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: \[ \sum_{k=-j}^i b_k \times 2^k \]
Representable Numbers

- **Limitation #1**
  - can exactly represent only numbers of the form \( n/2^k \)
    - other rational numbers have repeating bit representations
  - value representation
    - \( 1/3 \) 0.010101010101[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
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.
Precision options

- **Single precision: 32 bits**
  
<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8-bits</td>
<td>23-bits</td>
</tr>
</tbody>
</table>

- **Double precision: 64 bits**
  
<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11-bits</td>
<td>52-bits</td>
</tr>
</tbody>
</table>

- **Extended precision: 80 bits (Intel only)**
  
<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15-bits</td>
<td>64-bits</td>
</tr>
</tbody>
</table>

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: \( M = 1.x \ldots x_2 \)
  - \( x \ldots x \): bits of frac
  - minimum when frac=000...0 (\( M = 1.0 \))
  - maximum when frac=111...1 (\( M = 2.0 - \epsilon \))
  - get extra leading bit for “free”
Normalized Encoding Example

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

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

- **Exponent**
  - \( E = \quad 13 \)
  - \( bias = \quad 127 \)
  - \( exp = \quad 140 = \quad 10001100_2 \)

- **Result:**

```plaintext
  0 10001100 1101101101101000000000000
s  exp        frac
```

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: $M = 0.xxx...x2$
  - $xxx...x$: bits of $frac$
- Cases
  - $\exp = 000...0$, $frac = 000...0$
    » represents zero value
    » note distinct values: $+0$ and $-0$ (why?)
  - $\exp = 000...0$, $frac \neq 000...0$
    » numbers closest to 0.0
    » equispaced

Supplied by CMU.
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., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$

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

---

Supplied by CMU.
Visualization: Floating-Point Encodings

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 \texttt{frac}

- **Same general form as IEEE Format**
  - normalized, denormalized
  - representation of 0, NaN, infinity

Supplied by CMU.
## Dynamic Range (Positive Only)

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
<th>E</th>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>-6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>001</td>
<td>-6</td>
<td>1/8 \times 1/64 = 1/512</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>010</td>
<td>-6</td>
<td>2/8 \times 1/64 = 2/512</td>
<td>close to zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denormalized numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>110</td>
<td>-6</td>
<td>6/8 \times 1/64 = 6/512</td>
<td>largest denorm</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>111</td>
<td>-6</td>
<td>7/8 \times 1/64 = 7/512</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>000</td>
<td>-6</td>
<td>8/8 \times 1/64 = 8/512</td>
<td>smallest norm</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>001</td>
<td>-6</td>
<td>9/8 \times 1/64 = 9/512</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normalized numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>010</td>
<td>110</td>
<td>-1</td>
<td>14/8 \times 1/2 = 14/16</td>
<td>closest to 1 below</td>
</tr>
<tr>
<td>0</td>
<td>010</td>
<td>111</td>
<td>-1</td>
<td>15/8 \times 1/2 = 15/16</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>000</td>
<td>0</td>
<td>8/8 \times 1   = 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>001</td>
<td>0</td>
<td>9/8 \times 1   = 9/8</td>
<td>closest to 1 above</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>010</td>
<td>0</td>
<td>10/8 \times 1  = 10/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>110</td>
<td>7</td>
<td>14/8 \times 128 = 224</td>
<td>largest norm</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>111</td>
<td>7</td>
<td>15/8 \times 128 = 240</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>111</td>
<td>000</td>
<td>n/a</td>
<td>inf</td>
<td></td>
</tr>
</tbody>
</table>
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

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

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

Basic idea
- first compute exact result
- make it fit into desired precision
  » possibly overflow if exponent too large
  » possibly round to fit into 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 (−∞)</td>
<td>$1</td>
<td>$1</td>
<td>$1</td>
<td>$2</td>
<td>−$2</td>
</tr>
<tr>
<td>round up (+∞)</td>
<td>$2</td>
<td>$2</td>
<td>$2</td>
<td>$3</td>
<td>−$1</td>
</tr>
<tr>
<td>nearest integer</td>
<td>$1</td>
<td>$2</td>
<td>?</td>
<td>?</td>
<td>?</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.
Creating a Floating Point Number

- **Steps**
  - normalize to have leading 1
  - round to fit within fraction
  - postnormalize to deal with effects of rounding

- **Case study**
  - convert 8-bit unsigned numbers to tiny floating-point format

<table>
<thead>
<tr>
<th>Example numbers</th>
<th>Binary representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10000000</td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
</tr>
<tr>
<td>33</td>
<td>00010001</td>
</tr>
<tr>
<td>35</td>
<td>00010011</td>
</tr>
<tr>
<td>138</td>
<td>10001010</td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
</tr>
</tbody>
</table>

Supplied by CMU.
Normalize

- Requirement
  - set binary point so that numbers of form 1.xxxxx
  - adjust all to have leading one
    » decrement exponent as shift left

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Fraction</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10000000</td>
<td>1.0000000</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
<td>1.1010000</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>00010001</td>
<td>1.0001000</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>00010011</td>
<td>1.0011000</td>
<td>4</td>
</tr>
<tr>
<td>138</td>
<td>10001010</td>
<td>1.0001010</td>
<td>7</td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
<td>1.1111100</td>
<td>5</td>
</tr>
</tbody>
</table>
### Rounding

#### Guard bit: LSB of result

#### Sticky bit: OR of remaining bits

#### Round bit: 1st bit removed

- **Round-up conditions**
  - `round = 1`, `sticky = 1` $\Rightarrow$ > 0.5
  - `guard = 1`, `round = 1`, `sticky = 0` $\Rightarrow$ round to even

<table>
<thead>
<tr>
<th>Value</th>
<th>Fraction</th>
<th>GRS</th>
<th>Incr?</th>
<th>Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.0000000</td>
<td>000</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>13</td>
<td>1.1010000</td>
<td>100</td>
<td>N</td>
<td>1.101</td>
</tr>
<tr>
<td>17</td>
<td>1.0001000</td>
<td>010</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>19</td>
<td>1.0011000</td>
<td>110</td>
<td>Y</td>
<td>1.010</td>
</tr>
<tr>
<td>138</td>
<td>1.0001010</td>
<td>011</td>
<td>Y</td>
<td>1.001</td>
</tr>
<tr>
<td>63</td>
<td>1.1111100</td>
<td>111</td>
<td>Y</td>
<td>10.000</td>
</tr>
</tbody>
</table>

---

Supplied by CMU.
Postnormalize

- **Issue**
  - rounding may have caused overflow
  - handle by shifting right once & incrementing exponent

<table>
<thead>
<tr>
<th>Value</th>
<th>Rounded</th>
<th>Exp</th>
<th>Adjusted</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.000</td>
<td>7</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>13</td>
<td>1.101</td>
<td>3</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>1.000</td>
<td>4</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>1.010</td>
<td>4</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>138</td>
<td>1.001</td>
<td>7</td>
<td></td>
<td>134</td>
</tr>
<tr>
<td>63</td>
<td>10.000</td>
<td>5</td>
<td>1.000*2^6</td>
<td>64</td>
</tr>
</tbody>
</table>
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: \(s_1 ^ s_2\)
  - 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 precision

- Implementation
  - biggest chore is multiplying significands

Note that to compute E, one must first convert \(\exp_1\) and \(\exp_2\) to \(E_1\) and \(E_2\), then add them together and check for underflow or overflow (corresponding to \(-\infty\) and \(+\infty\)), and then convert to \(\exp\).
Note that, by default, overflow results in either $+\infty$ or $-\infty$. 

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 \texttt{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) \text{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!
    - $2 + f (1 e 20 + f -1 e 20) = 2$
    - $(2 + f 1 e 20) + f -1 e 20 = 0$

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=2/(x-y)
    z==2?
    » not necessarily!
      • consider y = 1e20