Bitwise operators and instructions

In computer programming, a bitwise operation operates on a bit string, a bit array or a binary numeral at the level of its individual bits. It is a fast and simple action, basic to the higher-level arithmetic operations and directly supported by the processor.

Pay attention to the difference between operators and instructions !!!

Mov ah, 01110111b << 3 ; AH :=10111000b

Vs.

Mov ah, 01110111b
Shl ah, 3

In the descriptions below x represents ONE BIT, 0 and 1 represent bit values and ~x represents the complementary value of the value bit x. The descriptive sequences below exemplify the mode of action of **the AND, OR and XOR operations** AT THE BIT LEVEL as the MECHANISM of action, regardless of whether the respective operation is triggered at the source code level by the respective OPERATOR or by the corresponding INSTRUCTION.

& - bitwise AND operator	x AND 0 = 0	; x AND x = x
AND – instruction	x AND 1 = x	; x AND ~x = 0

Operation useful for FORCING the values of certain bits to 0 !!!!

| - bitwise OR operator	x OR 0 = x	; x OR x = x
OR – instruction	x OR 1 = 1	; x OR ~x = 1

Operation useful for FORCING the values of certain bits to 1 !!!!

^ - bitwise EXCLUSIVE OR operator;	x XOR 0 = x	x XOR x = 0
XOR – instruction	x XOR 1 = ~x	x XOR ~x = 1

Operation useful for COMPLEMENTING the value of some bits !!!

XOR ax, ax ; AX=0 !!! = 00000000 0000000b

Operator ! (NOT)

In C - !0 = 1 (0 = false, anything different from 0 = TRUE, but a predefined function will set TRUE =1)

In ASM - !0 = same as in C, so ! - Logic Negation: !X = 0 when X ≠ 0, otherwise = 1

a d?....
b d?...

Mov eax, ![a] - because [a] is not something computable/determinable at assembly time, this instruction will issue a syntax error ! – (expression syntax error)

Mov eax, [!a] - ! can only be applied to SCALAR values !! (a = pointer data type ≠ scalar !)

Mov eax, !a - ! can only be applied to SCALAR values !!

Mov eax, !(a+7) - ! can only be applied to SCALAR values

Mov eax, !(b-a) – ok ! because a,b – pointers, but b-a = SCALAR !
Mov eax, ![a+7] - expression syntax error
Mov eax, !7 - EAX = 0
Mov eax, !0 – EAX = 1

Mov eax, !ebx ; syntax error !

```
aa equ 2
mov ah, !aa   ; AH=0
```

Logical bitwise INSTRUCTIONS (AND, OR, XOR and NOT instructions).

AND is recommended for forcing the value of some bits to 0 (useful for isolating certain bits).

OR is suitable for forcing certain bits to 1.

XOR is suitable for complementing the value of some bits.

NOT is used for complementing the operand's contents (reg/mem).

Shifts and rotates.

Bit shifting instructions can be classified in the following way:

- Logic shifting instructions
  - left - **SHL**
  - right - **SHR**
- Arithmetic shifting instructions
  - left - **SAL**
  - right - **SAR**

Bit rotating instructions can be classifed in the following way:

- Rotating instructions without carry
  - left - **ROL**
  - right - **ROR**
- Rotating instructions with carry
  - left - **RCL**
  - right - **RCR**

For giving a suggestive definition for shifts and rotates let's consider as an initial configuration one byte having the value X = abcdefgh, where a-h are binary digits, h is the least significant bit, bit 0, a is the most significant one, bit 7, and k is the actual value from CF (CF=k). We then have:

```
SHL X,1 ;has the effect X = bcdefgh0 and CF = a
SHR X,1 ;has the effect X = 0abcdefg and CF = h
SAL X,1 ; identically to SHL
SAR X,1 ;has the effect X = aabcdefg and CF = h
ROL X,1 ;has the effect X = bcdefgha and CF = a
ROR X,1 ;has the effect X = habcdefg and CF = h
RCL X,1 ;has the effect X = bcdefghk and CF = a
RCR X,1 ;has the effect X = kabcdefg and CF = h
```