CP- EITC

Lecture 5

CP Cell Leads:
Idriss EL ABIDI
Souhail LOUZI

Bitwise Operators

February, 2025

Introduction

Definition :

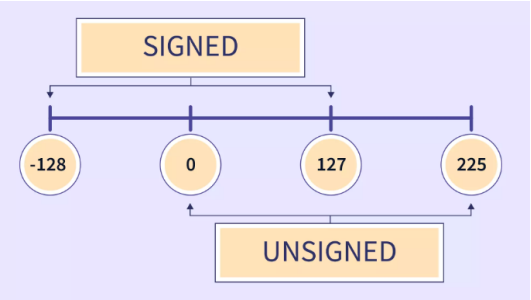
In programming, an n bit integer is internally stored as a binary number that consists of n bits. For example, the C++ type int is a 32-bit type, which means that every int number consists of 32 bits.
The bits in the representation are indexed from right to left. To convert a bit representation bk...b2b1b0 into a number, we can use the formula :

$$b_k 2^k + \ldots + b_2 2^2 + b_1 2^1 + b_0 2^0.$$

Here is the bit representation of the int number 43:
00000000000000000000000000101011

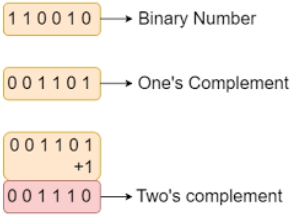
Signed & Unsigned

The bit representation of a number is either signed or unsigned. Usually a signed representation is used, which means that both negative and positive numbers can be represented. A signed variable of n bits can contain any integer between - $2^{(n-1)}$ and $2^{(n-1)}$-1. For example, the int type in C++ is a signed type, so an int variable can contain any integer between -2^{31} and 2^{31}-1.



Twos complement

The first bit in a signed representation is the sign of the number (0 for non negative numbers and 1 for negative numbers), and the remaining n-1 bits contain the magnitude of the number. Twos complement is used, which means that the opposite number of a number is calculated by first inverting all the bits in the number, and then increasing the number by one.



int in C++

```
int x = 2147483647
cout << x << "\n"; // 2147483647
x++;
cout << x << "\n"; // -2147483648
```

Initially, the value of x is $2^{31}-1$. This is the largest value that can be stored in an `int` variable, so the next number after $2^{31}-1$ is -2^{31}.

And operation

- The and operation x & y produces a number that has one bits in positions where both x and y have one bits.
- Using the and operation, we can check if a number x is even because x & 1 = 0 if x is even, and x & 1 = 1 if x is odd.
- More generally, x is divisible by (2^k) exactly when x & $(2^k - 1) = 0$.

Or operation

- The or operation x | y produces a number that has one bits in positions where at least one of x and y have one bits.
- The bitwise OR of two numbers is simply the sum of those two numbers if there is no carry involved; otherwise, you add their bitwise AND
- Lets say, we have a=5(101) and b=2(010), since there is no carry involved, their sum is just a|b. Now, if we change b to 6 which is 110 in binary, their sum would change to a|b + a&b since there is a carry involved.

Xor operation

- The xor operation $x \oplus y$ produces a number that has one bits in positions where exactly one of x and y have one bits.
- It is used in many problems. A simple example could be **Given a set of numbers where all elements occur an even number of times except one number, find the odd occurring number** This problem can be efficiently solved by doing XOR to all numbers.
- The bitwise XOR operator is the most useful operator from a technical interview perspective.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Not operation

- The NOT operation $\sim x$ produces a number where all the bits of x have been inverted. The formula $\sim x = -x - 1$ holds. For example, $\sim 29 = -30$.
- The result of the $\sim$ operator on a small number can be a large number if the result is stored in an unsigned variable.
- The result may be a negative number if the result is stored in a signed variable (assuming that negative numbers are stored in 2's complement form, where the leftmost bit is the sign bit).

Left Shift

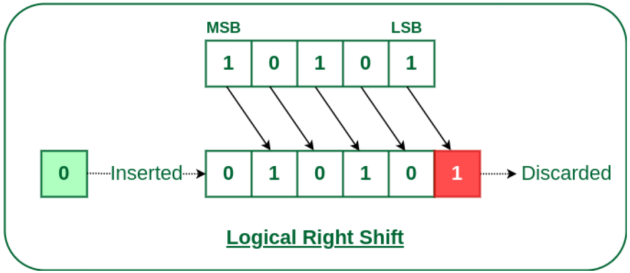
- The left bit shift x « k appends k zero bits to the number.
- Note that if there is no overflow, x « k corresponds to multiplying x by 2k.
- The left-shift and right-shift operators should not be used for negative numbers.

<< Shift Left

SYNTAX	BINARY FORM	VALUE
x = 7;	00000111	7
x=x<<1;	00001110	14
x=x<<3;	01110000	112
x=x<<2;	11000000	192

Right Shift

- the right bit shift x » k removes the k last bits from the number
- Note that x » k corresponds to dividing x by 2k rounded down to an integer
- As mentioned, it works only if numbers are positive.



Set

Every subset of a set 0,1,2,...,n-1 can be represented as an n bit integer whose one bits indicate which elements belong to the subset. This is an efficient way to represent sets, because every element requires only one bit of memory, and set operations can be implemented as bit operations

For example, since `int` is a 32-bit type, an `int` number can represent any subset of the set $\{0,1,2,\ldots,31\}$. The bit representation of the set $\{1,3,4,8\}$ is

$$00000000000000000000000100011010,$$

which corresponds to the number $2^8+2^4+2^3+2^1=282$.

Set implementation

The following code declares an int variable x that can contain a subset of 0,1,2,...,31. After this, the code adds the elements 1, 3, 4 and 8 to the set and prints the size of the set.

```
int x = 0;
x |= (1<<1);
x |= (1<<3);
x |= (1<<4);
x |= (1<<8);
cout << __builtin_popcount(x) << "\n"; // 4
```

```
for (int i = 0; i < 32; i++) {
    if (x&(1<<i)) cout << i << " ";
}
// output: 1 3 4 8
```

Set operations

	set syntax	bit syntax
intersection	$a \cap b$	a & b
union	$a \cup b$	a | b
complement	$\bar{a}$	$\sim a$
difference	$a \setminus b$	a & $(\sim b)$

For example, the following code first constructs the sets $x = \{1,3,4,8\}$ and $y = \{3,6,8,9\}$, and then constructs the set $z = x \cup y = \{1,3,4,6,8,9\}$:

```
int x = (1<<1)|(1<<3)|(1<<4)|(1<<8);
int y = (1<<3)|(1<<6)|(1<<8)|(1<<9);
int z = x|y;
cout << __builtin_popcount(z) << "\n"; // 6
```

C++ Functions

The g++ compiler provides the following functions for counting bits:

__builtin_clz(x): the number of zeros at the beginning of the number

__builtin_ctz(x): the number of zeros at the end of the number

__builtin_popcount(x): the number of ones in the number

__builtin_parity(x): the parity (even or odd) of the number of ones

```
int x = 5328; // 00000000000000000001010011010000
cout << __builtin_clz(x) << "\n"; // 19
cout << __builtin_ctz(x) << "\n"; // 4
cout << __builtin_popcount(x) << "\n"; // 5
cout << __builtin_parity(x) << "\n"; // 1
```

Iterating through subsets

The following code goes through the subsets of $\{0, 1, \ldots, n-1\}$:

```
for (int b = 0; b < (1<<n); b++) {
    // process subset b
}
```


PRACTICE
TIME
EITC
ENSIAS IT CLUB
Go Cell
++

Thank you for listening !

Idriss El Abidi
Souhail Louzi