← course home (/table-of-contents)

Bit Shifting

A **bit shift** moves each digit in a number's binary representation left or right. There are three main types of shifts:

Left Shifts

When shifting left, the most-significant bit is lost, and a 0 bit is inserted on the other end.

The left shift operator is usually written as "<<".

```
0010 << 1  →  0100
0010 << 2  →  1000
```

A single left shift multiplies a binary number by 2:

```
0010 << 1  →  0100

0010 is 2
0100 is 4
```

Logical Right Shifts

When shifting right with a **logical right shift**, the least-significant bit is lost and a `0` is inserted on the other end.

```
1011 >> 1  →  0101
1011 >> 3  →  0001
```

For positive numbers, a single logical right shift divides a number by 2, throwing out any remainders.

```
0101 >> 1  →  0010

0101 is 5
0010 is 2
```

Arithmetic Right Shifts

When shifting right with an **arithmetic right shift**, the least-significant bit is lost and the most-significant bit is *copied*.

Languages handle arithmetic and logical right shifting in different ways. Most Objective-C compilers implement the right shift operator (>>) as *arithmetic* right shifting on signed types and *logical* right shifting on unsigned types. Tricky!

```
1011 >> 1  →  1101
1011 >> 3  →  1111

0011 >> 1  →  0001
0011 >> 2  →  0000
```

The first two numbers had a `1` as the most significant bit, so more `1`'s were inserted during the shift. The last two numbers had a `0` as the most significant bit, so the shift inserted more `0`'s.

If a number is encoded using two's complement, (/concept/binary-numbers#twos-complement) then an arithmetic right shift preserves the number's sign, while a logical right shift makes the number positive.

```
// Arithmetic Shift
1011 >> 1  →  1101
    1011 is -5
    1101 is -3

// Logical Shift
1111 >> 1  →  0111
    1111 is -1
    0111 is  7
```

🡐 course home (/table-of-contents)

Next up: Integer Overflow 🡒 (/concept/integer-overflow?course=fc1&section=bit-manipulation)