\) and \(+\infty\)), and then convert to \(\exp\).
Floating-Point Addition

• \((-1)^{s_1} M_1 \ 2^{E_1} + (-1)^{s_2} M_2 \ 2^{E_2}\)
  
  – assume \(E_1 > E_2\)

• Exact result: \((-1)^{s} M \ 2^{E}\)
  
  – sign \(s\), significand \(M\):
    » result of signed align & add
  
  – exponent \(E\): \(E_1\)

• 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 frac precision

Supplied by CMU.
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
Quiz 2

Suppose f, declared to be a float, is assigned the largest possible floating-point positive value (other than $+\infty$). What is the value of $g = f+1.0$?

a) f  
b) $+\infty$  
c) NAN  
d) 0
Float is not Rational …

• Floating addition
  – commutative: a +f b = b +f a
    » yes!
  – associative: a +f (b +f c) = (a +f b) +f c
    » no!
      • 2 +f (1e10 +f -1e10) = 2
      • (2 +f 1e10) +f -1e10 = 0

Note that the floating-point numbers in this and the next two slides are expressed in base 10, not base 2.
Float is not Rational …

• Multiplication
  – commutative: $a \times f b = b \times f a$
    » yes!
  – associative: $a \times f (b \times f c) = (a \times f b) \times f c$
    » no!
    • $1e20 \times f (1e20 \times f 1e-20) = 1e20$
    • $(1e20 \times f 1e20) \times f 1e-20 = +\infty$
Float is not Rational …

• More …
  – multiplication distributes over addition:
    \[ a \times^f (b +^f c) = (a \times^f b) +^f (a \times^f c) \]
    » no!
    » \[ 1e20 \times^f (1e20 +^f -1e20) = 0 \]
    » \[ (1e20 \times^f 1e20) +^f (1e20 \times^f -1e20) = NaN \]
  – loss of significance:
    \[ x = y + 1 \]
    \[ z = \frac{2}{x-y} \]
    \[ z = 2? \]
    » not necessarily!
    • consider \[ y = 1e20 \]