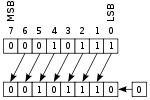
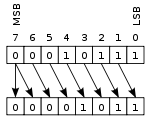
**Arithmetic shift**

Main article: [Arithmetic shift](https://en.wikipedia.org/wiki/Arithmetic_shift)

[](https://en.wikipedia.org/wiki/File:Rotate_left_logically.svg)

Left arithmetic shift

[](https://en.wikipedia.org/wiki/File:Rotate_right_arithmetically.svg)

Right arithmetic shift

In an *arithmetic shift*, the bits that are shifted out of either end are discarded. In a left arithmetic shift, zeros are shifted in on the right; in a right arithmetic shift, the [sign bit](https://en.wikipedia.org/wiki/Sign_bit) (the MSB in two's complement) is shifted in on the left, thus preserving the sign of the operand. This statement is not reliable in the latest C language draft standard, however. If the value being shifted is negative, the result is "implementation-defined," indicating the result is not necessarily consistent across platforms.[[2]](https://en.wikipedia.org/wiki/Bitwise_operation#cite_note-2)

This example uses an 8-bit register:

00010111 (decimal +23) LEFT-SHIFT

= 0010111**0** (decimal +46)

10010111 (decimal −105) RIGHT-SHIFT

= **1**1001011 (decimal −53)

In the first case, the leftmost digit was shifted past the end of the register, and a new 0 was shifted into the rightmost position. In the second case, the rightmost 1 was shifted out (perhaps into the [carry flag](https://en.wikipedia.org/wiki/Carry_flag)), and a new 1 was copied into the leftmost position, preserving the sign of the number (but not reliably, according to the most recent C language draft standard, as noted above). Multiple shifts are sometimes shortened to a single shift by some number of digits. For example:

00010111 (decimal +23) LEFT-SHIFT-BY-TWO

= 010111**00** (decimal +92)

A left arithmetic shift by *n* is equivalent to multiplying by 2*n* (provided the value does not [overflow](https://en.wikipedia.org/wiki/Arithmetic_overflow)), while a right arithmetic shift by *n* of a [two's complement](https://en.wikipedia.org/wiki/Two%27s_complement) value is equivalent to dividing by 2*n* and [rounding toward negative infinity](https://en.wikipedia.org/wiki/Rounding#Round_half_down). If the binary number is treated as [ones' complement](https://en.wikipedia.org/wiki/Ones%27_complement), then the same right-shift operation results in division by 2*n* and [rounding toward zero](https://en.wikipedia.org/wiki/Rounding#Round_half_towards_zero).

**Logical shift**

Main article: [Logical shift](https://en.wikipedia.org/wiki/Logical_shift)

|  |
| --- |
| [https://upload.wikimedia.org/wikipedia/commons/thumb/5/5c/Rotate_left_logically.svg/150px-Rotate_left_logically.svg.png](https://en.wikipedia.org/wiki/File:Rotate_left_logically.svg) |

|  |  |
| --- | --- |
| Left logical shift | [https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/Rotate_right_logically.svg/150px-Rotate_right_logically.svg.png](https://en.wikipedia.org/wiki/File:Rotate_right_logically.svg) |

|  |
| --- |
| Right logical shift |

In a *logical shift*, zeros are shifted in to replace the discarded bits. Therefore the logical and arithmetic left-shifts are exactly the same.

However, as the logical right-shift inserts value 0 bits into the most significant bit, instead of copying the sign bit, it is ideal for unsigned binary numbers, while the arithmetic right-shift is ideal for signed [two's complement](https://en.wikipedia.org/wiki/Two%27s_complement) binary numbers.