Floating Point

Introduction to Computer Systems
3rd Lecture, Sep. 8, 2015

Instructor:
Younghoon Kim
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>101.11₂</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.111₂</td>
</tr>
<tr>
<td>1 7/16</td>
<td>1.0111₂</td>
</tr>
</tbody>
</table>

Observations

- Divide by 2 by shifting right (unsigned)
- Multiply by 2 by shifting left
- Numbers of form 0.111111\ldots₂ are just below 1.0
  - \(1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0\)
  - Use notation \(1.0 - \varepsilon\)
Representable Numbers

- **Limitation #1**
  - 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]...\(_2\)

- **Limitation #2**
  - Just one setting of binary 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 \( M \))
**Precision options**

- **Single precision: 32 bits**
  - s: 1 bit
  - exp: 8 bits
  - frac: 23 bits

- **Double precision: 64 bits**
  - s: 1 bit
  - exp: 11 bits
  - frac: 52 bits

- **Extended precision: 80 bits (Intel only)**
  - s: 1 bit
  - exp: 15 bits
  - frac: 63 or 64 bits
“Normalized” Values

- When: exp ≠ 000...0 and exp ≠ 111...1

- Exponent coded as a **biased** value: \( E = \text{Exp} - \text{Bias} \)
  - \( \text{Exp} \): unsigned value of exp field
  - \( \text{Bias} = 2^{k-1} - 1 \), 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 frac field
  - Minimum when frac=000...0 (M = 1.0)
  - Maximum when frac=111...1 (M = 2.0 – \( \epsilon \))
  - Get extra leading bit for “free”

\[ v = (-1)^s \, M \, 2^E \]
Normalized Encoding Example

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

- Significand
  - \( M = 1.1101101101101_2 \)
  - \( \text{frac} = 11011011011010000000000000_2 \)

- Exponent
  - \( E = 13 \)
  - \( \text{Bias} = 127 \)
  - \( \text{Exp} = 140 = 10001100_2 \)

- Result:
  - \( v = (-1)^s \ M \ 2^E \)
  - \( E = \text{Exp} - \text{Bias} \)

\[
\begin{array}{ccc}
0 & 10001100 & 1101101101101000000000000000 \\
\text{s} & \text{exp} & \text{frac}
\end{array}
\]
Denormalized Values

- **Condition:** \( \text{exp} = 000\ldots0 \)

- **Exponent value:** \( E = 1 - \text{Bias} \) (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  +∞

NaN  −0  +0  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 \texttt{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></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>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></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>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></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>0</td>
<td>1111 000</td>
<td>n/a</td>
<td>inf</td>
<td></td>
</tr>
</tbody>
</table>

**Denormalized numbers**

- closest to zero
- largest denorm
- smallest norm

**Normalized numbers**

- closest to 1 below
- closest to 1 above
- largest norm

\[ v = (-1)^s \cdot M \cdot 2^E \]

\[ n: E = \text{Exp} - \text{Bias} \]

\[ d: E = 1 - \text{Bias} \]
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.**

---

<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>

8 values

- 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 of the 6-bit IEEE-like format with 3 exponent bits and 2 fraction bits. Denormalized, normalized values, and infinity are shown on a number line.](image)
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
Floating Point Puzzles

For each of the following C expressions, either:

- Argue that it is true for all argument values
- Explain why not true


- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (double)(float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0  ⇒  ((d*2) < 0.0)`
- `d > f  ⇒  -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Assume neither `d` nor `f` is NaN
# Interesting Numbers

<table>
<thead>
<tr>
<th>Description</th>
<th>exp</th>
<th>frac</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>00...00</td>
<td>00...00</td>
<td>0.0</td>
</tr>
<tr>
<td>Smallest Pos. Denorm.</td>
<td>00...00</td>
<td>00...01</td>
<td>(2^{-{23,52}} \times 2^{-{126,1022}})</td>
</tr>
<tr>
<td>- Single</td>
<td></td>
<td></td>
<td>(\approx 1.4 \times 10^{-45})</td>
</tr>
<tr>
<td>- Double</td>
<td></td>
<td></td>
<td>(\approx 4.9 \times 10^{-324})</td>
</tr>
<tr>
<td>Largest Denormalized</td>
<td>00...00</td>
<td>11...11</td>
<td>((1.0 - \varepsilon) \times 2^{-{126,1022}})</td>
</tr>
<tr>
<td>- Single</td>
<td></td>
<td></td>
<td>(\approx 1.18 \times 10^{-38})</td>
</tr>
<tr>
<td>- Double</td>
<td></td>
<td></td>
<td>(\approx 2.2 \times 10^{-308})</td>
</tr>
<tr>
<td>Smallest Pos. Normalized</td>
<td>00...01</td>
<td>00...00</td>
<td>(1.0 \times 2^{-{126,1022}})</td>
</tr>
<tr>
<td>One</td>
<td>01...11</td>
<td>00...00</td>
<td>1.0</td>
</tr>
<tr>
<td>Largest Normalized</td>
<td>11...10</td>
<td>11...11</td>
<td>((2.0 - \varepsilon) \times 2^{{127,1023}})</td>
</tr>
<tr>
<td>- Single</td>
<td></td>
<td></td>
<td>(\approx 3.4 \times 10^{38})</td>
</tr>
<tr>
<td>- Double</td>
<td></td>
<td></td>
<td>(\approx 1.8 \times 10^{308})</td>
</tr>
</tbody>
</table>