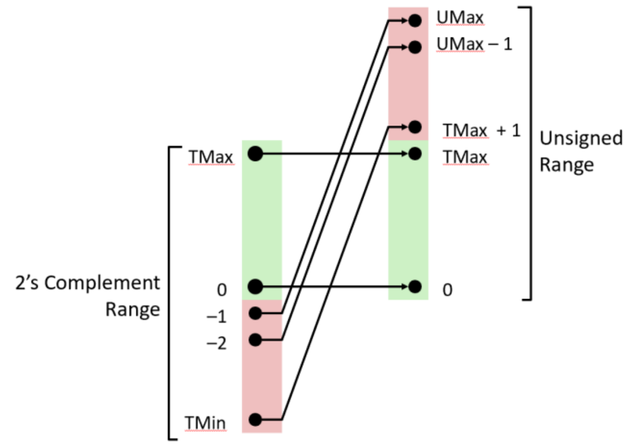
CSE 351 Section 3 – Integers and Floating Point

Welcome back to section, we're happy that you're here ☺ .. . .

Integers and Arithmetic Overflow

Arithmetic overflow occurs when the result of a calculation can't be represented in the current encoding scheme (*i.e.*, it lies outside of the representable range of values), resulting in an incorrect value.

- **Unsigned overflow:** the result lies outside of [UMin, UMax]; an indicator of this is when you add two numbers and the result is smaller than either number.
- **Signed overflow:** the result lies outside of [TMin, TMax]; an indicator of this is when you add two numbers with the same sign and the result has the opposite sign.



<u>Exercises</u>:

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        110001        011001        101111
+ 110110      + 111011      + 001100      + 011111
```

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]

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]

Goals of Floating Point

Representation should include: [1] a large range of values (both very small and very large numbers), [2] a high amount of precision, and [3] real arithmetic results (*e.g.* ∞ and NaN).

IEEE 754 Floating Point Standard

The value of a real number can be represented in scientific binary notation as:

$$\text{Value} = (-1)^{\text{sign}} \times \text{Mantissa}_2 \times 2^{\text{Exponent}} = (-1)^S \times 1.M_2 \times 2^{E\text{-bias}}$$

The binary representation for floating point values uses three fields:

- **S**: encodes the *sign* of the number (0 for positive, 1 for negative)
- **E**: encodes the *exponent* in **biased notation** with a bias of 2^{w-1}-1
- **M**: encodes the *mantissa* (or *significand*, or *fraction*) – stores the fractional portion, but does not include the implicit leading 1.

	S	E	M
`float`	1 bit	8 bits	23 bits
`double`	1 bit	11 bits	52 bits

How a `float` is interpreted depends on the values in the exponent and mantissa fields:

E	M	Meaning
0	anything	denormalized number (denorm)
1-254	anything	normalized number
255	zero	infinity (∞)
255	nonzero	not-a-number (NaN)

Exercises:

Bias Notation

1) Suppose that instead of 8 bits, E was only designated 4 bits. What is the bias in this case? ____________

2) Compare these two representations of E for the following values:

Exponent	E (4 bits)	E (8 bits)
1		
0		
-1		

Notice any patterns?

Floating Point / Decimal Conversions

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:

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

4) What are the decimal values of the following `float`s?

0x80000000   0xFF94BEEF   0x41180000

5) [Summer 2018 Midterm 1E-G] For the rest of this problem 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:

Sign (1)	Exponent (3)	Mantissa (4)

E) What is the value of the numeral **`0b 1010 1000`** in this representation?

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

G) What will occur if we cast **`flo f = (flo) x`** (*i.e.* try to represent the value stored in x as a `flo`)? Note: from a previous problem **`signedchar x = 0b10101000`** = -88.

Rounding   Underflow   Overflow   None of these

Floating Point Mathematical Properties

- Not <u>associative</u>: $(2 + 2^{50}) - 2^{50} \neq 2 + (2^{50} - 2^{50})$
- Not <u>distributive</u>: $100 \times (0.1 + 0.2) \neq 100 \times 0.1 + 100 \times 0.2$
- Not <u>cumulative</u>: $2^{25} + 1 + 1 + 1 + 1 \neq 2^{25} + 4$

<u>Exercises</u>:

6) Based on floating point representation, explain why each of the three statements above occurs.

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

IEEE 754 Float (32 bit) Flowchart