Administrivia

- **Lab 1b:**
  - Due **Monday, 10/17**
  - Run `make clean` and then `make` to ensure your program compiles!
  - Late days are always an option

- **Homeworks on Ed:**
  - HW 6 (Floating Point I) due **tomorrow, 10/14**
  - HW 7 (Floating Point II) due **Monday, 10/17**
Integer Overflow
Arithmetic Overflow

**Occurs when:**

- The result of a calculation can’t be represented in the current encoding scheme
  - i.e. Lies outside the representable range of values
- Results in an incorrect result
Unsigned Overflow

Occurs when:

➔ The result lies outside $[\text{UMin}, \text{UMax}]$
  ✷ **Indicator:** Adding two numbers and the result is smaller than either number

$$
\begin{array}{c}
0b 1 1 0 0 \\
+ 0b 0 1 1 1 \\
\hline
1|0 0 1 1
\end{array}
$$

12 + 7 = 3
Signed Overflow

Occurs when:

➔ The result lies outside \([TMin, TMax]\)

◆ **Indicator:** Adding two numbers with the same sign and the result has the opposite sign

\[
\begin{array}{ccc}
0b 0110 & + & 0b 0011 \\
6 & + & 3 \\
\hline
1001 & = & -7
\end{array}
\]
Exercises!
Overflow: Exercise 1

Assuming these are all signed two’s complement 6-bit integers, compute the result of each of the following additions. For each, indicate if it resulted in overflow. [Spring 2016 Midterm 1C]

\[
\begin{array}{cccc}
001001 & 110001 & 011001 & 101111 \\
+ 110110 & + 111011 & + 001100 & + 011111 \\
\end{array}
\]
Overflow: Exercise 1

Assuming these are all signed two’s complement 6-bit integers, compute the result of each of the following additions. For each, indicate if it resulted in overflow. [Spring 2016 Midterm 1C]

001001 + 110110 = 111111 (No)

110001 + 111011 = ±101100 (No)

011001 + 001100 = 100101 (Yes)

101111 + 011111 = ±001110 (No)
Overflow: Exercise 2

Find the largest 8-bit unsigned numeral (answer in hex) such that $c + 0x80$ causes NEITHER signed nor unsigned overflow in 8 bits. [Autumn 2019 Midterm 1C]
Find the largest 8-bit unsigned numeral (answer in hex) such that \( c + 0x80 \) causes NEITHER signed nor unsigned overflow in 8 bits. [Autumn 2019 Midterm 1C]

Unsigned overflow will occur for \( c \geq 0x80 \). Signed overflow can only happen if \( c \) is negative (also \( \geq 0x80 \)). Largest is therefore, \( 0x7F \)
Overflow: Exercise 3

Find the smallest 8-bit numeral (answer in hex) such that c + 0x71 causes signed overflow, but NOT unsigned overflow in 8 bits. [Autumn 2018 Midterm 1C]

For signed overflow, need (+) + (+) = (-). For no unsigned overflow, need no carryout from MSB. The first (-) encoding we can reach from 0x71 is 0x80. 0x80 – 0x71 = 0xF.
Floating Point
How can we build a representation that has a large range of values, high precision, and handle real arithmetic results (including special values like infinity and NaN)?
Encode the sign, mantissa, and exponent in three “fields” (S, E, M).

For a single-precision floating point number (float), we have the following field widths:

- **S**: 1 bit
- **E**: 8 bit
- **M**: 23 bits

Floating Point Representation

\[ +1.011_2 \times 2^1 \]
Special Cases in Floating Point

<table>
<thead>
<tr>
<th>E</th>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0...0</td>
<td>0b0...0</td>
<td>+/- 0</td>
</tr>
<tr>
<td>0b0...0</td>
<td>non-zero</td>
<td>denormalized number</td>
</tr>
<tr>
<td>everything else</td>
<td>anything</td>
<td>normalized number</td>
</tr>
<tr>
<td>0b1...1</td>
<td>0b0...0</td>
<td>+/- ∞</td>
</tr>
<tr>
<td>0b1...1</td>
<td>non-zero</td>
<td>Not-a-Number (NaN)</td>
</tr>
</tbody>
</table>
Exponent & E-Field

- We only care about the exponent value, not the base
- **Biased notation** to represent + and - values (unsigned with a *bias*/offset)
- The bias is $2^{w-1} - 1$ where the width of the E field is $w$

**Ex)** $1.011_2 \times 2^1$ stored as a 32-bit float

- $Exp = 1$
- $Bias = 2^{8-1} - 1 = 127$
- $E = Exp + Bias = 1 + 127 = 128$

Note: **Overflow/underflow** occurs when we try to calculate numbers outside of the representable range.
Significand is stored with the leading 1 implied
Numbers of this form are normalized

Ex) $1.011_2 \times 2^1$, 32-bit float:

- Mantissa $= 1.011$
- E is not 0, so the leading 1 is implied (normalized)
- $M = 0b01100 \ldots 0$

Note: Rounding errors may occur due to limitations of the precision we have (i.e., the field width).
Limitations of Floating Point

➔ Not associative! \[(2 + 2^{50}) - 2^{50} \neq 2 + (2^{50} - 2^{50})\]

➔ Not distributive! \[100 \times (0.1 + 0.2) \neq 100 \times 0.1 + 100 \times 0.2\]

➔ Not cumulative! \[2^{25} + 1 + 1 + 1 + 1 \neq 2^{25} + 4\]
Exercises!
Exercises 1 & 2

Bias (4 bits) = ?

<table>
<thead>
<tr>
<th>Exponent</th>
<th>E (4 bits)</th>
<th>E (8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Exercises 1 & 2

Bias (4 bits) = \(2^{(4-1)} - 1 = 7\)

\[E = \text{Exp} + \text{Bias}\]

<table>
<thead>
<tr>
<th>Exponent</th>
<th>E (4 bits)</th>
<th>E (8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>0 1 1 1 1</td>
<td>0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>-1</td>
<td>0 1 1 1 0</td>
<td>0 1 1 1 1 1 1 1 1 0</td>
</tr>
</tbody>
</table>
Exercise 3

Represent $3145728.125_{10}$ as a single precision floating point number $(2^{21}+2^{20}+2^{-3})$

a) Convert to single precision floating point
Exercise 3

Represent $3145728.125_{10}$ as a single precision floating point
$(2^{21}+2^{20}+2^{-3})$

a) Convert to single precision floating point

01001010010000000000000000000000
Exercise 3

Represent $3145728.125_{10}$ as a single precision floating point

$(2^{21}+2^{20}+2^{-3})$

b) How does this number highlight a limitation of floating point representation?
Exercise 3

Represent $3145728.125_{10}$ as a single precision floating point
$(2^{21}+2^{20}+2^{-3})$

b) How does this number highlight a limitation of floating point representation?

Could only represent $2^{21} + 2^{20}$. Not enough bits in the mantissa to hold $2^{-3}$, which caused rounding
Exercise 4

0x80000000  -0

0xFF94BEEF  NaN

0x41180000  +9.5
Exercise 5

Based on the floating point representation, explain why each of the three mathematical property examples shown on the previous page occurs.

a) Not Associative

b) Not Distributive

c) Not Cumulative
Exercise 5

Based on the floating point representation, explain why each of the three mathematical property examples shown on the previous page occurs.

a) Not Associative

Only 23 bits of mantissa, so $2 + 2^{50} = 2^{50}$ (2 gets rounded off). So LHS = 0, RHS = 2.
Exercise 5

Based on the floating point representation, explain why each of the three mathematical property examples shown on the previous page occurs.

b) Not Distributive

0.1 and 0.2 have infinite representations in binary point (0.2 = 0b 0.00110011...), so the LHS and RHS suffer from different amounts of rounding (try it!).
Exercise 5

Based on the floating point representation, explain why each of the three mathematical property examples shown on the previous page occurs.

c) Not Cumulative

1 = 2^0 is 25 powers of 2 away from 2^25, so 2^25 + 1 = 2^25, but 4 = 2^2 is 23 powers of 2 away from 2^25, so it doesn’t get rounded off.
Exercise 6

If x and y are variable type float, give two different reasons why (x+2*y)-y == x+y might evaluate to false.
Exercise 6

If x and y are variable type float, give two different reasons why (x+2*y)-y == x+y might evaluate to false.

1. Rounding Error
2. Overflow - if x and y are large enough, then x+2y may result in infinity while x+y does not
It can seem a bit confusing to interpret a floating point from a bit-level representation.

Thankfully, the process is very methodical, so we can illustrate it using a diagram like so →

From Exercise 4
0xFF94BEEF
0b1111 1111 1001 ...
NaN