Bit Magic

Bit Algorithms |Important Tactics

Let's look at some of the useful tactics of the Bitwise Operators which can be helpful in solving a lot of problems really easily and quickly.

1. **How to set a bit in the number 'num' :** If we want to set a bit at nth position in number 'num' ,it can be done using 'OR' operator(|).
    - First we left shift '1' to n position via (1 << n)
    - Then, use 'OR' operator to set bit at that position.'OR' operator is used because it will set the bit even if the bit is unset previously in binary representation of number 'num'.

2. **How to unset/clear a bit at n'th position in the number 'num' :**
   Suppose we want to unset a bit at nth position in number 'num' then we have to do this with the help of 'AND' (&) operator.
    - First we left shift '1' to n position via (1 << n) than we use bitwise NOT operator '~' to unset this shifted '1'.
    - Now after clearing this left shifted '1' i.e making it to '0' we will 'AND'(&) with the number 'num' that will unset bit at nth position position.

3. **Toggling a bit at nth position :** Toggling means to turn bit 'on'(1) if it was 'off'(0) and to turn 'off'(0) if it was 'on'(1) previously.We will be using 'XOR' operator here which is this '^'. The reason behind 'XOR' operator is because of its properties.
    - Properties of 'XOR' operator.
        - 1^1 = 0
        - 0^0 = 0
        - 1^0 = 1
        - 0^1 = 1
    - If two bits are different then 'XOR' operator returns a set bit(1) else it returns an unset bit(0).

4. **Checking if bit at nth position is set or unset:**
   It is quite easily doable using 'AND' operator.
    - Left shift '1' to given position and then 'AND'('&').

   If the result of the AND operation is 1 then the bit at nth position is set otherwise it is unset.

5. **Divide by 2**:
   ```
   x = x >> 1;
   ```
   Logic: When we do arithmetic right shift, every bit is shifted to right and blank position is substituted with sign bit of number, 0 in case of positive and 1 in case of negative number. Since every bit is a power of 2, with each shift we are reducing the value of each bit by factor of 2 which is equivalent to division of x by 2.
   Example:
   ```
   x = 18(00010010)
   x >> 1 = 9 (00001001)
   ```

6. **Multiplying by 2**:
   ```
   x = x << 1;
   ```
   Logic: When we do arithmetic left shift, every bit is shifted to left and blank position is substituted with 0 . Since every bit is a power of 2, with each shift we are increasing the value of each bit by a factor of 2 which is equivalent to multiplication of x by 2.

Example:

```
x = 18(00010010)
x << 1 = 36 (00100100)
```

7. **Find log base 2 of a 32 bit integer**:

```

int log2(int x)
{
    int res = 0;
    while (x >>= 1)
        res++;
    return res;
}

```

Logic: We right shift x repeatedly until it becomes 0, meanwhile we keep count on the shift operation. This count value is the log2(x).

8. **Flipping the bits of a number**: It can be done by a simple way, just simply subtract the number from the value obtained when all the bits are equal to 1 .

For example:

```
Number : Given Number
Value  : A number with all bits set in given number.
Flipped number = Value – Number.

Example :
Number = 23,
Binary form: 10111;
After flipping digits number will be: 01000;
Value: 11111 = 31;
```

9. **Swapping Two Numbers**: We can easily swap two numbers say **a** and **b** by using the XOR(^) operator by applying below operations:

```
a ^= b;
b ^= a;
a ^= b;
```