Bitwise operators

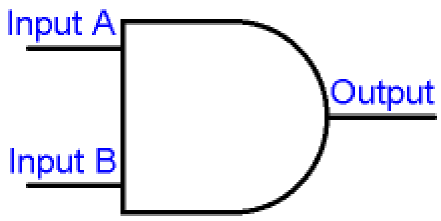
These operators perform on individual bits(0 and 1) of the operand. These operators are used to performing the operations on binary data. We can use these bitwise operators directly on binary numbers or on integers. When we use these operators on integers, these numbers are converted into bits and the bitwise operators act upon those bits. The results given by these operators are always the form of integers. The bitwise operators take the input as a decimal and convert into binary data type and perform the required operations on binary data and then the output of binary data is converted into decimal data and the decimal data is shown as output for us.

Types of Bitwise operators:

1) Bitwise AND operator(&)
2) Bitwise OR operator(|)
3) Bitwise XOR operator(^)
4) Bitwise left shift operator(<<)
5) Bitwise Right shift operator(>>)
6) Bitwise complement Operator(~)

1) Bitwise AND operator (&):

This operators performs AND operation on the individual bits of numbers. The symbol for this operator is &, which is called ampersand. To understand in detail check the Truth Table



x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

Example:

X=10

Y=2

X & Y = 2

Step-1: In the above example The x value that means 10 is converted into binary data then 10 becomes as 1010 The y value that means 2 is converted into binary data then 2 becomes as 0010

Step-2: In this step, the and operation is performed on both x and y values which are converted into binary data that suggests

```
 1010
 0010
______
  0010
```

In the above, the and operation is performed on both values and the output is 0010

Step-3: In this step, the output we got as 0010 now these binary data is converted into decimal and shown as output. Then if 0010 is converted into decimal the output will be 2. So the output of x & y is 2.

2) Bitwise OR operator (|):

This operators performs OR operation on the bits of numbers. The symbol for this operator is |, which is called pipe. To understand the bitwise OR operation. To understand in detail check the Truth Table.

x	y	x&y
0	0	0
0	1	1
1	0	1
1	1	1

Example:

X=10

Y=2

X | Y = 2

Step-1: In the above example The x value that means 10 is converted into binary data then 10 becomes as 1010 The y value that means 2 is converted into binary data then 2 becomes as 0010

Step-2: In this step, the and operation is performed on both x and y values which are converted into binary data that suggests

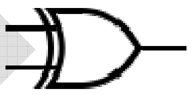
```
  1010
  0010
--------------
   1010
```

In the above, the and operation is performed on both values and the output is 1010

Step-3: In this step, the output we got as 1010 now these binary data is converted into decimal and shown as output. Then if 1010 is converted into decimal the output will be 10. So the output of x | y is 10.

2) Bitwise XOR operator (^):

This operators performs exclusive or (XOR) operation on the bits of numbers. The symbol for this operator is ^, which is called cap or circumflex. To understand the bitwise XOR operation in detail check the Truth Table.



x	y	x&y
0	0	0
0	1	1
1	0	1
1	1	0

Example:

X=10

Y=2

X | Y = 2

We can conclude that when we have odd number of 1's in the input bits we can get the output bit as 1.

Step-1: The x value that means 10 is converted into binary data then 10 becomes as 1010 The y value that means 2 is converted into binary data then 2 becomes as 0010

Step-2: In this step, the and operation is performed on both x and y values which are converted into binary data that suggests.

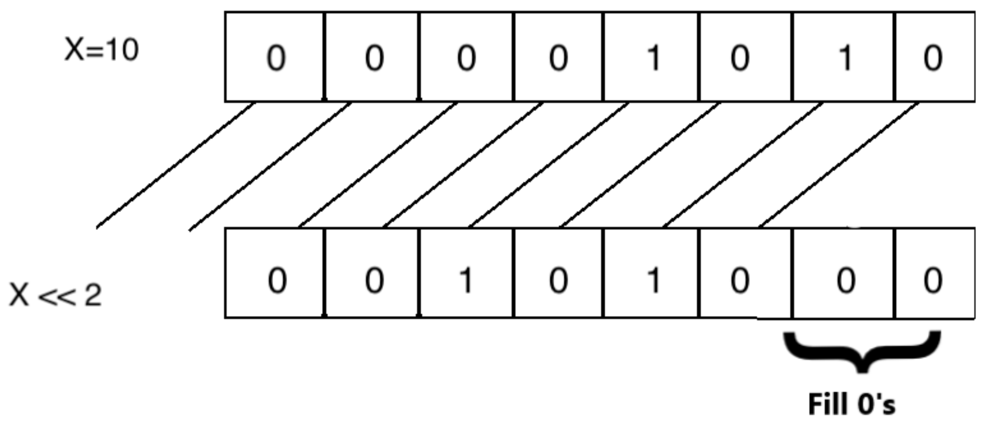
```
  1010
  0010
----------------
   1000
```

In the above, the and operation is performed on both values and the output is 1000

Step-3: In this step, the output we got as 1000 now these binary data is converted into decimal and shown as output. Then if 1000 is converted into decimal the output will be 8. So the output of x ^ y is 8.

3) Bitwise left shift operator (<<):

This operators shifts the bits of the number towards left a specified number of positions. The symbol for this operator is <<, read as double less than. If we write x<<\n,the meaning is to shift the bits of x towards left n position.



Example:

X=10

X<< 2 = 40

We can conclude that when we have odd number of 1's in the input bits we can get the output bit as 1.

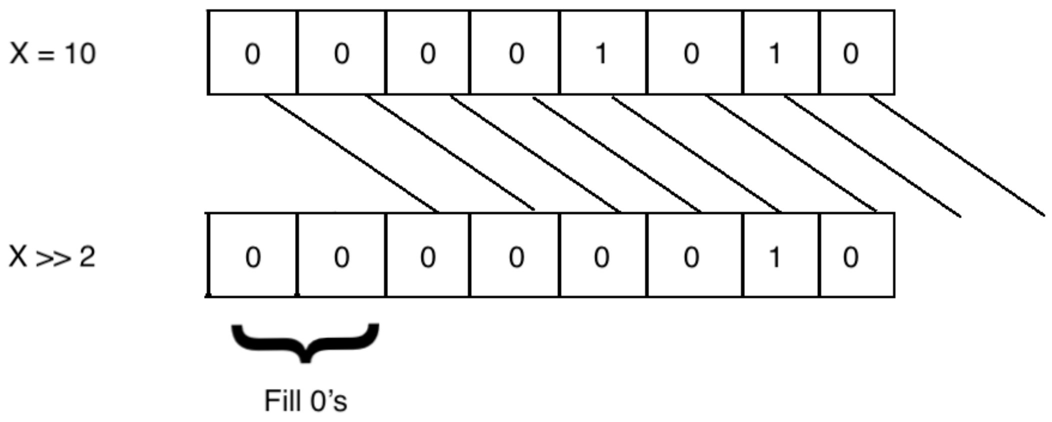
Step-1: The x value that means 10 is converted into binary data then 10 becomes as 1010 If we shift two digits to left then in the memory another two digits will be added.

Step-2: Then when we shift the bits to the left side with two digits data will be like 101000. Here two bits are added to right side when we shift two digits to the left side.

Step-3: the binary output 101000 is converted into decimal then the output will be 40.

5)Bitwise right shift operator (>>):

This operators shifts the bits of the number towards right a specified number of positions. The symbol for this operator is >>, read as double greater than. If we write x>>n,the meaning is to shift the bits of x towards right n position.



Example:

X=10

X>>2 = 2

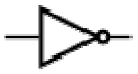
Step-1: The x value that means 10 is converted into binary data then 10 becomes as 1010 If we shift two digits to left then in the memory another two digits will be added. X>>2 If we shift two digits to right then in the memory another two digits will be added but the digits which are before there are exited from the memory in this scenario when we shift bits to the right side the existed bits will be deleted and the new bits will be replaced with it.

Step-2: Then when we shift the bits to the right side with two digits the binary data will be like 0010. See here two bits are added to the left side when we shift two digits to the right side.it means the existed two digits are got deleted and added new bits to the left side.

Step-3: Then the binary output 0010 is converted into decimal then the output will be 2.

6)Bitwise Negation Operator (~):

This operators gives the Negation from a given number. This operator symbol is '~', which is pronounced as tilde. This function form of a postive number can be obtained by changing 0's and 1's and vice cersa.



Example:

X=4

~~~~x

Output is 4

In the above example Negation is in even numbers so the input and output will be the same.

Example:

X=4

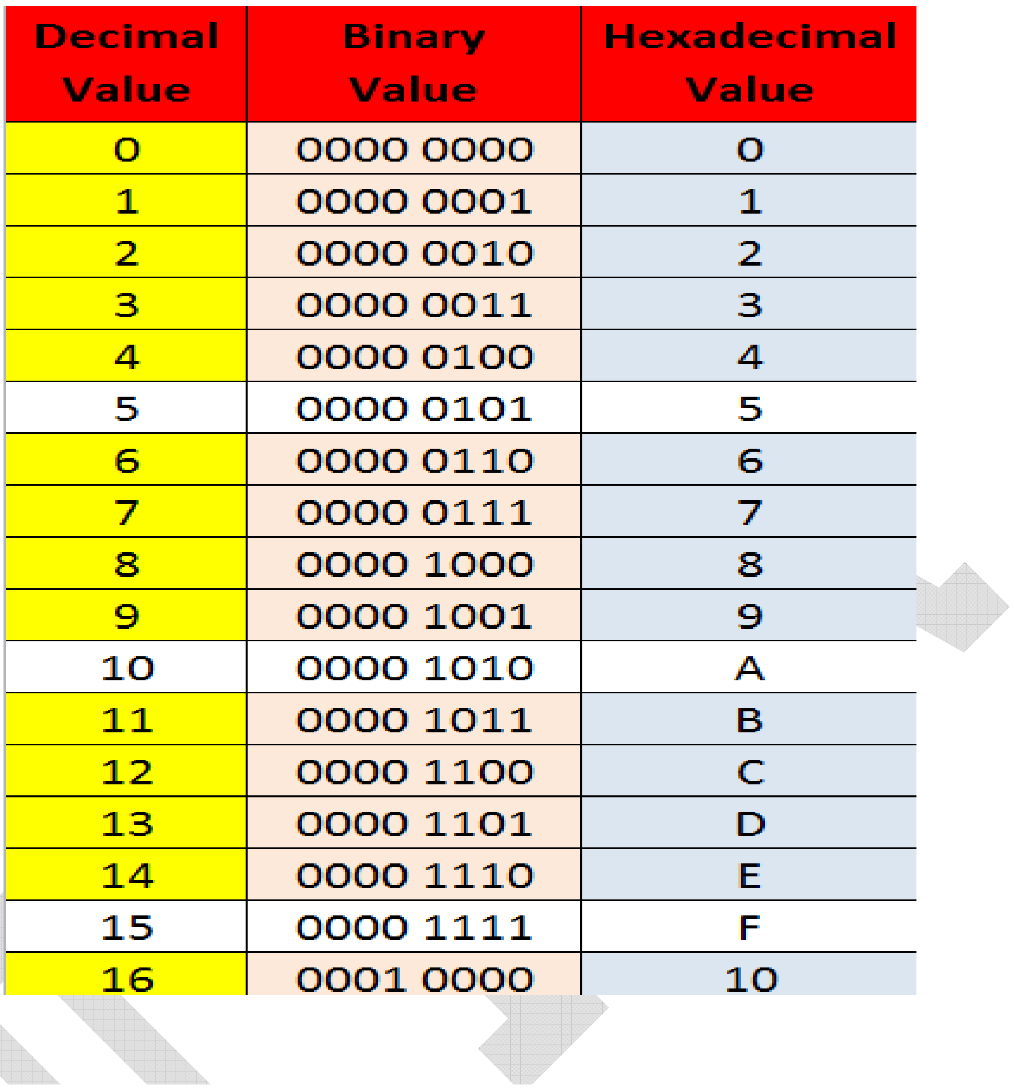
~~~~~x

Output is -5

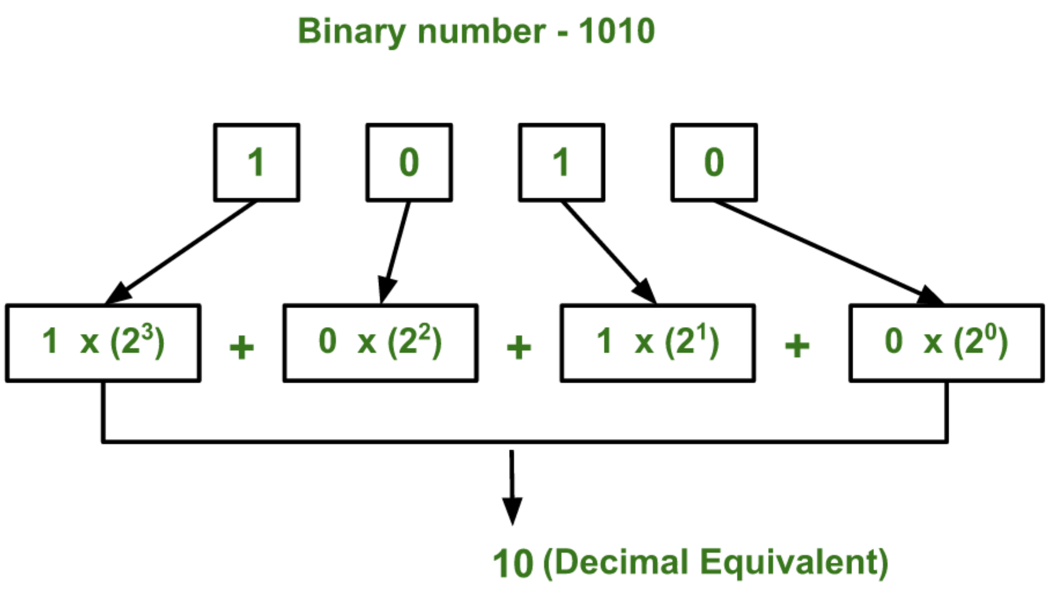
In the above example negations are in odd numbers then the output will be -5

x	y
0	1
1	0

Binary To Decimal Conversion



Decimal Value	Binary Value	Hexadecimal Value
0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
3	0000 0011	3
4	0000 0100	4
5	0000 0101	5
6	0000 0110	6
7	0000 0111	7
8	0000 1000	8
9	0000 1001	9
10	0000 1010	A
11	0000 1011	B
12	0000 1100	C
13	0000 1101	D
14	0000 1110	E
15	0000 1111	F
16	0001 0000	10


Binary number - 1010
1
0
1
0
1 x (2^3)
+
0 x (2^2)
+
1 x (2^1)
+
0 x (2^0)
10 (Decimal Equivalent)