Representing and Manipulating Floating Points

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
The Problem

• How to represent fractional values with finite number of bits?
  – 0.1
  – 0.612
  – 3.14159265358979323846264338327950288...

• Wide ranges of numbers
  – 1 Light-Year = 9,460,730,472,580.8 km
  – The radius of a hydrogen atom: 0.000000000025 m
Fractional Binary Numbers (1)

• Representation
  – Bits to right of “binary point” represent fractional powers of 2
  – Represents rational number: \[ \sum_{k=-j}^{i} b_k \cdot 2^k \]
Fractional Binary Numbers (2)

• 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>63/64</td>
<td>0.111111₁₂</td>
</tr>
</tbody>
</table>

• Observations
  – Divide by 2 by shifting right
  – Multiply by 2 by shifting left
  – Numbers of form 0.111111..₂ just below 1.0
    • \(1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0\)
    • Use notation 1.0 – \(\epsilon\)
Fractional Binary Numbers (3)

• Representable numbers
  – Can only exactly represent numbers of the form $x / 2^k$
  – Other numbers have repeating bit representations

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>0.010101010101[01]...₂</td>
</tr>
<tr>
<td>1/5</td>
<td>0.001100110011[0011]...₂</td>
</tr>
<tr>
<td>1/10</td>
<td>0.0001100110011[0011]...₂</td>
</tr>
</tbody>
</table>
Fixed-Point Representation (1)

- **p.q** Fixed-point representation
  - Use the rightmost q bits of an integer as representing a fraction
  - Example: 17.14 fixed-point representation
    - 1 bit for sign bit
    - 17 bits for the integer part
    - 14 bits for the fractional part
    - An integer \(x\) represents the real number \(x / 2^{14}\)
    - Maximum value: \((2^{31} - 1) / 2^{14} \approx 131071.999\)

![Binary representation of 17.14](image)
Fixed-Point Representation (2)

• Properties
  – Convert \( n \) to fixed point: \( n \times f \) (\( = n \ll q \))
  – Add \( x \) and \( y \): \( x + y \)

\[
\begin{array}{c}
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3.5
\end{array}
\]

\[
+\begin{array}{c}
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1.75
\end{array}
\]

\[
\begin{array}{c}
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 15.25
\end{array}
\]

  – Subtract \( y \) from \( x \): \( x - y \)
  – Add \( x \) and \( n \): \( x + n \times f \)
  – Multiply \( x \) by \( n \): \( x \times n \)
  – Divide \( x \) by \( n \): \( x / n \)

\( x, y \): fixed-point number
\( n \): integer
\( f = 1 \ll q \)
Fixed-Point Representation (3)

• Pros
  – Simple
  – Can use integer arithmetic to manipulate
  – No floating-point hardware needed
  – Used in many low-cost embedded processors or DSPs (digital signal processors)

• Cons
  – Cannot represent wide ranges of numbers
Representing Floating Points

- IEEE standard 754
  - Established in 1985 as uniform standard for floating-point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs
  - William Kahan, a primary architect of IEEE 754, won the Turing Award in 1989
  - Driven by numerical concerns
    - Nice standards for rounding, overflow, underflow
    - Hard to make go fast
    - Numerical analysts predominated over hardware types in defining standard
FP Representation

• Numerical form: \(-1^s \times M \times 2^E\)
  – 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

\[
\begin{array}{c|c|c}
  s & exp & frac \\
\end{array}
\]

  – MSB is sign bit \(s\)
  – \(exp\) field encodes \(E\) (Exponent)
  – \(frac\) field encodes \(M\) (Mantissa)
FP Precisions

- **Single precision**
  - 8 \texttt{exp} bits, 23 \texttt{frac} bits (32 bits total)

- **Double precision**
  - 11 \texttt{exp} bits, 52 \texttt{frac} bits (64 bits total)

- **Extended precision**
  - 15 \texttt{exp} bits, 63 \texttt{frac} bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits (1 bit wasted)
Normalized Values

• Condition: $\text{exp} \neq 000\ldots0$ and $\text{exp} \neq 111\ldots1$

• Exponent coded as a biased value
  
  $E = \text{Exp} - \text{Bias}$
  
  $\text{Exp}$: unsigned value denoted by $\text{exp}$
  
  $\text{Bias}$: Bias value ($=2^{k-1}-1$, where $k$ is the number of $\text{exp}$ bits)
    
    • Single precision ($k=8$): 127 ($\text{Exp}: 1..254$, $E$: -126..127)
    • Double precision ($k=11$): 1023 ($\text{Exp}: 1..2046$, $E$: -1022..1023)

• Significand coded with implied leading 1
  
  $M = 1.xxx\ldots x_2$
    
    • 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 Values: Example

- float f = 2003.0;
  - \(2003_{10} = 11111010011_2 = 1.1111010011_2 \times 2^{10}\)

- **Significand**
  - \(M = 1.1111010011_2\)
  - \(frac = 111101001100000000000000000_2\)

- **Exponent**
  - \(E = 10\)
  - \(Exp = E + Bias = 10 + 127 = 137 = 10001001_2\)

<table>
<thead>
<tr>
<th>Hex:</th>
<th>4</th>
<th>4</th>
<th>F</th>
<th>A</th>
<th>6</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary:</td>
<td>0100</td>
<td>0100</td>
<td>1111</td>
<td>1010</td>
<td>0110</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>137:</td>
<td>100</td>
<td>0100</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003:</td>
<td>1111</td>
<td>1010</td>
<td>0110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Denormalized Values

- **Condition:** $\exp = 000\ldots0$
- **Value**
  - Exponent value $E = 1 - \text{Bias}$
  - Significand value $M = 0.xxx\ldots x_2$ (no implied leading 1)
- **Case 1:** $\exp = 000\ldots0, \frac{\text{c}}{\text{a}} = 000\ldots0$
  - Represents value $0.0$
  - Note that there are distinct values $+0$ and $-0$
- **Case 2:** $\exp = 000\ldots0, \frac{\text{c}}{\text{a}} \neq 000\ldots0$
  - Numbers very close to $0.0$
  - “Gradual underflow”: possible numeric values are spaced evenly near $0.0$
Special Values

- **Condition:** $\exp = 111\ldots1$
- **Case 1:** $\exp = 111\ldots1, \frac{\text{frac}} = 000\ldots0$
  - 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 2:** $\exp = 111\ldots1, \frac{\text{frac}} \neq 000\ldots0$
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - e.g. $\sqrt{-1}$, $\infty - \infty$, $\infty \ast 0$, ...
Tiny FP Example (1)

• 8-bit floating point representation
  – The sign bit is in the most significant bit
  – The next four bits are the \( \text{exp} \), 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
Tiny FP Example (2)

- Values related to the exponent (*Bias* = 7)

<table>
<thead>
<tr>
<th>Description</th>
<th>Exp</th>
<th>exp</th>
<th>$E = \text{Exp} - \text{Bias}$</th>
<th>$2^E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denormalized</td>
<td>0</td>
<td>0000</td>
<td>-6</td>
<td>1/64</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0001</td>
<td>-6</td>
<td>1/64</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0010</td>
<td>-5</td>
<td>1/32</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0011</td>
<td>-4</td>
<td>1/16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0100</td>
<td>-3</td>
<td>1/8</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0101</td>
<td>-2</td>
<td>1/4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0110</td>
<td>-1</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0111</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1001</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1010</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1011</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1100</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>1101</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1110</td>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>inf, NaN</td>
<td>15</td>
<td>1111</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Tiny FP Example (3)

- **Dynamic range**

<table>
<thead>
<tr>
<th>Description</th>
<th>Bit representation</th>
<th>e</th>
<th>E</th>
<th>f</th>
<th>M</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Zero</strong></td>
<td>0 0000 000</td>
<td>0</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Smallest pos.</strong></td>
<td>0 0000 001</td>
<td>0</td>
<td>-6</td>
<td>1/8</td>
<td>1/8</td>
<td>1/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 010</td>
<td>0</td>
<td>-6</td>
<td>2/8</td>
<td>2/8</td>
<td>2/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 011</td>
<td>0</td>
<td>-6</td>
<td>3/8</td>
<td>3/8</td>
<td>3/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 110</td>
<td>0</td>
<td>-6</td>
<td>6/8</td>
<td>6/8</td>
<td>6/512</td>
</tr>
<tr>
<td><strong>Largest denorm.</strong></td>
<td>0 0000 111</td>
<td>0</td>
<td>-6</td>
<td>7/8</td>
<td>7/8</td>
<td>7/512</td>
</tr>
<tr>
<td><strong>Smallest norm.</strong></td>
<td>0 0001 000</td>
<td>1</td>
<td>-6</td>
<td>0</td>
<td>8/8</td>
<td>8/512</td>
</tr>
<tr>
<td></td>
<td>0 0001 001</td>
<td>1</td>
<td>-6</td>
<td>1/8</td>
<td>9/8</td>
<td>9/512</td>
</tr>
<tr>
<td></td>
<td>0 0110 110</td>
<td>6</td>
<td>-1</td>
<td>6/8</td>
<td>14/8</td>
<td>14/16</td>
</tr>
<tr>
<td></td>
<td>0 0110 111</td>
<td>6</td>
<td>-1</td>
<td>7/8</td>
<td>15/8</td>
<td>15/16</td>
</tr>
<tr>
<td><strong>One</strong></td>
<td>0 0111 000</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8/8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 0111 001</td>
<td>7</td>
<td>0</td>
<td>1/8</td>
<td>9/8</td>
<td>9/8</td>
</tr>
<tr>
<td></td>
<td>0 0111 010</td>
<td>7</td>
<td>0</td>
<td>2/8</td>
<td>10/8</td>
<td>10/8</td>
</tr>
<tr>
<td></td>
<td>0 1110 110</td>
<td>14</td>
<td>7</td>
<td>6/8</td>
<td>14/8</td>
<td>224</td>
</tr>
<tr>
<td><strong>Largest norm.</strong></td>
<td>0 1110 111</td>
<td>14</td>
<td>7</td>
<td>7/8</td>
<td>15/8</td>
<td>240</td>
</tr>
<tr>
<td><strong>Infinity</strong></td>
<td>0 1111 000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+∞</td>
</tr>
</tbody>
</table>
Tiny FP Example (4)

- Encoded values (nonnegative numbers only)

0 1101 XXX = (8/8 ~ 15/8) * 2^6

0 1110 XXX = (8/8 ~ 15/8) * 2^7

0 0111 XXX = (8/8 ~ 15/8) * 2^0

0 1000 XXX = (8/8 ~ 15/8) * 2^1

0 0000 XXX = (0/8 ~ 7/8) * 2^-6

0 0001 XXX = (8/8 ~ 15/8) * 2^-6

0 0011 XXX = (8/8 ~ 15/8) * 2^-4

(Without denormalization)

0 00000 XXX = (8/8 ~ 15/8) * 2^-7
# 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>000 ... 00</td>
<td>000 ... 00</td>
<td>0.0</td>
</tr>
<tr>
<td>Smallest Positive</td>
<td>000 ... 00</td>
<td>000 ... 01</td>
<td>Single: $2^{-23} \times 2^{-126} \approx 1.4 \times 10^{-45}$ Double: $2^{-52} \times 2^{-1022} \approx 4.9 \times 10^{-324}$</td>
</tr>
<tr>
<td>denormalized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest</td>
<td>000 ... 00</td>
<td>111 ... 11</td>
<td>Single: $(1.0 - \varepsilon) \times 2^{-126} \approx 1.18 \times 10^{-38}$ Double: $(1.0 - \varepsilon) \times 2^{-1022} \approx 2.2 \times 10^{-308}$</td>
</tr>
<tr>
<td>Denormalized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Positive</td>
<td>000 ... 01</td>
<td>000 ... 00</td>
<td>Single: $1.0 \times 2^{-126}$, Double: $1.0 \times 2^{-1022}$ (Just larger than largest denormalized)</td>
</tr>
<tr>
<td>Normalized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>011 ... 11</td>
<td>000 ... 00</td>
<td>1.0</td>
</tr>
<tr>
<td>Largest</td>
<td>111 ... 10</td>
<td>111 ... 11</td>
<td>Single: $(2.0 - \varepsilon) \times 2^{127} \approx 3.4 \times 10^{38}$ Double: $(2.0 - \varepsilon) \times 2^{1023} \approx 1.8 \times 10^{308}$</td>
</tr>
<tr>
<td>Normalized</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Special Properties

• FP zero same as integer zero
  – All bits = 0

• Can (almost) use unsigned integer comparison
  – Must first compare sign bits
  – Must consider –0 = 0
  – NaNs problematic
    • Will be greater than any other values
  – Otherwise OK
    • Denormalized vs. normalized
    • Normalized vs. Infinity
Rounding

- For a given value $x$, finding the “closest” matching value $x'$ that can be represented in the FP format
- IEEE 754 defines four rounding modes
  - Round-to-even avoids statistical bias by rounding upward or downward so that the least significant digit is even

<table>
<thead>
<tr>
<th>Rounding modes</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>Round-toward-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><strong>Round-to-even (default)</strong> or Round-to-nearest</td>
<td>$1$</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$-2$</td>
</tr>
</tbody>
</table>
Round-to-Even

- Round up conditions
  - \( R = 1, S = 1 \rightarrow > 0.5 \)
  - \( G = 1, R = 1, S = 0 \)
  - Round to even

### 

<table>
<thead>
<tr>
<th>Value</th>
<th>Fraction</th>
<th>GRS</th>
<th>Up?</th>
<th>Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.0000000 ((x2^7))</td>
<td>000</td>
<td>No</td>
<td>1.000</td>
</tr>
<tr>
<td>13</td>
<td>1.1010000 ((x2^3))</td>
<td>100</td>
<td>No</td>
<td>1.101</td>
</tr>
<tr>
<td>17</td>
<td>1.0001000 ((x2^4))</td>
<td>010</td>
<td>No</td>
<td>1.000</td>
</tr>
<tr>
<td>19</td>
<td>1.0011000 ((x2^4))</td>
<td>110</td>
<td>Yes</td>
<td>1.010</td>
</tr>
<tr>
<td>138</td>
<td>1.0001010 ((x2^7))</td>
<td>011</td>
<td>Yes</td>
<td>1.001</td>
</tr>
<tr>
<td>63</td>
<td>1.1111000 ((x2^5))</td>
<td>111</td>
<td>Yes</td>
<td>10.000</td>
</tr>
</tbody>
</table>

1.BBG

- **Guard** bit: LSB of result
- **Round** bit: 1\textsuperscript{st} bit removed
- **Sticky** bit: OR of remaining bits
FP Addition

• Adding two numbers: \[
\begin{array}{c}
\text{\(s_1\)} & \text{\(E_1\)} & \text{\(M_1\)} & \text{+} & \text{\(s_2\)} & \text{\(E_2\)} & \text{\(M_2\)}
\end{array}
\]

(Assume \(E_1 > E_2\))

– Align binary points
  • Shift right \(M_2\) by \(E_1 - E_2\)

– Add significands
  • Result: Sign \(s\), Significand \(M\), Exponent \(E\) (= \(E_1\))

– Normalize result
  • if \((M \geq 2)\), shift \(M\) right, increment \(E\)
  • if \((M < 1)\), shift \(M\) left \(k\) positions, decrement \(E\) by \(k\)

– Check for overflow (\(E\) out of range?)

– Round \(M\) and renormalize if necessary
FP Multiplication

• Multiplying two numbers: $s_1 E_1 M_1 \times s_2 E_2 M_2$
  
  – Obtain exact result
    • Sign $s = s_1 \wedge s_2$
    • Significand $M = M_1 \times M_2$
    • Exponent $E = E_1 + E_2$
    • The biggest chore is multiplying significands
  
  – Normalize result
    • if ($M \geq 2$), shift $M$ right, increment $E$
    • if ($M < 1$), shift $M$ left $k$ positions, decrement $E$ by $k$
  
  – Check for overflow ($E$ out of range?)
  
  – Round $M$ and renormalize if necessary
Floating Points in C

• C guarantees two levels
  – float (single precision) vs. double (double precision)

• Conversions
  – double or 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
FP Example 1

```c
#include <stdio.h>

int main ()
{
    int n = 123456789;
    int nf, ng;
    float f;
    double g;

    f = (float) n;
    g = (double) n;
    nf = (int) f;
    ng = (int) g;
    printf ("nf=%d ng=%d\n", nf, ng);
}
```
FP Example 2

```c
#include <stdio.h>

int main ()
{
    double d;

    d = 1.0 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;

    printf ("d = %.20f\n", d);
}
```
FP Example 3

```c
#include <stdio.h>

int main ()
{
    float f1 = (3.14 + 1e20) - 1e20;
    float f2 = 3.14 + (1e20 - 1e20);

    printf ("f1 = %f, f2 = %f\n", f1, f2);
}
```
Ariane 5

• Ariane 5 tragedy (June 4, 1996)
  – Exploded 37 seconds after liftoff
  – Satellites worth $500 million

• Why?
  – Computed horizontal velocity as floating-point number
  – Converted to 16-bit integer
    • Careful analysis of Ariane 4 trajectory proved 16-bit is enough
  – Reused a module from 10-year-old software
    • Overflowed for Ariane 5
    • No precise specification for the software
Summary

• IEEE floating point has clear mathematical properties

• Represents numbers of form $M \times 2^E$

• Can reason about operations independent of implementation
  – As if computed with perfect precision and then rounded

• Not the same as real arithmetic
  – Violates associativity / distributivity
  – Makes life difficult for compilers and serious numerical applications programmers