Arhitectura Sistemelor de Calcul

Lect. Dr. Șotropa Diana
diana.sotropa@ubbcluj.ro

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

Elementele de bază ale limbajului de asamblare

Elementele de bază ale limbajului de asamblare

- Elementele cu care lucrează un asamblor sunt:
  - Etichete
    - Nume scrise de utilizator cu ajutorul cărora se pot referi date sau zone de memorie

```
JMP label
 a DB 2
```

  - Instrucțiuni
  - Directive
  - Contor de locații

Elementele de bază ale limbajului de asamblare

- Elementele cu care lucrează un asamblor sunt:
  - Etichete
  - Instrucțiuni
    - Scrise sub forma unor mnemonice care sugerează acțiunea; ASAMBLORUL generează octeții care codifică instrucțiunea respectivă
    - Sunt menite pentru a ajunge la procesor
    - În EU -> ALU
    - Daca operanzii sunt din memorie intervine BIU -> ADR
  - Directive
  - Contor de locații

Elementele de bază ale limbajului de asamblare

- Elementele cu care lucrează un asamblor sunt:
  - Etichete
  - Instrucțiuni
  - Directive
    - Sunt indicații date asamblorului în scopul generării corecte a octeților
    - Cel mai important task pentru asamblor este generarea octeților (vezi coloana 2 din debugger)
    - Relația dintre modulele obiect, definirea unor segmente, indicații de asamblare condiționată, directive de generare a datelor
    - Intersecția cu instrucțiunile este vidă
  - Contor de locații

Elementele de bază ale limbajului de asamblare

- Elementele cu care lucrează un asamblor sunt:
  - Etichete
  - Instrucțiuni
  - Directive
  - Contor de locații
    - număr întreg gestionat de asamblor.
    - valoarea contorului de locatii exprima deplasamentul curent în cadrul segmentului.
    - mai exact, ca si valoare concreta $ reprezinta offset-ul începutului de linie care conține expresia respectiva.
    - programatorul poate utiliza această valoare (accesare doar în citire!) prin simbolul '$'.
    - fiecare segment are propriul său contor de locaţii
    - Este low level, specific limbajului de asamblare

Elementele de bază ale limbajului de asamblare

- Particularitate NASM
  - $ - pointează către “aici” (POINTER)
  - $$ - pointează către începutul secțiunii curente (POINTER)

`($-$$)` – distanța de la începutul segmentului / secțiunii (SCALAR)
= dimensiunea curentă a secțiunii
= numărul de octeţi generaţi corespunzător instrucţiunilor şi directivelor deja întâlnite în cadrul segmentului sau sectiunii respective.

```
section .data
 db 'hello'
 db 'h', 'e', 'l','l','o'
 data_segment_size equ $-$$; 10 octeți scriși
```

Elementele de bază ale limbajului de asamblare

Formatul unei linii sursă

[eticheta[:]][prefixe][mnemonică][operanzi][;comentariu]

```
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
```

Elementele de bază ale limbajului de asamblare

- La nivelul limbajului de asamblare se întâlnesc două categorii de etichete:
  - **etichete de cod**, care apar în cadrul secvenţelor de instrucţiuni cu scopul de a defini destinaţiile de transfer ale controlului în cadrul unui program.
    Pot apărea şi în segmente de date !
  - **etichete de date**, care identifică simbolic unele locaţii de memorie, din punct de vedere semantic ele fiind echivalentul noţiunii de variabilă din alte limbaje.
    Pot apărea şi în segmente de cod !

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

Elementele de bază ale limbajului de asamblare

- Distincția dintre referirea adresei unei variabile sau a conținutului asociat acesteia in NASM se face după regulile:
  - Când este specificat **între paranteze drepte**, **numele variabilei** desemnează **valoarea variabilei**, de exemplu [p] specifică accesarea valorii variabilei p
  - În orice alt context **numele variabilei** reprezintă **adresa variabilei**, spre exemplu, p este întotdeauna adresa variabilei p;

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

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

Elementele de bază ale limbajului de asamblare

- Moduri de adresare:
  - operanzi imediaţi, operanzi registru şi operanzi în memorie
- Valoarea operanzilor este calculată:
  - **în momentul asamblării** pentru *operanzii imediaţi* şi pentru offset-urile reprezentând *adresarea directă*,
  - **în momentul încărcării programului** pentru *adresarea directă* (adresa FAR)
  - **în momentul execuţiei** pentru *operanzii registru* şi cei *adresaţi indirect.*

```
??:offset (assembly time)          0708h:offset (loading time)
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor imediaţi

- Operanzii imediaţi sunt formaţi din date numerice constante calculabile la momentul asamblării.
- Constantele întregi se specifică prin valori binare, octale, zecimale sau hexazecimale.
- Adiţional, este permisă folosirea caracterului _ (underscore) pentru a separa grupuri de cifre.
- Baza de numeraţie poate fi precizată în mai multe moduri:
  - Folosind sufixele H sau X pentru hexazecimal, D sau T pentru zecimal, Q sau O pentru octal şi B sau Y pentru binar; în aceste cazuri numărul trebuie sa înceapa obligatoriu cu o cifră între 0 şi 9, pentru a nu exista confuzii între constante şi 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 oq pentru octal, rESPectiv 0b sau 0y pentru binar;

- 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

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor imediaţi

- **Deplasamentele etichetelor de date şi de cod** reprezinta valori determinabile la momentul asamblarii care rămân constante pe tot parcursul execuţiei programului

```
MOV EAX, et ; transfer în registrul EAX a adresei asociate etichetei et
```

va putea fi evaluată la momentul asamblării drept de exemplu

```
MOV EAX, 8 ; distanţă de 8 octeţi faţă de începutul segmentului de date
```

- "Constanţa" acestor valori derivă din regulile de alocare adoptate de limbajele de programare în general şi care statuează că **ordinea de alocare în memorie** a variabilelor declarate (mai precis distanţa faţă de începutul segmentului de date în care o variabilă este alocată) sau rESPectiv distanţele salturilor destinaţie în cazul unor instrucţiuni de tip **goto** sunt valori constante pe parcursul execuţiei unui program.
- Adică: o variabilă odată alocată în cadrul unui segment de memorie nu îşi va schimba niciodată locul alocării (adică poziţia sa faţă de începutul acelui segment) iar această informaţie determinabilă la momentul asamblării derivă din **ordinea specificării variabilelor la declarare** în cadrul textului sursă şi din **dimensiunea de reprezentare** dedusă pe bază **informaţiei de tip asociate**.

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor registru

- Modul de invocare /accesare directă

```
MOV EAX, EBX
```

- Invocare/accesare indirecta - pentru a indica locaţiile de memorie

```
MOV EAX, [EBX]
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor din memorie

- Operanzii din memorie: cu adresare **directă** şi cu adresare **indirectă**.
- Operandul cu **adresare directă** este o constantă sau un simbol care reprezintă adresa (*segment şi deplasament*) unei instrucţiuni sau a unor date.
- Aceşti 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 locaţii (`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 execuţie** (*loading time* – acest proces final de ajustare a adreselor numindu-se RELOCAREA ADRESELOR = Address Relocation).

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor din memorie

- Un deplasament utilizat ca operand in 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 destinaţie ale unor salturi `(jmp, call, ret, jz etc);`
  - **SS** în adresări SIB ce foloseste EBP sau ESP drept bază (indiferent de index sau scală);
  - **DS** pentru restul accesărilor de date.

Elementele de bază ale limbajului de asamblare

Operatorul de specificare a segmentului

- Operatorul de specificare a segmentului (:) comandă calcularea adresei FAR a unei variabile sau etichete în funcţie de un anumit segment. Sintaxa este:

```
segment:expresie
```

```
ss:[EBX+4]; deplasamentul e relativ la SS
ES:[082h]  ; deplasamentul e relativ la ES
10h:var    ; segmentul este indicat de selectorul 10h, iar
offsetul este valoarea etichetei var.
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor din memorie

- Specificarea explicită a unui registru de segment se face cu ajutorul **operatorului de prefixare segment** (notat **":"** şi care se mai numeşte, 'operatorul de specificare a segmentului').
- ES poate fi utilizat numai in specificari explicite (ca de exemplu `ES:[Var]` sau `ES:[EBX+EAX*2-a]`) sau in cadrul unor instructiuni pe siruri (`MOVSB`)

```
JMP FAR CS:…

JMP FAR DS:…

JMP FAR [label2]
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor din memorie

```
MOV EAX, [v] ; MOV EAX, DWORD PTR DS:[405000]
MOV EAX, [EBX] ; MOV EAX, DWORD PTR DS:[EBX]
MOV EAX, [EBP] ; MOV EAX, DWORD PTR SS:[EBP]
MOV EAX, [EBP*2] ; MOV EAX, DWORD PTR SS:[EBP+EBP]
MOV EAX, [EBP*3] ; MOV EAX, DWORD PTR SS:[EBP+EBP*2]
MOV EAX, [EBP*4] ; MOV EAX, DWORD PTR DS:[EBP*4]
MOV EAX, [EBX+ESP] ; EAX <- dword care incepe la adresa SS:[ESP+EBX]
MOV EAX, [ESP+EBX] ; EAX <- dword care incepe la adresa SS:[ESP+EBX]
MOV EAX, [EBX+ESP*2] ; syntax error – ESP nu poate fi index !
MOV EAX, [EBX+EBP*2] ; EAX <- dword care incepe la adresa DS:[EBX+2*EBP]
MOV EAX, [EBX+EBP] ; EAX <- …DS:…
MOV EAX, [EBP+EBX] ; EAX <- …SS:…
MOV EAX, [EBX*2+EBP] ; EAX <- …SS:…
MOV EAX, [EBX*1+EBP] ; EAX <- …SS:…
MOV EAX, [EBP*1+EBX] ; EAX <- …DS:…
MOV EAX, [EBX*1+EBP*1] ; EAX <- …SS… ; Primul gasit cu * e index ! EBP - baza
MOV EAX, [EBP*1+EBX*1] ; EAX <- …DS….; Primul gasit cu * e index! EBX - baza
MOV EAX, [EBP*1+EBX*2] ; EAX <- …SS:…_
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor cu adresare indirectă

- Operanzii cu *adresare indirectă* utilizează regiştri pentru a indica **adrese din memorie.**
- Deoarece valorile din regiştri se pot modifica la momentul execuţiei, 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:

```
[ registru_de_bază ] + [ registru_index * scală ] + [ constantă ]
```

- **Constanta** este o expresie a cărei valoare este determinabilă la momentul asamblării

```
[EBX + edi + table + 6]; table, 6 sunt constante
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor cu adresare indirectă

- Operanzii **registru_de_bază** şi **registru_index** sunt folosiţi de obicei pentru a indica o adresă de memorie referitoare la un tablou.
- În combinaţie cu factorul de scalare, mecanismul este suficient de flexibil pentru a permite acces direct la elementele unui tablou de înregistrări, cu condiţia ca dimensiunea în octeţi a unei înregistrări să fie 1, 2, 4 sau 8.

```
mov DH, [EDX + ECX * 4 + 3]; octetul superior al cuvuantului supe elementului de tip DWORD cu index dat în ECX, parte a unui vector de înregistrări al cărui adresă (a vectorului) este în EDX este încărcat în DH
```

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor cu adresare indirectă

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

```
push dword [EAX + EBX]; salvează pe stivă dublucuvântul de
la adresa EAX + EBX, asamblorul având libertatea de a
considera EAX drept bază şi EBX drept index sau invers, EBX
drept bază şi EAX drept index
```

```
pop DWORD [ecx]; restaurează vârful stivei în variabila cu
adresa dată de ECX, asamblorul putând interpreta ECX fie ca
bază fie ca index.
```

Toate codificările luate în considerare de către asamblor sunt echivalente iar decizia finală a asamblorului nu are impact asupra funcţionalităţii codului rezultat.

Elementele de bază ale limbajului de asamblare

Utilizarea operanzilor cu adresare indirectă

- asamblorul permite şi exprimări non-standard cu condiţia ca acestea să fie transformabile într-un final în forma standard de mai sus.

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

- asamblorul poate decide între codificare de tip `bază = EAX + index = EAX şi scală = 1` sau `index = EAX şi scală = 2.`

```
lea EAX, [EAX*9 + 12]; EAX ia valoarea EAX * 9 + 12
```

- Deşi 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, instrucţiunea putea fi precizată mai clar sub forma `lea EAX, [EAX + EAX * 8 + 12].`

Să reţinem deci că pentru adresarea indirectă, esenţială este **specificarea între paranteze drepte** a cel puţin unuia dintre elementele componente ale formulei de calcul a offsetului.

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

Str. Mihail Kogălniceanu nr. 1
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro