Logical and Arithmetic shifts

In computer science, a **logical shift** is a bitwise operation that shifts all the bits of its operand. Unlike an arithmetic shift, a logical shift does not preserve a number's sign bit or distinguish a number's exponent from its mantissa; every bit in the operand is simply moved a given number of bit positions, and the vacant bit-positions are filled in, generally with zeros (compare with a circular shift).

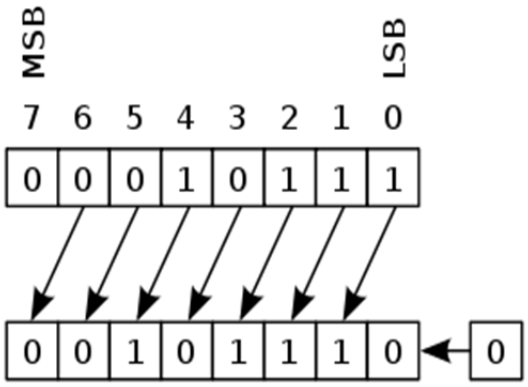
A logical shift is often used when its operand is being treated as a sequence of bits rather than as a number.
Logical shifts can be useful as efficient ways of performing multiplication or division of unsigned integers by powers of two. Shifting left by *n* bits on a signed or unsigned binary number has the effect of multiplying it by 2^n. Shifting right by *n* bits on an *unsigned* binary number has the effect of dividing it by 2^n (rounding towards 0).

Because arithmetic right shift differs from logical right shift, many languages have different operators for them. For example, in Java and JavaScript, the arithmetic right shift operator is >>; whereas the logical right shift operator is >>>. (Java only has one left shift operator (<<), because arithmetic left shift and logical left shift have the same effect.)

The C, C++, and Go programming languages, however, have only one right shift operator, >>. Most C and C++ implementations, as well as the Go language, choose which right shift to perform depending on what type of integer is being shifted -- signed integers are shifted using the arithmetic shift, and unsigned integers are shifted using the logical shift.

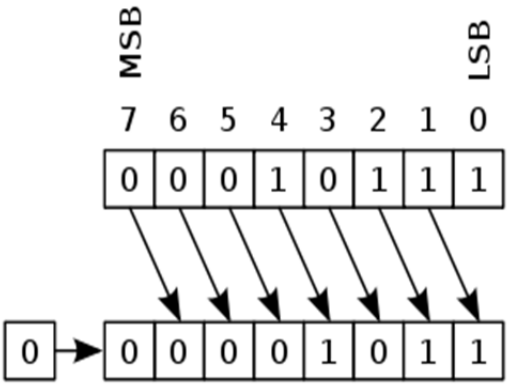
If the bit sequence 0001 0111 (decimal 23) were subjected to a logical shift of one bit position... (please refer to images below)

to the left would yield: 0010 1110 (decimal 46) ...



Logical left shift one bit

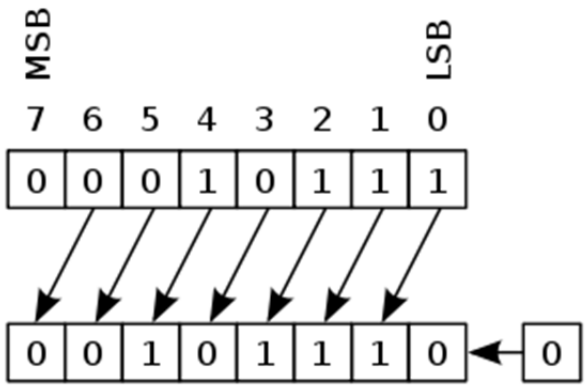
to the right would yield: 0000 1011 (decimal 11)



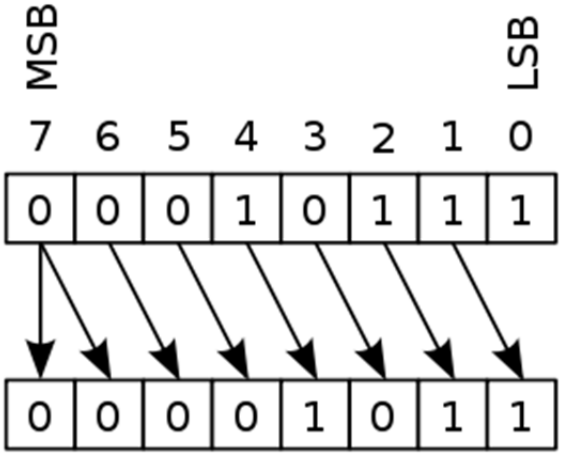
Logical right shift one bit

In computer programming, an **arithmetic shift** is a shift operator, sometimes known as a **signed shift** (though it is not restricted to signed operands). For binary numbers it is a bitwise operation that shifts all of the bits of its operand; every bit in the operand is simply moved a given number of bit positions, and the vacant bit-positions are filled in. Instead of being filled with all 0s, as in logical shift, when shifting to the right, the leftmost bit (usually the sign bit in signed integer representations) is replicated to fill in all the vacant positions (this is a kind of sign extension).

Arithmetic shifts can be useful as efficient ways of performing multiplication or division of signed integers by powers of two. Shifting left by *n* bits on a signed or unsigned binary number has the effect of multiplying it by 2^n. Shifting right by *n* bits on a two's complement *signed* binary number has the effect of dividing it by 2^n, but it always rounds down (towards negative infinity). This is different from the way rounding is usually done in signed integer division (which rounds towards 0). This discrepancy has led to bugs in more than one compiler.



A left arithmetic shift of a binary number by 1. The empty position in the least significant bit is filled with a zero. Note that arithmetic left shift may cause an overflow; this is the only way it differs from logical left shift.



A right arithmetic shift of a binary number by 1. The empty position in the most significant bit is filled with a copy of the original MSB