BIT MANIPULATION

Bit-wise Operators

-> Binary AND (&)

-> Binary OR (|)

-> Binary XOR (^)

-> Binary One's Complement (~)

-> Binary Left Shift (<<)

-> Binary Right Shift (>>)

Binary AND (&)

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Eg. 101 (5)
(&) 110 (6)
100 (4)
=> 5 & 6 = 4

Binary OR (|)

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

Eg. 101 (5)
(|) 110 (6)
111 (7)
=> 5 | 6 = 7

Binary XOR (^)

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Eg. 101 (5)
$(^{\wedge})\ 110$ (6)
011 (2)

$5 \wedge 6 = 2$

Binary One's Complement (~)

Eg. $\sim 5 = -6$

Explanation →

$(5)_{10} = (101)_2 = (00000101)_2$

1's complement = $(11111010)_2$

↑ MSB i.e. 1 (which means it would be a negative number)

Now find the magnitude of this negative number again calculate 1's complement = $(00000101)_2$

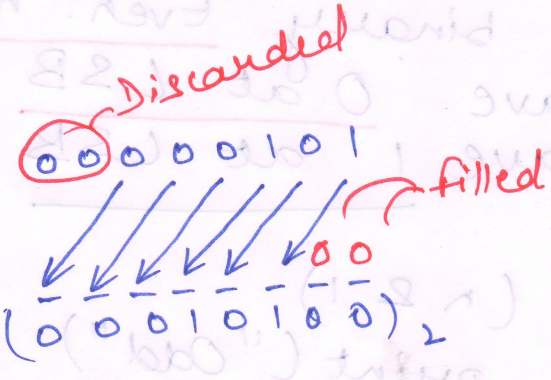
Now add 1 ⇒ $00000101 + 1$

$00000110 = (6)_{10}$

That's why $\sim 5 = -6$

Binary left Shift (<<)

Eg. $5 << 2 = 10$

$(5)_{10} = (101)_2 =$ 

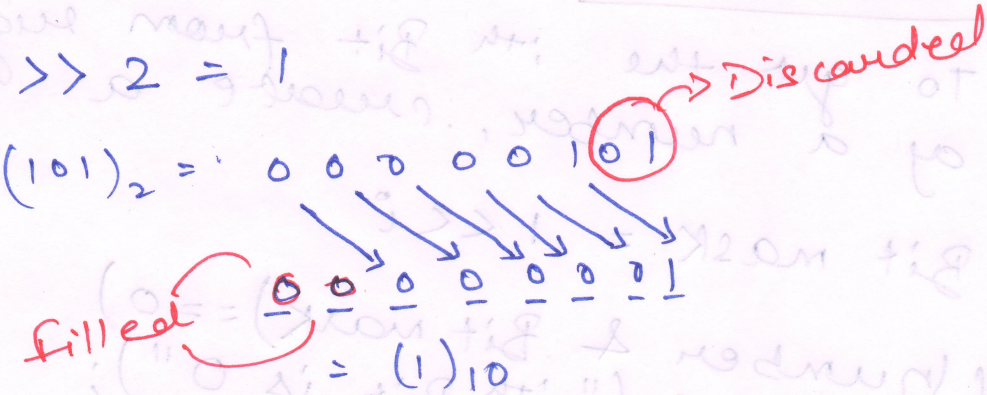
$= (20)_{10}$

Imp.

$a << b = a \times 2^b$

Binary Right Shift (>>)

Eg. $5 >> 2 = 1$

$(5)_{10} = (101)_2 =$ 

$= (1)_{10}$

Imp.

$$a >> b = \frac{a}{2^b}$$

Check if Odd or Even

- In binary, Even numbers have 0 at LSB and Odd numbers have 1 at LSB.

```
if (n & 1)
    print ("Odd")
else
    print ("Even")
```

Eg. $(5)_{10} = (101)_2$

```
 101
&001
 001   (true)
```

$(6)_{10} = (110)_2$

```
 110
&001
 000   (false)
```

Get i^{th} Bit

- To get the i^{th} Bit from right side of a number, create a bit mask.
- Bit mask = 1 << i
- if (number & Bit mask == 0)
    print ("i^{th} Bit is 0");
  else
    print ("i^{th} Bit is 1");

```
   3 2 1 0
Eg. 1 0 1 0
```

get 0^{th} → Bit mask = 1 << 0 = 1

```
 1010
&0001
 0000 ⇒ 0
```

get 3^{rd} → Bit mask = 1 << 3 = 1000

```
 1010
&1000
 1000 ⇒ 1
```

Set i^{th} Bit

Set i^{th} Bit means, make i^{th} bit 1.

Bit mask = 1 << i

number = number | bitmark

Clear i^{th} Bit

Clear i^{th} Bit means, make i^{th} bit 0.

Bit mark = ~(1 << i)

number = number & bit mark

Clear range of bits

Need to clears the bits from i to j

a = (~0) << (j+1)

b = (1 << i) - 1

Bitmark = a | b

number = number & bitmark

Count set bits in a number

set bits : 1's bit

```
count = 0
while (n > 0)
{
  if (n&1 != 0)
    count++;

  n=n>>1;
}
```

Fast Exponentiation

Take an example -

find 3^5

Now $(5)_{10} = (101)_2$

create a variable ans = 1; a = 3; n = 5

```
while(n > 0)
{
    if ((n&1) != 0)
        ans = ans * a;

    a = a*a
    n = n >> 1;
}
```

Dry Run:- 3^5

ans = 1, a = 3

(5 > 0) true

((5&1) != 0) true

ans = 3;

a = 3*3 = 9

n = 2;

(2 > 0) true

((2&1) != 0) false

a = 9*9 = 81

n = 1;

(1 > 0) true

((1&1) != 0) true

ans = 3*81 = 243

a = 6561

n = 0

ans = 243

5^2

ans = 1, a = 5

(2 > 0) true

((2&1) != 0) false

a = 5*5 = 25

n = 1

(1 > 0) true

((1&1) != 0) true

ans = 1*25

n = 0

ans = 25