Introduction to Computer Organization CSCI 2121
Fixed Width Representations

Adding a sign to an integer takes one bit or information (Why?) regardless of representation. Internally, processors represent integers and reals using a *fixed number* of bits (12, 16, 32 *etc*). Consequently, all arithmetic operation on such integers are based on **modular arithmetic**. Possible fixed width representations of signed integers are: *(i)* Sign-magnitude *(ii)* Excess representations (more abut this when we cover floating point numbers) and *(iii)* Two's complement.
The most common fixed-width representation of integers is **2's complement**.

2's Complement: The system is called **2'sC** since to find the n-bit representation of a negative number, you subtract it from 2^n. *e.g.* To represent -3 in 3 bits: $2^3 - 3 = 5$ So, -3 is repr. as: 101 in **2'sC**.
Notice that this is the same as the algorithm shown in class (Why?):

(1) Start with the unsigned binary repr. of the integer.
(2) Flip all bits
(3) Add 1

To represent -5 in three bits: $2^3 - 5 = 3$ This gives: 011 Does not seem to give a -ve answer! (Why?)

An n bit 2'sC can repr. integers in the range: $[-2^{n-1}, 2^{n-1} - 1]$.

Fixed width addition and subtraction can be done using the same circuits. The only problem here arises from the fact that this is a fixed-width representation *i.e.* only the least significant n bits are stored. Usually the CPU flags the condition by setting a bit **V** to indicate an overflow.
How do we detect an overflow?

What if you want to represent an $n-$bit **2'sC** number with a $q-$bit **2'sC**, $(q > n)$?