CS429: Computer Organization and Architecture
Floating Point

Warren Hunt, Jr. and Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: August 26, 2014 at 08:53
IEEE Floating Point Standard
Rounding
Floating point operations
Mathematical properties
Floating Point Puzzles

For each of the following C expressions, either:

- argue that it is true for all argument values, or
- explain why it is not true.

```c
int x = ...;
float f = ...;
double d = ...;
```

Assume neither d nor f is NaN.

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f)`
- `2/3 == 2/3.0`
- `d < 0.0`  \(\rightarrow\)  `(d*2) < 0.0)`
- `d > f`  \(\rightarrow\)  `-f < -d`
- `d*d >= 0.0`
- `(d+f)-d == f`
IEEE Standard 754

- Established in 1985 as a uniform standard for floating point arithmetic
- It is supported by all major CPUs.
- Before 1985 there were many idiosyncratic formats.

Driven by Numerical Concerns

- Nice standards for rounding, overflow, underflow
- Hard to make go fast: numerical analysts predominated over hardware types in defining the standard
- Now all (add, subtract, multiply) operations are fast except divide.
The binary number \( b_ib_{i-1}b_2b_1 \ldots b_0b_{-1}b_{-2}b_{-3} \ldots b_{-j} \) represents a particular sum. Each digit is multiplied by a power of two according to the following chart:

<table>
<thead>
<tr>
<th>Bit:</th>
<th>( b_i )</th>
<th>( b_{i-1} )</th>
<th>( \ldots )</th>
<th>( b_2 )</th>
<th>( b_1 )</th>
<th>( b_0 )</th>
<th>( \ldots )</th>
<th>( b_{-1} )</th>
<th>( b_{-2} )</th>
<th>( b_{-3} )</th>
<th>( \ldots )</th>
<th>( b_{-j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight:</td>
<td>( 2^i )</td>
<td>( 2^{i-1} )</td>
<td>( \ldots )</td>
<td>( 4 )</td>
<td>( 2 )</td>
<td>( 1 )</td>
<td>( \ldots )</td>
<td>( 1/2 )</td>
<td>( 1/4 )</td>
<td>( 1/8 )</td>
<td>( \ldots )</td>
<td>( 2^{-j} )</td>
</tr>
</tbody>
</table>

**Representation:**

- Bits to the right of the *binary point* represent fractional powers of 2.
- This represents the 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>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 the form 0.11111...₂ are just below 1.0
  - \(1/2 + 1/4 + 1/8 + \ldots + 1/2^i \rightarrow 1.0\)
  - We use the notation 1.0 − \(\epsilon\).
Limitation

- You can only represent numbers of the form $y + x/2^i$.
- Other fractions 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>
Numerical Form

\[-1^s \times M \times 2^E\]

- Sign bit $s$ determines whether number is negative or positive.
- Significand $M$ is normally a fractional value in the range $[1.0 \ldots 2.0)$.
- Exponent $E$ weights value by power of two.

Encoding

<table>
<thead>
<tr>
<th>$s$</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
</table>

- The most significant bit is the sign bit.
- The exp field encodes $E$.
- The frac field encodes $M$. 
Floating Point Precisions

Encoding

| s | exp | frac |

- The most significant bit is the sign bit.
- The exp field encodes $E$.
- The frac field encodes $M$.

Sizes

- Single precision: 8 exp bits, 23 frac bits, for 32 bits total
- Double precision: 11 exp bits, 52 frac bits, for 64 bits total
- Extended precision: 15 exp bits, 63 frac bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits: an explicit “1” bit appears in the format, except when exp is 0.
Condition: \( \exp \neq 000\ldots 0 \) and \( \exp \neq 111\ldots 1 \)

Exponent is coded as a biased value
\( E = \Exp - \Bias \)
- \( \Exp \): unsigned value denoted by \( \exp \).
- \( \Bias \): Bias value
  - Single precision: 127 (\( \Exp: 1\ldots 254, \; E : -126\ldots 127 \))
  - Double precision: 1023 (\( \Exp: 1\ldots 2046, \; E : -1022\ldots 1023 \))
  - In general: \( \Bias = 2^{\num{\text{exponent\ bits}}} - 1 \), where \( \num{\text{exponent\ bits}} \) is the number of exponent bits.

Significand coded with implied leading 1
\( M = 1.xxx\ldots x_2 \)
- \( xxx\ldots x \): bits of frac
- Minimum when 000\ldots 0 (\( M = 1.0 \))
- Maximum when 111\ldots 1 (\( M = 2.0 - \epsilon \))
- We get the extra leading bit “for free.”
**Value:**

```plaintext
float F = 15213.0;

15231_{10} = 11101101101101_{2} = 1.1101101101101_{2} \times 2^{13}
```

**Significand**

```plaintext
M = 1.1101101101101_{2}
frac = 1101101101101000000000000
```

**Exponent**

```plaintext
E = 13
Bias = 127
Exp = 140 = 10001100
```
Floating Point Representation
Hex: 466DB400
Binary: 0100 0110 0110 1101 1011 0100 0000 0000

140: 100 0110 0
15213: 1110 1101 1011 01
Denormalized Values

**Condition:** \( \exp = 000 \ldots 0 \)

**Value**
- Exponent values: \( E = -\text{Bias} + 1 \) *Why this value?*
- Significand value: \( M = 0.\text{xxx} \ldots x_2 \), where \( \text{xxx} \ldots x \) are the bits of frac.

**Cases**
- \( \exp = 000 \ldots 0 \) and \( \text{frac} = 000 \ldots 0 \)
  - represents values of 0
  - notice that we have distinct \( +0 \) and \( -0 \)
- \( \exp = 000 \ldots 0 \) and \( \text{frac} \neq 000 \ldots 0 \)
  - These are numbers very close to 0.0
  - Lose precision as they get smaller
  - Experience “gradual underflow”
**Condition:** $\exp = 111\ldots1$

**Cases**

- $\exp = 111\ldots1$ and $\frac{\text{frac}}{} = 000\ldots0$
  - Represents value of infinity ($\infty$)
  - Result returned for operations that overflow
  - Sign indicates positive or negative
  - E.g., $1.0/0.0 = -1.0/ - 0.0 = +\infty$, $1.0/ - 0.0 = -\infty$

- $\exp = 111\ldots1$ and $\frac{\text{frac}}{} \neq 000\ldots0$
  - Not-a-Number (NaN)
  - Represents the case when no numeric value can be determined
  - E.g., $\sqrt{-1}$, $\infty - \infty$
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.

This has the general form of the IEEE Format

- Has both normalized and denormalized values.
- Has representations of 0, NaN, infinity.
<table>
<thead>
<tr>
<th>Exp</th>
<th>exp</th>
<th>E</th>
<th>$2^E$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>-6</td>
<td>1/64</td>
<td>(denorms)</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>-6</td>
<td>1/64</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>-5</td>
<td>1/32</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>-4</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>-3</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>-2</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>-1</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>+1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>+2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>+3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>+4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>+5</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>+6</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>+7</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>n/a</td>
<td></td>
<td>(inf, NaN)</td>
</tr>
</tbody>
</table>
### Dynamic Range

| s | exp | frac | E | Value
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>$1/8 \times 1/64 = 1/512$ closest to zero</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>010</td>
<td>-6</td>
<td>$2/8 \times 1/64 = 2/512$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>110</td>
<td>-6</td>
<td>$6/8 \times 1/64 = 6/512$</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>111</td>
<td>-6</td>
<td>$7/8 \times 1/64 = 7/512$ largest denorm</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>000</td>
<td>-6</td>
<td>$8/8 \times 1/64 = 8/512$ smallest norm</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>001</td>
<td>-6</td>
<td>$9/8 \times 1/64 = 9/512$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>110</td>
<td>-1</td>
<td>$14/8 \times 1/2 = 14/16$ closest to 1 below</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>111</td>
<td>-1</td>
<td>$15/8 \times 1/2 = 15/16$</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>000</td>
<td>0</td>
<td>$8/8 \times 1 = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>001</td>
<td>0</td>
<td>$9/8 \times 1 = 9/8$ closest to 1 above</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>010</td>
<td>0</td>
<td>$10/8 \times 1 = 10/8$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>110</td>
<td>7</td>
<td>$14/8 \times 128 = 224$</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>111</td>
<td>7</td>
<td>$15/8 \times 128 = 240$ largest norm</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
<td>000</td>
<td>n/a</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

**Denormalized numbers**

**Normalized numbers**
## Interesting FP 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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Single $\approx 1.4 \times 10^{-45}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Double $\approx 4.9 \times 10^{-324}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest Denorm.</td>
<td>00...00</td>
<td>11...11</td>
<td>$(1.0 - \epsilon) \times 2{-126,-1022}$</td>
</tr>
<tr>
<td></td>
<td>• Single $\approx 1.18 \times 10^{-38}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Double $\approx 2.2 \times 10^{-308}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Pos. Norm.</td>
<td>00...01</td>
<td>00...01</td>
<td>$1.0 \times 2{-126,-1022}$</td>
</tr>
<tr>
<td></td>
<td>• Just larger than the largest denormalized.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>01...11</td>
<td>00...00</td>
<td>1.0</td>
</tr>
<tr>
<td>Largest Norm.</td>
<td>11...11</td>
<td>11...11</td>
<td>$(2.0 - \epsilon) \times 2^{127,1023}$</td>
</tr>
<tr>
<td></td>
<td>• Single $\approx 3.4 \times 10^{38}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Double $\approx 1.8 \times 10^{308}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Special Properties of Encoding

FP Zero is the Same as Integer Zero: All bits are 0.

Can (Almost) Use Unsigned Integer Comparison

- Must first compare sign bits.
- Must consider $-0 = 0$.
- NaNs are problematic:
  - Will be greater than any other values.
  - What should the comparison yield?
- Otherwise, it’s OK.
  - Denorm vs. normalized works.
  - Normalized vs. infinity works.
Conceptual View

- First compute the exact result.
- Make it fit into the desired precision.
  - Possibly overflows if exponent is too large.
  - Possibly round to fit into frac.

<table>
<thead>
<tr>
<th>Rounding Modes (illustrated with $ rounding)</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>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>

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.
Closer Look at Round to Even

Default Rounding Mode

- Hard to get any other kind without dropping into assembly.
- All others are statistically biased; the sum of a set of integers will consistently be under- or over-estimated.

Applying to Other Decimal Places / Bit Positions

When exactly halfway between two possible values, round so that the least significant digit is even.

E.g., round to the nearest hundredth:

- 1.2349999 1.23 Less than half way
- 1.2350001 1.24 Greater than half way
- 1.2350000 1.24 Half way, round up
- 1.2450000 1.24 Half way, round down
Binary Fractional Numbers

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

Examples

E.g., Round to nearest 1/4 (2 bits to right of binary point).

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Rounded</th>
<th>Action</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 2/32</td>
<td>10.000112</td>
<td>10.00</td>
<td>(&lt; 1/2: down)</td>
<td>2</td>
</tr>
<tr>
<td>2 3/16</td>
<td>10.001102</td>
<td>10.01</td>
<td>(&gt; 1/2: down)</td>
<td>2 1/4</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.111002</td>
<td>11.00</td>
<td>(1/2: up)</td>
<td>3</td>
</tr>
<tr>
<td>2 5/8</td>
<td>10.101002</td>
<td>10.10</td>
<td>(1/2: down)</td>
<td>2 1/2</td>
</tr>
</tbody>
</table>
**FP Multiplication**

**Operands:** \((-1)^{S_1} \times M_1 \times 2^{E_1}, (-1)^{S_2} \times M_2 \times 2^{E_2}\)

**Exact Result:** \((-1)^{S} \times M \times 2^{E}\)

- **Sign S:** \(S_1\) xor \(S_2\)
- **Significant M:** \(M_1 \times M_2\)
- **Exponent E:** \(E_1 + E_2\)

**Fixing**

- If \(M \geq 2\), shift \(M\) right, increment \(E\)
- \(E\) is out of range, overflow
- Round \(M\) to fit frac precision

**Implementation**

Biggest chore is multiplying significands.
FP Addition

**Operands:** \((-1)^{S_1} \times M_1 \times 2^{E_1}, (-1)^{S_2} \times M_2 \times 2^{E_2}\)

Assume \(E_1 > E_2\)

**Exact Result:** \((-1)^S \times M \times 2^E\)

- Sign \(S\), Significant \(M\); result of signed align and 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 \(k\)
- if \(E\) is out of range, overflow
- Round \(M\) to fit frac precision
Mathematical Properties of FP Add

**Compare to those of Abelian Group**

- Closed under addition? Yes, but may generate infinity or NaN.
- Commutative? Yes.
- Associative? No, because of overflow and inexactness of rounding.
- O is additive identity? Yes.
- Every element has additive inverse? Almost, except for infinities and NaNs.

**Monotonicity**

- $a \geq b \implies a + c \geq b + c$? Almost, except for infinities and NaNs.
Compare to those of Commutative Ring

- Closed under multiplication? Yes, but may generate infinity or NaN.
- Multiplication Commutative? Yes.
- Multiplication is Associative? No, because of possible overflow and inexactness of rounding.
- 1 is multiplicative identity? Yes.
- Multiplication distributes over addition? No, because of possible overflow and inexactness of rounding.

Monotonicity

- $a \geq b \land c \geq 0 \implies a \times c \geq b \times c$? Almost, except for infinities and NaNs.
Floating Point in C

C guarantees two levels

- float: single precision
- double: double precision

Conversions

- Casting among int, float, and double changes numeric values
- Double or 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 ≤ 53-bit word size
- int to float: will round according to rounding mode.
int x = ...;
float f = ...;
double d = ...;

Assume neither d nor f is NaN.

- `x == (int)(float) x`: No, 24 bit significand
- `x == (int)(double) x`: Yes, 53 bit significand
- `f == (float)(double) f`: Yes, increases precision
- `d == (float) d`: No, loses precision
- `f == -(-f)`: Yes, just change sign bit
- `2/3 == 2/3.0`: No, 2/3 == 0
- `d < 0.0 → ((d*2) < 0.0)`: Yes
- `d > f → -f < -d`: Yes
- `d*d >= 0.0`: Yes
- `(d+f)-d == f`: No, not associative
On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. The cause of the failure was a software error in the inertial reference system. Specifically a 64-bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16-bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16-bit signed integer, and thus the conversion failed.
IEEE Floating Point has Clear Mathematical Properties

- Represents numbers of the 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 and distributivity.
  - Makes life difficult for compilers and serious numerical application programmers.