Arhitectura Sistemelor de Calcul

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

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

Instrucțiuni ale limbajului de asamblare

Instrucțiuni ale limbajului de asamblare

- Forma generală a unui program în NASM + scurt exemplu:

```
bits 32 ; solicităm asamblarea pentru un procesor X86 (pe 32 biţi)

global start ; solicităm asamblorului sa confere vizibilitate globală simbolului denumit start
(eticheta start va fi punctul de intrare în program)

extern ExitProcess, printf ; informăm asamblorul că simbolurile ExitProcess şi printf au
provenienţă străină,evitând astfel a fi semnalate erori cu privire la lipsa definirii acestora

import ExitProcess kernel32.dll ;precizăm care sunt bibliotecile externe care definesc cele
două simboluri: ExitProcess e parte a bibliotecii kernel32.dll (bibliotecă standard a
sistemului de operare)

import printf msvcrt.dll ; printf este funcţie standard C şi se regăseşte în biblioteca
msvcrt.dll (SO)

segment data use32 class=DATA ; variabilele vor fi stocate în segmentul de date (denumit data)

string: db "Salut din ASM!", 0

segment code use32 class=CODE ; codul progr. va fi emis ca parte a unui segment numit code

start:

; apel printf(”Salut din ASM”)

push dword string ;transmitem param. fcţiei printf (adresa şirului) pe stivă (aşa cere printf)

call [printf] ;printf este numele unei funcţii (etichetă = adresă , trebuie indirectată cu [])

; apel ExitProcess(0), 0 reprezentând ”execuţie cu succes”

push dword 0

call [ExitProcess]
```

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Instrucțiuni de transfer de uz general

Instrucțiune	Descriere / Efect
`MOV d, s`	<d> ← <s> (b-b, w-w, d-d)
`PUSH s`	ESP = ESP - 4 și depune <s> în stivă (s – dublucuvânt)
`POP d`	Extrage elementul curent din stivă și îl depune în d (d – dublucuvânt), ESP = ESP + 4
`XCHG d, s`	<d> ↔ <s> ; s,d trebuie să fie L-values
`[reg_segment]XLAT`	AL ← DS:[EBX+AL] sau AL ← reg_segment:[EBX+AL]
`CMOVcc d, s`	<d> ← <s> dacă cc (cod condiție) este adevărat
`PUSHA / PUSHAD`	Depune pe stivă: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
`POPA / POPAD`	Extrage de pe stivă: EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX
`PUSHF / PUSHFD`	Depune EFlags pe stivă
`POPF / POPFD`	Extrage vârful stivei și îl depune în EFlags
`SETcc d`	<d> ← 1 dacă cc este adevărat, altfel <d> ← 0

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Dacă operandul destinaţie al instrucţiunii MOV este unul dintre cei 6 regiştrii de segment atunci sursa trebuie să fie unul dintre cei opt regiştri generali de 16 biţi ai UE sau o variabilă de memorie. Încărcătorul de programe al sistemului de operare preiniţializează în mod automat regiştrii de segment, iar schimbarea valorilor acestora, deşi posibilă din punct de vedere al procesorului, nu aduce nici o utilitate (un program este limitat la a încărca doar valori de selectori ce indică înspre segmente preconfigurate de către sistemul de operare, fără a putea să definească segmente adiţionale).
- Instrucţiunile PUSH şi POP au sintaxa PUSH s şi POP d
- Operanzii trebuie să fie reprezentaţi pe dublucuvânt, deoarece stiva este organizată pe dublucuvinte. Stiva creşte de la adrese mari spre adrese mici, din 4 în 4 octeţi, ESP punctând întotdeauna spre dublucuvântul din vârful stivei.

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Funcţionarea acestor instrucţiuni poate fi ilustrată prin intermediul unei secvenţe echivalente de instructiuni MOV si ADD sau SUB:
- `push eax` ⇔ `sub esp,4`; alocăm spaţiu pentru a stoca valoarea
  `mov [esp],eax`; stocăm valoarea in locaţia alocată
- `pop eax` ⇔ `mov eax,[esp]`; încărcăm în eax valoarea din vârful stivei
  `add esp, 4`; eliberăm locaţia

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- In perspectiva evaluarii efectului unor instructiuni ca PUSH ESP sau POP dword [ESP] trebuie precizat si mai clar ordinea in care se efectueaza (sub)operatiile componente ale instructiunilor PUSH si POP:
  a) **Se evaluează operandul sursă al instrucţiunii**
  (ESP pt PUSH sau respectiv elementul din vârful stivei pt POP)
  b) **Se actualizează corespunzător ESP**
  (ESP := ESP-4 pt PUSH şi respectiv ESP := ESP+4 pt POP)
  c) **Se efectuează atribuirea implicată de efectul instrucţiunii asupra operandului destinaţie**
  (noul vârf al stivei pt PUSH şi respectiv dword [ESP] (acesta fiind acum după scăderea ESP de la pct b) elementul de sub vârful initial al stivei) - pt POP)
- Presupunand ca situatia initiala este ESP = 0019FF74, dupa PUSH ESP vom avea ESP = 0019FF70 si continutul din varful stivei va fi acum 0019FF74.
- Presupunand ca situatia initiala este ESP = 0019FF74 si ca in aceasta locatie se afla valoarea 7741FA29 (varful stivei), dupa POP dword [ESP] vom avea ESP = 0019FF78 si continutul acestei locatii (continutul locatiei din varful stivei) va fi 7741FA29 (deci am putea spune ca „vf.stivei se mută cu o pozitie mai jos”!!).

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Instrucţiunile PUSH si POP permit doar depunerea şi extragerea de valori reprezentate pe cuvânt şi dublucuvânt.
- Ca atare, PUSH AL nu reprezintă o instrucţiune validă (syntax error), deoarece operandul nu este permis a fi o valoare pe octet. Pe de altă parte, secvenţa de instrucţiuni

```
PUSH ax ; depunem ax
PUSH ebx ; depunem ebx
POP ecx ; ecx <- dublucuvântul din vârful stivei (valoarea lui ebx)
POP dx ; dx <- cuvântul ramas în stivă (deci valoarea lui ax)
```

este corectă şi echivalentă prin efect cu

```
MOV ecx, ebx
MOV dx, ax
```

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Adiţional acestei constrângeri (inerentă tuturor procesoarelor x86), sistemul de operare impune ca **operarea stivei să fie obligatoriu făcută doar prin accese pe dublucuvânt** sau multipli de dublucuvânt, din motive de compatibilitate între programele de utilizator şi nucleul şi bibliotecile de sistem. Implicaţia acestei constrângeri este că o instrucţiune de forma PUSH operand16 sau POP operand16 (de exemplu PUSH word 10), deşi este suportată de către procesor şi asamblată cu succes de către asamblor, nu este recomandată, putând cauza ceea ce poartă numele de **eroare de dezaliniere e stivei: stiva este corect aliniată dacă şi numai dacă valoarea din registrul ESP este în permanenţă divizibilă cu 4!**

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Instrucţiunea XCHG permite interschimbarea conţinutului a doi operanzi de aceeaşi dimensiune (octet, cuvânt sau dublucuvânt), cel puţin unul dintre ei trebuind să fie registru. Sintaxa ei este

```
XCHG operand1, operand2
```

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Instrucţiunea XLAT "traduce" octetul din AL într-un alt octet, utilizând în acest scop o tabelă de corespondenţă creată de utilizator, numită tabelă de translatare. Instrucţiunea are sintaxa

```
[reg_segment] XLAT
```

- tabelă_de_translatare este **adresa directă** a unui şir de octeţi.
- Instrucţiunea XLAT pretinde la intrare adresa far a tabelei de translatare furnizată sub unul din următoarele două moduri:
  - DS:EBX (implicit, dacă lipseşte precizarea registrului segment)
  - registru_segment:EBX, dacă registrul segment este precizat explicit
- **Efectul instrucţiunii XLAT este înlocuirea octetului din AL cu octetul din tabelă ce are numărul de ordine valoarea din AL (primul octet din tabelă are indexul 0).**

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- De exemplu, secvenţa

```
mov ebx, Tabela
mov al,6
ES xlat;
```

AL = < ES:[EBX+6] >

depune conţinutul celei de-a 7-a locaţii de memorie (de index 6) din Tabela în AL.

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al informației

- Dăm un exemplu de secvenţă care translatează o valoare zecimală 'numar' cuprinsă între 0 şi 15 în cifra hexazecimală (codul ei ASCII) corespunzătoare:

```
segment data use32
. . .
TabHexa db '0123456789ABCDEF'
numar resb 1
. . .
segment code use32
mov ebx, TabHexa
. . .
mov al, numar
xlat ; AL = < DS:[EBX+AL] >
```

- O astfel de strategie este des utilizată şi se dovedeşte foarte utilă în cadrul pregătirii pentru tipărire a unei valori numerice întregi (practic este vorba despre o conversie valoare numerică registru – string de tipărit).

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al adreselor LEA

- `LEA reg_general, continutul unui operand din memorie`
  ; reg_general <-- offset(mem)
- Instrucţiunea LEA (**L**oad **E**ffective **A**ddress) transferă deplasamentul operandului din memorie *mem* în registrul destinaţie. De exemplu
  ```
  lea eax,[v]
  ```
  încarcă în EAX offsetul variabilei v, instrucţiune echivalentă cu
  ```
  mov eax, v
  ```
- Instrucţiunea LEA are însă avantajul că operandul sursă poate fi o expresie de adresare (spre deosebire de instructiunea mov care nu accepta pe post de operand sursă decât o variabilă cu adresare directă într-un astfel de caz). De exemplu, instrucţiunea
  ```
  lea eax,[ebx+v-6]
  ```
  avand ca efect
  ```
  „mov eax, ebx+v-6”
  ```

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al adreselor LEA

- `LEA reg_general, continutul unui operand din memorie`
  ; reg_general <-- offset(mem)
- Instrucţiunea LEA (**L**oad **E**ffective **A**ddress) transferă deplasamentul operandului din memorie *mem* în registrul destinaţie. De exemplu
  ```
  lea eax,[v]
  ```
  încarcă în EAX offsetul variabilei v, instrucţiune echivalentă cu
  ```
  mov eax, v
  ```
- Instrucţiunea LEA are însă avantajul că operandul sursă poate fi o expresie de adresare (spre deosebire de instructiunea mov care nu accepta pe post de operand sursă decât o variabilă cu adresare directă într-un astfel de caz). De exemplu, instrucţiunea
  ```
  lea eax,[ebx+v-6]
  ```
  avand ca efect
  ```
  „mov eax, ebx+v-6”
  ```
  nu are ca echivalent direct o singură instrucţiune MOV, instrucţiunea
  ```
  mov eax, ebx+v-6
  ```
  fiind incorectă sintactic deoarece expresia `ebx+v-6` nu este determinabilă la momentul asamblării.

Instrucțiuni

Manipularea datelor – Instrucțiuni de transfer al adreselor LEA

- Prin utilizarea directă a valorilor deplasamentelor ce rezultă în urma calculelor de adrese (in contrast cu folosirea memoriei indicate de către acestea), LEA se evidenţiază prin versatilitate şi eficienţă sporite:
  - versatilă prin combinarea unei înmulţiri cu adunări de regiştri şi/sau valori constante
  - eficienţă ridicată datorată execuţiei întregului calcul într-o singură instrucţiune, fără a ocupa 7 circuitele ALU care rămân astfel disponibile pentru alte operaţii (timp în care calculul de adresă este efectuat de către circuite specializate, separate, ale BIU).
- Exemplu: inmulţirea unui număr cu 10

```
mov eax, [număr] ; eax <- valoarea variabilei număr
lea eax, [eax * 2] ; eax <- număr * 2
lea eax, [eax * 4 + eax] ; eax <- (eax * 4) + eax = eax * 5 = (număr * 2) * 5
```

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

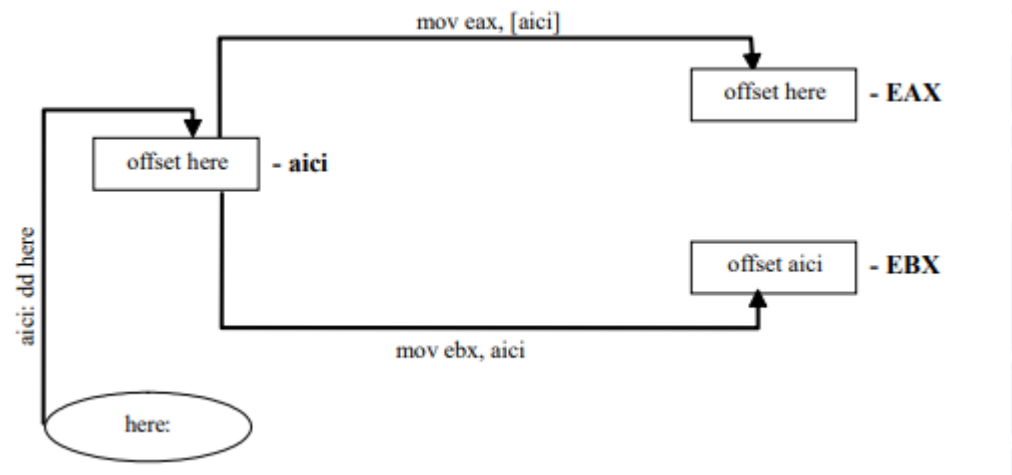
- Prezentăm în continuare un exemplu edificator pentru modul de transfer al controlului la o etichetă, punând în evidenţă deosebirile dintre un transfer direct şi unul indirect.

```
segment data
        aici DD here        ;echivalent cu aici := offsetul etichetei here din segmentul de cod
segment code
        . . .
        here:
        . . .
        mov eax, [aici]     ;se încarcă în EAX conţinutul variabilei aici (adică
                            deplasamentul lui here în cadrul segmentului code – echivalent
                            deci ca efect cu: mov eax, here)
        mov ebx, aici       ;se încarcă în EBX deplasamentul lui aici în cadrul segmentului
                            data (probabil 00401000h)
```

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.



Inițializarea variabilei `aici` și a regiștrilor `EAX` și `EBX`

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

- Prezentăm în continuare un exemplu edificator pentru modul de transfer al controlului la o etichetă, punând în evidenţă deosebirile dintre un transfer direct şi unul indirect.

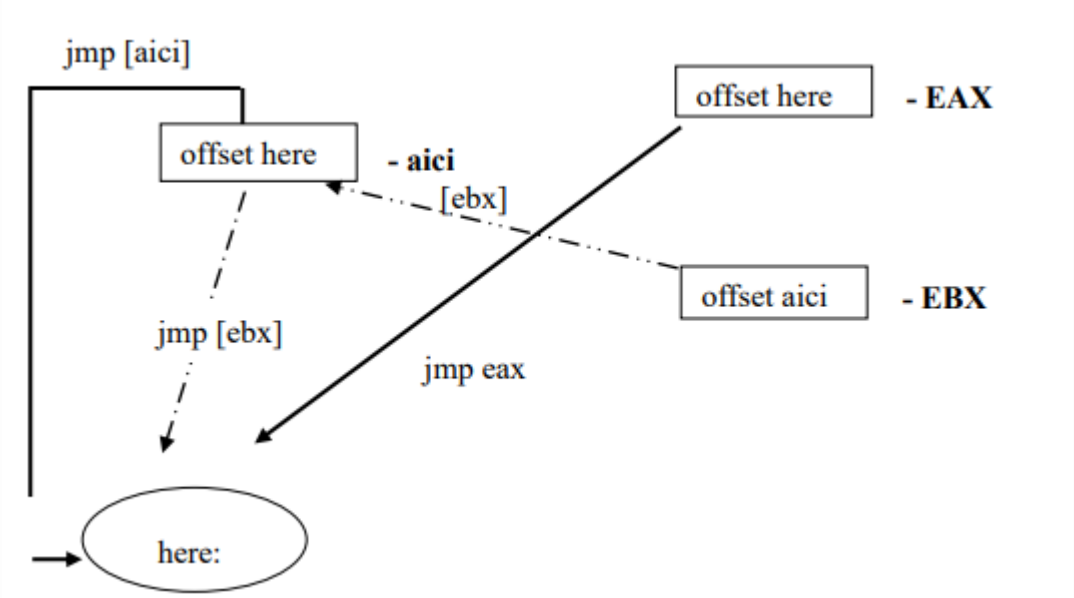
```
jmp [aici]      ; salt la adresa desemnată de valoarea variabilei aici (care este adresa lui here),
                  deci instrucţiune echivalentă cu jmp here
                ; ce face jmp aici? – acelasi lucru ca si jmp ebx ! salt la CS:EIP cu EIP=offset
                  aici) din SEGMENT DATA (00401000h)
                ; salt la niste instr. care merg pana la primul access violation
jmp here        ; salt la adresa lui here (sau, echivalent, salt la eticheta here);
                ; ce face jmp [here] ? – JMP DWORD PTR DS:[00402014] – cel mai probabil
                  Access violation....
jmp eax         ; salt la adresa conţinută în EAX (adresare registru în mod direct), adică la here
                ; ce face prin comparatie jmp [eax]? - JMP DWORD PTR DS:[EAX] – cel mai
                ; probabil Access violation....
jmp [ebx]       ; salt la adresa conţinută în locaţia de memorie a cărei adresă este conţinută în EBX
                  adresare registru în mod indirect - singura situaţie de apel indirect din acest exemplu) – ce
                  face prin comparatie jmp ebx ? - salt la CS:EIP cu EIP=offset (aici) din
                  SEGMENT DATA (00401000h)
                ; salt la niste instr. care merg pana la primul access violation
```

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

- Prezentăm în continuare un exemplu edificator pentru modul de transfer al controlului la o etichetă, punând în evidenţă deosebirile dintre un transfer direct şi unul indirect.
- în `EBX` se află adresa variabilei `aici`, deci se accesează conţinutul acestei variabile
- În această locaţie de memorie se găseşte offset-ul etichetei `here`, deci se va efectua saltul la adresa `here` - ca urmare, ultimele 4 instrucţiuni de mai sus sunt toate echivalente cu `jmp here`



Modalităţi alternative de efectuare a salturilor la eticheta `here`.

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

- Explicaţii asupra interactiunii dintre regulile de asociere implicită a unui offset cu registrul segment corespunzător şi efectuarea corespunzătoare a saltului la offset-ul precizat

```
JMP [var_mem]     ; JMP DWORD PTR DS:[00401704]
                  ; salt NEAR la offset-ul din DS:[00401704]
JMP [EBX]         ; JMP DWORD PTR DS:[EBX]
                  ; salt NEAR la offset-ul din DS:[EBX]
JMP [EBP]         ; JMP DWORD PTR SS:[EBP]
                  ; salt NEAR la offset-ul din SS:[EBP]
JMP here          ; JMP [SHORT] 00402024
                  ; va face saltul DIRECT la offsetul respectiv in code segment conform
                  regulilor de asociere implicită a unui offset cu registrul segment coresp.
```

- Operandul cu adresare **INDIRECTĂ** de după `JMP` ne indică DE UNDE să luăm OFFSET-ul la care să se efectueze saltul **NEAR** (salt în interiorul segmentului de cod curent).
- Ultima instrucţiune exprimă un salt DIRECT la offset-ul calculat ca valoare asociată etichetei here (salt la `CS:here`).

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

- Chiar dacă folosim prefixarea explicită cu un registru segment a operandului destinaţie saltul NU va fi unul FAR. FAR va fi doar ADRESA de la care se va prelua OFFSET-ul unde se va face saltul NEAR.

```
jmp [ss: ebx + 12]
```

```
jmp [ss:ebp +12] ⇔ jmp [ebp +12]
```

- Saltul rămâne în continuare unul NEAR şi se va efectua în cadrul aceluiaşi segment de cod, adică la `CS : valoare offset preluată din SS:[ebp+12]`
- Dacă dorim însă efectuarea saltului într-un segment diferit (salt FAR) trebuie să specificăm explicit acest lucru prin intermediul operatorului de tip FAR, care va impune tratarea operandului destinaţie a instrucţiunii JMP drept O ADRESĂ FAR:

```
jmp far [ebx + 12] => CS : EIP <- adresa far (48 biţi = 6 octeţi)
⇔
jmp far [DS: ebx + 12] => CS : EIP <- adresa far (48 biţi = 6 octeţi)
(9b 7a 52 61 c2 65) -> EIP = 61 52 7a 9b ; CS= 65 c2
„Mov CS:EIP, [adr_far]”
```

- sau prin prefixare explicită:

```
jmp far [ss: ebx + 12] => CS : EIP <- adresa far (48 biţi)
```

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

- Operatorul FAR precizeaza aici faptul că nu doar EIP trebuie populat cu ce se află la adresa de memorie indicată de operandul destinaţie (adresă NEAR = offset) ci şi `CS`-ul trebuie încărcat cu o nouă valoare (`CS:EIP` = adresă FAR).
- Pe scurt avem:
  - valoarea pointer-ului poate fi stocată oriunde în memorie, rezultând că orice specificare de adresă validă la un mov poate apărea la fel de bine şi la jmp (de exemplu `jmp [gs:ebx + esi*8 - 1023]`)
  - pointer-ul în sine (deci octeţii luaţi de la respectiva adresă de memorie) poate fi FAR sau NEAR, fiind, după caz, aplicat fie lui doar lui `EIP` (dacă e NEAR), fie perechii `CS:EIP` daca saltul e FAR.
- Saltul poate fi imaginat ca fiind echivalent, ipotetic, cu instructiuni `MOV` de forma:

```
jmp [gs:ebx + esi * 8 - 1023]      ⇔   mov EIP, [gs:ebx + esi * 8 - 1023]
jmp FAR [gs:ebx + esi * 8 - 1023]  ⇔   mov EIP, DWORD [gs:ebx + esi * 8 - 1023] +
                                       mov CS, WORD [gs:ebx + esi * 8 - 1023 + 4]
```

- La adresa `[gs:ebx + esi * 8 - 1023]` am gasit in exemplul concret utilizat configuraţia de memorie: `7B 8C A4 56 D4 47 98 B7.....`

```
Mov CS:EIP, [memory] -> EIP = 56 A4 8C 7B, CS = 47 D4
```

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR. JMP prin etichete – întotdeauna NEAR!

```
segment data use32 class=DATA
 a db 1,2,3,4
 start3:
        mov edx, eax    ; ok ! – controlul este transferat la start3 şi această instrucţiune este executată !

segment code use32 class=code  ; offset code segment = 00402000
 start:
        mov edx, eax
        jmp start2      ; ok – salt NEAR - JMP 00403000 (offset code1 segment = 00403000)
        jmp start3      ; ok – salt NEAR – JMP 00401004 (offset data segment = 00401000)
        jmp far start2  ; Segment selector relocations are not supported in PE file – syntax error !
        jmp far start3  ; Segment selector relocations are not supported in PE file– syntax error !
                        ; Cele două salturi de mai sus jmp start2 si respectiv jmp start3 se vor efectua la etichetele
                          menţionate, start2 si start3 însă ele nu vor fi considerate salturi FAR (dovada este că precizarea
                          acestui atribut mai sus în celelalte două variante ale instrucţiunilor va furniza syntax error !). Ele
                          vor fi considerate salturi NEAR datorita modelului de memorie FLAT utilizat de catre SO
        add eax,1

 final:
        push dword 0
        call [exit]

segment code1 use32 class=code
 start2:

        mov eax, ebx

 push dword 0
 call [exit]
```

De ce ?... Din cauza “Flat memory model”

Instrucțiuni

Ramificări, salturi, cicluri (modul de transfer al controlului la o eticheta, pag. 142-143 – carte curs)

Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR. JMP prin etichete – întotdeauna NEAR! Concluzii finale.

- **Salturi NEAR – se pot realiza prin oricare dintre cele 3 tipuri de operanzi (etichetă, registru, operand cu adresare la memorie)**
- **Salturi FAR (asta însemnând modificarea şi a valorii din CS, nu numai a valorii din EIP) – se pot realiza DOAR prin intermediul unui operand adresare la memorie pe 48 de biţi (pointer FAR = 6 octeţi). De ce doar aşa şi prin etichete sau regiştri nu ?**
- **Prin etichete, chiar dacă se sare într-un alt segment nu se consideră ca este salt FAR deoarece nu se modifica CS-ul (din cauza modelului de memorie implementat – Flat Memory Model). Se va modifica doar EIP-ul şi saltul se consideră dpdv tehnic ca fiind un salt NEAR.**
- **Prin regiştri nu este posibil deoarece regiştri sunt pe 32 de biţi şi se poate astfel specifica drept operand al unui JMP doar un offset (salt NEAR), deci practic suntem în imposibilitatea de a preciza un salt FAR cu un operand limitat la 32 de biţ**

Instrucțiuni

Ramificări, salturi, cicluri

Instrucțiuni CALL și RET

- Apelul unei proceduri se face cu ajutorul instrucţiunii `CALL`, acesta putând fi apel direct sau apel indirect.
- Apelul direct are sintaxa

  ```
  CALL operand
  ```

  Asemănător instrucţiunii `JMP` şi instrucţiunea `CALL` transferă controlul la adresa desemnată de `operand`.
- În plus faţă de aceasta, înainte de a face saltul, instrucţiunea `CALL` salvează în stivă adresa următoarei instrucţiuni de după `CALL` (adresa de revenire).
- Cu alte cuvinte, avem echivalenţa

  ```
  CALL operand          push A
  A: . . .      ⇔       jmp operand
  ```

Instrucțiuni

Ramificări, salturi, cicluri

Instrucțiuni CALL și RET

- Terminarea execuţiei secvenţei apelate este marcată de întâlnirea unei instrucţiuni `RET`.
- Aceasta preia din stivă adresa de revenire depusă acolo de `CALL`, predând controlul la instrucţiunea de la această adresă.
- Sintaxa instrucţiunii RET este

  ```
  RET [n]
  ```

  unde `n` este un parametru opţional. El indică eliberarea din stivă a `n` octeţi aflaţi sub adresa de revenire.
- Instrucţiunea `RET` poate fi ilustrată prin echivalenţa

```
RET n
(revenire near)  ⇔
```

```
B dd ?
. . .
pop [B]
add esp,[n]
jmp [B]
```

Instrucțiuni

Ramificări, salturi, cicluri

Instrucțiuni CALL și RET

- De cele mai multe ori, după cum este şi natural, instrucţiunile `CALL` şi `RET` apar în următorul context

```
etichetă_procedură:
. . .
ret n
. . .
CALL etichetă_procedură
```

- Instrucţiunea `CALL` poate de asemenea prelua adresa de transfer dintr-un registru sau dintr-o variabilă de memorie.
- Un asemenea gen de apel este denumit **apel indirect**.
  Exemple:

```
call ebx          ;adresă preluată din registru
call [vptr]       ;adresă preluată din memorie (similar cu apelul funcţiei printf)
```

- Rezumând, operandul destinaţie al unei instrucţiuni `CALL` poate fi:
  - numele unei proceduri
  - numele unui registru în care se află o adresă
  - o adresă de memorie

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

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

www.cs.ubbcluj.ro