În cadrul unei instrucţiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:

* *modul registru*, dacă pe post de operand se află un registru al maşinii; mov ax, bx
* *modul imediat*, atunci când în instrucţiune se află chiar valoarea operandului (nu adresa lui şi nici un registru în care să fie conţinut); mov eax,2
* *modul adresare la memorie*, dacă operandul se află efectiv undeva în memorie. În acest caz, adresa offsetului lui se calculează după următoarea formulă:

***adresa\_offset* = [ bază ] + [ index × scală ] + [ constanta]**

constanta = offset constant sau/si val.imediata [EBX+ECX\*2-7]

Deci *adresa\_offset* se obţine din următoarele (maxim) patru elemente:

* conţinutul unuia dintre regiştrii EAX, EBX, ECX, EDX, EBP, ESI, EDI sau ESP ca bază;
* conţinutul unuia dintre regiştrii EAX, EBX, ECX, EDX, EBP, ESI sau EDI drept index;
* factor numeric (scală) pentru a înmulţi valoarea registrului index cu 1, 2, 4 sau 8
* valoarea unei constante numerice, pe octet sau dublucuvânt.

De aici rezultă următoarele moduri de adresare la memorie:

* *directă*, atunci când apare numai *constanta*;
* *bazată*, dacă în calcul apare unul dintre regiştrii bază;
* *scalat-indexată*, dacă în calcul apare unul dintre regiştrii index;

Adresarea care NU este directa se numeste ***adresare indirecta*** (bazata si/sau indexata). Deci o adresare indirecta este cea pt care avem specificat cel putin un registru intre parantezele drepte.

In cadrul sistemului de adresare se efectueaza operatii cu adrese (pointeri). Care sunt operatiile ARITMETICE cu pointeri permise IN INFORMATICA ?...

**Raspuns**: Orice operatie care are sens... aceasta insemnand orice operatie ce exprima ca rezultat o localizare in memorie corecta si utila ca informatie pt programator.

* Singura exceptie de la regulile aritmeticii de pointeri o constituie formula de calcul a offsetului unui operand unde sunt permise adunari de valori de registri... In rest nu exista exceptii (de exemplu la nivel de adrese de variabile cu care se opereaza intr-un program).

Pointer arithmetic...?

Adresa – adresa = ok (q-p = scadere de 2 pointeri = sizeof(array))

Adresa – offset = adresa – adresa = ok

Adresa + constanta numerica (identificarea unui element prin indexare – a[7]) , q+9

Adresa – constanta numerica - a[-4] , p-7

* scaderea a 2 pointeri
* adunarea unei constante la un pointer
* scaderea unei constante dintr-un pointer

p+q = ???? (allowed in NASM...sometimes...)

Atribuire: i:=i+1 Adresa lui I 🡨 valoarea lui I + 1

LHS (valoare stanga a unei atribuiri L-value = adresa) := RHS (valoarea dreapta a unei atribuiri = R-Value = CONTINUT !!)

Exemple/studii de caz

Mov ax, ebx - reg16 🡨 reg32 – Sintax error ! invalid combination of opcode and operands

Mov ebx, ch - Sintax error ! invalid combination of opcode and operands

Mov op\_size\_dest, op\_SAME\_size b,b w,w dw,dw

Mov eax, ebx - eax🡨EBX

Mov eax, [ebx] - eax 🡨 [valoarea DIN MEMORIE preluata de la adresa indicata de continutul reg. EBX]

Mov edx, [eax+ebx] edx 🡨 [valoarea DIN MEMORIE preluata de la adresa indicata de rezultatul adunarii continuturilor registrilor EAX si EBX]

Mov edx, eax+ebx ; syntax error ! A se intelege diferenta intre OPERATORUL + si INSTRUCTIUNEA ADD !

Mov edx, [ebx+eax] – corecta , face acelasi lucru ca mai sus; NU STIM CARE va fi pt processor BAZA si care INDEXUL !!

Mov edx, [esp+ecx] ; ESP=baza; ECX = index

Mov edx, [ecx+esp]; ESP=baza; ECX = index

Mov edx, [esp+2\*ecx] – correct…ESP – baza; ECX – index ; 2 = scala

Mov edx, [ecx+2\*esp] – syntax error !!!! pt ca ESP NU poate fi index

mov dh, [edx + ecx \* 4 + 3] ; CORECTA ; DH🡨PRIMUL OCTET de la offsetul desemnat de formula de calcul edx + ecx \* 4 + 3

mov dx, [edx + ecx \* 4 + 3] ; CORECTA ; DX🡨PRIMUL CUVANT (WORD) de la offsetul desemnat de formula de calcul edx + ecx \* 4 + 3

mov eax, [eax\*3] = mov eax, [eax+2\*eax] – CORECT !!!!!!!

mov eax, [ebx\*9 + 12] - EBX + 8\*EBX + 12 – CORECT !!!!!

mov eax, [esp\*5] – syntax error – ESP NU POATE FI INDEX !!!!

Mov reg, [var] – in care dintre modurile de specificare a operanzilor se incadreaza ?

Mov eax, [a] - corecta!!! Pt ca OFFSETUL ORICAREI VARIABILE DEFINITE IN PROGRAM ESTE O VALOARE NUMERICA CONSTANTA DETERMINABILA LA MOMENTUL ASAMBLARII !!!!

Mov eax, dword ptr [DS:2] – vezi OllYDBG !!!!

Var d? ….

Mov eax, var ; EAX 🡨 offsetul (var) = pe 32 biti

Mov eax, [var] ; EAX 🡨 se iau 4 octeti de la adresa var (indiferent de cum am definit var) si se transfera in EAX !

Mov ax, var ; va merge cu Warning (16 bits reloc of 32 bits value)

* se vor prelua cei doi octeti INFERIORI din reprezentarea IN MEMORIE a offset-ului

Mov ax, [var] ; AX 🡨 se iau 2 octeti de la adresa var (indiferent de cum am definit var) si se transfera in AX !

Mov ah, var ; Syntax error ! OBJ file can only handle 16 – or 32 bits addresses

Mov ah, [var] ; AH 🡨 se ia 1 octet de la adresa var (indiferent de cum am definit var) si se transfera in AH !

Var db 17, 18, 19, 29, 2ah, -3

Mov [var], eax; mov dword ptr [var], eax

A db 17

B db 19

C db 21

D db 23

Mov eax, [A]; ia 4 octeti de la adresa A si ii transfera in EAX !!!

Mov ah, ebx ; syntax error -

Mov ah, [ebx] ; se ia 1 octet de la [ebx] si se transfera in ah

Mov ax, [ebx]; 2 octeti

Mov eax, [ebx] ; 4 octeti

Offset\_spec16 = [BX|BP] + [SI|DI] + [constanta]

Mov ah, [bx] ; 1 octet in AH luat pe baza formulei de calcul pe 16

Mov ax, [bx] ; 2 octeti in AX

Mov eax,[bx] ; 4 octeti in EAX

Mov ah, [bh] ; syntax error !!!

a db …

b dw…

c dd….

Rolul directivelor de definire a datelor NU este in NASM de a preciza tipul de data al variabilelor definite, ci DOAR de a genera octetii corespunzatori acelor zone de memorie pe care le ocupa in conformitate cu directive specifica aleasa si respectand ordinea de plasare de tip little-endian !!!

Deci a NU este de tip byte – ci doar un offset/deplasament si atat… un simbol desemnand inceputul unei zone de memorie FARA VREUN TIP ASOCIAT !

Deci b NU este de tip word – ci doar un offset/deplasament si atat… un simbol desemnand inceputul unei zone de memorie FARA VREUN TIP ASOCIAT !

Deci c NU este de tip doubleword – ci doar un offset/deplasament si atat… un simbol desemnand inceputul unei zone de memorie FARA VREUN TIP ASOCIAT !

Instructiunile – adresate PROCESORULUI

Directivele – adresate ASAMBLORULUI

Add ebx, ecx

Mov eax, ebx+ecx

**Contor de locatii**

**Segment data**

**……**

**a db 17, -2, 0ffh, ‘xyz’,…**

**db ….**

**db….**

**lg dw lg-a**

**lg dw $-a //aritmetica de pointeri – scaderea a 2 pointeri = scalar (constanta numerica) - lg=variabila de memorie (mov [lg],…)**

**lg db $-a**

**lg EQU $-a //lg = constanta !!! (mov [lg],… - illegal !)**

**lg dw $-$$ ; OK !!!**

**lg dw $-data ; syntax error – “Expression is not simple or relocatable”**

**lg dw lg-a ; !!!!!! OK !!!!!!!**

Segment data

A db 17,-2

B dw -20345, “x”

Start2:

C dd 12345678h

Segment code

……

Jmp start2

……

a[7] = \*(a+7) = \*(7+a) = 7[a]

[] = dereferencing operator (operator de dereferentiere = \*)

Mov ebx,[var] – registru, adresare la memorie

A db 17

Int \*a

Int\* a = int \*a

Int\* a,b,c = Int \*a, \*b, \*c