

CAPITOLUL 3

ELEMENTELE DE BAZA ALE LIMBAJULUI DE ASAMBLARE

Limbajul mașină al unui sistem de calcul (SC) este format din totalitatea instrucțiunilor mașină puse la dispoziție de procesorul SC. Acestea se reprezintă sub forma unor șiruri de biți cu semnificație prestabilită.

Limbajul de asamblare al unui calculator este un limbaj de programare în care setul de bază al instrucțiunilor coincide cu operațiile mașinii și ale cărui structuri de date coincid cu structurile primare de date ale mașinii. **Limbaj simbolic. Simboluri - Mnemonice + etichete.**

Elementele cu care lucrează un **asamblor** sunt:

- * **etichete** - nume scrise de utilizator, cu ajutorul cărora se pot referi date sau zone de memorie.
- * **instrucțiuni** - scrise sub forma unor mnemonice care sugerează acțiunea. Asamblorul generează octeții care codifică instrucțiunea respectivă.
- * **directive** - sunt indicații date asamblorului în scopul generării corecte a octetilor. Ex: relații între modulele obiect, definirea unor segmente, indicații de asamblare condiționată, directive de generare a datelor.
- * **contor de locații** - număr întreg gestionat de asamblor. În fiecare moment, valoarea contorului coincide cu numărul de octeți generați corespunzător instrucțiunilor și directivelor deja întâlnite în cadrul segmentului respectiv (deplasamentul curent în cadrul segmentului). Programatorul poate utiliza această valoare (accesare doar în citire!) prin simbolul '\$'.

NASM supports two special tokens in expressions, allowing calculations to involve the current assembly position: the \$ and \$\$ tokens. \$ evaluates to the assembly position at the beginning of the line containing the expression; so you can code an infinite loop using `JMP $`.

\$\$ evaluates to the start of the current section; so you can tell how far into the section are by using `($-$)`.

Directiva SECTION

```
section .data
    db 'hello'
    db 'h', 'e', 'l', 'l', 'o'
    data_segment_size equ $-$$
```

\$-\$\$ = the distance from the beginning of the segment AS A SCALAR (constant numerical value)!!!!!!!!!!

\$ - is an offset = POINTER TYPE !!!! It is an address !!!!

\$\$ - is an offset =POINTER TYPE !!!! It is an address !!!!

\$ means "address of here".

\$\$ means "address of start of current section".

So \$-\$\$ means "current size of section".

For the example above, this will be 10, as there are 10 bytes of data given.

Daca nu am definite secțiuni explicite , atunci \$\$=începutul segmentului respectiv

3.1. FORMATUL UNEI LINII SURSĂ

Formatul unei linii sursă în limbajul de asamblare x86 este următorul:

[*etichetă*[:]] [*prefixe*] [*mnemonică*] [*operandi*] [*;comentariu*]

Ilustrăm conceptul prin intermediul a câteva exemple de linii sursă:

aici: jmp acolo	; avem etichetă + mnemonică + operand + comentariu
repz cmpsd	; prefix + mnemonică + comentariu

start: ; etichetă + comentariu
 ; doar un comentariu (care putea lipsi si el)
a dw 19872, 42h ; etichetă + mnemonică + 2 operanzi + comentariu

Caracterele din care poate fi constituită o *etichetă* sunt următoarele:

- Litere, atât A-Z cât și a-z;
- Cifre de la 0 la 9;
- Caracterele `_`, `$`, `$$`, `#`, `@`, `~`, `.` și `?`

Ca și prim caracter al unei etichete sunt permise doar litere, `_` și `?`

Aceste reguli sunt valabile pentru toți *identificatorii* valizi (denumiri simbolice, precum nume de variabile, etichete, macro, etc).

Identificatorii NASM sunt *case sensitive*, limbajul diferențiind literele mari de cele mici privitor la denumirile utilizator. Aceasta înseamnă că un identificator `Abc` este diferit de identificatorul `abc`. Pentru denumirile implicit parte a limbajului, cum ar fi cuvintele cheie, mnemonicile și numele regiștrilor, nu se diferențiază literele mari de cele mici (acestea sunt *case insensitive*).

La nivelul limbajului de asamblare se întâlnesc două categorii de etichete:

- 1). ***etichete de cod***, care apar în cadrul secvențelor de instrucțiuni cu scopul de a defini destinațiile de transfer ale controlului în cadrul unui program. **Pot apărea și în segmente de date !**
- 2). ***etichete de date***, care identifică simbolic unele locații de memorie, din punct de vedere semantic ele fiind echivalentul noțiunii de *variabilă* din alte limbaje. **Pot apărea și în segmente de cod !**

Valoarea unei etichete în limbaj de asamblare este un număr întreg reprezentând adresa instrucțiunii, directivei sau datelor ce urmează etichetei.

Distincția dintre referirea adresei unei variabile sau a conținutului asociat acesteia în NASM se face după regulile:

- Când este specificat **între paranteze drepte, numele variabilei desemnează valoarea (conținutul) variabilei**, de exemplu [p] specifică accesarea valorii variabilei p, similar cu modul în care *p semnifică dereferențierea unui pointer (accesul la conținutul indicat prin valoarea pointerului) în C;
- În orice alt context **numele variabilei reprezintă adresa variabilei**, spre exemplu, p este întotdeauna adresa variabilei p;

Exemple:

```
mov EAX, et      ; încarcă în registrul EAX adresa datelor sau a codului marcat cu eticheta et (4 octeți)
mov EAX, [et]    ; încarcă în registrul EAX conținutul de la adresa et (4 octeți)
lea eax, [v]     ; încarcă în registrul eax adresa (offsetul) variabilei v (4 octeți)
```

Ca generalizare, **folosirea parantezelor pătrate indică întotdeauna accesarea unui operand din memorie**. De exemplu, mov EAX, [EBX] semnifică un transfer în EAX a conținutului memoriei a cărei adresă este dată de valoarea lui EBX.

Există două tipuri de *mnemonice*: mnemonice de *instrucțiuni* și nume de *directive*. Directivele dirijează asamblorul. Ele specifică modul în care asamblorul va genera codul obiect. Instrucțiunile dirijează procesorul.

Operanzii sunt parametri care definesc valorile ce vor fi prelucrate de instrucțiuni sau de directive. Ei pot fi **registri, constante, etichete, expresii, cuvinte cheie sau alte simboluri**. Semnificația operanzilor depinde de mnemonica instrucțiunii sau directivei asociate.

3.2. EXPRESII

expresie - operanzi + operatori. *Operatorii* indică modul de combinare a operanzilor în scopul formării expresiei. **Expresiile sunt evaluate în momentul asamblării** (adică, valorile lor sunt determinabile la momentul asamblării, cu excepția acelor părți care desemnează conținuturi de regiștri și care vor fi determinate la execuție).

3.2.1. Moduri de adresare

Operanzii instrucțiunilor pot fi specificați sub forme numite *moduri de adresare*.

Cele trei tipuri de operanzi sunt: *operanzi imediați*, *operanzi registru* și *operanzi în memorie*.

Valoarea operanzilor este calculată în momentul asamblării pentru operanzii imediați, în momentul încărcării programului pentru adresarea directă (adresa FAR) și în momentul execuției pentru operanzii registru și cei adresați indirect.

3.2.1.1. Utilizarea operanzilor imediați

Operanzii imediați sunt formați din date numerice constante calculabile la momentul asamblării.

Constantele întregi se specifică prin valori binare, octale, zecimale sau hexazecimale. Adicional, este permisă folosirea caracterului _ (underscore) pentru a separa grupuri de cifre. Baza de numerație poate fi precizată în mai multe moduri:

- Folosind sufixele H sau X pentru hexazecimal, D sau T pentru zecimal, Q sau O pentru octal și B sau Y pentru binar; în aceste cazuri numărul trebuie să înceapă obligatoriu cu o cifră între 0 și 9, pentru a nu exista confuzii între constante și simboluri, de exemplu, OABCH este interpretat ca număr hexazecimal, dar ABCH este interpretat ca simbol
- În stil C, prin prefixare cu 0x sau 0h pentru hexazecimal, 0d sau 0t pentru zecimal, 0o sau 0q pentru octal, respectiv 0b sau 0y pentru binar;

Exemple:

- constanta hexazecimală B2A poate fi exprimată ca 0xb2a, 0xb2A, 0hb2a, 0b12Ah, 0B12AH, etc;
- valoarea zecimală 123 poate fi specificată ca 123, 0d123, 0d0123, 123d, 123D, ...
- 11001000b, 0b11001000, 0y1100_1000, 001100_1000Y reprezintă diferite exprimări ale numărului binar 11001000

Deplasamentele etichetelor de date și de cod reprezintă valori determinabile la momentul asamblării care rămân constante pe tot parcursul execuției programului

mov eax, et ; transfer în registrul EAX a adresei (offsetului) asociate etichetei et

va putea fi evaluată la momentul asamblării drept de exemplu

mov eax, 8 ; distanță de 8 octeți față de începutul segmentului de date

“Constanța” acestor valori derivă din regulile de alocare adoptate de limbajele de programare în general și care statuează că ordinea de alocare în memorie a variabilelor declarate (mai precis distanța față de începutul segmentului de date în care o variabilă este alocată) sau respectiv distanțele salturilor destinație în cazul unor instrucțiuni de tip **goto** sunt valori constante pe parcursul execuției unui program.

Adică: o variabilă odată alocată în cadrul unui segment de memorie nu își va schimba niciodată locul alocării (adică poziția sa față de începutul acelui segment) iar această informație determinabilă la momentul asamblării derivă din ordinea specificării variabilelor la declarare în cadrul textului sursă și din dimensiunea de reprezentare dedusă pe bază informației de tip asociate.

3.2.1.2. Utilizarea operanzilor registru

Modul de *invocare/accesare directă* - **mov eax, ebx**

Invocare/accesare *indirectă* - pentru a indica locațiile de memorie - **mov eax, [ebx]**

3.2.1.3. Utilizarea operanzilor din memorie

Operanzii din memorie : cu *adresare directă* și cu *adresare indirectă*.

Operandul cu *adresare directă* este o constantă sau un simbol care reprezintă adresa (segment și deplasament) unei instrucțiuni sau a unor date. Acești operanzi pot fi *etichete* (de ex: jmp et, var1 dw 324, add eax,[b]), *nume de proceduri* (de ex: call proc1) sau *valoarea contorului de locații* (de ex: b db \$-a).

Deplasamentul unui operand cu adresare directă este calculat în momentul asamblării (assembly time). Adresa fiecărui operand raportată la structura programului executabil (mai precis stabilirea segmentelor la care se raportează deplasamentele calculate) este calculată în momentul editării de legături (linking time). Adresa fizică efectivă este calculată în momentul încărcării programului pentru execuție (loading time – acest proces final de ajustare a adreselor numindu-se RELOCAREA ADRESELOR = Address Relocation).

Un deplasament utilizat ca operand în cadrul unui program este întotdeauna raportat la un registru de segment. Acest registru poate fi specificat explicit sau, în caz contrar, se asociază de către asamblor în mod implicit un registru de segment. Regulile limbajului de asamblare pentru asocierile implicite sunt:

- **CS** pentru etichete de cod destinație ale unor salturi (jmp, call, ret, jz etc);
- **SS** în adresări SIB ce folosește EBP sau ESP drept bază (indiferent de index sau scală);
- **DS** pentru restul accesărilor de date

Specificarea explicită a unui registru de segment se face cu ajutorul operatorului de prefixare segment (notat ":" și care se mai numește, 'operatorul de specificare a segmentului'). ES poate fi utilizat numai în specificări explicite (ca de exemplu ES:[Var] sau ES:[ebx+eax*2-a]) sau în cadrul unor instrucțiuni pe siruri (MOVSB)

JMP FAR CS:...

JMP FAR DS:... or JMP FAR [label2]

3.2.1.4. Operanzi cu adresare indirectă

Operanzii cu *adresare indirectă* utilizează regiștri pentru a indica adrese din memorie. Deoarece valorile din regiștri se pot modifica la momentul execuției, adresarea indirectă este indicată pentru a opera în mod dinamic asupra datelor.

Forma generală pentru accesarea indirectă a unui operand de memorie este dată de formula de calcul a offset-ului unui operand:

$$[\text{registru_de_bază}] + [\text{registru_index} * \text{scală}] + [\text{constantă}]$$

Constanta este o expresie a cărei valoare este determinabilă la momentul asamblării. De exemplu, $[\text{ebx} + \text{edi} + \text{table} + 6]$ desemnează un operand prin adresare indirectă, unde atât *table* cât și 6 sunt constante.

Operanzii *registru_de_bază* și *registru_index* sunt folosiți de obicei pentru a indica o adresă de memorie referitoare la un tablou. În combinație cu factorul de scalare, mecanismul este suficient de flexibil pentru a permite acces direct la elementele unui tablou de înregistrări, cu condiția ca dimensiunea în octeți a unei înregistrări să fie 1, 2, 4 sau 8. De exemplu, octetul superior al elementului de tip DWORD cu index dat în ecx, parte a unui vector de înregistrări al cărui adresă (a vectorului) este în edx poate fi încărcat în dh prin intermediul instrucțiunii

mov dh, [edx + ecx * 4 + 3]

Din punct de vedere sintactic, atunci când operandul nu este specificat prin formula completă, lipsind unele dintre componente (de exemplu lipsește “* scală”), asamblorul va rezolva ambiguitatea care rezultă printr-un proces de analiză a tuturor formele echivalente de codificare posibile și alegerea celei mai scurte dintre acestea. Cu alte cuvinte, având

push dword [eax + ebx] ; salvează pe stivă dublucuvântul de la adresa eax+ebx

asamblorul are libertatea de a considera eax drept bază și ebx drept index sau invers, ebx drept bază și eax drept index. Analog, pentru

pop DWORD [ecx] ; restaurează vârful stivei în variabila cu adresa dată de ecx

asamblorul poate interpreta ecx fie ca bază fie ca index. Ce este realmente important de reținut este faptul că toate codificările luate în considerare de către asamblor sunt echivalente iar decizia finală a asamblorului nu are impact asupra funcționalității codului rezultat.

De asemenea, în plus față de rezolvarea unor astfel de ambiguități, asamblorul permite și exprimări non-standard cu condiția ca acestea să fie transformabile într-un final în forma standard de mai sus.

Alte exemple:

lea eax, [eax*2] ; încarcă în eax valoarea lui eax*2 (adică, eax devine 2*eax)

În acest caz, asamblorul poate decide între codificare de tip bază = $\text{eax} + \text{index} = \text{eax}$ și scală = 1 sau index = eax și scală = 2.

lea eax, [eax*9 + 12] ; eax ia valoarea $\text{eax} * 9 + 12$

Deși scală nu poate fi 9, asamblorul nu va emite aici un mesaj de eroare. Aceasta deoarece el va observa posibila codificare a adresei drept: bază= eax + index= eax cu scală=8, unde de această dată valoarea 8 este corectă pentru scală. Evident, instrucțiunea putea fi precizată mai clar sub forma

lea eax, [eax + eax * 8 + 12].

Să reținem deci că pentru adresarea indirectă, esențială este specificarea între paranteze drepte a cel puțin unuia dintre elementele componente ale formulei de calcul a offsetului.