Floating Point
C Bitwise Operations...

- We have the boolean operations
  - || boolean or
  - && boolean and

- We also have bitwise operations
  - Treat the data as raw bits and apply them on a bit-by-bit basis
  - | bitwise or, 0b0011 | 0b0101 = 0b0111
  - & bitwise and, 0b0011 & 0b0101 = 0b0001
  - ^ bitwise exclusive or, 0b0011 ^ 0b0101 = 0b0110
And bit shift operations
(Example using 5 bit values)

- \( a << b \): Shift the value in \( a \) to the left by \( b \) bits, shifting in 0
  - Equivalent to multiplying by \( 2^b \)
  - \( 0b00101 << 2 = 0b10100 \)
  - Bits off the left are just dropped
    - \( 0b10010 << 2 = 0b01000 \)

- \( a >> b \): Shift the value in \( a \) to the right by \( b \) bits
  - If \( a \) is signed, we sign extend (copy the MSB)
    - \( 0b10100 >> 2 = 0b11101 \)
    - \( 0b00100 >> 2 = 0b00001 \)
  - If \( a \) is unsigned, we zero extend
    - \( 0b10100 >> 2 = 0b00101 \)
  - Not quite the same as dividing by \( 2^b \) due to how rounding works
Today, we’ll be learning about a standardized format for representing floating point numbers in computers.

IEEE (Institute of Electronics and Electrical Engineers)
- Standardizes methods for how we do things in computing

IEEE-754
- Established in 1985 to standardize how we represent floating point numbers in binary
- Most recent update was published in 2019
Goals for IEEE 754 Floating-Point Standard

• Standard arithmetic for all computers
  • Important because computer representation of real numbers is approximate. Want same results on all computers.

• Keep as much precision as possible

• Help programmer with errors in real arithmetic
  • $+\infty$, $-\infty$, Not-A-Number (NaN), exponent overflow, exponent underflow, +/- zero

• Keep encoding that is somewhat compatible with two’s complement
  • E.g., +0 in Fl. Pt. is 0 in two’s complement
  • Make it possible to sort without needing to do floating-point comparisons
Scientific Notation

- In decimal, we use scientific notation to shorten the number of digits that numbers take up

  \[ 3.0 \times 10^8 \text{ m/s} \]

  \[ 6.022 \times 10^{23} \text{ mol}^{-1} \]
Scientific Notation (Normalized Form)

9.2318 \times 10^5

- Significand/Mantissa
- Exponent
- Decimal Point
- Base
Representing Fractions in Binary

\[
\begin{align*}
1 & 0 & 1 & 1 & 0 & . & 1 & 0 & 1 \\
2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3}
\end{align*}
\]

\[
\begin{align*}
2^4 + 2^2 + 2^1 & + 2^{-1} + 2^{-3} \\
21 & .625
\end{align*}
\]

\[
21.625
\]
Binary in Normalized Form

\[ 1.0101 \times 2^6 \]

- Significand/Mantissa
- Binary Point
- Base
- Exponent
Binary in Normalized Form Example

Convert 0b011010100 to normalized format.

1.10101 \times 2^7
Components of Floating Point Numbers

\[ +1.0101 \times 2^{6} \]
Floating point diagram (32-bit)

- **S**: Sign bit (1 bit)
- **Exponent**: 8 bits
- **Mantissa**: 23 bits
Sign

- 0 means positive
- 1 means negative
Mantissa

- In normalized form, there must be one non-zero number to the left of the point
  - In binary, the only non-zero number is 1, which means that any binary number written in normalized format will have a 1 to the left of the point (except 0)
  - We can save room by not storing this 1!
- Pad with zeros to the right

\[1.010110 \times 2^4\]

010110 0000000000000000
Exponent

- Exponent is written in biased notation so that the smallest number is written as all zeros.
- The range of the exponent is $[-126, 127]$.
- The exponent is biased by adding 127 to get the number into the range $[1, 254]$
  - 0 and 255 have special meanings
Exponent Review of Bias Notation

Original Number Line

Biased Number Line

A

A-N = 0

B

B-N
Confusion over bias notation

- There are different notations with bias encoding
- It’s not about memorizing a formula, I just gave one because I know some people prefer that
- It’s important to think about the direction in which we are trying to shift the number line
  - If we are trying to shift the number line to the right, then we should be increasing the lower and upper bounds
  - If we are trying to shift the number line to the left, then we should be decreasing the lower and upper bounds
Exponent

Why do we use bias notation?

- Comparison is a common operation (<, >, etc)
- It’s really easy to perform comparisons on biased values because you can just perform an unsigned comparison
Exponent

• Bias formula: \( N = -(2^{n-1} - 1) \)

• For IEEE-754 32-bit floating point numbers, there are 8 exponent bits
  • Bias = \(-(2^{8-1} - 1) = -127\)
Floating Point

\((-1)^S \times 1.mantissa \times 2^{\text{exponent}-127}\)
Floating Point Examples

Convert the following floating point number to decimal
\[ 0b1100000011110000000000000000000000 \]

\[
\begin{array}{c|c|c}
1 & 10000001 & 11100000000000000000000000 \\
\end{array}
\]

\[
(-1)^S \times 1.\text{mantissa} \times 2^{\text{exponent-127}}
\]

\[
(-1)^1 \times 1.111 \times 2^{129-127}
\]

\[-1.111 \times 2^2\]

\[-111.1\]

\[-7.5\]
Floating Point Examples

Convert 123.4375 to IEEE-754 32-bit notation

123 = 64 + 32 + 16 + 8 + 2 + 1  
0.4375 = 1/4 + 1/8 + 1/16

1111011  
0111

0.1110110111 x 2^6

Sign = 0  Exponent = 6 + 127 = 133  Mantissa = 1101110111

0 | 10000101 | 1110110111000000000000000
Floating Point Tool

- [https://www.h-schmidt.net/FloatConverter/IEEE754.html](https://www.h-schmidt.net/FloatConverter/IEEE754.html)
Rounding

• Rounding can occur
  • During a calculation
  • During conversion
    • Double precision -> single precision value
    • Floating point -> integer
Rounding Modes

- **Round to Nearest** – round to nearest number; if the number falls midway it is rounded to the nearest value with an even (zero) least significant bit, which means it is rounded up 50% of the time
  - $2.4 \rightarrow 2, \quad 2.5 \rightarrow 2$
  - $-2.6 \rightarrow -3, \quad -3.5 \rightarrow -4$

- **Round toward 0** (truncate)
  - $2.001 \rightarrow 2$
  - $-2.999 \rightarrow -2$

- **Round toward $+\infty$**
  - $2.001 \rightarrow 3$
  - $-2.999 \rightarrow -2$

- **Round toward $-\infty$**
  - $1.999 \rightarrow 1$
  - $-1.001 \rightarrow -2$
How to Represent 0?

- Sign = 0 or 1
- Exponent = all zeros
- Mantissa = all zeros
## Floating Point Chart

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Number</td>
<td>1-254</td>
<td>Anything</td>
</tr>
<tr>
<td>Zero</td>
<td>All zeros</td>
<td>All zeros</td>
</tr>
</tbody>
</table>
How to Represent Infinity?

- Sign = 0 or 1 (corresponds to if its positive or negative infinity)
- Exponent = all ones
- Mantissa = all zeros
## Floating Point Chart

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Number</td>
<td>1-254</td>
<td>Anything</td>
</tr>
<tr>
<td>Zero</td>
<td>All zeros</td>
<td>All zeros</td>
</tr>
<tr>
<td>Infinity</td>
<td>All ones (255)</td>
<td>All zeros</td>
</tr>
</tbody>
</table>
NaN (Not A Number)

- What happens if I take the square root of a negative number or divide by zero?
  - The result not representable or is undefined in computing systems

- Any operation that is not representable or is undefined is encoded as NaN (Not A Number)
What happens to NaN values?

• Usually, NaN values are propagated through arithmetic operations to allow the user to see that some error occurred during the calculation that resulted in a NaN somewhere along the way.

• There are a couple of exceptions. We don’t cover those in this class.
Encoding NaN in IEEE-754

- Sign = 0 or 1
- Exponent = all ones
- Mantissa = nonzero
  - Allows for the definition of multiple distinct NaN values
# Floating Point Chart

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Number</td>
<td>1-254</td>
<td>Anything</td>
</tr>
<tr>
<td>Zero</td>
<td>All zeros</td>
<td>All zeros</td>
</tr>
<tr>
<td>Infinity</td>
<td>All ones (255)</td>
<td>All zeros</td>
</tr>
<tr>
<td>NaN</td>
<td>All ones (255)</td>
<td>Nonzero</td>
</tr>
</tbody>
</table>
Range of Floating Point Values

- What is the smallest positive number that we can represent?

\[ (-1)^s \times 1.mantissa \times 2^{exponent-127} \]

\[ (-1)^0 \times 1. \ 00000000000000000000000 \times 2^{1-127} \]

\[ 1 \times 2^{-126} \]

\[ 2^{-126} \]
Range of Floating Point Values

- What is the largest positive number that we can represent?

\[ 0 \mid 11111110 \mid 11111111111111111111111 \]

\[ (-1)^S \times 1.\text{mantissa} \times 2^{\text{exponent}-127} \]

\[ (-1)^0 \times 1.11111111111111111111111 \times 2^{254-127} \]

\[ 11111111111111111111111 = 2^{-1} + 2^{-2} + \ldots + 2^{-23} \]

\[ \sum_{i=0}^{n-1} 2^i = 2^n - 1 \]

\[ 2^{-23}(2^{22} + 2^{21} + \ldots + 1) \]

\[ 2^{-23}(2^{23}-1) = 1 - 2^{-23} \]

Implicit 1

\[ 1 + 1 - 2^{-23} \]

\[ 2 - 2^{-23} \]

\[ (2 - 2^{-23}) \times 2^{127} \]
Range of Floating Point Values

- Positive Range
  - \([2^{-126}, (2 - 2^{-23}) \times 2^{127}]\)

- Negative Range
  - The only thing that’s different is the sign bit, so the range is the same
  - \([- (2 - 2^{-23}) \times 2^{127}, -2^{-126}]\)
Range of Floating Point Values

- **Overflow** = When the magnitude of the value is too large to represent (blue regions)
- **Underflow** = When the magnitude of the value is too small to represent (red region)

\[-(2 - 2^{-23}) \times 2^{127} \quad -1 \quad -2^{-126} \quad 0 \quad 2^{-126} \quad 1 \quad (2 - 2^{-23}) \times 2^{127}\]
Pause
Floating Point Step Size

- We cannot represent every value between $[2^{-126}, (2 - 2^{-23}) \times 2^{127}]$ because we have a limited number of bits.
- There are small gaps in the numbers that we can represent.
Floating Point Step Size

- What’s the next smallest number greater than 2 that we can represent?

2

\((-1)^0 \times 1.0 \times 2^1\)

Exponent = 1 + 127 = 128

\((-1)^0 \times 1.000000000000000000000000000000001 \times 2^{128-127}\)

\((1+2^{-23}) \times 2\)

\(2+2^{-22}\)
Floating Point Step Size

- What’s the next smallest number greater than 4 that we can represent?

\[ (-1)^0 \times 1.0 \times 2^2 \]

Exponent = 2 + 127 = 129

\[ 0 \mid 10000001 \mid 00000000000000000000000 \]

\[ (-1)^0 \times 1.00000000000000000000000 \times 2^{129-127} \]

\[ (1+2^{-23}) \times 2^2 \]

4 + 2^{-21}
Floating Point Step Size

(-1)^s \times 1.\text{mantissa} \times 2^{\text{exponent-127}}

- If \( x \) is the biased exponent and \( y \) is the significand

- How do we write our current number in terms of \( x \) and \( y \)?
  - \((1 + y) \times 2^{(x-127)}\)

- How do we write the next number in terms of \( x \) and \( y \)?
  - \((1 + y + 2^{-23}) \times 2^{(x-127)}\)

- Step-size = next_num - curr_num
  - \((1 + y + 2^{-23}) \times 2^{(x-127)} - (1 + y) \times 2^{(x-127)}\)
  - \(2^{-23} \times 2^{(x-127)}\)
  - \(2^{(x-150)}\)
Floating Point Step Size

- Step size = $2^{(x-150)}$
- The step size increases by a factor of 2 for every time the exponent increases by 1
Floating Point Step Size

- The gap between 0 and the smallest positive number is $2^{-126}$
- What is the gap between the smallest positive number and the next smallest positive number is
  - $2(x-150)$
  - $2(1-150)$
  - $2^{-149}$
- There is a larger gap between 0 and the smallest positive number due to the requirement of normalization with an implicit leading one
- Many calculations have values that fall near zero, so let’s find a way to represent more values near zero
## Floating Point Chart

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Number</td>
<td>1-254</td>
<td>Anything</td>
</tr>
<tr>
<td>Zero</td>
<td>All zeros</td>
<td>All zeros</td>
</tr>
<tr>
<td>Infinity</td>
<td>All ones (255)</td>
<td>All zeros</td>
</tr>
<tr>
<td>NaN</td>
<td>All ones (255)</td>
<td>Nonzero</td>
</tr>
<tr>
<td>???</td>
<td>All zeros</td>
<td>Nonzero</td>
</tr>
</tbody>
</table>
Denormalized Numbers

• **Sign**
  • Can be positive (0) or negative (1)

• **Exponent**
  • The exponent field is set to all zeros to encode the denormalized number

• **Significand**
  • We want to have an implicit leading 0 in order to be able to encode smaller values
Denormalized Numbers

\[
\begin{align*}
\text{Normalized} & \quad (-1)^S \times 1.\text{mantissa} \times 2^{\text{exponent}-127} \\
\text{Denormalized} & \quad (-1)^S \times 0.\text{mantissa} \times 2^{-126}
\end{align*}
\]

Exponent = 0 and we need to shift the binary point over by 1 to get an implicit leading 0
Denorm Range

What is the smallest positive denormalized number that can be represented?

\[ (-1)^S \times 0.\text{mantissa} \times 2^{-126} \]

\[ (-1)^0 \times 0.0000000000000000000001 \times 2^{-126} \]

\[ 2^{-23} \times 2^{-126} \]

\[ 2^{-149} \]

What is the largest positive denormalized number that can be represented?

\[ (-1)^S \times 0.\text{mantissa} \times 2^{-126} \]

\[ (-1)^0 \times 0.1111111111111111111111 \times 2^{-126} \]

\[ (1 - 2^{-23}) \times 2^{-126} \]

\[ 2^{-126} - 2^{-149} \]
Denorm Step Size

\((-1)^s \times 0.\text{mantissa} \times 2^{-126}\)

- If \(y\) is the significand
- How do we write our current number in terms of \(y\)?
  - \(y \times 2^{-126}\)
- How do we write the next number in terms of \(y\)?
  - \((y + 2^{-23}) \times 2^{-126}\)
- Step-size = next_num - curr_num
  - \((y + 2^{-23}) \times 2^{-126} - y \times 2^{-126}\)
  - \(2^{-149}\)
- The step size is the same for all denorm values because they all have the same exponent
## Floating Point Chart

<table>
<thead>
<tr>
<th>Type</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Number</td>
<td>1-254</td>
<td>Anything</td>
</tr>
<tr>
<td>Zero</td>
<td>All zeros</td>
<td>All zeros</td>
</tr>
<tr>
<td>Infinity</td>
<td>All ones (255)</td>
<td>All zeros</td>
</tr>
<tr>
<td>NaN</td>
<td>All ones (255)</td>
<td>Nonzero</td>
</tr>
<tr>
<td>Denorm</td>
<td>All zeros</td>
<td>Nonzero</td>
</tr>
</tbody>
</table>
Denorm Examples

Convert the following IEEE-754 floating point number to decimal

1 | 00000000 | 110100000000000000000000

Exponent is 0, mantissa is nonzero => denorm

\((-1)^S \times 0.\text{mantissa} \times 2^{-126}\)

\((-1)^1 \times 0.1101 \times 2^{-126}\)

\(1/2 + 1/4 + 1/16 = 0.8125\)

\(-0.8125 \times 2^{-126}\)

\(-9.55 \times 10^{-39}\)
Denorm Examples

Write $1.5_{10} \times 2^{-129}$ in IEEE-754 Format

Put in normalized binary form

$1.1_2 \times 2^{-129}$

Exponent is too big for normalized

Put in denorm form

$0.0011 \times 2^{-126}$

$(-1)^S \times 0.\text{mantissa} \times 2^{-126}$

0 | 00000000 | 00110000000000000000000000000000
Floating Point Associativity

- **Associativity**
  - \((X + Y) + Z = X + (Y + Z)\)

- Because of rounding errors, you can find Big and Small numbers such that:
  - \((\text{Small} + \text{Big}) + \text{Big} \neq \text{Small} + (\text{Big} + \text{Big})\)

- Ex: \(x = -1.5 \times 10^{38}, y = 1.5 \times 10^{38}, z = 1.0\)

  \[
  \begin{align*}
  x + (y + z) & \quad \text{(x + y) + z} \\
  -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0) & \quad (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0 \\
  -1.5 \times 10^{38} + (1.5 \times 10^{38}) & \quad 0 + 1.0 \\
  0 & \quad 1.0
  \end{align*}
  \]

  Floating Point Addition is not associative!
Other Floating Point Notations

- There are other floating point notations that exist to optimize for speed, precision, and/or accuracy

<table>
<thead>
<tr>
<th>Type</th>
<th>Sign</th>
<th>Exponent</th>
<th>Significand field</th>
<th>Total bits</th>
<th>Exponent bias</th>
<th>Bits precision</th>
<th>Number of decimal digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half (IEEE 754-2008)</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>16</td>
<td>15</td>
<td>11</td>
<td>~3.3</td>
</tr>
<tr>
<td>Single</td>
<td>1</td>
<td>8</td>
<td>23</td>
<td>32</td>
<td>127</td>
<td>24</td>
<td>~7.2</td>
</tr>
<tr>
<td>Double</td>
<td>1</td>
<td>11</td>
<td>52</td>
<td>64</td>
<td>1023</td>
<td>53</td>
<td>~15.9</td>
</tr>
<tr>
<td>x86 extended precision</td>
<td>1</td>
<td>15</td>
<td>64</td>
<td>80</td>
<td>16383</td>
<td>64</td>
<td>~19.2</td>
</tr>
<tr>
<td>Quad</td>
<td>1</td>
<td>15</td>
<td>112</td>
<td>128</td>
<td>16383</td>
<td>113</td>
<td>~34.0</td>
</tr>
</tbody>
</table>

https://en.wikipedia.org/wiki/Floating-point_arithmetic

(There are a lot more than this, these are just the basic ones)