CSE 351 Section 3 – Integers and Floating Point

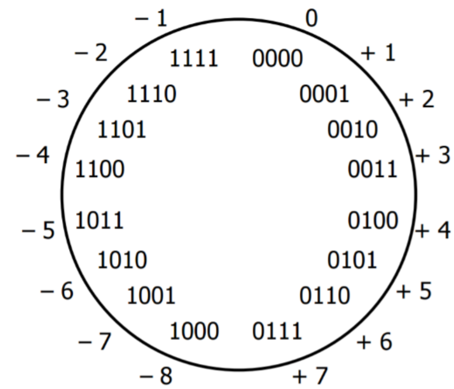
Welcome back to section, we're happy that you're here ☺

Signed Integers with Two's Complement

Two's complement is the standard for representing signed integers:

- The most significant bit (MSB) has a negative value; all others have positive values (same as unsigned)
- Binary addition is performed the same way for signed and unsigned
- The bit representation for the negative (additive inverse) of a two's complement number can be found by:

  <u>flipping all the bits and adding 1</u> (i.e. $-x = \sim x + 1$).



The "number wheel" showing the relationship between 4-bit numerals and their Two's Complement interpretations is shown on the right:

- The largest number is 7 whereas the smallest number is -8
- There is a nice symmetry between numbers and their negative counterparts except for -8

<u>Exercises</u>: (assume 8-bit integers)

1) What is the **largest integer**? The **largest integer + 1**?

<u>Unsigned</u>:	<u>Two's Complement</u>:
1111 1111 -> 0000 0000	0111 1111 -> 1000 0000

2) How do you represent (if possible) the following numbers: **39**, **-39**, **127**?

<u>Unsigned</u>:	<u>Two's Complement</u>:
39: 0010 0111	39: 0010 0111
-39: Impossible	-39: 1101 1001
127: 0111 1111	127: 0111 1111

3) Compute the following sums in binary using your Two's Complement answers from above. *Answer in hex.*

a. 39 -> 0b **0 0 1 0 0 1 1 1** +(-39) -> 0b **1 1 0 1 1 0 0 1** 0x **0 0** <- 0b **0 0 0 0 0 0 0 0**	**b.** 127 -> 0b **0 1 1 1 1 1 1 1** + (-39)-> 0b **1 1 0 1 1 0 0 1** 0x **5 8** <- 0b **0 1 0 1 1 0 0 0**
c. 39 -> 0b **0 0 1 0 0 1 1 1** +(-127)-> 0b **1 0 0 0 0 0 0 1** 0x **A 8** <- 0b **1 0 1 0 1 0 0 0**	**d.** 127 -> 0b **0 1 1 1 1 1 1 1** + 39 -> 0b **0 0 1 0 0 1 1 1** 0x **A 6** <- 0b **1 0 1 0 0 1 1 0**

4) Interpret your answers from 2 & 3 and indicate if overflow has occurred for each of the representations. (For values that cannot be represented, interpret as Two's Complement, then convert to unsigned.)

a. 39 + (-39) Unsigned: 0 overflow Two's Complement: 0 no overflow	**b.** 127 + (-39) Unsigned: 88 overflow Two's Complement: 88 no overflow
c. 39 + (-127) Unsigned: 168 no overflow Two's Complement: -88 no overflow	**d.** 127 + 39 Unsigned: 166 no overflow Two's Complement: -90 overflow

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

5) Suppose that instead of 8 bits, E was only designated 5 bits. What is the bias in this case? $2^{(5-1)} - 1 = 15$

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

Exponent	E (5 bits)	E (8 bits)
1	1 0 0 0 0	1 0 0 0 0 0 0 0
0	0 1 1 1 1	0 1 1 1 1 1 1 1
-1	0 1 1 1 0	0 1 1 1 1 1 1 0

Notice any patterns?

The representations are the same except the length of number of repeating bits in the middle are different.

Floating Point / Decimal Conversions

7) Convert the decimal number 1.25 into single precision floating point representation:

0	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

8) Convert the decimal number -7.375 into single precision floating point representation:

1	1	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

9) Add the previous two floats from exercise 7 and 8 together. = -6.125
Convert that number into single precision floating point representation:

1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10) 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:

0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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*.

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

0x80000000	0xFF94BEEF	0x41180000
–0	NaN	+9.5

0x41180000 = 0b 0|100 0001 0|001 1000 0…0.
S = 0, E = 128+2 = 130 → Exponent = E – bias = 3, Mantissa = 1.0011_2
$1.0011_2 \times 2^3 = 1001.1_2 = 8 + 1 + 0.5 = 9.5$

Floating Point Mathematical Properties

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

<u>Exercises:</u>

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

<u>Associative</u>: Only 23 bits of mantissa, so $2 + 2^{50} = 2^{50}$ (2 gets rounded off). So LHS = 0, RHS = 2.

<u>Distributive</u>: 0.1 and 0.2 have infinite representations in binary point ($0.2 = 0.\overline{0011}_2$), so the LHS and RHS suffer from different amounts of rounding (try it!).

<u>Cumulative</u>: 1 is 25 powers of 2 away from 2^{25}, so $2^{25} + 1 = 2^{25}$, but 4 is 23 powers of 2 away from 2^{25}, so it doesn't get rounded off.

13) 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: 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.