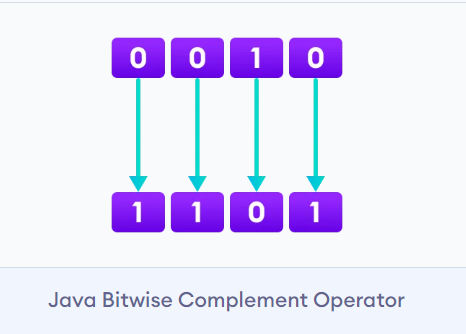
**BITWISE COMPLEMENT OPERATOR:**

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0".

For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

It changes binary digits **1** to **0** and **0** to **1**.



**NOTE:** It is important to note that the bitwise complement of any integer **N** is equal to **- (N + 1)**.

**EXAMPLE:**

Consider an integer **35**. As per the rule, the bitwise complement of **35** should be **-(35 + 1)** = **-36**. Now let's see if we get the correct answer or not.

35 = 00100011 (In Binary)

// using bitwise complement operator

~ 00100011

\_\_\_\_\_\_\_\_\_\_

11011100

In the above example, we get that the bitwise complement of **00100011** (**35**) is **11011100**. Here, if we convert the result into decimal we get **220**.

However, it is important to note that we cannot directly convert the result into decimal and get the desired output. This is because the binary result **11011100** is also equivalent to **-36**.

To understand this we first need to calculate the binary output of **-36**.

**2's Complement**

In binary arithmetic, we can calculate the binary negative of an integer using 2's complement.

1's complement changes **0** to **1** and **1** to **0**. And, if we add **1** to the result of the 1's complement, we get the 2's complement of the original number. For example,

// compute the 2's complement of 36

36 = 00100100 (In Binary)

1's complement = 11011011

2's complement:

11011011

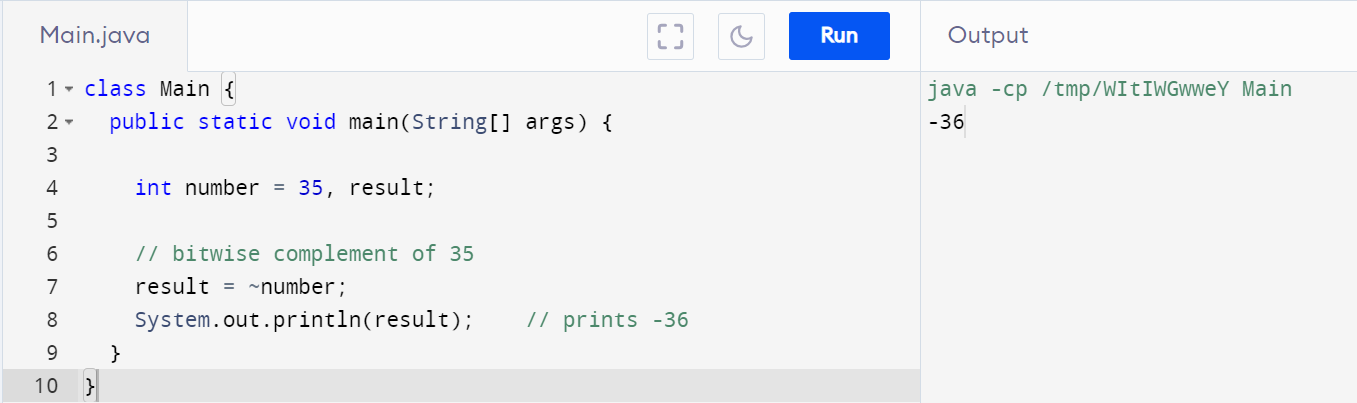
+ 1

\_\_\_\_\_\_\_\_\_

11011100

Here, we can see the 2's complement of **36** (i.e. **-36**) is **11011100**. This value is equivalent to the bitwise complement of **35**.

Hence, we can say that the bitwise complement of **35** is **-(35 + 1) = -36**.



## Introduction to Java BigInteger

Let's say you are building a factorial calculator that gives a factorial of any number that the user inputs. If the user inputs 5, then it gives output 120 as 5! = 120. Now let's say the user inputs **13**, the answer should be **6227020800**, however, your program will break if you have used int data type to store the answer. This is because the int data type can only store values up to **2147483647**, and as you can see **13!** is greater than what the int data type can store. However, if you used a long variable to store the answer, then your program will work. However, if a user inputs a bigger number such as 20 or 100, then again your program will break as the answer will become greater than what primitive data types can store.

To counter the above problem, you can use a Biginteger class in java. Biginteger class in java is used for mathematical operations that require integer values larger than what primitive data types can store. A biginteger class represents immutable arbitrary-precision integers. Apart from storing very large numbers, one can even perform operations on large numbers using biginteger in java. Before proceeding with how to use biginteger class in java, let's have a look at how the biginteger class works and how it overcomes the limitation of primitive data types.

## Working of BigInteger Class in Java

In Java, all the bytes are stored in 2's complement method. The first bit determines whether the number is positive or negative.

**BigInteger** class in java used the mag property to store numbers that excess storage capacity of primitive data types. What does this property mean? Well, it means that the **Biginteger** class in java stores values in an array using the binary representation. The **BigInteger** java groups the binary representation in 32-bit portions. This removes the limitation of the number of storage possessed by other data types in java.

BigInteger works with both numbers and strings. You can store a very large number converted to a string using a biginteger in java.