There will be a tutorial every week at the chosen time (Friday for CS1110, Tuesday for the other module code, I don't know the times).

The first assignment from the module is up on the webpage.

Representing Negative Numbers

We have only a fixed number of bits in our computer (e.g. 32-bit representation, 64-bit representation, etc.).

Thus, we have to split them between the positive and negative representations.

Using many bits

We try to divide the number of bit sequences in two as evenly as possible.
We settle on the following (unbalanced) convention:

000...000	0
000...001	1
...	
011...111	2,147,483,647
100...000	-2,147,483,648
100...001	-2,147,483,647
...	
111...111	-1

The point where the most significant bit (MSB) changes from 0 to 1 is halfway.
There is a negative number (-2,147,483,648) for which there is no positive number.
Interestingly, you can tell whether a number is positive or negative just from looking at the first bit.
Also, adding 1 to -1 gets you to 0 (by overflow), and adding 1 to -2 gives you -1. The direction of addition is consistent.
Where there is a problem is at the boundary between the positive and negative numbers. You get overflow (going above the top positive limit) or underflow (going below the bottom negative limit). If the MSB changes, you know this has happened.

Using only 4 bits

				binary	±	+/-	hex
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	2	2	2	2
0	0	1	1	3	3	3	3
0	1	0	0	4	4	4	4
0	1	0	1	5	5	5	5
0	1	1	0	6	6	6	6
0	1	1	1	7	7	7	7
1	0	0	0	8	-8	8	8
1	0	0	1	9	-7	9	9
1	0	1	0	10	-6	A	A
1	0	1	1	11	-5	B	B
1	1	0	0	12	-4	C	C
1	1	0	1	13	-3	D	D
1	1	1	0	14	-2	E	E
1	1	1	1	15	-1	F	F

Ranges

positive range: 0 -> $2^{n-1} - 1$

negative range: $-(2^{n-1})$ -> -1

most negative number = $-(2^{n-1})$

most positive number = $2^{n-1} - 1$

There is 1 more negative number than positive (counting 0 as neutral).

Two's Complement

This convention is called "Two's Complement".

- It ensures $x + (-x) = 0$

Example:

5	+	(-5)		
0101	+	1011	=	(1)0000

Subtraction

With this representation we can now add a negative number to another number using our multi-bit adders that we already have.

What algorithm can we use to negate a binary number? (i.e. go from one half to the other)

		binary		+/-
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Algorithm: Invert every bit, and then add 1.

Example:

Negate -6 = 1010

Step 1: 0101

Step 2: 0111 = 6

Negate -8 = 1000

Step 1: 0111

Step 2: 1000 = -8

So it's nice that -8 is unaffected by the algorithm!

We now have a method for subtraction

Simply negate the appropriate operand and add.

5 - 3 --> 5 + (-3)

We can use our full-adder to implement subtraction of 2 bits a-b by adding the two's complement of b to a.

This is done by negating b and adding 1 (which in turn can be done by setting the carry-in to 1).

If we carefully construct a multiplexor, we can use just the carry-in bit to determine whether b is sent through a NOT gate, and therefore whether we are adding or subtracting.