Floating Point
Sept 5, 2002

Topics
- IEEE Floating Point Standard
- Rounding
- Floating Point Operations
- Mathematical properties

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 go fast
- Numerical analysts predominated over hardware types in defining standard

Floating Point Puzzles

For each of the following C expressions, either:
- Argue that it is true for all argument values
- Explain why not true

- \( x \equiv (\text{int})(\text{float}) \ x \)
- \( x \equiv (\text{int})(\text{double}) \ x \)
- \( f \equiv (\text{float})(\text{double}) \ f \)
- \( d \equiv (\text{float}) \ d \)
- \( f \equiv -(-f) \);
- \( 2/3 \equiv 2/3.0 \)
- \( d < 0.0 \Rightarrow (d+2) < 0.0 \)
- \( d > f \Rightarrow -f > -d \)
- \( d \times d = -0.0 \)
- \( (d+f) - d = f \)

Fractional Binary Numbers

Representation
- Bits to right of “binary point” represent fractional powers of 2
- Represents rational number:
\[
\sum_{k=0}^{\infty} b_k 2^k
\]
### Frac. Binary Number 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>10.111₁₂</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… just below 1.0
- \( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \rightarrow 1.0 \)
- Use notation 1.0 - \( \varepsilon \)

### Representable Numbers

**Limitation**
- Can only exactly represent numbers of the form \( x \times 2^e \)
- Other numbers have repeating bit representations

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>0.0101010101₁₂</td>
</tr>
<tr>
<td>1/5</td>
<td>0.0011001100₁₂</td>
</tr>
<tr>
<td>1/10</td>
<td>0.0001100110₁₂</td>
</tr>
</tbody>
</table>

### Floating Point Representation

**Numerical Form**
- \(-1 \times M \times 2^E\)
  - Sign bit \( s \) determines whether number is negative or positive
  - Significant \( M \) normally a fractional value in range \([1.0, 2.0)\).
- Exponent \( E \) weights value by power of two

**Encoding**
- MSB is sign bit
- \( \exp \) field encodes \( E \)
- \( \frac{\text{frac}}{} \) field encodes \( M \)

### Floating Point Precisions

**Encoding**
- MSB is sign bit
- \( \exp \) field encodes \( E \)
- \( \frac{\text{frac}}{} \) field encodes \( M \)

**Sizes**
- Single precision: 8 \( \exp \) bits, 23 \( \frac{\text{frac}}{} \) bits
  - 32 bits total
- Double precision: 11 \( \exp \) bits, 52 \( \frac{\text{frac}}{} \) bits
  - 64 bits total
- Extended precision: 15 \( \exp \) bits, 63 \( \frac{\text{frac}}{} \) bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits
  - 1 bit wasted
“Normalized” Numeric Values

Condition
- exp #000...0 and exp #111...1

Exponent coded as biased value
- $E = \exp - \text{Bias}$
  - $\text{Exp: unsigned value denoted by } \exp$
  - $\text{Bias: bias value}$
  - $\text{Single precision: 127 (Exp: 1...254, } E: -126...127)$
  - $\text{Double precision: 1023 (Exp: 1...2046, } E: -1022...1023)$
  - $\text{in general: } \text{Bias} = 2^{e+1} - 1$, where $e$ is number of exponent bits

Significant coded with implied leading 1
- $M = 1.xxx...x_2$
  - $\text{xxxx...bits of frac}$
  - $\text{Minimum when } 0000.0 \text{ (} M = 1.0)$
  - $\text{Maximum when } 1111.1 \text{ (} M = 2.0 - e)$
  - $\text{Get extra leading bit for "free"}$

Normalized Encoding Example

Value
- Float $F = 15213.0$
- $15213.0 = 1.110110110101_2 = 1.11011011011 \times 2^{13}$

Significant
- $M = 1.110110110101_2$
- frac = $1101101101100000000_2$

Exponent
- $E = 13$
- $\text{Bias} = 127$
- $\exp = 140 = 10001100_2$

Floating Point Representation (Class 02):  
Hex: $4666666666666666$
Binary: $010001100110110110100000000000$
$140: \ 100\ 0110 \ 0$
$15213: \ 1110 \ 1101 \ 1011 \ 01$

Denormalized Values

Condition
- $\exp = 000...0$

Value
- Exponent value $E = -\text{Bias} + 1$
- Significant value $M = 0.000...x_2$
  - $\text{xxxx...bits of frac}$

Cases
- $\exp = 000...0, \text{frac} = 000...0$
  - Represents value 0
  - Note that have distinct values +0 and -0
- $\exp = 000...0, \text{frac} \neq 000...0$
  - Numbers very close to 0.0
  - Lose precision as get smaller
  - "Gradual underflow"

Special Values

Condition
- $\exp = 111...1$

Cases
- $\exp = 111...1, \text{frac} = 000...0$
  - Represents value $\pm \infty$
  - Operation that overflows
  - Both positive and negative
  - E.g., $1.0/0.0 = -1.0/0.0 = +\infty, -1.0/0.0 = -\infty$
- $\exp = 111...1, \text{frac} \neq 000...0$
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., $\sqrt{-1} = \pm \infty$
Summary of Floating Point Real Number Encodings

Tiny Floating Point Example
8-bit Floating Point Representation
- the sign bit is 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

Values Related to the Exponent

Dynamic Range

<table>
<thead>
<tr>
<th>Exp</th>
<th>exp</th>
<th>frac</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0000 000 -6 1/64 (denorm)</td>
<td>0 0000 001 -6 1/8*1/64 = 1/512</td>
<td>closest to zero</td>
<td></td>
</tr>
<tr>
<td>1 0001 000 -5 1/64</td>
<td>0 0000 010 -6 2/8*1/64 = 2/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 0010 001 -4 1/32</td>
<td>0 0000 110 -6 6/8*1/64 = 6/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0011 000 -3 1/16</td>
<td>0 0000 111 -6 7/8*1/64 = 7/512</td>
<td>largest denorm</td>
<td></td>
</tr>
<tr>
<td>4 0100 010 -2 1/8</td>
<td>0 0001 000 -6 8/8*1/64 = 8/512</td>
<td>smallest norm</td>
<td></td>
</tr>
<tr>
<td>5 0101 000 -1 1/4</td>
<td>0 0001 001 -6 9/8*1/64 = 9/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 0110 010 0</td>
<td>0 0001 011 -1 14/8*1/2 = 14/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 0111 100 0</td>
<td>0 0111 000 0 8/8*1 = 1</td>
<td>closest to 1 below</td>
<td></td>
</tr>
<tr>
<td>8 1000 100 0</td>
<td>0 0111 001 0 9/8*1 = 9/8</td>
<td>closest to 1 above</td>
<td></td>
</tr>
<tr>
<td>9 1001 101 0</td>
<td>0 0111 010 0 10/8*1 = 10/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 1010 110 7</td>
<td>0 1110 110 7 14/8*128 = 224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 1011 110 7</td>
<td>0 1110 111 7 15/8*128 = 240</td>
<td>largest norm</td>
<td></td>
</tr>
<tr>
<td>12 1100 111 7</td>
<td>0 1111 000 0 (inf, NaN)</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 3

Notice how the distribution gets denser toward zero.

**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>$2^{-123.25} \times 2^{-126.1023}$</td>
</tr>
<tr>
<td>Single</td>
<td>1.4 X 10^{-35}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>4.9 X 10^{-24}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest Denormalized</td>
<td>00..00 11...11</td>
<td>1.0 X 2^{-126.1023}</td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td>1.18 X 10^{-30}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>2.2 X 10^{-25}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Pos. Normalized</td>
<td>00...01 00...00</td>
<td>1.0 X 2^{-126.1023}</td>
<td></td>
</tr>
<tr>
<td>Just larger than largest denormalized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>01...11 00...00</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Largest Normalized</td>
<td>11...10 11...11</td>
<td>(2.0 - ε) X 2^{127.1023}</td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td>3.4 X 10^{38}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>1.8 X 10^{206}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Special Properties of Encoding**

**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
- What should comparison yield?
- Otherwise OK
  - Denorm vs. normalized
  - Normalized vs. infinity
Floating Point Operations

Conceptual View
- First compute exact result
- Make it fit into desired precision
  - Possibly overflow if exponent too large
  - Possibly round to fit into $\pm \infty$

Rounding Modes (illustrate with $\oplus$ rounding)

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Rounded</th>
<th>Action</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.40</td>
<td>$1.011_2$</td>
<td>$1.1_2$</td>
<td>(1/2—down) 1</td>
<td>1</td>
</tr>
<tr>
<td>1.60</td>
<td>$1.100_2$</td>
<td>$1.0_2$</td>
<td>(1/2—up) 2</td>
<td>1</td>
</tr>
<tr>
<td>1.50</td>
<td>$1.01_2$</td>
<td>$1.0_2$</td>
<td>(1/2—up) 2</td>
<td>1</td>
</tr>
</tbody>
</table>

Note:
1. Round down: rounded result is close to but no greater than true result.
2. Round up: rounded result is close to but no less than true result.

Closing Look at Round-To-Even

Default Rounding Mode
- Hard to get any other kind without dropping into assembly
- All others are statistically biased
  - Sum of set of positive numbers will consistently be over- or underestimated

Applying to Other Decimal Places / Bit Positions
- When exactly halfway between two possible values
  - Round so that least significant digit is even
- E.g., round to nearest hundredth
  - 1.2349999 $\rightarrow$ 1.23
    - (Less than half way)
  - 1.2350001 $\rightarrow$ 1.24
    - (Greater than half way)
  - 1.2350000 $\rightarrow$ 1.24
    - (Half way—round up)
  - 1.2450000 $\rightarrow$ 1.24
    - (Half way—round down)

Rounding Binary Numbers

Binary Fractional Numbers
- “Even” when least significant bit is 0
- Half way when bits to right of rounding position = 100...2

Examples
- Round to nearest 1/4 (2 bits right of binary point)
  - Value | Binary | Rounded Action | Rounded Value
  - 2/3/2 | 10.00011$^0$, 10.00$^1$ | (<1/2—down) 2 | 2
  - 2/3/16 | 10.00110$^0$, 10.01$^1$ | (>1/2—up) 2 | 1/4
  - 2/7/8 | 10.11100$^0$, 11.00$^1$ | (1/2—up) 3 | 1/2
  - 2/5/8 | 10.10100$^0$, 10.10$^1$ | (1/2—down) 2 | 1/2

FP Multiplication

Operands
- $(-1)^s M_1 2^{E_1}$ * $(-1)^t M_2 2^{E_2}$

Exact Result
- $(-1)^{s+t} M 2^{E_1+E_2}$
  - Sign: $s_1 + s_2$
  - Significant $M$: $M_1 * M_2$
  - Exponent $E$: $E_1 + E_2$

Fixing
- If $M \geq 2$, shift $M$ right, increment $E$
- If $E$ out of range, overflow
- Round $M$ to fit precision

Implementation
- Biggest chore is multiplying significands
**Mathematical Properties of FP Add**

**FP Addition**

**Operands**

\[-1\]^{s_1}M_12^{E_1} + \[-1\]^{s_2}M_22^{E_2}

- Assume \(E_1 > E_2\)

**Exact Result**

\[-1\]^{s_1}M_12^{E_1} + \[-1\]^{s_2}M_22^{E_2}

- 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 \(f\)rac precision

*Assume \(E_1 > E_2\)*

**Math. Properties of FP Mult**

**Compare to Commutative Ring**

- Closed under multiplication? \(\text{YES}\)
- But may generate infinity or NaN
- Multiplication Commutative? \(\text{YES}\)
- Multiplication is Associative? \(\text{NO}\)
- Possibility of overflow, inexactness of rounding
- 1 is multiplicative identity? \(\text{YES}\)
- Multiplication distributes over addition? \(\text{NO}\)
- Possibility of overflow, inexactness of rounding

**Monotonicity**

- \(a > b \Rightarrow a\cdot c > b\cdot c\) \(\text{ALMOST}\)
- \(\text{Except for infinities & NaNs}\)

**Floating Point in C**

**C Guarantees Two Levels**

- \text{float} \quad \text{single precision}
- \text{double} \quad \text{double precision}

**Conversions**

- Casting between int, float, and double changes numeric values
- Double of float to int
  - Truncates fractional part
  - Like rounding toward zero
  - Not defined when out of range
    - Generally saturates to TMin or TMax
- int to double
  - Exact conversion, as long as int has \(\leq 53\) bit word size
- int to float
  - Will round according to rounding mode
**Answers to Floating Point Puzzles**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \text{(int)}(\text{float}) \times )</td>
<td>Assume neither ( d ) nor ( f ) is ( \text{NAN} )</td>
</tr>
<tr>
<td>( x = \text{(int)}(\text{double}) \times )</td>
<td>No: 24 bit significand</td>
</tr>
<tr>
<td>( f = \text{(float)}(\text{double}) \times f )</td>
<td>Yes: 53 bit significand</td>
</tr>
<tr>
<td>( d = \text{(float)} \times d )</td>
<td>Yes: increases precision</td>
</tr>
<tr>
<td>( f = -(-f) )</td>
<td>No: loses precision</td>
</tr>
<tr>
<td>( 2/3 = 2/3.0 )</td>
<td>Yes: Just change sign bit</td>
</tr>
<tr>
<td>( d &lt; 0.0 \Rightarrow (d^2) &lt; 0.0 )</td>
<td>No: 2/3 == 0</td>
</tr>
<tr>
<td>( d + f \Rightarrow f &gt; -d )</td>
<td>Yes!</td>
</tr>
<tr>
<td>( d + d \Rightarrow 0.0 )</td>
<td>Yes!</td>
</tr>
<tr>
<td>( d(f)-d \Rightarrow f )</td>
<td>No: Not associative</td>
</tr>
</tbody>
</table>

---

**Ariane 5**

- Exploded 37 seconds after liftoff
- Cargo worth $500 million

**Why**

- Computed horizontal velocity as floating point number
- Converted to 16-bit integer
- Worked OK for Ariane 4
- Overflowed for Ariane 5
- Used same 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 & serious numerical applications programmers