Problem Solving Through programming in C

Course Code:ONL1001

Bitwise operators

Ms. Shubhra dwivedi
Assistant Professor
School - SCOPE
VIT-AP Amaravati

What is bitwise structure?

- The smallest type is of 8 bits (char). Sometimes we need only a single bit.
- For instance, storing the status of the lights in 8 rooms:

- We need to define an array of at least 8 chars. If the light of room 3 is turned on the value of the third char is 1, otherwise 0.
- Total array of 64 bits.

EXPENSIVE in place and time!!!

What is bitwise structure?

- It is better to define only 8 bits since a bit can also store the values 0 or 1.
- But the problem is that there is no C type which is 1 bit long (char is the longer with 1 byte).
- Solution: define a char (8 bits) but refer to each bit separately.
- Bitwise operators, introduced by the C language, provide one of its more powerful tools for using and manipulating memory. They give the language the real power of a “low-level language”.
- Accessing bits directly is fast and efficient, especially if you are writing a real-time application.

Bitwise Operators

- In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

- The language introduces the bitwise operators, which help in manipulating a single bit of a byte. bitwise operators may be used on integral types only (unsigned types are preferable).

Bitwise Operators

&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	1's compliment
<<	Shift left
>>	Shift right

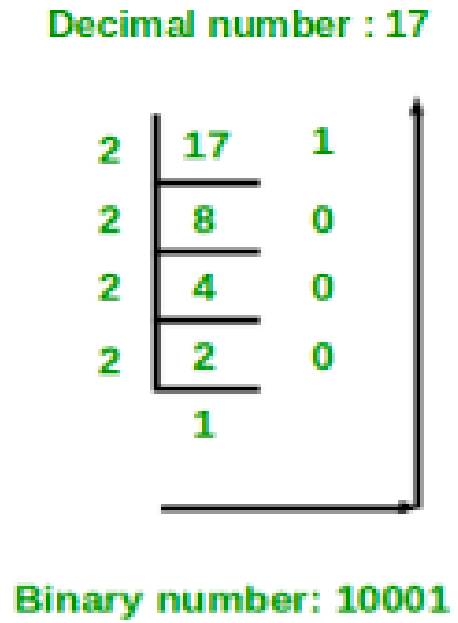
All these operators can be suffixed with =
For instance a &= b; is the same as a = a & b;

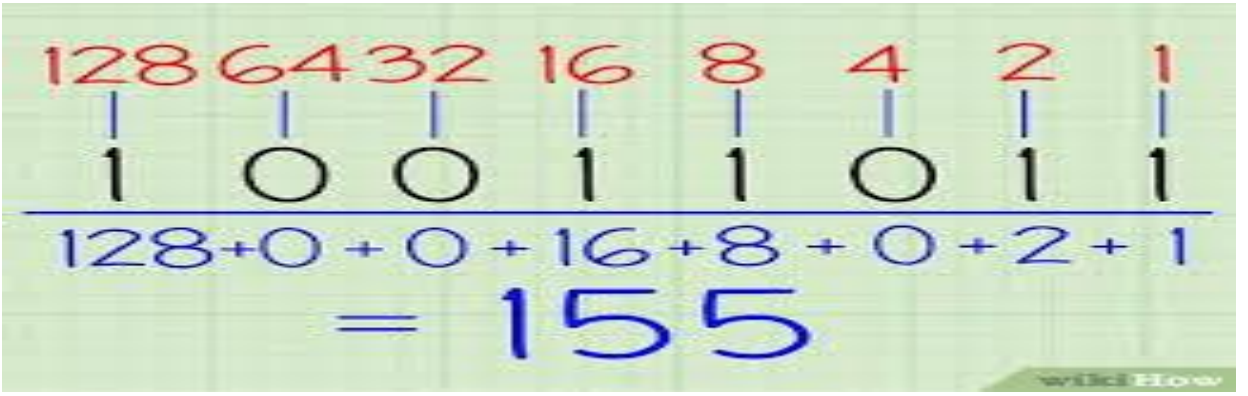
Bitwise operators: truth table

a	**b**	**a&b**	**a|b**	**a^b**	**~a**
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Decimal to Binary conversion

Decimal	Binary
1	0
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010





Bitwise AND operator &

- The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

- 12 = 00001100 (In Binary)
- 25 = 00011001 (In Binary)

- Bit Operation of 12 and 25
- 00001100 & 00011001
- 00001000 = 8 (In decimal)

Example 1: Bitwise AND

Example #1: Bitwise AND

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a&b);
   return 0;
}
```

Output

Output = 8

Bitwise OR operator |

- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

- 12 = 00001100 (In Binary)
- 25 = 00011001 (In Binary)

- Bitwise OR Operation of 12 and 25
- 00001100| 00011001
- 00011101 = 29 (In decimal)

Example 2: Bitwise OR

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a|b);
   return 0;
}
```

Output

Output = 29

Bitwise XOR (exclusive OR) operator ^

- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

- 12 = 00001100 (In Binary)
- 25 = 00011001 (In Binary)

- Bitwise XOR Operation of 12 and 25
- 00001100^ 00011001
- 00010101 = 21 (In decimal)

Example 3: Bitwise XOR

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a^b);
   return 0;
}
```

Output

Output = 21

Bitwise complement operator ~

- Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

- 35 = 00100011 (In Binary)

- Bitwise complement Operation of 35
- ~ 00100011
- 11011100 = 220 (In decimal)

Twist in bitwise complement operator in C Programming

- The bitwise complement of 35 (~35) is -36 instead of 220, but why?
- For any integer n, bitwise complement of n will be -(n+1). To understand this, you should have the knowledge of 2's complement.

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

Decimal	Binary	2's complement
0	00000000	-(11111111+1) = -00000000 = -0(decimal)
1	00000001	-(11111110+1) = -11111111 = -255(decimal)
12	00001100	-(11110011+1) = -11110100 = -244(decimal)
220	11011100	-(00100011+1) = -00100100 = -36(decimal)

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise complement of any number N is -(N+1). Here's how:

bitwise complement of N = ~N (represented in 2's complement form)
2'complement of ~N= -(~(~N)+1) = -(N+1)

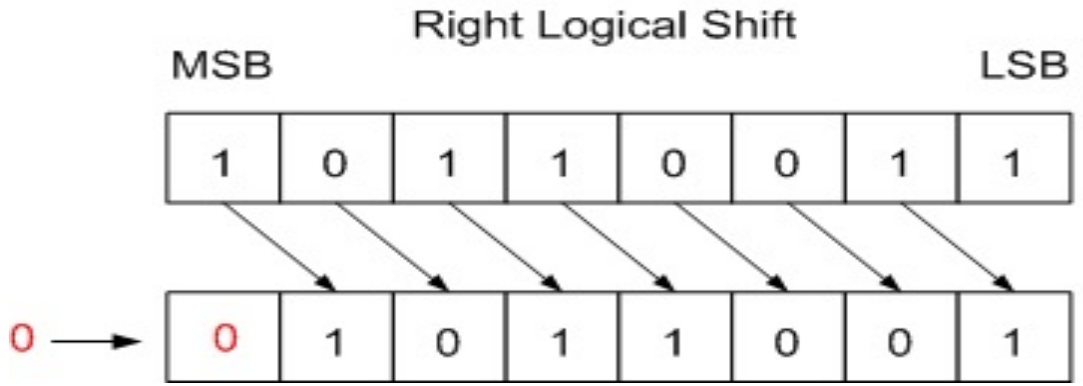
Example 4: Bitwise complement

```
#include <stdio.h>
int main()
{
   printf("Output = %d\n",~35);
   printf("Output = %d\n",~-12);
   return 0;
}
```

Output
Output = -36
Output = 11

Right Shift Operator

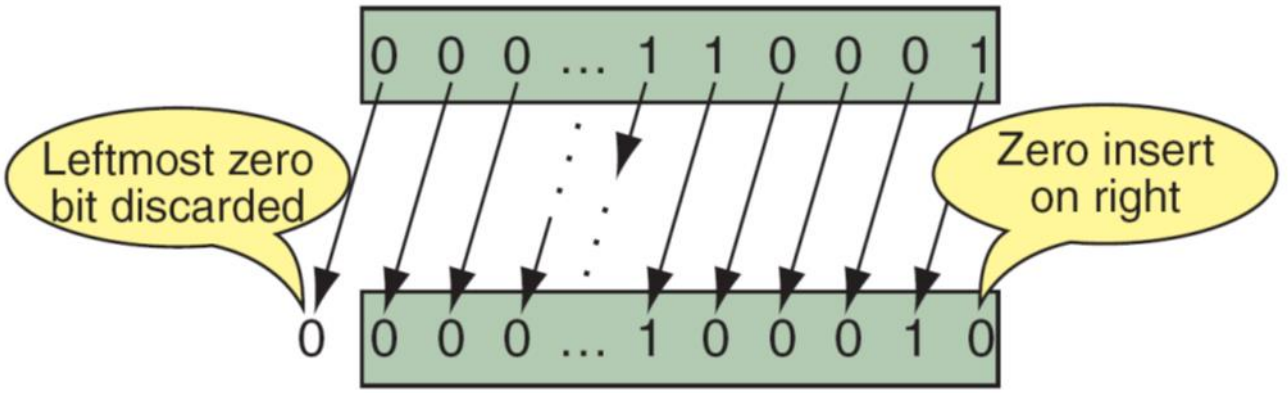
- Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.



- 212 = 11010100 (In binary)
- 212>>2 = 00110101 (In binary) [Right shift by two bits]
- 212>>7 = 00000001 (In binary)
- 212>>8 = 00000000
- 212>>0 = 11010100 (No Shift)

Left Shift Operator

- Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is <<.



- 212 = 11010100 (In binary)
- 212<<1 = 110101000 (In binary) [Left shift by one bit]
- 212<<0 = 11010100 (Shift by 0)
- 212<<4 = 110101000000 (In binary) =3392(In decimal)

1. The left shift and right shift operators should not be used for negative numbers. The result of is undefined behaviour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined.

2. If the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits. See this for more details.

3. The left-shift by 1 and right-shift by 1 are equivalent to the product of first term and 2 to the power given element(1<<3 = 1*pow(2,3)) and division of first term and second term raised to power 2 (1>>3 = 1/pow(2,3)) respectively.

As mentioned in point 1, it works only if numbers are positive.

```
#include<stdio.h>
int main()
{
   int x = 19;
   printf ("x << 1 = %d\n", x << 1);
   printf ("x >> 1 = %d\n", x >> 1);
   return 0;
}
```

Output:

```
x << 1 = 38
x >> 1 = 9
```

The left-shift of 1 by i is equivalent to 2 raised to power i. As mentioned in point 1, it works only if numbers are positive.

```
#include<stdio.h>
int main()
{
  int i = 3;
  printf("pow(2, %d) = %d\n", i, 1 << i);
  i = 4;
  printf("pow(2, %d) = %d\n", i, 1 << i);
  return 0;
}
```

Output:

```
pow(2, 3) = 8
pow(2, 4) = 16
```

Example 5: Shift Operators

```
#include <stdio.h>
int main()
{
   int num=212, i;
   for (i=0; i<=2; ++i)
      printf("Right shift by %d: %d\n", i, num>>i);

    printf("\n");

    for (i=0; i<=2; ++i)
      printf("Left shift by %d: %d\n", i, num<<i);

    return 0;
}
```

```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

Thank you