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

Directive de definire de date

a1 db 0,1,2,'xyz'	; 00h|01h|02h|'x'|'y'|'z' ⇔ 00h|01h|02h|78h|79h|7Ah ; offset(a1) - determinat la încărcarea lui OllyDbg) = 00401000h ; offset(a1) - determinat la asamblare de către NASM = 0
db 300, "F"+3	; 2Ch|49h -Warning - byte data (300 = 1 2Ch) exceeds bounds!
a2 TIMES 3 db 44h	; 44h|44h|44h ; offset a2 = 00401008h, însă offsetul determinat la asamblare de către NASM va fi 8
a3 TIMES 11 db 5,1,3	; 05h|01h|03h … de 11 ori (33 octeţi)
a4 dw a2+1, 'bc'	; offset(a2)=00401008h; a2+1=00401009h; deci se generează 09h|10h|'b'|'c' = 09h|10h|62h|63h (2 cuvinte - words) 09 10 (corect, DAR… această valoare particulară 10h este calculabilă doar DUPA INCARCAREA PROGRAMULUI (LOADING) - deci offset-ul începuturilor de segmente este şi el determinabil doar la momentul încărcării programului - LOADING TIME) Offset-ul variabilelor faţă de începutul segmentelor în care apar sunt constante (de tip pointer, NU scalar !) determinabile la momentul asamblării
a41 dw a2+1, 'b', 'c'	; 09h|10h|'b'|00h|'c'|00h = 09h|10h|62h|00h|63h|00h (3 cuvinte - words)
a42 db a2+1	; - syntax error - OBJ format can only handle 16 or 32 bits relocation !
a44 dw 1009h	; 09h|10h

Elementele de bază ale limbajului de asamblare

Directive de definire de date

a5 dd a2+1, 'bcd'	; 09h|10h|40h|00h|62h|63h|64h|00h
a6 TIMES 4 db '78'	; 37h|38h|37h|38h|37h|38h|37h|38h
a61 TIMES 4 db '7','8'	; 37h|38h|37h|38h|37h|38h|37h|38h
a62 TIMES 4 dw '78'	; 37h|38h|37h|38h|37h|38h|37h|38h
a7 db a2	; syntax err. OBJ format can only handle 16- or 32- relocation (echiv. cu mov ah,a2)
a8 dw a2	; 08h|10h
a9 dd a2	; 08h|10h|40h|00h
a10 dq a2	; 08h|10h|40h|00h|00h|00h|00h|00h
a11 db [a2]	- expression syntax error – pt că [a2] NU este o expresie validă acceptată de către asamblor, nereprezentând "o valoare constantă determinabilă la momentul asamblării" ! Dereferenţierea implicată aici este ceea ce deranjează asamblorul - Conţinutul unei zone de memorie sau al unui registru NU sunt valori constante determinabile la momentul asamblării. Acestea sunt accesibile şi determinabile DOAR la momentul execuţiei !! (run-time).

Elementele de bază ale limbajului de asamblare

Directive de definire de date

`a12 dw [a2]`	`; expression syntax error`
`a13 dd dword [a2]`	`; expression syntax error`
`a14 dq [a2]`	`; expression syntax error`
`a15 dd eax`	`; expression syntax error`
`a16 dd [eax]`	`; expression syntax error`
`mov ax, v`	`; Warning - 32 bit offset in 16 bit field`

Elementele de bază ale limbajului de asamblare

Directive de definire de date

- Pașii urmați de un program de la codul sursă până la rulare:
  - Verificarea sintactică (realizată de *asamblor/compiler/interpretor*)
  - Fișierele **.OBJ** sunt generate de către *asamblor/compiler*
  - Faza de linking (realizată de LINKER = o unealtă furnizată de sistemul de operare, care verifică posibilele DEPENDENȚE dintre aceste fișiere/module OBJ); Rezultatul → fișier **.EXE**
  - Tu (utilizatorul) activezi fișierul exe (prin clic sau apăsarea tastei Enter…)
  - LOADER-ul sistemului de operare caută spațiul necesar în memoria RAM pentru fișierul EXE. Când îl găsește, încarcă fișierul EXE ȘI realizează RELOCALIZAREA ADRESELOR
  - La final, loader-ul cedează controlul procesorului prin specificarea PUNCTULUI DE INTRARE al programului (ex: eticheta start)
  - Faza de run-time începe ACUM…

Elementele de bază ale limbajului de asamblare

Directive de definire de date - 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 octeţii corespunzători acelor zone de memorie pe care le ocupă în conformitate cu directiva specifică aleasă şi respectând ordinea de plasare de tip little-endian
- Deci a NU este de tip byte – ci doar un offset/deplasament şi 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 şi 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 şi atât … un simbol desemnand începutul unei zone de memorie FARA VREUN TIP ASOCIAT
  ```

Elementele de bază ale limbajului de asamblare

Directive de definire de date - Tipuri de date asociate operanzilor

- Atunci DE CE mai asociem în definiţie o directivă de tip de dată ? Pt a INDICA ASAMBLORULUI CUM să populeze cu date/iniţializeze zona de memorie respectivă (fie ca o secvenţă de bytes, fie ca una de words, fie ca una de doublewords)
- Directivele de date se referă la modalitatea de iniţializare concretă a unei zone de memorie şi 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 iniţializare a zonei de memorie alocate variabilei cu valorile dorite

Elementele de bază ale limbajului de asamblare

Directive de definire de date - Tipuri de date asociate operanzilor

- Numele unei variabile este asociat, în limbaj de asamblare, cu offset-ul ei relativ la segmentul în care apare definiția acesteia.
- Offset-urile variabilelor definite într-un program sunt întotdeauna valori constante, determinabile în momentul asamblării/compilării.
- Limbajul de asamblare și C sunt limbaje orientate pe valori, ceea ce înseamnă că totul se reduce, în final, la o valoare numerică — aceasta fiind o caracteristică de nivel scăzut.
- Într-un limbaj de programare de nivel înalt, programatorul poate accesa memoria doar folosind nume de variabile; în schimb, în limbajul de asamblare, memoria este/poate/trebuie accesată DOAR prin folosirea formulei de calcul a offset-ului („formula de la două noaptea”), unde este utilizată și aritmetica pointerilor (aceasta fiind folosită și în C!).
- `mov ax, [ebx]` – operandul sursă nu are un tip de date asociat (reprezintă doar începutul unei zone de memorie) și, din acest motiv, în cazul instrucțiunii MOV, operandul destinație este cel care decide tipul de date al transferului (un word în acest caz), iar transferul se va face în conformitate cu reprezentarea little endian.

Elementele de bază ale limbajului de asamblare

Directive de definire de date

- Segment code (incepe intotdeauna la offset 00402000 - CINE decide asta ?)
- Linkeditorul ia deciziile de acest tip. Adresa de baza pentru incarcarea PE-urilor, cel putin cea implicita (setata de catre linkeditorul de la Microsoft si nu numai) este 0x400000 in cazul executabilelor (respectiv 0x10000000 pentru biblioteci).
- Alink respecta aceasta conventie si completeaza in campul ImageBase al structurii IMAGE_OPTIONAL_HEADER din fisierul P.E. nou construit valoarea 0x400000.
- Cum fiecare “segment”/sectiune din program poate prevedea drepturi diferite de acces (codul este executabil, putem avea segmente read-only etc...), acestea sunt planificate sa inceapa fiecare la adresa cate unei noi pagini de memorie (4KiB, deci multiplu de 0x1000), fiecare pagina de memorie putand fi configurata cu drepturi specifice de catre incarcatorul de programe.

Elementele de bază ale limbajului de asamblare

Directive de definire de date

- In cazul unor programe de mici dimensiuni, implicatia este ca se va obtine urmatoarea harta a programului in memorie (la executare):
  - programul este planificat a fi incarcat in memorie la exact adresa 0x400000 (insa aici vor ajunge structurile de metadate ale fisierului, nu codul sau datele programului in sine)
  - primul “segment” va fi incarcat la 0x401000 (pun ghilimele deoarece nu este un segment propriu-zis ci doar o diviziune logica a programului, nu este asociat direct “segmentul” unui registru de segment – din aceasta pricina se prefera de multe ori denumirea de sectiune in loc de cea de segment)
  - al doilea “segment” va fi incarcat la 0x402000 (segmentul pe care il va folosi procesorul pentru segmentare incepe la adresa 0 si are limita de 4GiB, indiferent de adresele si dimensiunile sectiunilor)
  - va fi pregatit “segment” (sectiune) de importuri, “segment” de exporturi si “segment” de stack in ordinea decisa de catre likeditor (si de dimensiuni prevazute tot de catre acesta), segmente ce vor fi incarcate de la 0x403000, 0x404000 si asa mai departe (incremente de 0x1000 cat timp au dimensiune suficient de mica, in caz contrar fiind nevoie a se folosi pentru increment cel mai mic multiplu de 0x1000 care permite suficient spatiu pentru continutul intregului segment)

Elementele de bază ale limbajului de asamblare

Directive de definire de date

- Conform logicii de decizie a adreselor de inceput ale sectiunilor, putem concluziona ca aici avem o sectiune (de date probabil) inaintea celei de cod, continand sub 0x1000 octeti, motiv pentru care codul porneste imediat dupa, de la 0x402000, harta programului fiind la final: metadate (antete) de la 0x400000, cod la 0x401000 si date la 0x402000 (urmat bineinteles de alte “segmente” pentru stiva, importuri si, optional, exporturi).

Elementele de bază ale limbajului de asamblare

Directive de definire de date

```
Segment code ;starts always at offset 00402000
 Start:
      Jmp Real_start ; offset(instr. JMP) = 00402000 => + 2 octeti
      a db 17 ; offset(a) = 00402002 => + 1 octet
      b dw 1234h ; offset(b) = 00402003 => + 2 octeti
      c dd 12345678h ; offset(c) = 00402005 => + 4 octeti
 Real_start:
      ……
      Mov eax, c ; EAX = 00402005
      Mov edx, [c] ; mov edx, DWORD PTR DS:[00402005]
      ……
      Mov edx, [CS:c] ; mov edx, DWORD PTR CS:[402005]
      Mov edx, [DS:c] ; mov edx, DWORD PTR DS:[402005]
      Mov edx, [SS:c] ; mov edx, DWORD PTR SS:[402005]
      Mov edx, [ES:c] ; mov edx, DWORD PTR ES:[402005]
```

- Efectul fiind in toate cele 5 cazuri EDX:=12345678h DE CE ?

Elementele de bază ale limbajului de asamblare

Directive de definire de date

- Explicaţia este direct legată de modelul de memorie flat – toate segmentele descriu în realitate întreaga memorie, începând de la 0 şi până la capătul primilor 4GiB ai memoriei. Ca atare, [CS:c] sau [DS:c] sau [SS:c] sau [ES:c] vor accesa aceeaşi locaţie de memorie însă cu drepturi de acces potenţial diferite.
- Deşi toţi selectorii (potenţiali DIFERITI – vezi OllyDbg) indică segmente IDENTICE ca adresă şi dimensiune, aceştia pot avea diferenţe în cum le sunt completate alte câmpuri de control şi de acces ale descriptorilor de segment indicaţi de către ei.
- Modelul flat ne asigura ca mecanismul de segmentare este transparent pentru noi, noi nu sesizam diferente intre segmente si, ca atare, scapam complet de grija segmentarii (insa ne intereseaza impartirea logica in segmente a programului, motiv pentru care folosim sectiuni/”segmente” separate pentru cod / date).
- Acest lucru este valabil insa doar cat timp ne limitam la CS/DS/ES si SS!
- **Selectorii FS si GS indica inspre segmente speciale care nu respectă întotdeauna modelul flat (rezervate interacţiunii programului cu S.O-ul), mai precis, [FS:c] nu garantează indicarea aceleiaşi zone de memorie ca şi [CS:c]!**

Elementele de bază ale limbajului de asamblare

Clasificarea erorilor în Informatică

- **Eroare de sintaxă** – ea este diagnosticată de asamblor/compilator (**eroare de asamblare**)
- **Run-time error (eroare la execuţie)** – programul “crapă” – programul se opreste = program crash
- **Eroare logică** = programul funcţionează până la capăt sau ramâne blocat în ciclu infinit, însă GRESIT dpdv LOGIC obţinându-se cu totul alte rezultate decât cele aşteptate
- **Fatal: Linking Error** (de ex în cazul unei definiţii duble de variabilă… 17 module şi o variabilă trebuie să fie DEFINITĂ DOAR într-un singur modul ! Dacă ea este definită în 2 sau mai multe module se va obţine Fatal: Linking Error – Duplicate definition for symbol A1 !!!)

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

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

www.cs.ubbcluj.ro