CS 33

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
  
  - 8-bits for the exponent
  - 23-bits for the fraction

- **Double precision**: 64 bits
  
  - 11-bits for the exponent
  - 52-bits for the fraction

- **Extended precision**: 80 bits (Intel only)
  
  - 15-bits for the exponent
  - 64-bits for the fraction
“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: \( M = 1.xxx\ldots x2 \)
  – \( xxx\ldots x \): bits of \( \text{frac} \)
  – minimum when \( \text{frac} = 000\ldots0 \) (\( M = 1.0 \))
  – maximum when \( \text{frac} = 111\ldots1 \) (\( M = 2.0 - \varepsilon \))
  – 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**
  - \( M = 1.1101101101101_{2} \)
  - \( \text{frac} = 1101101101101000000000000_{2} \)

- **Exponent**
  - \( E = 13 \)
  - \( bias = 127 \)
  - \( exp = 140 = 10001100_{2} \)

- **Result:**
  - \( s \) 10001100 110110110110100000000000000
  - \( \text{exp} \) 10001100
  - \( \text{frac} \) 110110110110100000000000000000
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

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

+0  -0

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 $frac$

- **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 000</td>
<td>-6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0000 001</td>
<td>-6</td>
<td>1/8*1/64 = 1/512</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000 010</td>
<td>-6</td>
<td>2/8*1/64 = 2/512</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>closest to zero</td>
</tr>
<tr>
<td>0</td>
<td>0000 110</td>
<td>-6</td>
<td>6/8*1/64 = 6/512</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000 111</td>
<td>-6</td>
<td>7/8*1/64 = 7/512</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>largest denorm</td>
</tr>
<tr>
<td>0</td>
<td>0001 000</td>
<td>-6</td>
<td>8/8*1/64 = 8/512</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0001 001</td>
<td>-6</td>
<td>9/8*1/64 = 9/512</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>smallest norm</td>
</tr>
<tr>
<td>0</td>
<td>0110 110</td>
<td>-1</td>
<td>14/8*1/2 = 14/16</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0110 111</td>
<td>-1</td>
<td>15/8*1/2 = 15/16</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>closest to 1 below</td>
</tr>
<tr>
<td>0</td>
<td>0111 000</td>
<td>0</td>
<td>8/8*1 = 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0111 001</td>
<td>0</td>
<td>9/8*1 = 9/8</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0111 010</td>
<td>0</td>
<td>10/8*1 = 10/8</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>closest to 1 above</td>
</tr>
<tr>
<td>0</td>
<td>1110 110</td>
<td>7</td>
<td>14/8*128 = 224</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1110 111</td>
<td>7</td>
<td>15/8*128 = 240</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>largest norm</td>
</tr>
<tr>
<td>0</td>
<td>1111 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.

8 values

<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>

Denormalized ▲ Normalized ☐ Infinity ☐
Distribution of Values (close-up view)

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

![Diagram showing distribution of values with 6-bit format: s (sign), exp (exponent), frac (fraction), 3-bits, 2-bits. The diagram illustrates denormalized, normalized, and infinity values.]
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 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 (−∞)</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 even (default)</td>
<td>$1</td>
<td>$2</td>
<td>$2</td>
<td>$2</td>
<td>−$2</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 \(\text{frac}\) 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
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) \( \text{NAN} \)

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!
  - $2 +^f (1e10 +^f -1e10) = 2$
  - $(2 +^f 1e10) +^f -1e10 = 0$
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!
    » \(1e20 \times_f (1e20 +_f -1e20) = 0\)
    » \((1e20 \times_f 1e20) +_f (1e20 \times_f -1e20) = \text{NaN}\)
  – loss of significance:
    \[ x = y + 1 \]
    \[ z = 2/(x-y) \]
    \[ z == 2? \]
    » not necessarily!
    • consider \(y = 1e20\)