CSE 351 Section 3 – Integers and Floating Point
Welcome back to section, we’re happy that you’re here 😊

**Integers and Arithmetic Overflow**

**Exercises:**

1) [Spring 2016 Midterm 1C] 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.

- \[001001 + 110110 = 111111\] (+) + (–) = (–) No
- \[110001 + 111011 = 1110100\] (–) + (–) = (–) No
- \[011001 + 001100 = 100101\] (+) + (+) = (–) Yes
- \[101111 + 011111 = 001110\] (–) + (+) = (–) No

2) [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.

Unsigned overflow (i.e., a carry-out) will occur for \(c \geq 0x80\). Signed overflow can only happen if \(c\) is negative (looking for neg + neg = pos), which also occurs when \(c \geq 0x80\). Therefore, the largest numeral that doesn’t cause overflow is \(0x7F\).

**IEEE 754 Floating Point Standard**

**Exercises:**

3) Let’s say that we want to represent the number 3145728.125 (broken down as \(2^{21} + 2^{20} + 2^{-3}\))

a) Convert this number to into single precision floating point representation:

\[
\begin{array}{cccccccccccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

\(2^{21} + 2^{20} + 2^{-3} = 2^{21}(1+2^{-1}+2^{-24}) = 1.10...01_2 \times 2^{21}\) with 22 zeros in the mantissa.

Therefore, \(S = 0, E = 21+127 = 128 + 16 + 4 = 0b10010100, M = 0b10...0\).

b) Which limitation of floating point representation does this result highlight?

Not enough bits in the mantissa to hold \(2^{-3}\), which caused **rounding**.

4) [Summer 2018 Midterm 1E-G] We are working with a new floating point datatype (flo) that follows the same conventions as IEEE 754 except using 8 bits split into the following fields:

<table>
<thead>
<tr>
<th>Sign (1)</th>
<th>Exponent (3)</th>
<th>Mantissa (4)</th>
</tr>
</thead>
</table>

a) What is the encoding of the most negative real number that we can represent (\(\infty\) is not a real number) in this floating point scheme (binary)?

Largest normalized number, but negative means \(S = 1, E = 0b110, M = 0b1111 \rightarrow 0b11101111\).
b) If we have signed char \( x = 0b10101000 = -88 \), what will occur if we cast \( \text{flo } f = (\text{flo }) x \) (i.e., try to represent the value stored in \( x \) as a \( \text{flo} \))?

<table>
<thead>
<tr>
<th>Rounding</th>
<th>Underflow</th>
<th>Overflow</th>
<th>None of these</th>
</tr>
</thead>
</table>

From part (a), the largest normalized magnitude we can represent is \( 1.1111_2 \times 2^{6-\text{bias}} \). Here, \( \text{bias} = 2^{3-1} - 1 = 3 \), so \( 1.1111_2 \times 2^3 = 1111.1_2 = 15.5 \). 88 > 15.5, so this number is too large in magnitude to encode, so we end up with overflow (i.e., the result is \( \infty \)).

Another way of seeing this is that -88 = -(64 + 16 + 8) = -1.0112 \( \times 2^6 \) and the exponent 6 is too large to encode (6 + bias = 9, which requires 4 bits).

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.

b) Not distributive:
   0.1 and 0.2 have infinite representations in binary point (0.2 = 0.0011...) so the LHS and RHS suffer from different amounts of rounding (try it!).

c) Not cumulative:
   \( 1 = 2^6 \) 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.

6) If we have float \( x, y \), give two different reasons why \( (x+2*y) - y == x+y \) might evaluate to false.

   (1) Rounding error: like what is seen in the examples above.
   (2) Overflow: if \( x \) and \( y \) are large enough, then \( x + 2*y \) may result in infinity when \( x + y \) does not.