Everything in the computer is a number. Its meaning (“semantics”) depends on context:

- numeric value (integer, floating point, signed/unsigned)
- address in memory
- character
- instruction be executed
- boolean value (true/false)

Computers use **binary numbers**: a number system in which there are only two digits, 0 and 1.

A binary digit (i.e., a single 0 or 1) is referred to as a **bit**.

**Unsigned integers**

An **integer** is a “whole number”, i.e. no fractional part.

**Unsigned** means that there is no indication of positive or negative (thus it’s assumed positive).

An example of a binary unsigned integer is 1001. How can we interpret this?

Well, think of the decimal number 345. That means: \((3\times10^2) + (4\times10^1) + (5\times10^0)\)

So 1001 means \((1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0) = 8 + 0 + 0 + 1 = 9\)

Simple, right?

In this case, the first (leftmost) bit is referred to as the “most significant bit”, because the power of 2 to which it is raised is greatest. The last (rightmost) bit is referred to as the “least significant bit”.

Two super-duper important questions:

1. **How many values can you represent with \(k\) bits?**

   Well, let’s say \(k = 3\). The different values are 000, 001, 010, 011, 100, 101, 110, and 111. That’s a total of 8 different values.

   What if \(k = 4\)? Then we have 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111. That’s a total of 16. See a trend emerging?

   A simpler way to think about it is that each of the \(k\) bits have two possible values, so you have \(2 \times 2 \times \ldots \) and so on \(k\) times. Thus, the answer here is \(2^k\).

2. **What is the biggest (unsigned integer) value you can represent with \(k\) bits?**

   Well, if \(k = 3\), then clearly the biggest unsigned integer would be 111, which is 7.

   And if \(k = 4\), then it’s 1111, which is 15. See a trend emerging?

   The biggest value is **not** \(2^k\), but is rather \((2^k – 1)\).

Just like decimal numbers, we can add binary numbers, of course. Adding unsigned integers is simple:

```
carry ->  1  1
          1  1  1  1
          + 1  0  1  0
          -------
           1  1  0  0  1
```
If you get a question like this on a test (which you almost certainly will!), you can check your work:

1111 = 15
1010 = 10
11001 = 16 + 8 + 1 = 25 w00t!

What happens when you add a number to itself?

```
carry ->          1 1
                 1 0 1 1
                + 1 0 1 1
                   -------
                 1 0 1 1 0
```

Look at what happened. The “1011” part got shifted to the left and we stuck a 0 at the end. Coincidence? No! When you add a number to itself, you multiply it by 2. In decimal, what happens when you multiply a number by 10? You just put a 0 at the end, right? Same thing here.