Bit Magic

Basic problem On Bit Manipulation

Below are some problems which can be solved very easily using some of the basic concepts of Bit Manipulation. Let's look at each of these problems and the Bitwise approach of solving them:

- **Problem 1**: Given a number N, the task is to check whether the given number is a power of 2 or not.

  For Example:

  ```
  Input : N = 4
  Output : Yes
  2^2 = 4

  Input : N = 7
  Output : No

  Input : N = 32
  Output : Yes
  2^5 = 32
  ```

  Bitwise Solution: If we subtract a number which is a power of 2 1 then all of it's unset bits after the only set bit become set; and the set bit become unset.

  For example, consider **4** (Binary representation: 100) and **16**(Binary representation: 10000), we get following after subtracting 1 from them:

  ```
  3 -> 011
  15 -> 01111
  ```

  You can clearly see that **bitwise-AND(&)** of 4 and 3 gives zero, similarly 16 and 15 also gives zero.

  So, if a number N is a power of 2 then **bitwise-AND(&)** of **N and N-1** will be zero. We can say that N is a power of 2 or not based on the value of N&(N-1).

- **Problem 2**: Given a number N, find the most significant set bit in the given number.

  Examples:

  ```
  Input : N = 10
  Output : 8
  Binary representation of 10 is 1010
  The most significant bit corresponds
  to decimal number 8.

  Input : N = 18
  Output : 16
  ```

  Bitwise Solution: The most-significant bit in binary representation of a number is the highest ordered bit, that is it is the bit-position with highest value.

One of the solution is first find the bit-position corresponding to the MSB in the given number, this can be done by calculating logarithm base 2 of the given number, i.e., $\log_2(N)$ gives the position of the MSB in N.

Once, we know the position of the MSB, calculate the value corresponding to it by raising 2 by the power of calculated position. That is, value = $2^{\log_2(N)}$.

- **Problem 3**: Given a number N, the task is to find the XOR of all numbers from 1 to N.

Examples :

```
Input : N = 6
Output : 7
// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6  = 7

Input : N = 7
Output : 0
// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 = 0
```

Solution:

1. Find the remainder of N by moduling it with 4.
2. If rem = 0, then xor will be same as N.
3. If rem = 1, then xor will be 1.
4. If rem = 2, then xor will be N+1.
5. If rem = 3 ,then xor will be 0.

How does this work?

When we do XOR of numbers, we get 0 as XOR value just before a multiple of 4. This keeps repeating before every multiple of 4.

```
Number Binary-Repr  XOR-from-1-to-n
1         1           [0001]
2        10           [0011]
3        11           [0000]  <----- We get a 0
4       100           [0100]  <----- Equals to n
5       101           [0001]
6       110           [0111]
7       111           [0000]  <----- We get 0
8      1000           [1000]  <----- Equals to n
9      1001           [0001]
10     1010           [1011]
11     1011           [0000] <------ We get 0
12     1100           [1100] <------ Equals to n
```