CSE 1320 - Intermediate Programming

Bitwise Operations

Alex Dillhoff

University of Texas at Arlington

Bitwise Operations

Virtually any programming language will provide some way of operating on individual bits.

Depending on your application, bitwise operators can be essential for reaching peak performance or keeping your data footprint small.

Bitwise Operations

C provides the following bitwise operators. All but the NOT operator are binary.

Symbol	Description
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	left shift
>>	right shift
~	bitwise NOT

Bitwise AND

Bitwise AND Table

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR

Bitwise OR Table

a	**b**	**a | b**
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise Exclusive OR (XOR)

Bitwise XOR Table

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Shift

A `char` in C has a minimum size of 1 byte (8 bits).

Consider the value 10 stored in a `char`.

This is 00001010 in binary.

Bitwise Left Shift

We can shift the bits left or right by some number using the shift operators.

For example,

```
char val = 10;
val << 2;
```

The second line produces 40... How?

Bitwise Shift

00001010 shift left by 2 becomes 00101000.

Shifting the original by 2 to the right via `val >> 2` yields 00000010.

The bits do not loop around.

Bitwise NOT

Applying ~ to a value negates the bits.

That is, the 1's are flipped to 0 and vice versa.

Bitwise NOT

Consider the following example:

```
char val = 10;
printf("%d\n", ~val);
```

This produces -11...Why?

Two's Complement

Two's complement is a mathematical operation used for signed number representation.

Definition

The two's complement of an n-bit number is defined as its complement with respect to 2^n. The sum of a number and it's two's complement is 2^n.

Two's Complement

For example, the two's complement of the 4 bit number 1010 is 0110.

$1010 + 0110 = 10000 = 16_{10}$

In base 10 that is $10 + 6 = 16$.

Two's Complement

An easier way to calculate the two's complement is by inverting the bits and adding 1.

Let's try on the original example of 00001010.

Applying bitwise NOT produces $11110101_2 = -11_{10}$.

Two's Complement

Finishing the Two's complement by adding the 1 yields $11110110_2 = -10_{10}$.

This is how signed numbers are representing in computing.

A natural question to ask is **why use two's complement over one's complement** $(-10 = 11110101)$?

Two's Complement

This has to do with the efficiency of arithmetic operations.

Consider adding $-127 + 127$ using one's complement.

$10000000 + 01111111 = 11111111$

We would expect this to be 0!

Two's Complement

Let's see the same example with two's complement.

$$10000001 + 01111111 = 00000000$$

Bitmasks

Another useful application of bitwise operations is that of a **bitmask**.

Bitmasks are used to isolate a certain value or range of values of a group of bits.

They are usually applied with the AND operator.

Bitmasks

For example, the bitmask 00001111 will select the right-most 4 bits of a byte.

$11001010 \,\&\, 00001111 = 00001010.$

Bitmasks

A practical example of this is subnetting IP addresses.

Link: Cisco Subnetting Guide