Lecture 10: Floating Point

Spring 2020
Jason Tang
Topics

• Representing floating point numbers

• Floating point arithmetic

• Floating point accuracy
Floating Point Numbers

• So far, all arithmetic has involved numbers in the set of $\mathbb{N}$ and $\mathbb{Z}$

• But what about:

  • Very large integers, greater than $2^n$, given only $n$ bits
  
  • Very small numbers, like 0.000123
  
  • Rational numbers in $\mathbb{Q}$ but not in $\mathbb{Z}$, like $\frac{2}{3}$
  
  • Irrational and transcendental numbers, like $\sqrt{2}$ and $\pi$
Floating Point Numbers

• Issues:
  • Arithmetic (addition, subtraction, multiplication, division)
  • Representation, normal form
  • Range and precision
  • Rounding
  • Illegal operations (divide by zero, overflow, underflow)
Normal Form

• The value $411_{10}$ could be stored as $4110 \times 10^{-1}$, $411 \times 10^{0}$, $41.1 \times 10^{1}$, $4.11 \times 10^{2}$, etc.

• In scientific notation, values are usually normalized such that one non-zero digit to left of decimal point: $4.11 \times 10^{2}$

• Computers numbers use base-2: $1.01 \times 2^{1101}$

• Because the digit to the left of the decimal point ("binary point"?) will always be a 1, that digit is omitted when storing the number

  • In this example, the stored significand will be 01, and exponent is 1101
Floating Point Representation

- Size of exponent determines the range of represented numbers

- Accuracy of representation depends on size of significand
  - Trade-off between accuracy and range

- **Overflow**: required positive exponent too large to fit in given number of bits for exponent

- **Underflow**: required negative exponent too large to fit in given number of bits for exponent
# IEEE 754 Standard Representation

- Same representation used in nearly all computers since mid-1980s
- In general, a floating point number $= (-1)^\text{Sign} \times [1].\text{Significand} \times 2^{\text{Exponent}}$
- Exponent is **biased**, instead of Two’s Complement
  - For single precision, actual magnitude is $2^{(\text{Exponent} - 127)}$

<table>
<thead>
<tr>
<th>Type</th>
<th>Sign</th>
<th>Exponent</th>
<th>Significand</th>
<th>Total Bits</th>
<th>Exponent Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>Single</td>
<td>1</td>
<td>8</td>
<td>23</td>
<td>32</td>
<td>127</td>
</tr>
<tr>
<td>Double</td>
<td>1</td>
<td>11</td>
<td>52</td>
<td>64</td>
<td>1023</td>
</tr>
<tr>
<td>Quad</td>
<td>1</td>
<td>15</td>
<td>112</td>
<td>128</td>
<td>16383</td>
</tr>
</tbody>
</table>
Single-Precision Example

• Convert $-12.625_{10}$ to single precision IEEE-754 format

• Step 1: Convert to target base 2: $-12.625_{10} \rightarrow -1100.101_{2}$

• Step 2: Normalize: $-1100.101_{2} \rightarrow -[1].100101_{2} \times 2^{3}$

• Step 3: Add bias to exponent: $3 \rightarrow 130$

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11000000101010100000000000000000000000000</td>
<td>Leading 1 of significand is implied</td>
</tr>
</tbody>
</table>
Single and Double Precision Example

• Convert \(-0.75_{10}\) to single and double precision IEEE-754 format

\[-0.75_{10} \rightarrow (-3/4)_{10} \rightarrow (-3/2^2)_{10} \rightarrow -11_2 \times 2^{-2} \rightarrow [-1].1_2 \times 2^{-1}\]

• Single Precision:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand (23 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 1 1 1 1 0</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

• Double Precision:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand (upper 20 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 1 1 1 1 1 1 1 1 0</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Significand (lower 32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>


Denormalized Numbers

- Smallest single precision normalized number is 1.000 0000 0000 0000 0000 0001 × 2^{-126}

- Use denormalized (or subnormal) numbers to store values between 0 and above number
  - Denormal numbers have a leading implicit 0 instead of 1
  - Needed when subtracting two values, where the difference is not 0 but close to it
  - Denormalized are allowed to degrade in significance until it becomes 0 (gradual underflow)
Special Values

• Some bit patterns are special:

• Negative zero

• ±Infinity, for overflows

• **Not a Number** (NaN), for invalid operations like 0/0, \( \infty - \infty \), or \( \sqrt{-1} \)

<table>
<thead>
<tr>
<th></th>
<th>Single Precision</th>
<th>Double Precision</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>Significand</td>
<td>Exponent</td>
<td>Significand</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Nonzero</td>
<td>0</td>
<td>Nonzero</td>
</tr>
<tr>
<td>1-254</td>
<td>Anything</td>
<td>1-2046</td>
<td>Anything</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>2047</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>Nonzero</td>
<td>2047</td>
<td>Nonzero</td>
</tr>
</tbody>
</table>

0 or -0

± denormalized

± normal floating point

± infinity

NaN
Normal and Denormal Exponent

• A normal number stores its exponent $e$ as $e + \text{bias}$

• Single-precision floating point has a bias of 127

• If a single-precision’s stored exponent bits are 0000…1, then its value is $\pm[1].XXX…X \times 2^{-126}$

• When exponent bits are all zeroes, the number is denormal

• Implied exponent is defined as $2^{-(\text{bias} - 1)}$

• Largest denormal single-precision value is $\pm[0].1111…1 \times 2^{-126}$
Floating Point Arithmetic

• Floating point arithmetic differs from integer arithmetic in that exponents are handled as well as the significands

  • For addition and subtraction, exponents of operands must be equal

  • Significands are then added/subtracted, and then result is normalized

• Example: \([1].101 \times 2^3 + [1].111 \times 2^4\)

  • Adjust exponents to equal larger exponent: \(0.1101 \times 2^4 + 1.1110 \times 2^4\)

  • Sum is thus \(10.1011 \times 2^4 \rightarrow [1].01011 \times 2^5\)
Floating Point Addition / Subtraction

• Compute $E = A_{\text{exp}} - B_{\text{exp}}$

• Right-shift $A_{\text{sig}}$ to form $A_{\text{sig}} \times 2^E$

• Compute $R = A_{\text{sig}} + B_{\text{sig}}$

• Left shift $R$ and decrement $E$, or right shift $R$ and increment $E$, until MSB of $R$ is implicit 1 (normalized form)

  • If cannot left shift enough, then keep $R$ as denormalized
Floating Point Addition Example

- Calculate $0.5_{10} - 0.4375_{10}$, with only 4 bits of precision
  
  - $0.5_{10} \rightarrow (1/2)_{10} \rightarrow 0.1_2 \times 2^0 \rightarrow [1].000_2 \times 2^{-1}$
  
  - $0.4375_{10} \rightarrow (1/4)_{10} + (1/8)_{10} + (1/16)_{10} \rightarrow 0.0111_2 \times 2^0 \rightarrow [1].110_2 \times 2^{-2}$
  
  - Shift operand with lesser exponent: $1.110_2 \times 2^{-2} \rightarrow 0.111_2 \times 2^{-1}$

  - Subtract significands: $1.000_2 \times 2^{-1} - 0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

  - Normalize: $0.001_2 \times 2^{-1} \rightarrow [1].000_2 \times 2^{-4} = (2^{-4})_{10} \rightarrow (1/16)_{10} \rightarrow 0.0625_{10}$

  - No overflow/underflow, because exponent is between -126 and +127
Floating Addition Hardware
Floating Point Multiplication and Division

• For multiplication and division, the sign, exponent, and significand are computed separately
  
  • Same signs → positive result, different signs → negative result
  
  • Exponents calculated by adding/subtracting exponents
  
  • Significands multiplied/divided, and then normalized
Floating Point Multiplication Example

- Calculate $0.5_{10} \times -0.4375_{10}$, with only 4 bits of precision
  
  - $[1].0002 \times 2^{-1} \times [1].1102 \times 2^{-2}$

- Sign bits differ, so result is negative (sign bit = 1)

- Add exponents, **without bias**: $(-1) + (-2) = -3$

- Multiply significands:

- Keep result to 4 precision bits: $1.1102 \times 2^{-3}$

- Normalize results: $-[1].1102 \times 2^{-3}$
Floating Point Multiplication

- Compute \( E = A_{\text{exp}} + B_{\text{exp}} - \text{Bias} \)
- Compute \( S = A_{\text{sig}} \times B_{\text{sig}} \)
- Left shift \( S \) and decrement \( E \), or right shift \( S \) and increment \( E \), until MSB of \( S \) is implicit 1 (normalized form)
  - If cannot left shift enough, then keep \( S \) as denormalized
- Round \( S \) to specified size
- Calculate sign of product
Accuracy

• Floating-point numbers are approximations of a value in $\mathbb{R}$

  • Example: $\pi$ stored as a single-precision floating point is $[1].10010010000111111011011_2 \times 2^1$

  • This floating point value is exactly 3.1415927410125732421875

  • Truer value is 3.1415926535897932384626…

  • Floating point value is accurate to only 8 decimal digits

• Hardware needs to round accurately after arithmetic operations

http://www.exploringbinary.com/pi-and-e-in-binary/
Rounding Errors

- **Unit in the last place (ulp)**: number of bits in error in the LSB of significand between actual number and number that can be represented.

- Example: store a floating point number in base 10, maximum 3 significand digits
  
  - $3.14 \times 10^1$ and $3.15 \times 10^1$ are valid, but $3.145 \times 10^1$ could not be stored
  
  - ULP is thus 0.01

- If storing $\pi$, and then rounding to nearest (i.e., $3.14 \times 10^1$), the rounding error is 0.0015926...

- If rounding to nearest, then maximum rounding error 0.005, or 0.5 of a ULP

https://matthew-brett.github.io/teaching/floating_error.html
Accurate Arithmetic

• When adding and subtracting, append extra guard digits to LSB

• When rounding to nearest even, use guard bits to determine to round up or down

• Example: $2.34_{10} \times 10^0 + 2.56_{10} \times 10^{-2}$, with and without guard bits, maximum 3 significand digits

\[
\begin{array}{c}
2.34 \quad 4 \\
+ 0.0 \quad 2 \\
\hline
2.36
\end{array}
\quad
\begin{array}{c}
2.34 \quad 0 \quad 0 \\
+ 0.0 \quad 2 \quad 5 \quad 6 \\
\hline
2.36 \quad 5 \quad 6
\end{array}
\rightarrow \text{rounded to } 2.37 \times 10^0
IEEE 754 Rounding Modes

- Always round up (towards $+\infty$)

- Always round down (towards $-\infty$)

- Truncate

- Round to nearest even (RNE)
  - Most commonly used, including on IRS tax forms
  - If LSB is 1, then round up (resulting in a LSB of 0); otherwise round down (again resulting in a LSB of 0)