Operators ! and ~ usage (page illustrating the way in which we TOGETHER obtained the results !)

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

In ASM - !0 = ? It the same mechanism as in C

! Logic Negation: !X = 0 when X ≠ 0, otherwise = 1 (X-bit)
~ 1's Complement: mov al, ~0 => mov AL, 0ffh

a is defined… RESB

Mov eax, ![a] - Expression syntax error ! [a] – is not a SCALAR value…
Mov eax, [!a] - !a is NOT a SCALAR – is a POINTER !!! a is an offset, it is determinable at assembly time, but IT IS NOT A SCALAR !!!!

Mov eax, !a - !a is NOT a SCALAR – syntax error

Mov eax, !(a+7) - !(a+7) is NOT a SCALAR – syntax error

Mov eax, !(b-a) – OK !!! because the difference of 2 pointers IS A SCALAR !!! (usually you will obtain a zer !)

Mov eax, ![a+7] - Expression syntax error !
Mov eax, !7 - EAX = 0
Mov eax, !0 – EAX = 1

Mov eax, ~7 ; 7 = 00 00 00 07h = … 00000111b, so ~7 = 0 ff ff ff f8h

Mov eax, !ebx ; syntax error !

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

Mov AH, 17^(~17) ; AH = 0 ffh = -1
Mov ax, value ^ (~value); eax= 0 ff ffh = -1
Mov eax, value ^ (~value); eax= 0 ff ff ff ffh = -1
(in the general case we can say that we obtain -1)

Operators ! and ~ usage (examples prepared for me in advance – with previously completed answers…)

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 (X-bit)

~ 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…)

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

Mov eax, [!a] - ! can 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 !
Mov eax, ![a+7] - expression syntax error
Mov eax, !7 - EAX = 0
Mov eax, !0 – EAX = 1

Mov eax, ~7 ; 7 = 00000111b , so ~7 = 11111000b = 0f8h,
EAX=0 ff ff ff f8h

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 = 1

Operands data type (discussed with you…)

Push v – stack ß offset v (32 bits)

Push [v] - Syntax error ! – Operation size not specified !
Push byte [v] – syntax error !
Push word [v] – ok !
Push dword [v] – ok !
Push qword [v] – syntax error !

Mov eax,[v] - ok ! EAX=dword ptr [v] = mov eax, dword ptr [DS:v]

Push [eax] - Syntax error ! – Operation size not specified !
Push word/dword [eax] ; ok !
…? – is it a correct, valid and accessible address [DS:EAX] ?? Possible run-time error "Memory violation"… but this is something decided at run-time based on the value from EAX…

Push 15 – PUSH DWORD 15 – ok !

Pop [v] - Syntax error ! – Operation size not specified !
Pop word/dword [v] – ok !

Pop v ; v is an address !! BUT… it is a CONSTANT address… You can not change a CONSTANT address !! It would be exactly like attempting to write 2=3 !!!... v is NOT a L-value !!

Pop [eax] ; Syntax error ! – Operation size not specified !
Pop word/dword [eax] ; ok !

Pop 15 - 15 is NOT a L-value !! (15 = 3 !!!)
Pop [15] - Syntax error ! – Operation size not specified !
Pop word/dword [15] = [DS:15] – most probably will issue a run –time error …

Mov [v],0 - Syntax error ! – Operation size not specified !
Mov byte [v], 0 ; OK !!
Mov [v], byte 0 ; OK !!

Div [v] – syntax error
Div byte/word/dword [v] – OK !!!

Imul [v+2] – syntax error
Imul byte/word/dword [v+2] – OK !!!

a dd...
b dw...

Mov a,b – …error

Mov [a], b – syntax error – Op.size not specified !
Mov [a], word b - ok
Mov dword [a], b – ok
Mov byte [a], b – syntax error ! (similar to mov ah, b type of error…)
Mov qword [a], b ; syntax error !

Mov a,[b] – a NOT a L-value !!

Mov [a], [b] – NO asm instruction can have both operands from memory !!
Mov word [a], [b] - NO asm instruction can have both operands from memory !!

Mul v – MUL reg/mem – syntax error because it doesn't follow the syntax of MUL !

Mul [v] – Op.size not specified !
Mul byte/word/dword [v] - ok

Mul eax ; ok !
Mul [eax] ; Op.size not specified !
Mul byte/word/dword [eax] - ok

MUL 15 ; MUL reg/mem – syntax error because it doesn't follow the syntax of MUL !

Operands data type (examples + answers prepared by me in advance…)

Push v – stackßoffset v

Push [v] - Syntax error ! – Operation size not specified !! (a PUSH on a 32 bits programming stack accepts both 16 and 32 bits values as stack operands) ;

Push dword [v] - ok
Push word [v] - ok

Mov eax,[v] - ok ; EAX = dword ptr [v], in Olly dbg "mov eax, dword ptr [DS:v]"

Push [eax] - Syntax error ! – Operation size not specified !!
Push word/dword [eax]

Push 15 – PUSH DWORD 15
Pop [v] - Syntax error ! – Operation size not specified !! (a POP from the stack accepts both 16 and 32 bits values as stack operands) ;

Pop word/dword [v];

Pop v ; Invalid combination of opcode and operands , because v is an offset (R-value) and a R-value CANNOT be the destination of an assignment ! (like attempting 2=3)

Pop [eax] – Op size not specified !

Pop 15 - Invalid combination of opcode and operands , because v is an offset (R-value) and a R-value CANNOT be the destination of an assignment ! (like attempting 2=3)

Mov [v],0 - op size not spec.
Mov byte [v],0 ; ok !!!
Mov [v], byte 0 ; ok !!!!

Div [v] – Op. size not spec. – 3 possibilities …
Imul [v+2] - Op. size not spec

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

Mov a,b – Invalid combination of opcode and operands , because a is an offset (R-value) and a R-value CANNOT be the destination of an assignment ! (like attempting 2=3)

Mov [a], b – Op. size not spec.
Mov word [a], b or mov [a], word b - the lower word from the offset of b will be transferred into the first 2 bytes starting at offset a !

Mov dword [a], b or… - the offset of b will be transferred into the first 4 bytes starting at offset a !

Mov byte [a], b or…. – SYNTAX ERROR ! because AN OFFSET is EITHER a 16 bits value or a 32 bits value, NEVER an 8 bit value !!!!!
(the same effect as mov ah, v)

Mov a,[b] - Invalid combination of opcode and operands , because a is an offset (R-value) and a R-value CANNOT be the destination of an assignment ! (like attempting 2=3)

Mov [a], [b] - Invalid combination of opcode and operands, BECAUSE asm doesn't allow both explicit operands to be from memory !!!

Mul v – Invalid combination of opcode and operands, BECAUSE syntax is MUL reg/mem

Mul [v] – op size not spec.

Mul eax ; ok !
Mul [eax] ; op size not spec.

MUL 15 ; Invalid combination of opcode and operands, BECAUSE syntax is MUL reg/mem

Pop byte [v] - Invalid combination of opcode and operands
Pop qword [v] – Instruction not supported in 32 bit mode !

Mov eax,0
Idiv eax; run-time error ! Zero divide…

Eroare de asamblare / assembly error = syntax error !

The need for XLAT emerged from situations like this:

How to generate the STRING of digits corresponding to a numeric value ?

fa26h in AX à ‘fa26’
3 + ‘0’ = Ascii code of CHARACTER ‘3’
I+’0’ = ascii code of whatever I is… 0..9…

If the value is between 10..16 à i+’a’-10