Recently Nelson asked why ~6 is -7. This is because in Java, integers are stored using a binary representation called **2's complement**. This method is widely used because it simplifies the design of arithmetic circuits and allows for efficient computation.

What is 2's Complement?

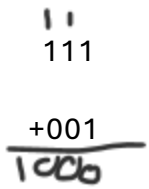
2's complement is a way of encoding signed integers so that positive and negative numbers can be represented uniformly. In this system, the most significant bit (MSB) is used as the sign bit:

- **0** indicates a positive number.
- **1** indicates a negative number.

So for example. Suppose we had three bits in our register. _ _ _.

Then 0 would be 000, 1-001, 2-010, and 3-011.

Now suppose we wanted to represent -1. We want -1 + 1 =0. What could we add to 001 to get 000. The answer is 111.



We get 1000. But since our register has only three bits, we get 0.

What would -2, -3 and -4 be?

Note that if we take 2 (010) and complement the bits we get ~010=101 which is -3 (try adding 101 and 011 –you'll get 0.)

So in all systems, -1 is all ones. For example, if x is a byte (8 bits) -1 is 11111111

The most negative byte (8 bits) would be 10000001 which is -2^7 or -128.

In general if our system has n bits, the most negative number we can get is -2^(n-1). And the most positive number is 2^(n-1)-1.

Try listing the most positive numbers and most negative numbers

byte 8 bits

short 16 bits

int 32 bits

long 64 bits

Ans:

- byte (8 bits, signed)
  - Most Positive: 127 (2^7 - 1)
  - Most Negative: -128 (-2^7)
- short (16 bits, signed)
  - Most Positive: 32,767 (2^15 - 1)
  - Most Negative: -32,768 (-2^15)
- int (32 bits, signed)
  - Most Positive: 2,147,483,647 (2^31 - 1)
  - Most Negative: -2,147,483,648 (-2^31)
- long (64 bits, signed)
  - Most Positive: 9,223,372,036,854,775,807 (2^63 - 1)
  - Most Negative: -9,223,372,036,854,775,808 (-2^63)

If your memories are good, perhaps you could memorize these numbers?!

FINDING the the 2's complement of a negative number:

To find the 2's complement of a negative number:

1. Write the number in positive binary
2. **Invert all the bits** of the number (change 0s to 1s and 1s to 0s).
3. **Add 1** to the least significant bit (LSB) of the inverted number.

Example

Let's take an 8-bit representation for simplicity:

Positive Number is 5

For the number **5**:

- Binary representation: 0000 0101

Negative Number

For the number **-5**:

1. Start with the binary representation of 5: 0000 0101
2. Invert the bits: 1111 1010
3. Add 1: 1111 1010 + 1 = 1111 1011

So, **-5** is represented as 1111 1011 in 2's complement.

Java Example

In Java, the int type is a 32-bit signed integer. Here's how you can see 2's complement in action:

```
public class TwosComplementExample {

   public static void main(String[] args) {

      int positiveNumber = 5;

      int negativeNumber = -5;



      System.out.println("Binary representation of 5: " +
Integer.toBinaryString(positiveNumber));

      System.out.println("Binary representation of -5: " +
Integer.toBinaryString(negativeNumber));

   }

}
```

Output

Binary representation of 5: 101

Binary representation of -5: 11111111111111111111111111111011

In the output, you can see that the binary representation of -5 is a 32-bit number with the MSB set to 1, indicating a negative number.

Summary

2's complement is a powerful and efficient way to represent signed integers in binary. It allows for straightforward arithmetic operations and is the standard method used in Java and many other programming languages.