Logic Gates

The output is `1` when exactly one input is `1`; this is enumerated in the following table:

`p`	`q`	`p ^ q`
0	0	0
0	1	1
1	0	1
1	1	0

"Exclusive or" is can also be created from the basic three gates: `p ^ q` is equivalent to `(p | q) & ~(p & q)` or `(p & ~q) | (q & ~p)`.

Bit-wise Boolean operators in code

Virtually every C-derived language, including Java, Python, Javascript, and most other languages in common use today, have a set of bit-wise Boolean operators that can be combined to perform various tasks. These treat the integer datatype in the language in question as an array or list of bits (generally 32 bits, though that varies a little) and allow you to manipulate them directly.

Operator	Meaning	Example
`&`	Bit-wise and	1100_2 `&` $0110_2 \rightarrow 0100_2$ (i.e., `(12 & 6) == 4`)
`|`	Bit-wise or	1100_2 `|` $0110_2 \rightarrow 1110_2$ (i.e., `(12 | 6) == 14`)
`^`	Bit-wise xor	1100_2 `^` $0110_2 \rightarrow 1010_2$ (i.e., `(12 ^ 6) == 10`)
`>>`	Bit-shift to the right	1101001_2 `>>` $3 \rightarrow 1101_2$ (i.e., `(105 >> 3) == 13`)
`<<`	Bit-shift to the left	1101_2 `<<` $3 \rightarrow 1101000_2$ (i.e., `(13 << 3) == 104`)

When shifting, bits that no longer fit within the number are dropped. New bits are generally added to keep the number the same number of bits; for left shifts those new bits are always 0s, but for right shifts they are sometimes 0s and sometimes copies of whatever bit had been in the highest-order spot before the shift. Copying the high-order bit is called "sign-extending" because it keeps negative numbers negative in twos-complement. Which kind of right-shift is performed varies by language and by datatype shifted. Most languages use sign-extending shifts for signed

Masks

A bit-mask or simply **mask** is a value used to select a set of bits from another value. Typically, these have a sequential set of bits set to 1 while all others are 0, and are used with an `&` to select particular bits out of a value.

Bit-mask constants are generally written in hexadecimal; for example, `0x3ffe0` (or 0011 1111 1111 1110 0000_2) selects 13 bits, the 5th-least-significant through the 17th.

Bit-mask computed values are generally built using shifts and negations; for example, `((~0)<<5) ^ ((~0)<<14)` generates `0x3fe0`:

Expression	binary	description	alternative constructions
`0`	... `00000000000000000`	all zeros	
`~0`	... `11111111111111111`	all ones	`-1`
`(~0)<<5`	... `11111111111100000`	ones with 5 zeros in the bottom place	`~((1<<5)-1)`
`(~0)<<14`	... `11100000000000000`	ones with 14 zeros in the bottom place	`~((1<<14)-1)`
`((~0)<<5) ^ ((~0)<<14)`	... `00011111111100000`	9 ones, 5 places from bottom	`((1<<9)-1)<<5`, `(~((~0)<<9))<<5`

Set operation	Bit-wise parallel
$a \in x$	`(A & x) != 0`
$\{a\} \cup x$	`A | x`
$x \setminus \{a\}$	`x & ~A`

Set datatype action	Bit-wise parallel
`x.contains(A)`	`(x & A) != 0`
`x.add(A)`	`x |= A`
`x.remove(A)`	`x &= ~A`