

## Operatori de tip și tipuri de date asociate operanzilor

Operatorii efectuează calcule cu valori constante SCALARE determinabile la momentul asamblării (valori scalare = valori imediate), cu excepția adunării și scăderii unei constante la un/dintr-un pointer (care va furniza întotdeauna "pointer data type") și cu excepția formulei de calcul al offset-ului unui operand (care permite operatorul '+').

Specificatorii BYTE / WORD / DWORD / QWORD au rolul de a clarifica o ambiguitate.

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

Push v - ok, stack ← offset v (pe 32 biți)

Push [v] - syntax error – Operation size not specified ! (PUSH pe o stivă pe 32 biți acceptă atât operanzi pe 32 biți cât și pe 16 biți)

Push dword [v] – ok

Push word [v] - ok

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

Push [eax] – syntax error.... Operation size not specified !

Push byte [eax] – syntax error....

Push word [eax] – ok

Push 15 ; PUSH DWORD 15

Pop [v] ; Op size not specified (a POP from the stack accepts both 16 and 32 bits values as stack operands) ;

Pop word/dword [v] – ok !!

Pop v ; sintaxa este POP destinatie; destinatie must be a L-value !!!!!

...dar v este R-value !!!!! ; acest pop v este similar ca operatie cu 2:=3 !!!

(Invalid combination of opcode and operands)

Pop dword b ; syntax error !

Pop [eax] ; Op size not specified

Pop (d)word [eax] ; ok!

Pop 15 ; 15 is NOT a L-value !! – syntax error

Pop [15] ; syntax error - Op size not specified

Pop dword [15] ; syntactic ok , cel mai probabil **run-time error** deoarece probabil [DS:15] va provoca Access violation !!

Mov [v],0 ; syntax error - Op size not specified

Mov byte [v], 0 ; ok !

Mov [v], byte 0

Div [v] ; op. size ?

Div word [v]; ok !

Imul [v+2] ; op. size ?

Imul word [v+2]; DX:AX = AX\*word de la adresa v+2

a d?...

b d?...

Mov a,b ; syntax error pt că a NU este L-value, ci R-value fiind un offset determinabil ca și constantă la momentul asamblării

Mov [a], b ; syntax error – op. size not specified !

Mov byte [a], b or mov [a], byte b – 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 word [a], b ; ok !! – 2 octeți inferiori din valoarea offset-ului lui b !

Mov dword [a], b ; full offset 32 bits

Mov a,[b] ; Invalid comb. of opcode and operands (a = R-value)

Mov [a], [b] ; Invalid comb. Of opcode and operands (NU putem avea 2 operanzi simultan din memorie)

Mul v – syntax error – MUL reg/mem  
Mul word v - syntax error – MUL reg/mem  
Mul [v] - op. size not specified  
Mul dword [v]; ok !

Mul eax ; ok !  
Mul [eax] ; op. size not specified  
Mul byte [eax] ; ok !!!  
Mul 15 ; Invalid comb. of opcode and operands – MUL reg/mem

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

### Clasificarea erorilor in informatică

- Eroare de sintaxă – ea este diagnosticată de asamblor/compiler !  
(eroare de asamblare)
- Run-time error (eroare la execuție) – programul “crapă” – programul se opreste = program crash !!
- Eroare logică = programul funcționează până la capăt sau ramâne blocat în ciclu infinit, însă GRESIT dpdv LOGIC obținându-se cu totul alte rezultate decât cele așteptate...
- Fatal: Linking Error !!! (de ex în cazul unei definiții duble de variabilă... 17 module și o variabilă trebuie să fie DEFINITA DOAR într-un singur modul ! Dacă ea este definită în 2 sau mai multe module se va obține Fatal: Linking Error !!! – Duplicate definition for symbol A1 !!!)

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

- Syntactic checking (done by assembler/compiler/interpreter)
- OBJ files are generated by the assembler/compiler
- 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 – semnătura EXE = 'MZ'

## Tipuri de date asociate operanzilor

### Directivele de definire a datelor in NASM NU sunt mecanisme de definire a tipurilor de date !!

a db 17,19  
b dw 1234h ; 34h 12h  
c dd....

Rolul directivelor de definire a datelor NU este **în NASM** de a preciza tipul de dată al variabilelor definite, ci DOAR de a genera octeții corespunzători acelor zone de memorie pe care le ocupă în conformitate cu directiva specifică aleasă și respectând ordinea de plasare de tip little-endian !!!

Deci a NU este de tip byte – ci doar un offset/deplasament și atât... un simbol desemnand începutul unei zone de memorie FARA VREUN TIP ASOCIAT !

Iar b NU este de tip word – ci doar un offset/deplasament și atât ... un simbol desemnand începutul unei zone de memorie FARA VREUN TIP ASOCIAT !

Si nici c NU este de tip doubleword – ci doar un offset/deplasament și atât ... un simbol desemnand începutul unei zone de memorie FARA VREUN TIP ASOCIAT !

Atunci DE CE mai asociem în definiție o directivă de tip de dată ???? Pt a INDICA ASAMBLORULUI CUM să populeze cu date/inițializeze zona de memorie respectivă !!! (fie ca o secvență de bytes, fie ca una de words, fie ca una de doublewords !) Directivele de date se referă la modalitatea de inițializare concretă a unei zone de memorie și nu asociază atributul de tip de dată cu un simbol !

Deci: directiva de definire a unei date NU este un mecanism de asociere de tip de dată pentru o variabilă, ci doar un mecanism de inițializare a zonei de memorie alocate variabilei cu valorile dorite !!!

Stiute, dar bine de a fi reamintite:

The *name of a variable* is *associated* in assembly language *with its offset relative to the segment* in which its *definition appears*. The *offsets of the variables* defined in a program are *always constant* values, determinable at assembly/compiling time.

*Assembly language* and *C* are *value oriented languages*, meaning that everything is reduced in the end to a numeric value, this being a low level feature.

In a *high-level programming language*, the *programmer can access the memory only* by using *variable names*, in contrast, in *assembly language*, the *memory is/can/must be accessed ONLY* by using the *offset computation formula* ("formula de la doua noaptea") where *pointer arithmetic* is also used (pointer arithmetic is also used in C !).

*mov ax, [ebx]* – the source operand *doesn't* have an *associated data type* (it represents only a start of a memory area) and because of that, in the case of our MOV instruction the *destination operand* is the one that *decides the data type of the transfer (a word in this case)*, and the transfer will be made accordingly to the little endian representation.