Operators And Expression
Bit wise Operators

Dr. Sheak Rashed Haider Noori
Professor & Associate Head
Department of Computer Science

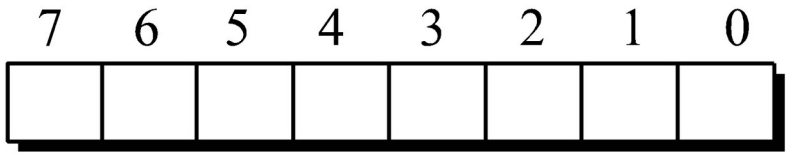
Bit wise Operators

- One of C's powerful features is a set of bit manipulation operators.
- These permit the programmer to access and manipulate individual bits within a piece of data to perform bit operations.
- Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
- These operators can operate upon **int**s and **char**s but not on **float**s and **double**s.

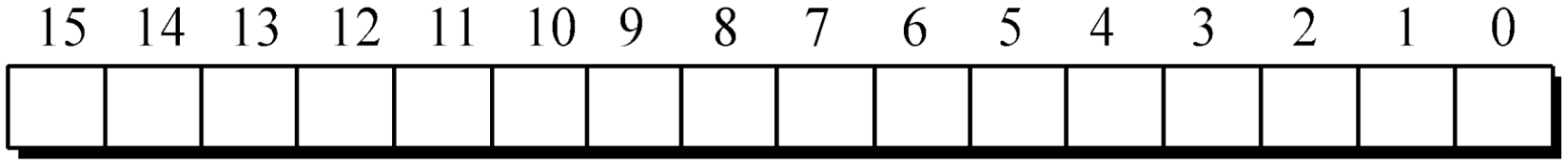
Operator	Meaning
~	One's complement
>>	Right shift
<<	Left shift
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR(Exclusive OR)

Bit wise Operators

- Let us first take a look at the bit numbering scheme in integers and characters. Bits are numbered from zero onwards, increasing from right to left as shown below:



Character



16-bit Integer

One's Complement Operator (~)

- One's complement operator is represented by the symbol ~
- On taking one's complement of a number, all 1's present in the number are changed to 0's and all 0's are changed to 1's.
- For example one's complement of 1010 is 0101. Similarly, one's complement of 1111 is 0000. Note that here when we talk of a number we are talking of binary equivalent of the number.
- Thus, one's complement of 65 means one's complement of 0000 0000 0100 0001, which is binary equivalent of 65. One's complement of 65 therefore would be, 1111 1111 1011 1110.

Use of one's complement operator

- In real-world situations where could the one's complement operator be useful?
- Since it changes the original number beyond recognition, one potential place where it can be effectively used is in development of a file encryption utility

Right Shift Operator (>>)

- The right shift operator is represented by >>.
- It needs two operands. It shifts each bit in its left operand to the right.
- The number of places the bits are shifted depends on the number following the operator (i.e. its right operand).
- Thus, **ch >> 3** would shift all bits in **ch** three places to the right. Similarly, **ch >> 5** would shift all bits 5 places to the right.
- For example, if the variable **ch** contains the bit pattern 11010111, then, **ch >> 1** would give 01101011 and **ch >> 2** would give 00110101.
- Note that as the bits are shifted to the right, blanks are created on the left. These blanks must be filled somehow. They are always filled with zeros.

Left Shift Operator (<<)

- The left shift operator is represented by <<.
- This is similar to the right shift operator, the only difference being that the bits are shifted to the left, and for each bit shifted, a 0 is added to the right of the number.

Bitwise AND Operator (&)

- This operator is represented as **&**.
- Remember it is different than **&&**, the logical AND operator.
- The **&** operator operates on two operands. While operating upon these two operands they are compared on a bit-by-bit basis. Hence both the operands must be of the same type (either **char** or **int**).
- The second operand is often called an AND mask. The **&** operator operates on a pair of bits to yield a resultant bit.

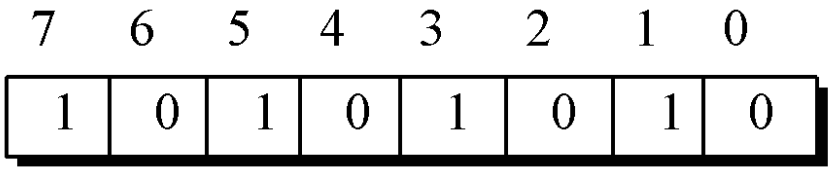
Bitwise AND Operator (&)

- The rules that decide the value of the resultant bit are shown below:

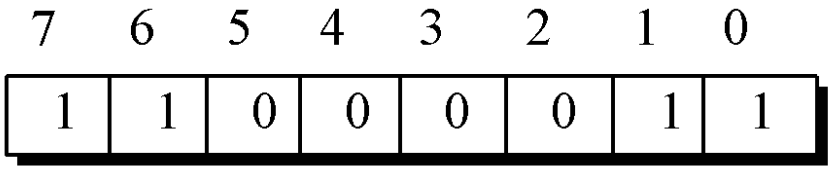
First bit	Second bit	First bit & Second bit
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise AND Operator (&)

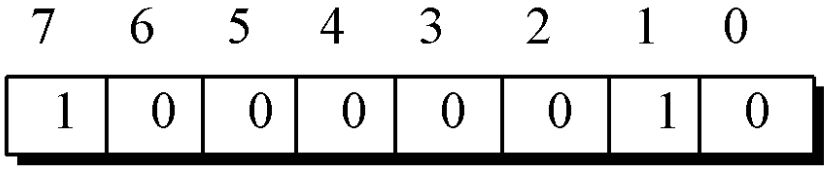
- The example given below shows more clearly what happens while ANDing one operand with another.



This operand when ANDed bitwise



With this operand yields



this result

Bitwise AND Operator (&)

- Probably, the best use of the AND operator is to check whether a particular bit of an operand is ON or OFF. This is explained in the following example.
- Suppose, from the bit pattern 10101101 of an operand, we want to check whether bit number 3 is ON (1) or OFF (0).

Then the ANDing operation would be,

```
10101101        Original bit pattern
00001000        AND mask
--------------
00001000        Resulting bit pattern
```

Bitwise AND Operator (&)

- In every file entry present in the directory, there is an attribute byte. The status of a file is governed by the value of individual bits in this attribute byte. The AND operator can be used to check the status of the bits of this attribute byte. The meaning of each bit in the attribute byte is shown in Figure

Bit numbers								Meaning
7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	1	Read only
.	.	.	.	.	.	1	.	Hidden
.	.	.	.	.	1	.	.	System
.	.	.	.	1	.	.	.	Volume label entry
.	.	.	1	.	.	.	.	Sub-directory entry
.	.	1	.	.	.	.	.	Archive bit
.	1	.	.	.	.	.	.	Unused
1	.	.	.	.	.	.	.	Unused

Bitwise AND Operator (&)

- Now, suppose we want to check whether a file is a hidden file or not. A hidden file is one, which is never shown in the directory, even though it exists on the disk.
- From the above bit classification of attribute byte, we only need to check whether bit number 1 is ON or OFF.
- So, our first operand in this case becomes the attribute byte of the file in question, whereas the second operand is the $1 * 2^1 = 2$.
- Similarly, it can be checked whether the file is a system file or not, whether the file is read-only file or not, and so on.
- The second important use of the AND operator is in changing the status of the bit, or more precisely to switch OFF a particular bit.

Bit wise AND operator

- Let's summarize the uses of bitwise AND operator:
  1. It is used to check whether a particular bit in a number is ON or OFF.
  2. It is used to turn OFF a particular bit in a number.

Bitwise XOR Operator (^)

- The XOR operator is represented as ^ and is also called an Exclusive OR Operator.
- The OR operator returns 1, when any one of the two bits or both the bits are 1, whereas XOR returns 1 only if one of the two bits is 1.
- XOR operator is used to toggle a bit ON or OFF.
- The truth table for the XOR operator is given below.

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Truth table for bit wise operations

x	y	x|y	x & y	x ^ y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bit wise Operators

```c
#include <stdio.h>

int main()
{
   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;          /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15

Example program for bit wise operators in C

```c
#include <stdio.h>
int main()
{
    int m=40,n=80,AND_opr,OR_opr,XOR_opr,NOT_opr ;

    AND_opr = (m&n);
    OR_opr = (m|n);
    NOT_opr = (~m);
    XOR_opr = (m^n);

    printf("AND_opr value = %d\n",AND_opr );
    printf("OR_opr value = %d\n",OR_opr );
    printf("NOT_opr value = %d\n",NOT_opr );
    printf("XOR_opr value = %d\n",XOR_opr );
    printf("left_shift value = %d\n", m << 1);
    printf("right_shift value = %d\n", m >> 1);
}
```

Output:
AND_opr value = 0
OR_opr value = 120
NOT_opr value = -41
XOR_opr value = 120
left_shift value = 80
right_shift value = 20

Summary Bit wise Operators

1. To help manipulate hardware oriented data—individual bits rather than bytes a set of bitwise operators are used.
2. The bitwise operators include operators like one's complement, right-shift, left-shift, bitwise AND, OR, and XOR.
3. The one's complement converts all zeros in its operand to 1s and all 1s to 0s.
4. The right-shift and left-shift operators are useful in eliminating bits from a number—either from the left or from the right.
5. The bitwise AND operators is useful in testing whether a bit is on/off and in putting off a particular bit.
6. The bitwise OR operator is used to turn on a particular bit.
7. The XOR operator works almost same as the OR operator except one minor variation.