# Analiza instrucţiunii JMP. Salturi NEAR şi salturi FAR.

**Exemplul 4.3.1.2**. 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
        .       .       .
        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)
```

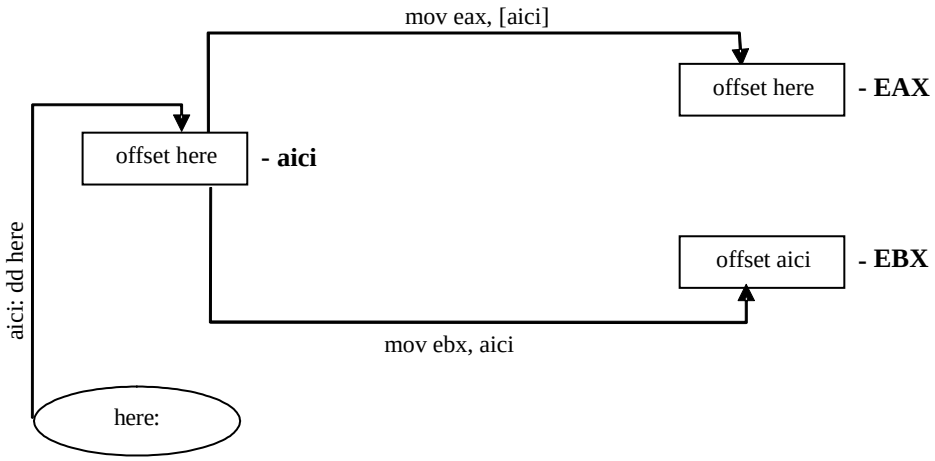


**Fig. 4.4.** Iniţializarea variabilei aici şi a regiştrilor EAX şi EBX.

```
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); 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
```

;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**

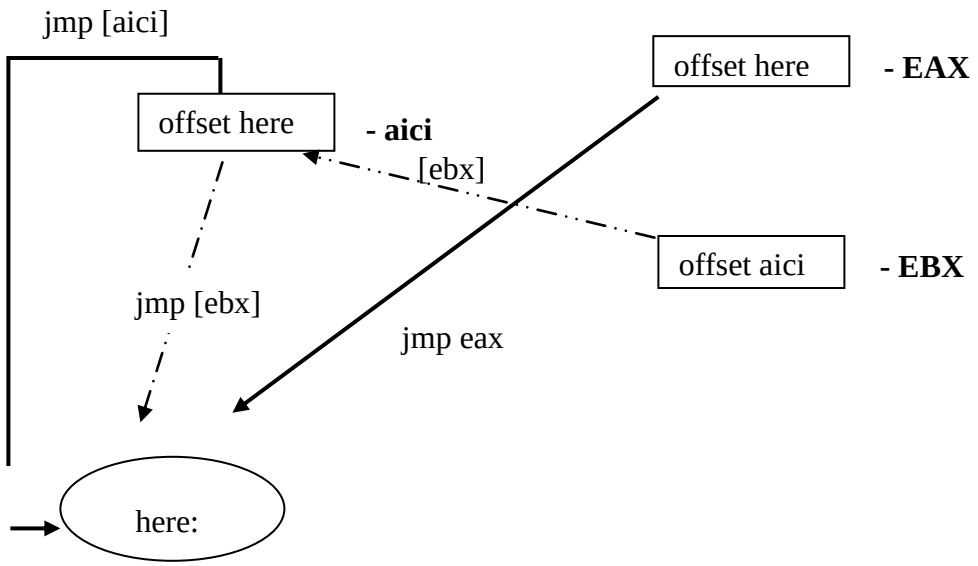


**Fig. 4.5.** Modalităţi alternative de efectuare a salturilor la eticheta *here*.

```
        jmp   [ebp]   ;  JMP DWORD PTR SS:[EBP]
.....................................................................
        here:
              mov ecx, 0ffh
```

---

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 corespunzător.

**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).**

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] equiv with 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) echivalent cu
## 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)**

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

# JMP prin etichete – always 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"**

**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 biţi.**