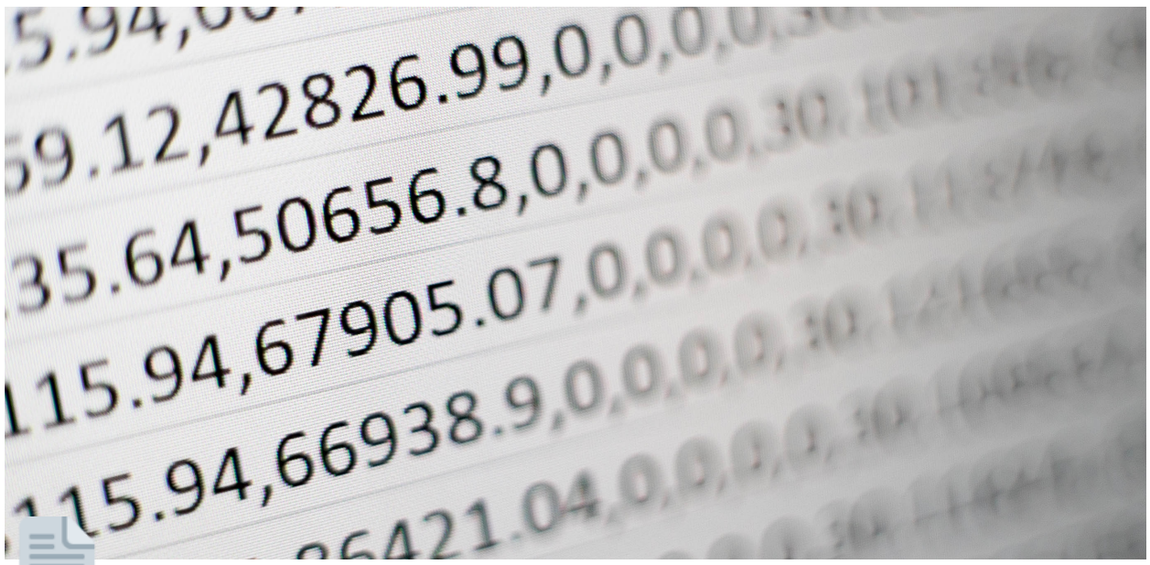



# Bit Manipulation

## What is Bit Manipulation?

Bit Manipulation refers to manipulation of the bits of a number using bit-wise operators.

## Data representation in memory

## Uses of Bit Manipulation

- Data compression
- Encryption
- Optimisation

## Bitwise Operators

**Bitwise Operators**

| Aa Operator | Resemblance |
|---|---|
| & | Bitwise AND operator |
| \| | Bitwise OR Operator |
| ^ | Bitwise XOR operator |
| ~ | Bitwise NOT operator |
| << | Left shift operator |
| >> | Right shift operator |

Bitwise operators cannot be directly applied to primitive data types such as `float` , `double` , etc.

They are mostly used with `int` data type because of its compatibly

The result of the computation of bitwise logical operators is shown in the table given below.

### Bitwise AND

This is one of the most commonly used logical bitwise operators. It is represented by a single ampersand sign (`&`). Two integer expressions are written on each side of the (`&`) operator.

The result of the bitwise AND operation is 1 if both the bits have the value as 1; otherwise, the result is always 0.

Let us consider that we have 2 variables `op1` and `op2` with values as follows:

`Op1 = 0000 1101`

`Op2 = 0001 1001`

The result of the AND operation on variables `op1` and `op2` will be

`Result = 0000 1001`

### Properties :-

- A & B = B & A ( Commutative Law )
- A & ( B & C ) = ( A & B ) & C ( Associative Law )
- A & 0 = 0
- A & 1 = A ( Identity )
- A & A = A ( Idempotence )

## Bitwise OR

It is represented by a single vertical bar sign `(|)`. Two integer expressions are written on each side of the `(|)` operator.

The result of the bitwise OR operation is 1 if at least one of the expression has the value as 1; otherwise, the result is always 0.

Let us consider that we have 2 variables `op1` and `op2` with values as follows:

`Op1 = 0000 1101`

`Op2 = 0001 1001`

The result of the OR operation on variables op1 and op2 will be

`Result = 0001 1101`

### Properties :-

- A | B = B | A ( Commutative Law )
- A | ( B | C ) = ( A | B ) | C ( Associative Law )
- A | 0 = A
- A | 1 = 1 ( Identity )
- A | A = A ( Idempotence )

## Bitwise NOT

It is represented by symbol (~). It is an unary operator that takes one number and inverts all the bits of it.

### Example:

Suppose

**A =** `00000000 00000000 00000000 00000111`

**~A =** `11111111 11111111 11111111 11111010`

## Bitwise XOR

It is represented by a symbol (`^`). Two integer expressions are written on each side of the (`^`) operator.

The result of the bitwise XOR operation is 1 if and only if the operands are different; otherwise, the result is 0.

Let us consider that we have 2 variables `op1` and `op2` with values as follows:

`Op1 = 0000 1101`

`Op2 = 0001 1001`

The result of the XOR operation on variables op1 and op2 will be

`Result = 0001 0100`

### Properties :-

- A ^ B = B ^ A ( Commutative Law )
- A ^ ( B ^ C ) = ( A ^ B ) ^ C ( Associative Law )
- A ^ 0 = A
- A ^ 1 = ~A ( Identity )
- A ^ A = 0
- A ^ B ^ B = A

**Bitwise Operations**

| Aa X | Y | X&Y | X\|Y | X^Y | ~(X) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

## Lets see everything we discussed in code

```
#include <stdio.h>
int main()
{
    int a = 20; /* 20 = 010100 */
    int b = 21; /* 21 = 010101 */
    int c = 0;
    int d=1;

    c = a & b;       /* 20 = 010100 */
    printf("AND - Value of c is %d\n", c );

    c = a | b;       /* 21 = 010101 */
    printf("OR - Value of c is %d\n", c );

    c = a ^ b;       /* 1 = 0001 */
    printf("XOR - Value of c is %d\n", c );

    int inverse = ~d;
    printf("Compliment of d is %d\n", inverse );
    -
    getch();
}
```

# Bit-wise shift

A bit-shift moves each digit in a number's binary representation left or right. In simple words, they shift bits.

## Left Shift

### Operator: `<<`

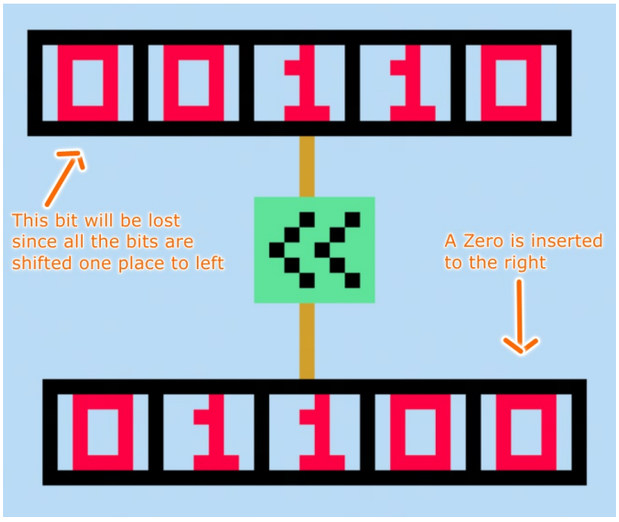
Integers are stored, in memory, as a series of bits. For example, the number 6 stored as a 32-bit int would be

`00000000 00000000 00000000 00000110` = `6`

Shifting this bit pattern to the left one position (`6 << 1`) would result in the number `12`

`00000000 00000000 00000000 00001100` = `12`

Notice, the digits have shifted to the left by one position, and the last digit on the right is filled with a zero.




Also notice that shifting left is equivalent to multiplication by powers of 2.

### Example:

`6<<1` = `6*2`

`6<<3` = `6*8`

so to generalize:

`n<<k` ⇒ $n * (2^k)$

## Right Shift

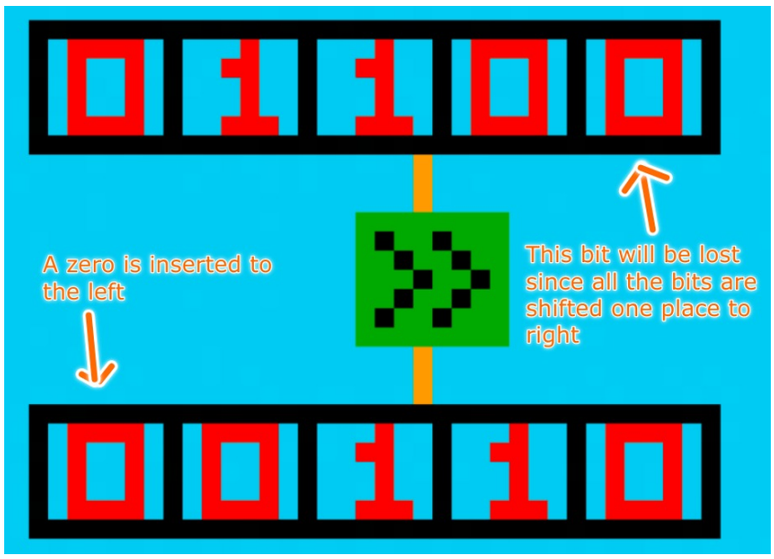
### Operator: `>>`

A logical right shift is the converse to the left shift. Rather than moving bits to the left, they simply move to the right.

`00000000 00000000 00000000 00001100` = `12`

Shifting this bit pattern to the right one position (`12 >> 1`) would result in the original number `6`

`00000000 00000000 00000000 00000110` = `6`




### Shifts cannot reclaim lost bits:

`00111000 00000000 00000000 00000110` = `939,524,102`

Shifting this bit pattern to the left 4 position (`939,524,102 << 4`) would result in the `2,147,483,744` = `10000000 00000000 00000000 01100000`

notice the shift, we lost 4 bits (`0011`) from the left. If we try to right shift 4 positions, it won't recover the lost bits like so:

`((939,524,102 << 4) >>> 4)` we get `134,217,734` and not `939,524,102`

so to generalize:

`n>>k` ⇒ $n/(2^k)$

# Some common patterns/properties

n<<k = n* 2^k

Even/odd nos

Set/unset bits

x&(x-1)

check power of 2.

Set bit A |= 1 << bit

Clear bit A &= ~(1 << bit)

# Problems

- https://leetcode.com/problems/single-number/

```
int x = 0;
for(int i =0; i<nums.size();i++){
    x = x^nums[i];
}
```

- https://leetcode.com/problems/missing-number/

```
int x = 0;
for(int i=0;i<=nums.size();i++){
  x = x^i;
}
for(int i=0;i<nums.size();i++){
  x = x^nums[i];
}
return x;
```

- https://www.codechef.com/LTIME97C/problems/UNONE

```
    int n;
    cin>>n;
    int even[n];   //to store all even nos
    int odd[n];    //to store all odd nos
    int a=0,b=0;   //iterators for even and odd arrays respectively
    for(int i=0;i<n;i++){
        int temp;
        cin>>temp;
        if(temp&1) //checking if ith input is even or odd
        {
            odd[a++] = temp;
        }
        else{
            even[b++] = temp;
        }
    }

    //display all even first
    for(int i=0;i<b;i++){
        cout<<even[i]<<" ";
    }

    for(int i = 0;i<a;i++){
        cout<<odd[i]<<" ";
    }

    cout<<endl;
    return;
```

- https://leetcode.com/problems/counting-bits/

```
vector <int> ans;
/*for even no 'X', no of set bits = no of set bits in 'X/2'
  for odd no 'X', no of set bits = no of set bits in 'X/2' +1. */
ans[0] = 0;   //No of set bits in '0' = 0.
for(int i =1;i<=n; i++){
      if(i & 1){
      ans[i] = ans[i/2] + 1;
    }
    else{
      ans[i] = ans[i/2];
    }
}
return ans;
```

- https://leetcode.com/problems/bitwise-and-of-numbers-range/

```
/*
No need to check for all elements. Only smallest and greatest element can give the ans.
Compare all the bits of leftmost and righmost element.
If bits a are same at an index 'i' (both set/both unset), then reflect same in the ith position in ans.
If bits are different at any positon break and return current ans.
*/
        int rangeBitwiseAnd(int left, int right) {
        int ans = 0;
        for(int i=31; i > -1;i--){
            if( (left & (1<<i)) != (right & (1<<i)) ){
                break;
```

```
            }
            else{
                if(left & (1<<i)){
                    ans = ans | 1<<i;
                }
            }
        }
        return ans;
```

- https://leetcode.com/problems/sum-of-two-integers/

```
int a,b;
a = a&b;
while(b){
  b<<1;
  a=a^b;
}
cout<<a;
```

- https://codeforces.com/contest/1527/problem/A

```
    ll n;
    cin>>n;
    int k=0;
//finding most significant bit in the no.
    while(n){
        k++;
        n = n>>1;
    }
//if n is a power of 2. Ans = n-1
    if(n == pow(2,k)){
        cout<<(int)pow(2,k)-1<<endl;
        return;
    }
//else ans is nearest power of two (which is less than n) -1.
    cout<< (int)pow(2,k-1)-1<<endl;
    return;
```

- https://codeforces.com/contest/1514/problem/B

```
//https://codeforces.com/blog/entry/89810
    int n,k;
    cin>>n>>k;
    long long product = 1;
    for(int i=0; i<k;i++){
        product = (product*n)%x;  // x = 10^9 + 7
    }
    cout<<product<<"\n";
    return;
```