Bit Manipulation

Introduction

Bit manipulation involves working on a particular number in its binary representation.

Applications:

1. Used in data compression.
2. In a wide variety of problems, helping us to optimize code efficiently

Shift Operators

There are two types of shift operators.

1. Left Shift Operator
2. Right Shift Operator

- ### Left Shift Operator

Syntax: n << x.

Here n is the number and x is the number of places to be left-shifted.

For example: 2<<1 , 100<<3.

Steps involved in a left shift

1. Convert the given decimal integer into its binary representation.
2. During the left shift, x number of trailing zeros are added at the end of the binary representation.
3. Finally, the binary number obtained by left shifting is converted to its decimal form.

Example:

1. 3 << 4 = 48.
2. 17 << 1 = 34.

   Note: In every left shift operation, the number is multiplied by a factor of 2. That is $N << i = N*2^i$.

- **Right Shift Operator**

Syntax: n >> x.

Here n is the number and x is the number of places to be right-shifted.

For example: 32>>1 , 144>>3.

Steps involved in a right shift

1. Convert the given decimal integer in its binary representation.
2. During the right shift, x number of trailing bits(i.e. from the end of number) are ignored and the new binary number is written.
3. Finally the binary number obtained by right shifting is converted to its decimal form.

Example:

1. 96 >> 4 = 6.
2. 17 >> 1 = 8.

   Note: In every right shift operation, the number is divided by a factor of 2. That is $N >> i = N/2^i$.

Some More Bitwise Operator

i. **Bitwise OR**

It is denoted by | in c++, java and with or in python. It is different than logical OR which returns true if either of the variables is true and false if both the variables are false. It is denoted by || operator.

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

Therefore we can conclude that

1. $x|1 = 1$
2. $x|0 = x$

For Example : 2|4 = 6.

ii. **Bitwise AND**

It is denoted by & in c++, java and with and in python. It is different than logical AND which returns true if both of the variables are true and false if either of the variables is false. It is denoted by && operator.

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

Therefore we can conclude that

1. *x*&1 = x
2. *x*&0 = 0

For Example : 2&4 = 0.

iii. **Bitwise NOT**

It is denoted by ~ (tilde) operator. It is used for complimenting the bits.

a	~ *a*
0	1
1	0

For example: In four bit representation of 4, ~4 = 11 but in reality, in 32 bit operation ~4 = -5.

iv. **Bitwise XOR**

It is denoted by ^in c++, java and python.

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

Therefore we can conclude that

1. *x*^*x* = 0
2. *x*^ ~ *x* = 1
3. *x*^1 = ~ *x*
4. *x*^0 = x

For Example : 2^4 = 6.

Applications of Bitwise Operators

1. Checking the i^{th} bit.
2. Flip i^{th} bit.
3. Check even/odd.
4. Check the power of 2.
5. Clear all bits from LSB.
6. Clear all bits from MSB.

There are many more applications of Bitwise operators and analyzing the question in the bitwise form.