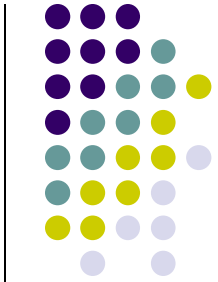


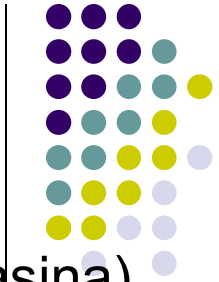
Microprocesorul Intel8086 Programarea in limbaj de asamblare Curs 7

Arhitectura sistemelor de calcul

Lect. Dr. Ozten CHELAI

Facultatea de Matematica si Informatica
Universitatea Ovidius Constanta

Programarea in limbaj de asamblare



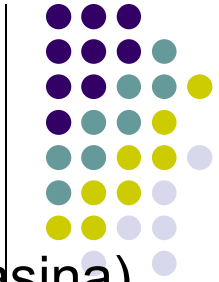
- Limbajul de asamblare = limbaj de programare cu nivel de translatare unu la unu (o instructiune l.a. = o instructiune masina)
- Etape in programare:
  - Editare fisier sursa cu extensia corespunzatoare asamblorului utilizat (*.asm pentru Borland)
  - Asamblare – translatarea fisierului sursa in fisier obiect (fisierul *.obj contine instructiunile masinii) cu asamblorul (tasm *.asm).
  - Link editare - obtinerea fisierului executabil (*.exe). Se utilizeaza link editorul (tlink *.obj pentru Borland)
- Structura fisierului sursa (programul):
  - Specificarea continutului segmentelor de memorie asociate unui program: COD, DATE, STIVA utilizand directive(comenzi) ale asamblorului.

    <nume> SEGMENT [<tip>]

    <corp segment>

    <nume> ENDS

Programarea in limbaj de asamblare



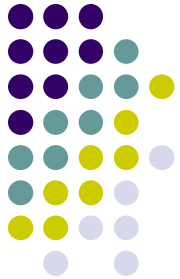
- Limbajul de asamblare = limbaj de programare cu nivel de translatare unu la unu (o instructiune l.a. = o instructiune masina)
- Etape in programare:
  - Editare fisier sursa cu extensia corespunzatoare asamblorului utilizat (*.asm pentru Borland)
  - Asamblare – translatarea fisierului sursa in fisier obiect (fisierul *.obj contine instructiunile masinii) cu asamblorul (tasm *.asm).
  - Link editare - obtinerea fisierului executabil (*.exe). Se utilizeaza link editorul (tlink *.obj pentru Borland)
- Structura fisierului sursa (programul):
  - Specificarea continutului segmentelor de memorie asociate unui program: COD, DATE, STIVA utilizand directive(comenzi) ale asamblorului.

    <nume> SEGMENT [<tip>]

    <corp segment>

    <nume> ENDS

Structura programului in limbaj de asamblare

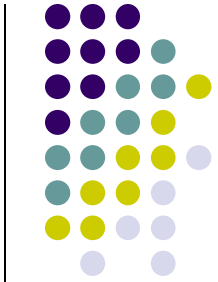


.model <tip_model_memorie>
.data [segment]
[.data ends]
[.stack nr]
.code [segment]
End

Unde:

- **directiva .model specifica tipul de model de memorie folosit dupa cum urmeaza** (tiny, small, medium, compact, huge)
- **directiva .data** precede declaratiile de date si initializarile
- **directiva .code** precede instructiunile ce reprezinta codul programului
- **directiva .stack nr** specifica numarul de octeti alocati stivei
- [segment] specifica numele asociat segmentului de memorie si este optional.

Programarea in limbaj de asamblare
Declaratiile de date



- Formatul general al declaratiilor de date:

  [<nume>] <tip> <lista_expresii>

- unde:
  - **nume** = prin care este referita data; valoarea este valoarea este adresa la care se gaseste in memorie la referire;
  - **tip** =tipul datelor ce specifica spatiul de memorie ocupat

    DB – pentru date de tip octet

    DW – date de tip cuvant

    DD – date de tip pointer (dublu cuvant)

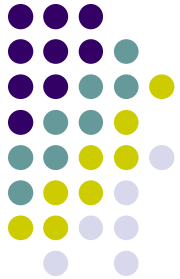
    DQ – date de tip virgula mobila, de 8 octeti; folosite pentru reprezentarea numerelor reale

    DT – date de 10 octeti pentru reprezentarea numerelor BCD
- lista expresii = valorile cu care se initializeaza zona de date rezervate pentru declaratia respectiva;
  - **?** indica rezervarea fara initializare.
  - pentru a initializa o zona de memorie cu aceeasi valoare se poate folosi functia de multiplicare dup cu urmatoarea sintaxa:

    <nr.> dup(<valoare>)

    unde <nr.> reprezinta factorul de multiplicare, iar <valoare> valoarea care se multiplica.

Programarea in limbaj de asamblare
Codul program



- **Initializarea segmentului de date** se va face cu instructiunile:

```
mov ax,@data
mov ds,ax
```

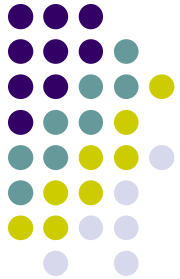
- **Terminarea programului** se face cu apelul functiei SO de terminare normala a programului

```
mov ah,4ch
int 21h
```

- **end** indica sfarsitul fisierului sursa.
- Functii sistem utilizate pentru operatiile de I/E (vezi documentatie techelp.bat)
  - Preluarea caracterelor de la tastatura:
    - functia cu numarul 1 - preia un caracter in registrul AL
    - functia cu numarul 3fh – preia un sir de caractere intr-o zona de memorie indicata de reg. DX
  - Afisarea pe ecran:
    - functia cu numarul 2 – afiseaza un caracter pe ecran in pozitia curenta a cursorului
    - functia cu numarul 9 – afiseaza pe ecran sirul pointat de reg. DXin pozitia curenta a cursorului
  - Utilizare

```
mov ah,nrFunctie
int 21h
```

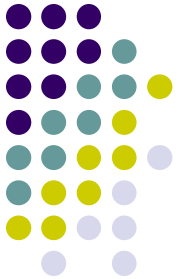
Programarea in limbaj de asamblare



Exemplu de program in limbaj de asamblare

```
        .model small
        .stack 100h
        .data
sir     db 80 dup('$')
m1      db 'Primul program in limbaj de asamblare',13,10,'$'
m3      db ‘Introduceti sirul:$’
m2      db 'Sirul preluat de la tastatura este: ','$'
        .code
;initializare segment de date
mov ax,@data
        mov ds,ax
;afisare mesaj m1
        mov ah,9h
        mov dx,offset m1
        int 21h
;afisare mesaj m3
mov ah,9h
        mov dx,offset m1
        int 21h
;preluare sir de la tastatura si depunere in memorie la adresa din dx
        mov bx,0
        mov cx,80
        mov ah,3fh
        mov dx,offset sir
        int 21h
;afisare mesaj m2
        mov ah,9h
        mov dx,offset m2
        int 21h
;afisare sir introdus
        mov ah,9h
        mov dx,offset sir
        int 21h
;apel functie de terminare normala a programului
        mov ah,4ch
        int 21h
end ; terminare program
```

Setul de instructiuni al I8086



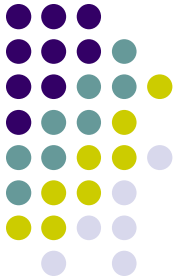
- Mnemonica generală (sintaxa) a instrucţiunilor este

  nume_instrucţiune destinaţie, sursă

  Operanzii destinaţie pot fi:
  - regiştri ai microprocesorului (*reg*)
  - locaţii de memorie (m*em*)

  Operanzii sursă pot fi:
  - regiştri ai microprocesorului (*reg*)
  - locaţii de memorie (*mem*)
  - date constante (data)
- La microprocesorul I8086, setul de instrucţiuni este organizat în 6 grupe.
  - instrucţiuni pentru transferuri de date
  - instrucţiuni aritmetice
  - instrucţiuni logice
  - instrucţiuni pentru manipularea şirurilor
  - instrucţiuni pentru controlul transferului programului
  - instrucţiuni pentru controlul procesorului.

Setul de instructiuni al I8086



1. **Instrucţiuni pentru transferuri de date**

Se împart în patru clase:

- **cu scop general**
- **specifice cu acumulatorul**
- **cu obiect adresă**
- **referitoare la indicatorii de condiţie**

a) **Instrucţiunile pentru transferuri de date cu scop general**:

- de tip **MOV** - transferă o valoare din sursă în destinaţie (realizeză o copie a sursei în destinaţie) - reprezintă instrucţiunea de **atribuire** în limbaj de asamblare

MOV dest,sursa

- instrucţiune are multe limitări =>tipuri de instrucţiuni de tip MOV:

```
mov    reg, reg
mov    mem, reg
mov    reg, mem
mov    mem, immediate data
mov    reg, immediate data
```

- Observaţii referitoare la instrucţiunea MOV:
  - nu afectează indicatorii de condiţie
  - unul din operanzi se află întotdeauna într-un registru.
  - nu sunt admise transferuri din memorie în memorie
  - nu se pot încărca regiştrii segment cu valori imediate de date
  - unele instrucţiuni MOV sunt mai rapide decât altele (de ex. MOV ax,mem e mai rapidă decât MOV reg, mem)
  - operanzii trebuie să fie de acelaşi tip şi pot fi de dimensiunea octeţilor, cuvintelor sau dublu cuvintelor (>I386). Se poate specifica dimensiunea transferului în instrucţiunea Mov dacă se foloseşte sintaxa:

```
mov    byte ptr [bx], 0
mov    word ptr [bx], 0
mov    dword ptr [bx], 0 – numai >I386
```

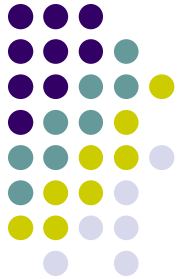
  - când încărca în regiştri valori mai mici decât dimensiunea regiştrilor, partea mai puţin semnificativă se stochează în partea low a registrului
  - pentru încărcarea constantelor în regiştri segment se pot folosi de exemplu instrucţiunile:

```
mov    ax, 40h
mov    es, ax
```

- Exemplu. Interschimbarea continutului registrilor AX si BX:

```
mov CX,AX
mov AX,BX
mov BX,AX
```

Instrucţiunile pentru transferuri de date cu scop general



- **instrucţiuni cu stiva (PUSH, POP)**
  - Registrul SP indica varful stivei
  - Push pentru salvarea datelor in stiva

    PUSH sursa

    operatii: [SP] <- sursa, SP=SP-2 si
  - Pop pentru extragerea datelor din stiva

    POP destinatie

    operatii: SP=SP+2, destinatie <- [SP]
  - Exemplu: interschimbarea continutului registrilor AX si BX:

    ```
    push AX
    push BX
    pop AX
    pop BX
    ```
- **instrucţiunea XCHG** (exchange)

  XCHG sursa, destinatie

  interschimbă sursa cu destinatia.
  - Exemplu: interschimbarea continutului registrilor AX si BX:

    ```
    XCHG AX,BX
    ```