Bitwise Operations Cheatsheet - https://github.com/shackspace/uc-basics

Operators

AND	1 iff[1] all inputs are 1 0 otherwise
OR	1 if at least one input is 1 0 only If all inputs are 0
XOR	1 if inputs are not the same
NOT	1 if input is 0 0 if input is 1

CAVEAT

Semantics pitfall!
OR corresponds to “and/or” in spoken language, while XOR corresponds to “either or”.
XOR stands for exclusive-or.

C-Style Operators

	Bitwise	Logical
AND	&	&&
OR	|	||
NOT	~	!
XOR	^	

CAVEAT

To fiddle with bits, use only bitwise operators!
Your compiler will allow you to use logical operators, but the result will not be what you want!

Bits & Bytes

8 bits make 1 byte[2].
Bits are numbered from right to left.

Bit #	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
	MSB[3]							LSB[4]

This is analogous to how the decimal system works, the more important digits are further left. But instead of every digit having a place value of 10^n binary has a place value of 2^n.

Shifting

Shift all bits to the left or right en-bloc.

Bits that are "shifted out" are lost.
Replacement bits "shifted in" are usually set to 0, with one exception: Right-shifting unsigned values repeats the MSB to preserve the sign.

	Operator
Shift Left	<<
Shift Right	>>

Fun Fact :)

Shifting left by n places is equivalent to multiplying by 2^n.
Likewise, shifting right by n places is equivalent to dividing by 2^n.

Operation	Unsigned Result	Signed Result
11001011 >> 2	00**110010**	11**110010**
11001011 << 2	**00101100**	**00101100**

1 iff: “if and only if”
2 For sake of simplicity and the platform we're using, we only consider bytes in this cheatsheet
3 MSB: Most significant bit
4 LSB: Least significant bit

Bitwise Operations Cheatsheet - https://github.com/shackspace/uc-basics

Masking

If you're only interested in a subset of bits, you can "select" them using a mask.
A mask is simply a byte with a number of bits set to 1.
You can think of such a 1-bit as a cut-out in a piece of paper.
Everything that is 1 you can see, everything else (0) you cannot.

Useful Operations

We've collected a bunch of useful operations for your convenience in the table to the right.

Enjoy & happy hacking!

Mask with n^{th} bit set	`(1 << n)`	
	`(1 << 3)`	`=> 00001000`
Mask with n^{th} and m^{th} bit set	`(1 << n) | (1 << m)`	
	`(1 << 3) | (1 << 7)`	`=> 10001000`
Mask with everything but n^{th} bit set	`~(1 << n)`	
	`~(1 << 3)`	`=> 11110111`
Set high-bits of mask m in v	`v | m`	
	`11001100 | 01100110`	`=> 11101110`
Clear high-bits of mask m in v	`v & ~m`	
	`11001100 & ~01100110`	`=> 10001000`
	`11001100 & 10011001`	`=> 10001000`
Toggle the n^{th} bit	`V ^ (1 << n)`	
	`11001100 ^ (1 << 3)`	`=> 11000100`
	`11001100 ^ (1 << 4)`	`=> 11011100`
Directly apply operations to a variable	`a = a | b`	`=> a |= b`
	`a = a & b`	`=> a &= b`
	`a = a ^ b`	`=> a ^= b`
	`a = a << b`	`=> a <<= b`
	`a = a >> b`	`=> a >>= b`