Data Representation (Part 3)
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  $0.0101010101[01]..._2$
      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
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 \( M \))
Precision options

- **Single precision: 32 bits**
  
  s \ exp \frac{1}{23} \ text{bits} \quad \text{fraction} \quad 23\text{-bits}

- **Double precision: 64 bits**
  
  s \ exp \frac{1}{52} \ text{bits} \quad \text{fraction} \quad 52\text{-bits}

- **Extended precision: 80 bits (Intel only)**
  
  s \ exp \frac{1}{64} \ text{bits} \quad \text{fraction} \quad 64\text{-bits}
“Normalized” Values

- When: $\exp \neq 000\ldots0$ and $\exp \neq 111\ldots1$
  - Exponent coded as *biased* value: $E = \Exp - \Bias$
    - $\exp$: unsigned value $\exp$
    - $\bias = 2^{k-1} - 1$, where $k$ is number of exponent bits
      » single precision: 127 ($\Exp: 1\ldots254$, $E: -126\ldots127$)
      » double precision: 1023 ($\Exp: 1\ldots2046$, $E: -1022\ldots1023$)

- Significand coded with implied leading 1: $M = 1.\xxx\ldots\ldotsx_2$
  - $\xxx\ldots\ldotsx$: bits of $\frac{1}{2} - 1$
  - minimum when $\frac{1}{2} = 000\ldots0$ ($M = 1.0$)
  - maximum when $\frac{1}{2} = 111\ldots1$ ($M = 2.0 - \epsilon$)
  - get extra leading bit for “free”
Normalized Encoding Example

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

• **Significand**
  
  \[ M = 1.1101101101101 \_2 \]
  \[ \text{frac} = \underbrace{11011011011010000000000000}_2 \]

• **Exponent**

  \[ E = 13 \]
  \[ \text{bias} = 127 \]
  \[ \text{exp} = 140 = 10001100 \_2 \]

• **Result:**

  
  \[ \begin{array}{cccccccccc}
    0 & 10001100 & 11011011011010000000000000 \\
    \text{s} & \text{exp} & \text{frac} 
  \end{array} \]
Denormalized Values

- **Condition**: $\text{exp} = 000\ldots0$
- **Exponent value**: $E = -\text{Bias} + 1$ (instead of $E = 0 - \text{Bias}$)
- **Significand coded with implied leading 0**: $M = 0.xxx\ldots x_2$
  - $xxx\ldots x$: bits of $\text{frac}$

**Cases**
- $\text{exp} = 000\ldots0$, $\text{frac} = 000\ldots0$
  » represents zero value
  » note distinct values: $+0$ and $-0$ (why?)
- $\text{exp} = 000\ldots0$, $\text{frac} \neq 000\ldots0$
  » numbers closest to $0.0$
  » equispaced
Special Values

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

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

- $\infty$
- $-\infty$
- Normalized
- Denorm
- $-0$
- $+0$
- +Denorm
- +Normalized
- $+\infty$
- NaN
- NaN
Tiny Floating-Point Example

- **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 $\text{frac}$

- **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>
</tbody>
</table>

**Denormalized numbers**

- $0 0000 110$ -6 $6/8*1/64 = 6/512$  
- $0 0000 111$ -6 $7/8*1/64 = 7/512$  

$0 0001 000$ -6 $8/8*1/64 = 8/512$  
$0 0001 001$ -6 $9/8*1/64 = 9/512$  

$0 0110 110$ -1 $14/8*1/2 = 14/16$  
$0 0110 111$ -1 $15/8*1/2 = 15/16$  

**Normalized numbers**

- $0 0111 000$ 0 $8/8*1 = 1$  
- $0 0111 001$ 0 $9/8*1 = 9/8$  
- $0 0111 010$ 0 $10/8*1 = 10/8$  

- $0 1110 110$ 7 $14/8*128 = 224$  
- $0 1110 111$ 7 $15/8*128 = 240$  

$0 1111 000$ n/a inf  

- **Smallest norm**
- **Closest to 1 below**
- **Closest to 1 above**
- **Largest norm**
- **Closest to zero**
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.

```
1 3-bits
2-bits
```

-15 -10 -5 0 5 10 15

- Denormalized  Normalized  Infinity

\[ s \, \text{exp} \, \text{frac} \]
Distribution of Values (close-up view)

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

-1 -0.5 0 0.5 1

- Denormalized
- Normalized
- Infinity
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

• $x +_f y = \text{Round}(x + y)$

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

• Basic idea
  – first compute exact result
  – make it fit into desired precision
    » possibly overflow if exponent too large
    » possibly round to fit into $\text{frac}$
### Rounding

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

<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>$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>
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

example numbers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10000000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>00010001</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>00010011</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>10001010</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
<td></td>
</tr>
</tbody>
</table>
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
Round bit: 1st bit removed

• Round-up conditions
  – round = 1, sticky = 1 ⇒ > 0.5
  – guard = 1, round = 1, sticky = 0 ⇒ 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>

1.BBG\textcolor{red}{RXXX}

Sticky bit: OR of remaining bits
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>128</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1.101</td>
<td>3</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1.000</td>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1.010</td>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>1.001</td>
<td>7</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>10.000</td>
<td>5</td>
<td>1.000*2^6 64</td>
<td></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 \(s\): \(s_1 \wedge s_2\)
  – 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 fraction precision

• Implementation
  – biggest chore is multiplying significands
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 \(\text{frac}\) precision
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
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: $a^f + b^f = b^f + a^f$
    » yes!
  – associative: $(a^f + b^f + c^f) = (a^f + b^f) + c^f$
    » no!
  • $2^f + (1 \times 10^{20} + f - 1 \times 10^{20}) = 2$
  • $(2^f + 1 \times 10^{20}) + f - 1 \times 10^{20} = 0$
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 \( y = 1e20 \)