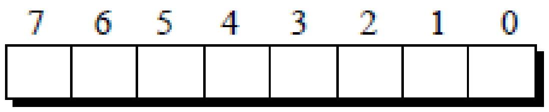
C Bitwise Operators

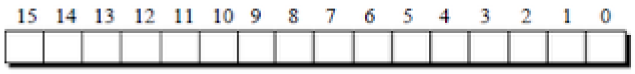
- The programming languages are byte oriented but the hardware are bit oriented. so bit level operationas are very important when the program interact directly with the hardware.
- C provides **6 bitwise operators** for bit manipulation.
- Bitwise operators can only applied to **integral operands such as char,short,int and long**. these may be **signed or unsigned.**
- Bitwise operators can not be applied on **floats and doubles**.

Operator Name	Operator Symbol
bitwise AND	&
bitwise inclusive OR	|
bitwise exclusive OR	^
one's complement(unary)	~
left shift	<<
right shift	>>

- Consider the bit numbering scheme. The bits are numbered from zero onwards,increasing from right to left
- for character,the bit representation



- For integer(considering int are 16 bits),the bit representation,



- The bitwise operators are applied to binary numbers ie number consist of 0 and 1. so the integer or char(ascii value) number is converted to binary number for operation.
- The program for converting an decimal number to binary number is given below
- Program to print the binary equivalent of a decimal number

1.Bitwise AND(&)

- The & operator is a binary operator so it requires 2 operands.
- When the AND operation is performed the 2 operands are compared on a bit by bit basis.So both the operands must be of the same data type (ie both the operands are char or int)
- The rules for the & operator is

First bit	Second bit	result=First bit & Second bit
0	0	0
0	1	0
1	0	0
1	1	1

- Example:

consider a=10 and b=6. to find a&b

```
8bit binary value of a-->0000 1010
8bit binary value of b-->0000 0110
result of a&b         -->0000 0010.
```

Each bit of a is compared with the corresponding bit in b.If both the bits are set as 1 then the result is 1 else the result is set as 0.Thus the result of a&b=2.
Note:The operation is performed on individual bits and the operation performed on one pair of bits is completely independent of the operation performed on the other pairs.

- **How to turn off a particular bit? The & operator is used to turn off a particular bit**
- **The & operator is often used to mask off some set of bits**.The second operand is called an AND mask.ie using AND operator we can switch off a particular bit..To switch off the 3rd bit the AND mask is created with setting 0 on the 3rd bit and all remaining bit are set as 1.

ie AND mask=1111 1011
Example

```
binay value of 12--> 0000 1100
AND mask         --> 1111 1011
Result           --> 0000 1000(3rd bit is now off)
```

- **The & operator also used to check whether a particular bit in a number is ON(1) or OFF(0)**.To find a particular bit for example 3rd bit is ON or OFF the AND mask is 2^3.ie 8. The binary value of 8 is 0000 1000.The original value is & with this AND mask. In the result if the 3rd bit is 1 then the 3rd bit in the original number is ON. if its 0 then the 3rd bit in the original number is OFF.

Example:check whether 2nd bit of decimal number 6 is On or OFF.The AND mask is 2^2 ie 4(0000 0100)

```
binary value of 6-->0000 0110
AND mask         -->0000 0100
Result is        -->0000 0100.
```

The result has 3rd bit as 1. So the 3rd bit on the original number is set as ON.

2.Bitwise OR (or) Bitwise Inclusive OR(|)

- The Bitwise OR operator is also a binary operator so it requires 2 operands.
- The rules for the bitwise OR operator is

First bit	Second bit	result=First bit | Second bit
0	0	0
0	1	1
1	0	1

1	1	1

- Example: consider a=10 and b=6. to find a | b

```
8bit binary value of a-->0000 1010
8bit binary value of b-->0000 0110
result of a|b         -->0000 1110.
```

Each bit of a is compared with the corresponding bit in b.If either of the bits are set as 1 then the result is 1.If both the bits are 0 then only the result is set as 0.Thus the result of a | b=13.

- **How to turn on a particular bit? Bitwise OR operator is used to turn bits ON**. To turn ON a particular bittThe OR mask is created with the particular bit as 1.

Example

If you want to set 3rd bit as 1 in the given number 0000 0111(decimal 7) then the OR mask=2^3 ie 8(0000 1000). This OR mask is OR with original number 0000 0111.

```
Binary value of 7-->0000 0111
OR mask          -->0000 1000
Result           -->0000 1111.
```

if the bit is already ON the OR operation wont alter the value since 1 | 1=1. If the bit is off then the bit is set to ON.

3.Bitwise XOR (or) Bitwise Exclusive OR (^)

- The XOR operator is a binary operator so it requires 2 operands.
- The rules for the bitwise XOR is

First bit	Second bit	result=First bit ^ Second bit
0	0	0
0	1	1
1	0	1
1	1	0

- XOR operator returns 1 only if one of the two bit is 1.else 0

Example

consider a=10 and b=6. to find a ^ b.

```
8bit binary value of a-->0000 1010
8bit binary value of b-->0000 0110
result of a^b         -->0000 1100.
```

Each bit of a is compared with the corresponding bit in b.If either of the bits are set as 1 then the result is 1.If both the bits are 0 or both the bits are 1 then the result is set as 0.Thus the result of a ^ b=12.

- **XOR operator is used to toggle a bit ON or OFF**.ie if the bit is 1 set to 0 and if the bit is 0 set to 1.To toggle bits 2 and 3 the XOR mask is created by setting 1 to the 2 and 3rd bit ie 0000 1100. To toggle 0th bit the XOR mask is 0000 0001.

Example

To toggle first 4 bits in the decimal number 65

The XOR mask for toggle 1st 4 bits=0000 1111

```
binary value of 65      -->0100 0001
XOR mask                -->0000 1111
Result of 65 ^ XOR mask -->0100 1110
```

- **A number XORed with another number twice gives the original number.**

Exampleconsider a=20 b=10.

```
binary value of a   -->0001 0100
binary value of b   -->0000 1010
result of a ^ b     -->0001 1110.
```
now the result is again XOR with b.
```
result of a^b      -->0001 1110
binary value of b  -->0000 1010
result of a^b^b    -->0001 0100
```
(20 ie value of a)

4.Ones's Complement Operator(~)

- The one's complement operator is a unary operator. So it requires only one operand
- In one's complement of a number, all 1's present in the number are changed to 0'sand all 0's are changed to 0's.
- The rules are

bit	~bit
0	1
1	0

- Example:if a =0000 0000(decimal 0) ones complement of a ie ~a= 1111 1111(decimal 255). If a=1111 1111 then ~a=0000 0000.

5.Left Shift Operator(<<)

- The left shift operator is a binary operator so it requires 2 operands namely left operand and right operand
- Left shift operator shifts each bit in the left operand to the left. The number of places the bits are shifted depends on the right operand.
- So A << 1 would shift all bits in A 1 place to the left.A << 3 shift all bits of A in 3 places to the left.
- When the bits are shifted to the left ,blanks are created on the right.These blanks are always filled with zeros.
- Example: if A=65 then binary value of A=0100 0001.

A << 1-->1000 001*
Now the blank is filled with 0. so A << 1 --> 1000 0010
A << 2--> 0000 01** so A << 2-->0000 0100
A << 3-->0000 1000

- **Note:**

1 left shifting is equivalent to multiplication of left operand by $2^{\text{right operand}}$ (ie)**A << B =A * 2^B**

Example1:
binary value of 32-->0010 0000
32 << 1 -->0100 0000(decimal 64 ie 32*2)
32 >> 2 --> 1000 0000(decimal 128 ie 32 * 2^2)
Example2
binary value of 25 -->0001 1001
25 << 1 -->0011 0010(decimal 50 ie 25*2)
25 >> 2 -->0110 0100(decimal 100 ie 25*2^2)

6.Right Shift Operator(>>)

- The right shift operator is a binary operator so it requires 2 operands namely left operand and right operand
- Right shift operator shifts each left operand to the right. The number of places the bits are shifted depends on the right operand.
- So A >> 1 would shift all bits in A 1 place to the right.A >> 3 shift all bits of A in 3 places to the right.
- When the bits are shifted to the right ,blanks are created on the left.These blanks are always filled with zeros.
- Example: if A=65 then binary value of A=0100 0001.

  A >> 1-->0010 0000
  A >> 2-->0000 1000
  A >> 3-->0000 0001

- Note:
  1. If the left operand is a multiple of 2 then shifting the operand to right is same as dividing it by $2^{\text{right operand}}$

     Example:
     binary value of 128-->1000 0000
     128 >> 1 -->0100 0000(decimal 64 ie 128/2)
     128 >> 2 --> 0010 0000(decimal 32 ie 128/22)
     128 >> 3 -->0001 0000(decimal 16 ie 128/23)
  2. If the left operand is not a multiple of 2 then shifting the operand to right is same as dividing it by $2^{\text{right operand}}$ and ignoring the remainder.

  Example
  binary value of 25 -->0001 1001
  25 >> 1 -->0000 1100(decimal 12 ie 25/2 discarding remainder 1)
  25 >> 2 -->0000 0110(decimal 6 ie 25/4)

- In A >> B, if B is negative the result is unpredictable.
- In A >> B, if A is negative then A's left most bit ie sign bit is 1
- Logical Shift and Arithmetic Shift:
  1. In Logical Shift the high order bits are filled with zero
  2. In Arithmetic shift, the high order bits are filled with the original sign bits. So if the sign bit is 1 then all bits are filled with 1's otherwise they are filled with 0's
  3. For unsigned data types logical shift is used,and for signed data types arithmetic shift is used.

Source:

http://datastructuresprogramming.blogspot.in/2010/03/c-bitwise-operators.html