**Examples reflecting the implicit rules for prefixing an offset with the corresponding segment register**

Mov eax, [ebx+esp] ; ESP – base… EBX – index ;EAX 🡨 …SS:…

Mov eax, [esp + ebx] ; ESP – base… EBX – index ;EAX 🡨 …SS:…

Mov eax, [ebx+esp\*2] ; syntactic error BECAUSE ESP can be ONLY a base register and here is specified as an index register because scale is present with it !

Mov eax, [ebx+ebp\*2] ; mov eax, DWORD PTR [DS:EBX+EBP\*2]

Mov eax, [ebx+ebp] ; …DS…

Mov eax, [ebp+ebx] ; …SS…

Mov eax, [ebx\*2+ebp] ; …SS…

Mov eax, [ebx\*1+ebp] ;…SS…

Mov eax, [ebp\*1+ebx] ; …DS…

Mov eax, [ebx\*1+ebp\*1] ; ;…SS…

Mov eax, [ebp\*1+ebx\*1] ; …DS…

Jmp et1 ; …CS:et1…

Jmp [et1] ; JMP short [DS:0f6795B4] - I have to take 4 bytes as the needed correct offset to be referred to the current CS !!!

* I go in memory to the address DS:0f6795B4 , because of [] I will take THE CONTENTS from this address (for example 0BA2F5C4) and BECAUSE of JMP this contents will be THE TARGET OFFSET to which I (the processor) will perform this JMP (this offset being relative to the current CS). So, the JMP will be made to the address CS: 0BA2F5C4 !!!!

What you will be as programmers confronted with in your checkings will be that DS=ES=SS=GS , a slightly different value for CS and a different value for FS (this comes from THE FLAT MEMORY MODEL !!!).

CS:correct\_offset ; “Access violation” run-time error !

JMP DWORD PTR [DS…] – to be performed at CS:the correct identified offset

Jmp 5 ; syntax error BECAUSE it does not obbey the JMP syntax , 5 is not a label, nor a register and nor a memory address !!!

* Relative call to absolute address not supported by OBJ format
* **CS** for code labels target of the control transfer instructions (jmp, call, ret, jz etc);
* **SS** in SIB addressing when using EBP or ESP as *base* (no matter of *index* or *scale*);
* **DS** for the rest of data accesses;

[eax+ebx] – indirect addressed operand

[v] – direct addressed operand (the contents !!!)

V – is determinable at assembly time as an offset !

**Bitwise operations and operators**

Attention to the difference between operators and instructions !!!

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

Vs.

Mov ah, 01110111

Shl ah, 3

& - 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 setting the values of some 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 !

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

Mov ax, value ^ ~value ax=11111111 11111111 = 0ffffh

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

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

In ASM - !0 =

(because a 0 in asm is a binary ZERO represented on 8, 16, 32 or 64 bits the logical BITWISE negation will issue a binary 8 of 1’s, 16 of 1’s, 32 of 1’s or 64 of 1’s… 0 = false, TRUE = 8 of 1’s, 16 of 1’s, 32 of 1’s or 64 of 1’s = -1 (0 = false, -1 = TRUE)

**Operators ! and ~ usage**

! Logic Negation: !X = 0 when X ≠ 0, otherwise = 1 (X-bit)

~ 1’s Complement: mov al, ~0 => mov AL, 0ffh

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, ~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

**Operands data type**

Push v – stack🡨offset 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 v 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 v 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 the syntax of MUL 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 the syntax of MUL is MUL reg/mem

Pop byte [v] - Invalid combination of opcode and operands

Pop qword [v] – Instruction not supported in 32 bit mode !

**Data definition directives**

Always your data segment starts at offset 00401000

Segment data

a1 db 0,1,2,’xyz’ ; 00 01 02 ‘x’ ‘y’ ‘z’ (ascii codes for these chars)

78 79 7A

db 300, “F”+3 ; 2C ‘ascii code for F + 3”

a2 TIMES 3 db 44h ; 44 44 44

a3 TIMES 11 db 5,1,3 ; 05 01 03… 11 times (33 bytes)

a4 dw a2+1, ‘bc’ ; offset a2 = 00401008 ; 09 10 ‘b’ ‘c’

a5 dd a2+1, ‘bcd’ ; 09 10 40 00 62 63 64 00

a6 TIMES 4 db ‘13’ ; 31 33 31 33 31 33 31 33

a7 db a2 ; syntax error – OBJ format can only handle 16 or 32 bits relocation (equiv. mov ah,a2)

a8 dw a2 ; 08 10

a9 dd a2 ; 08 10 40 00

a10 dq a2 ; 08 10 40 00 00 00 00 00

a11 db [a2] ; expression syntax error !

a12 dw [a2] ; expression syntax error !

a13 dd dword [a2] ; expression syntax error !

a14 dq [a2] ; expression syntax error !

mov ax, v ; Warning – 32 bit offset in 16 bit field !

Definition of a variable/subroutine = Declaration (to specify its attributes) + ALLOCATION (it’s a ONE TIME ONLY process)

**The steps followed by a program from source code to run-time:**

* Syntactic checking (done by assembler/compiler/interpreter)
* OBJ files are generated (….)
* Linking phase (performed by a LINKER = a tool provided by the OS, which checks the possible DEPENDENCIES between this OBJ files/modules); The result 🡪 .EXE file !!!
* You (the user) are activating your exe file by clicking or enter-ing…
* The LOADER of the OS is looking for the required RAM memory space for your EXE file. When finding it, it loads the EXE file AND performs ADDRESS RELOCATION !!!!
* In the end the loader gives control to the processor by specifying THE PROGRAM’s ENTRY POINT (ex: the start label) !!! The run-time phase begins NOW…

Mark Zbirkowski – semnatura EXE = ‘MZ’

The specifiers BYTE / WORD / DWORD / QWORD always have the task to clarify an ambiguity (inclusively when we talk about a memory variable, specifying mov BYTE [v], 0 or mov WORD [v], 0 is also an ambiguity elimination, because NASM is not associating the type to a label – v is not considered byte/ word / dword but just simply an address).

mov [v],0 ; syntax error – operation size not specified

**The QWORD specifier is never used explicitly** in 32 bits code.

Examples illustrating where a specifier is needed:

    - mov [mem], 12

- (i)div [mem] ; (i)mul [mem]

* push [mem] ; pop [mem]
* push 15 – allowed, but this is a NASM inconsistency, the assembler not issuing an error but “translating” the initial written instruction to push DWORD 15   
    
  Exemples of IMPLICITLY 64 bits operands (in 32 bits code):
* mul dword [v] ; multiplies eax with the dword from address v and stores the result in EDX:EAX

    - div dword [v] ; divides EDX:EAX to dword v