Bitwise and Boolean Operators

Bitwise Operators

Generally operands are one or two numbers (could be ASCII characters too) represented in binary with the same no of bits

(i) Bitwise NOT (also called 1’s complement) Symbol ~ (Tilde) Unary operator

e.g. ~0b10101010 = 0b01010101

(ii) Bitwise OR Symbol | Binary operator

e,g, 0b01101001 | 0b10000101

01101001

10000101

11101101

(iii) Bitwise AND Symbol & Binary operator

e,g, 0b01101001 & 0b10000101

01101001

10000101

00000001

(iv) Bitwise XOR Symbol ^ Binary Operator

e.g. 0b01101001 ^ 0b10000101

01101001

10000101

11101100

Bitwise NOR,NAND and XNOR do not have their own symbols

They must be computed as ~(X|Y) ~(X&Y) and ~(X^Y) respectively

Note: a paranthesis is needed in each case. Without it, the unary operator ~ has precedence and is computed first and then the binary operators |, & , ^ giving totally different results

Setting and clearing only certain bits in a byte without affecting the status of other bits

To set selected bits of the byte prepare a mask having 1 in the selected positions and 0 in all other positions and perform a bitwise OR with the byte

E,g to set bits 4 and 1 mask is 0b00010010 . Bits 4 and 1 will be ORed with 1 and will become 1 whether they were 0 or 1 before the mask operation. All other bits will be ORed with 0 and retain their status

To clear certain bits of the byte prepare a mask with 0 in the selected positions and 1 in all other positions and perform bitwise AND with the byte

e.g to clear bits 6,3 and 0 the mask is ~01001001= 0b10110110. Bits 6,3,0 will be ANDed with 0 and turn to 0 whether they were 0 or 1 before masking. All other bits will be ANDed with 1 and their status will remain unchanged

To toggle certain bits of the byte prepare a mask with 1 in the selected positions and 0 in all other positions and perform bitwise XOR with the byte

e.g to toggle bits 6,3 and 0 the mask is 01001001. Bits 6,3,0 will be XORed with 1 and will toggle. All other bits will be XORed with 0 and will remain unchanged

Testing whether a specified bit is 0 or 1

To test whether a specified bit of a byte is 0 or 1 prepare a mask with 1 in the selected position and 0 in all other positions and perform a bitwise AND with the byte. If the specified bit were 0 , the whole byte will be 0 after ANDing with the mask. On the other hand if the selected bit were 1 the byte will be non-zero after ANDing with the mask

If x is the byte and M is the mask if(x&M==0) is the test for the specified bit being 0

If (x&M!=0) is the test for the specified bit being 1

Boolean Operators

Operands are Boolean (TRUE or FALSE) . C language does not support Boolean types. They are declared as unsigned integers 0 being FALSE and 1 being TRUE. More generally any value other than 0 is treated as TRUE. The 0 and 1 here are distinct from the ‘0’ and ‘1’ in circuits which are two distinct voltage levels whose actual values depend on the type of components used e.g. ‘1’ may correspond to 3.3V or 5V. ‘0’ sometimes represents a negative voltage

Boolean NOT Symbol ! Unary operator

Changes 0 to 1 and 1 (or any non-zero value) to 0

Boolean OR Symbol || Binary operator

Follows rules of Boolean Algebra

0 || 0 is 0 . 0||1 is 1, 1||0 is 1 , 1||1 is 1 (any nonzero value in the operands is treated like 1 ; result will be always 0 or 1 and no other value)

Boolean AND Symbol && Binary operator

Follows rules of Boolean Algebra

0 && 0 is 0 . 0&&1 is 0, 1&&0 is 0 , 1&&1 is 1 (any nonzero value in the operands is treated like 1 ; result will be always 0 or 1 and no other value)

Note: There is no such thing like Boolean XOR ^^. If required it can be expressed In terms of !, || , && like (!A&&B) || (A&&!B) . Note the positions of the parantheses

Relational operators

There are 6 binary operators

Equal to ==

Not equal to !=

Greater than >

Greater than or Equal to >=

Less than <

Less than or Equal to <=

All relational operators return a Boolean value 0 or 1

e.g

c=(a<b) with a=7 and b=5 returns 0 and makes c=0

The result of any relational operator can be used in ‘if’ and ‘while’ statements to set up branches and loops in the code

e.g if(a<b)

{

S1;

}

else

{

S2;

}

The operator < returns 0 and if statement branches to code block S2

while(a<b)

{

loop

}

The operator < returns 0 and the code exits the loop

Compex branch and loop conditions can be created by applying Boolean operators to the values retuned by multiple relational operators e.g

if((a>b) && (p<q))

{

S1;

}

else

{

S2;

}