We have been assuming that numbers can be of any length.
We have been assuming that there is space for the Carry.
What about Negative integers/numbers?

The solution is predicated on the fact that computer memory is finite and organized in units

BITS
4 bits NIBBLES
8 bit BYTES
16 bit WORDS

Remember most Registers are 16 bits or 1 word sized

Bit numbering

For 1 byte
Bit 0 is rightmost bit
Bit 7 is leftmost bit

For 1 word
Bit 0 is rightmost bit
Bit 15 is leftmost bit

Suppose we are working with numbers that must fit in 8 bits.

Bit 0 corresponds to 2^0   Least significant bit (LSB)
Bit 1 corresponds to 2^1
Bit 2 corresponds to 2^2
...
Bit 7 corresponds to 2^7   Most significant bit (MSB)

With 16 bits   Bit 0 is LSB
                    Bit 15 is MSB

so

Binary	Hex	decimal
01011101	5D	93
11111111	FF	255
00000000	00	0
00110011	33	51

But suppose we add

```
          11001001              201
          01001111               79
          -------------         ------
1         00011000               24  but shd be 280
```

So addition (and subtraction) cause problems.

What about the sign? positive and negative integers??

One idea is to use 1 bit out of the 8 to represent the sign
This is the SIGNED MAGNITUDE Representation

We use the MSB to represent SIGN, the rest of the bits represent the MAGNITUDE

0 means +ive
1 means -ive

00000101 = +5

10000101 = -5

10000111 = -7

01111111 = +127

11111111 = -127

00000000 = +0
10000000 = -0 ???

Addition and subtraction:

Two rules:

**Rule 1: When adding two numbers of like sign
Add the two MAGNITUDES and assign the "like" SIGN
to the result**

Example: -50 + - 25 = -75

add two magnitudes = 75 ; sign is -ive so answer = -75

**Rule 2: When adding two numbers of unlike sign
Subtract the smaller magnitude from the larger magnitude
and assign the sign of the larger magnitude to the result**

Example: 25 + -50 = -25

subtract 50 - 25 = 25 assign sign of 50; answer = -25

**But there are problems with this kind of
representation and computers use a different kind
of representation call 2’s complement.**

We’ll use the idea of an ODOMETER.

Imagine person sitting on an exercycle with a 2-digit mileage indicator.

Initial Odometer reading = 00

For each mile
When person pedals forward, odometer shows 01, 02,....

When person pedals backward,odometer shows 99, 98,....

In this kind of representation, lets fix zero and say that
Pedalling Forward = Positive numbers
Pedalling Backward = Negative numbers

But there is a problem, you could get 60 by pedalling forward 60 miles or pedalling backward 40 miles. WE solve this ambiguity by splitting the range from 00 - 99 into two parts.

00 - 49 positive numbers
50 - 99 negative numbers

TWO digit odometer numbers

ODOMETER	DECIMAL
49	49
48	48
.....	....
2	2
1	1
0	0
99	-1
98	-2
97	-3
96	-4
95	-5
......	.....
52	-48
51	-49
50	-50

Addition and Subtraction with TWO digit odometer numbers

Addition:

Add normally but ignore carry from highest order digit.
Example:

98 + 03 = 01   ignoring the carry

(-2) + 3 = 1

Subtraction is simply adding the negative

A - B = A + (-B)

Example:

Signed number:
-5 - 3 = -8

Odometer

95 - 3 = 95 + 97 = 92 = -8 !!

Negative odometer numbers

The negative of N in the odometer representation, represents a trip of N miles in the backward direction.

You get -N by subtracting N from 100

$$-N = (100 - N) = 10^2 - N \text{ for two digit numbers.}$$

This is called the 10's complement representation for two digit numbers

Example:

$$-10 = 100 - 10 = 90$$

$$-23 = 100 - 23 = 77$$

$$-47 = 100 - 47 = 63$$

$$-57 = 100 - 57 = 43$$

Its easier to do the subtraction if we rewrite the equation

$$-N = (99 - N) + 1$$

99 - N ----> 9's complement of N

We use the same concept for binary numbers:

Size = 8 bits

2's complement	Decimal value
01111111	127
01111110	126
01111101	125
.....	.....
00000011	3
00000010	2
00000001	1
00000000	0
11111111	-1
11111110	-2
11111101	-3
.....	.....
10000010	-126
10000001	-127
10000000	-128