Integer representations

We have a byte, which consists of 8 bits. What integers can be represented using a byte? The answer depends on how we interpret the bits. With 8 bits, we have $2^8 = 256$ different possible combinations, so 256 different values can be stored.

For example, one everyday use of these 8 bits is to represent unsigned integers in the range [0..255]. These numbers are all *non-negative*; this data type is typically referred to as a byte or an unsigned short int in many programming languages.

Example: how to add two binary numbers?

```
   10110110   (182)
+  01101101   (109)
------------
  100100011   (291)
```

But what happens if we need to store negative numbers as well? How can we represent both positive and negative integers?

One common solution is reserving the *highest-order bit* (the most significant bit, also known as the 7th bit, since counting starts from zero) as a sign bit. This bit indicates the sign of the number, and it is often denoted as the sign bit, 's'.

- If the sign bit (s) is 0, the number is positive or zero.
- If the sign bit (s) is 1, the number is negative.

The remaining **7 bits** are used to represent the magnitude of the number, either positive or negative. This method allows us to represent integers in the range [-128..+127]. In other words:

- If the sign bit is 0, the number is in the range [0 .. 127].
- If the sign bit is 1, the number is in the range [-1 .. -128].

This is a simple form of sign-magnitude representation.

Two's Complement Representation

In modern computing, two's complement is a more commonly used method for representing signed integers. In this system, the highest-order bit is also used as the sign bit, but the way negative numbers are stored differs. Here's how it works:

- The most significant bit is still used as the sign bit, where 0 indicates a positive number, and 1 indicates a negative number.
- However, negative numbers are represented by taking the two's complement of their absolute value. This involves inverting all the bits and then adding 1 to the result.

Example: Encode -5 with two's complement method with 8 bits.

- The number 5 in binary (in an 8-bit system) is represented as **00000101**
- Invert the bits: **11111010**

- Add one: **11111011**