Working in base 2

- Consider the number 4,567.
  - The value is: 4*1000+5*100+6*10+7*1.
  - Back in grade school we'd have said that the 6 is "in the 10s place"

4,567

1000s 100s 10s 1s

$4,567_{10}$

1000s (10^3) 100s (10^2) 10s (10^1) 1s (10^0)

1010_2

8s (2^3) 4s (2^2) 2s (2^1) 1s (2^0)

Adding

```
  1001          1001          1111
+ 0010        + 0011        + 0011
======        ======        ======
```

Subtracting

```
  1001          1001          1111
- 0010        - 0011        - 0011
======        ======        ======
```

Negative numbers

- Say I want to use 8 bits to represent positive and negative numbers.
  - One solution is to use the “Most Significant Bit” (MSB – the one on the far left) as a “sign bit”
    - If it is 1 the number is negative.
    - What would be the value of:
      - 1000 0110
      - 0000 1000
    - Now what is the smallest and largest number we could represent?
- This scheme is called “signed magnitude”

More on negative numbers

- It turns out using signed-magnitude is pretty icky on a computer.
  - So instead they use “two’s complement”
    - If the MSB is 1 the number is negative
    - To get the magnitude of a negative flip all bits and add 1
      - 1111 1111 would be 0000 0000 +1 = 1. So 1111 1111 is negative 1.
      - 1000 0000 is an exception. That would be -128.
  - We use two’s complement because it turns out adding those numbers and adding unsigned numbers is really the same thing.
  - With 8 bits can represent 2^7-1 to -2^7 (127 to -128)

Consider adding

```
           c3c2c1
  1001     abcd
+ 0010     efgh
======     ====
  1011     wxyz
```

- Notice that
  - z= (d and !h) or (!d and h)
  - c_1=d and h
- Who cares?
  - It turns out we can build an adder out of “and”, “or” and “not” operations.
  - We can also build subtractors, multipliers, etc.
- **We can use these simple “ands”, “ors” and “nots” to build a computer. But we pretty much are stuck with base 2.**