asm3

Instrucţiuni de salt

Instrucţiunile de salt modifică valoarea registrului contor program (EIP), astfel încât următoarea instrucţiune care se execută să nu fie neapărat cea care urmează în memorie. Sunt utile pentru implementarea, la nivel de limbaj de asamblare, a structurilor de control (testări sau bucle).

Salturile pot fi:

- **necondiţionate**: instrucţiunea ***jmp***
- **condiţionate**: instrucţiuni de forma ***j<condiţie>***

Sintaxa: ***instructiune_salt adresa***

Vom considera în continuare doar cazul în care adresa este o constantă referită de o etichetă.

Exemplul de mai jos ilustrează modul de definire şi utilizare a etichetelor:

```
#include <stdio.h>

void main()
{
  int i;
  _asm {
    mov i, 11
    jmp eticheta
    sub i, 3;     // aceasta instructiune nu se executa
  eticheta:
    add i, 4
  }
  printf ("%d\n", i);
}
```

Saltul necondiţionat - instructiunea JMP

Nu introduce o ramificaţie în program, neexistînd variante de execuţie.

Este util, folosit împreună cu salturi condiţionate, pentru reluarea unei secvenţe de de cod într-o buclă, aşa cum se va vedea într-un exemplu ulterior.

Salturi condiţionate

Introduc o ramificaţie în program, deoarece avem două variante:

- condiţia de salt este adevărată – se face saltul la adresa indicată
- condiţia de salt este falsă – se continuă cu instrucţiunea următoare din memorie ca şi cum nu ar fi existat instrucţiune de salt.

Instrucţiuni care testează indicatorii individuali

Cele mai utile la acest nivel sunt cele care testează indicatorii:

- Carry
- Overflow
- Sign
- Zero

Pentru fiecare indicator există două instrucţiuni de salt condiţionat:

- una care face saltul când indicatorul testat are valoarea 1
- una care face saltul când are valoarea 0

indicator testat	salt pe valoarea 1	salt pe valoarea 0
Carry	jc	jnc
Overflow	jo	jno
Zero	jz	jnz
Sign	js	jns

Exemplu:

```
#include <stdio.h>

void main()
{
  int a, b, s=0;
  printf("a=");
  scanf("%x", &a);
  printf("b=");
  scanf("%x", &b);

  _asm {
    mov eax, a;
    add eax, b;
    jc semnaleaza_depasire; //in Visual C++,
                           //putem sari la o eticheta din codul C
    mov s, eax;
    jmp afiseaza_suma;     //sau in alt bloc asm
  }
semnaleaza_depasire:
  printf ("S-a produs depasire!\n");
  return;
  _asm {
    afiseaza_suma:
  }
  printf ("%x + %x = %x\n", a, b, s);
}
```

Instrucţiuni corespunzătoare operatorilor relaţionali

În practică, utilizăm mai des ramificări dictate de operatori relaţionali: <, <=, !=, etc.

În acest sens este utilă instrucţiunea de comparare ***cmp.***

cmp funcţionează ca şi *sub* (aceleaşi restricţii pentru operanzi, *aceiaşi indicatori modificaţi*), însă nu modifică primul operand (destinaţia). Prin verificarea indicatorilor se poate stabili în urma aceste operaţii relaţia dintre operanzi. Instrucţiunile care fac aceste verificări sunt:

Pentru numere ***fără semn:***

Pentru numere ***cu semn:***

relaţie	instrucţiune	Comentariu
op1 < op2	jb	"Jump if Below"
op1 <= op2	jbe	"Jump if Below or Equal"
op1 > op2	ja	"Jump if Above"
op1 >= op2	jae	"Jump if Above or Equal"

relaţie	instrucţiune	Comentariu
op1 < op2	jl	"Jump if Less than"
op1 <= op2	jle	"Jump if Less than or Equal"
op1 > op2	jg	"Jump if Greater than"
op1 >= op2	jge	"Jump if Greater than or Equal"

Sunt necesare instrucţiuni diferite pentru numere fără semn, respectiv cu semn, deoarece indicatorii ce trebuie verificaţi diferă. De exemplu, comparând 00100110 şi 11001101, ar trebui să obţinem relaţia 00100110 < 11001101 dacă sunt numere fără semn, şi 00100110 > 11001101 dacă sunt numere cu semn.

Independent de statutul bitului celui mai semnificativ (semn sau cifră) funcţionează instrucţiunile:

relaţie	instrucţiune	Comentariu
op1 == op2	je	"Jump if Equal" (identic cu jz)
op1 != op2	jne	"Jump if Not Equal" (identic cu jnz)