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
```

equivalent

---

1 & 1 = 1
0 & 0 = 1
1 & 0 = 0
0 & 1 = 0

& - 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 !!!!

1 | 0 = 1
0 | 1 = 0
1 | 1 = 1
0 | 0 = 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 !!!!

0 ^ 0 = 1
1 ^ 1 = 0
1 ^ 0 = 1
0 ^ 1 = 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 !!!

0 1 0 0 1 1 1 0 ^
1 1 1 1 1 1 1 1

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

0111 ^ ~ 0111
1
1000 = 8
F8

0 ^ 1 = 1
1 ^ 1 = ~ ~ 0 ~ 0 0 0 0
1 1 1 1

Operators ! and ~ usage

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

~ 1's Complement: mov al, ~0 => mov AL, 0ffh (bitwise operator !)
(because a 0 in asm is a binary ZERO represented on 8, 16, 32 or 64 bits the logical BITWISE negation – 1's complement - will issue a binary 8 of 1's, 16 of 1's, 32 of 1's or 64 of 1's…)

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

→ not determinable at assembly time

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 !)

error: '! may only be applied to scalar values'

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 7 = 111b => !7 = 0b
Mov eax, !0 – EAX = 1 00 00 00 01h

Mov eax, ~7 ; 7 = 00000111b , so ~7 = 11111000b = f8h,
EAX=ff ff ff f8h one's complement on 1 byte

=> neg ax 2's complement

Mov eax, !ebx ; syntax error !

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

Mov AH, 17^(~17) ; AH = 11111111b = 0ffh = -1
Mov ax, value ^ ~value ax=11111111 11111111 = 0ffffh

value ^ ~value ax=0ffffh