ARHITECTURA MICROPROCESOARELOR
LUCRAREA DE LABORATOR NR. 2

Programarea microprocesorului Intel 8086. Instructiuni de transfer. Instructiuni aritmetice

Scopul lucrării

a) Elemente de limbaj de asamblare.
b) Instructiuni de transfer.
c) Instructiuni aritmetice.

1. Elemente de limbaj de asamblare pentru microprocesorul Intel 8086

1.1. Tipuri de date utilizate de microprocesorul Intel 8086

Reprezentarea interna a informatiilor (instructiuni, date, adrese, comenzi, etc.) este realizata in binar. De aceea, *tipurile de date elementare* (grupurile de biti) utilizate de microprocesorul Intel 8086 au urmatoarele dimensiuni si denumiri:

- ***bitul*** (**b**it) = cifra binara,
- *octetul* (**B**yte) = grup de *8 biti*,
- *cuvantul* (**W**ord) = grup de *16 biti = 2 octeti* (MSB = octetul superior, LSB = octetul inferior),
- *dublul-cuvant* (**D**ouble-word) = grup de *32 biti = 4 octeti = 2 cuvinte* (MSW = cuvantul superior, LSW = cuvantul inferior).

Dublul-cuvint este tipul de data necesar pentru memorarea unei adrese logice, fapt pentru care se numeste si "*pointer*".

Un tip de date derivat este *inregistrarea de biti*, formata din 8 sau 16 biti (octet sau cuvant), compusa din campuri de biti de lungimi variabile, cu semnificatii diferite.

Dintre *tipurile de date compuse*, necesare pentru structuri de date complexe, cel mai utilizat este *sirul de date*, care permite memorarea tablourilor (vectorilor, matricilor, etc.). Un alt tip compus este *stuctura*, care este o multime de date de tip heterogen (octeti, cuvinte, etc.).

1.2. Structura unui program in limbaj de asamblare pentru microprocesorul Intel 8086

Un exemplu de program in limbaj de asamblare pentru microprocesorul Intel 8086 este dat in continuare. Programul efectueaza suma a N=5:

```
DSEG        SEGMENT                     ; segmentul de date
```

```
TAB         LABEL  BYTE              ; eticheta TAB
TABW        DW  5  DUP(7000H)  ; sirul de cuvinte (tabloul) TABW
REZ         DB  ? , ? , ? , ?; sirul de octeti (tabloul) REZ
ZERO        EQU  0                   ; constanta ZERO

DSEG        ENDS

STIVA       SEGMENT                  ; segment de stiva
            DW  40  DUP(?)
VIRF        LABEL  WORD              ; eticheta VIRF
STIVA       ENDS

CSEG   SEGMENT                       ; segment de cod (program)

ASSUME CS: CSEG, DS: DSEG, SS: STIVA, ES: DSEG

START:                               ; eticheta de inceput a programului
        MOV  AX, DSEG                ; initializari
        MOV  DS, AX
        MOV  ES, AX
        MOV  AX, STIVA
        MOV  SS, AX
        MOV  SP, OFFSET  VIRF     ; initializare pointer de stiva
        MOV  AX, 0                ; initializare registre utilizate  MOV  DX, 0
        MOV  CX, LENGTH  TABW        ; CX = 5 = lungimea TABW
        MOV  BP, SIZE  TABW          ; BP = 10 = 5*2 =  dimensiunea TABW
 NEXT:
        SUB  BP, TYPE  TABW          ; BP = BP - 2
        ADD  AX, DS: TABW [BP]  ; AX = AX + cuvant curent din TABW
        ADC  DX, ZERO                ; DX = DX + transportul anterior
        LOOP NEXT                 ; CX = CX-1
                                     ; daca CX <> 0 salt la NEXT
                                     ; altfel trece la instruct. urmatoare
        MOV  WORD PTR REZ, AX        ; stocarea LSW la adresa REZ
        MOV  WORD PTR REZ + 2 , DX   ; stocare MSW la adresa REZ+2       MOV  AH, 4CH
                  ; functie DOS 4CH (terminare proces)
        INT  21H                     ; intrerupere SW - apel fct. DOS 4CH

 CSEG ENDS
        END  START                ; sfarsitul programului
```

O varianta de plasare in memorie a codurilor obtinute in urma asamblarii programului anterior, reprezentata pe harta memoriei principale, este urmatoarea:

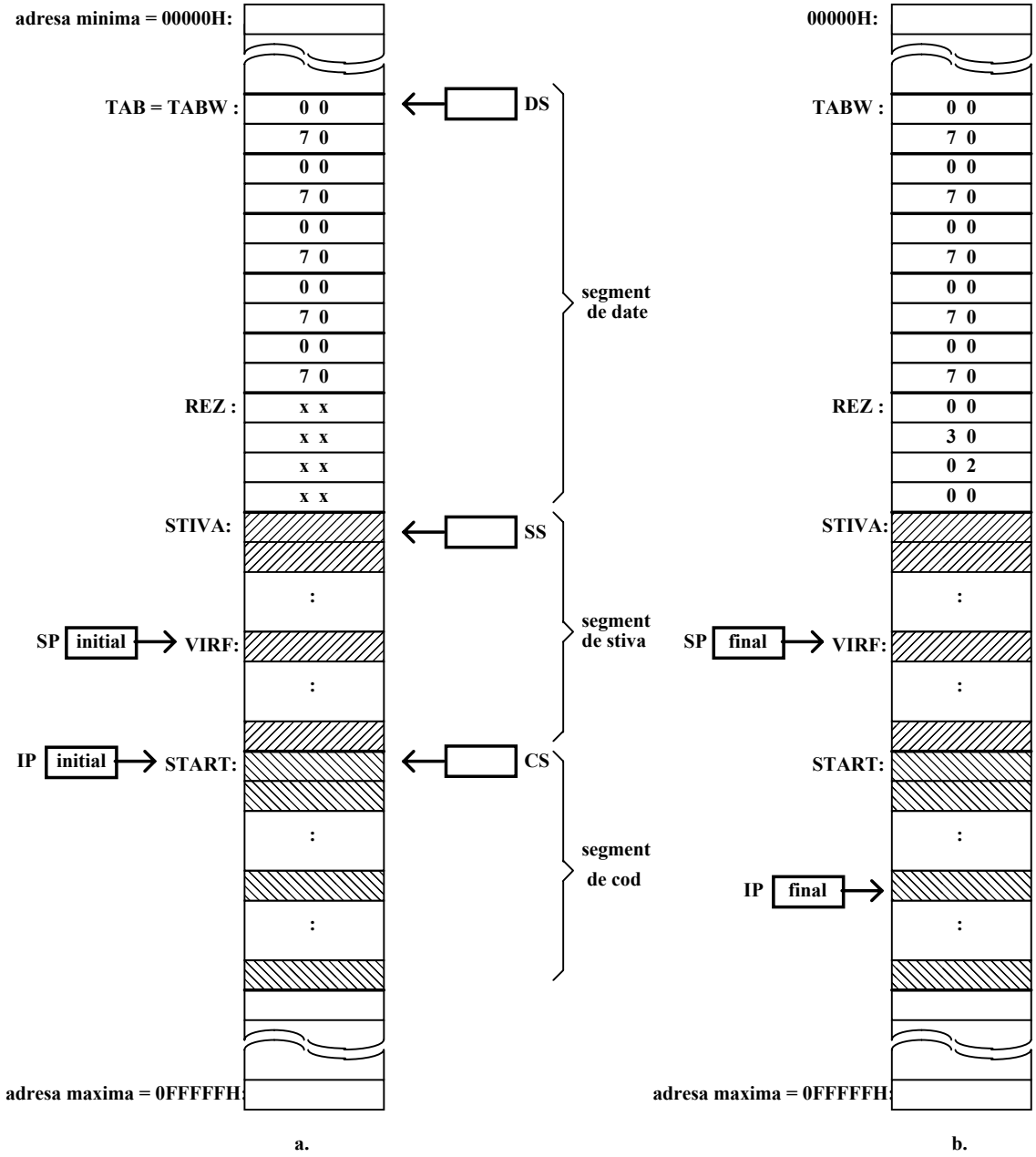

Fig. 1. Harta memoriei asociata programului de adunare a N = 5 numere de 16 biti din memorie, cu rezultat pe 32 de biti.

a. Configuratia initiala:
AX = 0, DX = 0, CX = 5, BP = 10
REZ = XXXX H : XXXX H

b. Configuratia finala:
AX = 3000H, DX =0002H, CX = 0, BP = 0
REZ = 0002 H : 3000 H = 23000H = 5 * 7000H

1.3. Instructiunile limbajelor de asamblare pentru microprocesorul I8086

Un *program in limbaj de asamblare este alcatuit din* mai multe linii sursa. O linie sursa este alcatuita din urmatoarele cimpuri:

| Eticheta: | Codul operatiei (mnemonica) Operanzi de tip imediat | ;Comentariu |

unde intre | | sunt cuprinse elementele optionale.

Cimpul eticheta este *facultativ* si reprezinta numele simbolic al adresei din memorie la care se afla codul numeric al unei instructiuni.

Mnemonica codului de operatie este numele simbolic al instructiunii.

Cimpul operanzi poate contine doi operanzi, unul sau nici unul, in functie de tipul instructiunii. Operanzii pot fi datele asupra carora actioneaza instructiunea, adrese sau alte informatii auxiliare.

Comentariul este optional. El serveste doar la marirea inteligibilitatii programului. La asamblare textul comentariului este ignorat de catre asamblor.

1.4. Operatorii asamblorului TASM

Asupra operanzilor instructiunilor *pot fi efectuate operatii si de catre programul asamblor*. Acest fel de operatii sunt specificate prin *operatori* si sunt efectuate de catre asamblor odata cu asamblarea textului sursa.

Operatorii utilizati de catre asamblorul TASM sunt de mai multe tipuri:

1. Operatori modificatori de valori:

a. *Operatori aritmetici* (pentru operatiile aritmetice elementare: +, -, *, /, MOD).

Exemplu:

MOV WORD PTR REZ + 2 , DX

Asamblorul efectueaza operatia de adunare intre adresa REZ (componenta de tip offset) si 2, rezultatul fiind utilizat de catre codul obiect.

b. *Operatorul de indexare* [ ], care permite este utilizat la adresare.

Exemplu:

ADD AX, DS: TABW [BP]

Asamblorul utilizeaza registrul BP pentru a calcula adresa efectiva *TABW + BP*, necesara pentru obtinerea adresei fizice din adresa logica *segment (DS) : offset (TABW + BP).*

2. Operatori generatori de valori:

a. *SEG* genereaza (returneaza) valoarea *adresei de baza a segmentului* in care se afla variabila/eticheta careia i se aplica.

b. *OFFSET* genereaza (returneaza) valoarea *adresei relative ("offset"-ului)* fata de adresa de inceputul segmentului in care este declarata variabila/eticheta careia i se aplica.

Exemplu:

MOV SP, OFFSET VIRF

Asamblorul inlocuieste OFFSET VIRF cu offset-ul etichetei VARF.

c. *TYPE* genereaza (returneaza) *numarul de octeti ai unui element al variabilei* careia i se aplica (1 pentru octeti, 2 pentru cuvinte si 4 pentru dublu-cuvinte).

Exemplu:

```
SUB  BP, TYPE  TABW
```

Asamblorul inlocuieste TYPE TABW cu 2.

d. *LENGTH* genereaza (returneaza) *numarul de elemente* pe care le are variabila careia i se aplica (lungimea, ex. LENGTH TABW = 5).

Exemplu:

```
MOV  CX, LENGTH  TABW
```

Asamblorul inlocuieste LENGTH TABW cu 5.

e. *SIZE* genereaza (returneaza) *numarul de octeti alocat* unei variabile (*dimensiunea* in octeti). Se observa ca TYPE*LENGTH = SIZE.

Exemplu:

```
MOV  BP, SIZE  TABW
```

Asamblorul inlocuieste SIZE TABW cu 2*5 = 10.

f. *HIGH si LOW* genereaza (returneaza) octetul cel mai semnificativ (MSB), respectiv cel mai putin semnificativ (LSB) al unei expresii.

3. Operatori modificatori de atribute:

a. *PTR* care modifica tipul unei variabile/etichete: BYTE, WORD, DWORD

Exemple:

```
MOV  WORD PTR REZ, AX
MOV  WORD PTR REZ + 2 , DX
```

In ambele cazuri asamblorul utilizeaza elementele variabilei REZ, declarata initial de tip octet, sub forma de cuvinte (grupuri de doi octeti). Astfel se obtine compatibilitate cu tipul cuvant al registrelor AX si DX.

b. *Operatorul :* care modifica segmentul implicit al unei variabile/etichete (utilizat pentru precizarea adresei fizice) intr-un segment explicit (ex. DS: TABW [BP]).

Exemplu:

```
ADD  AX, DS: TABW [BP]
```

Asamblorul utilizeaza registrul DS pentru a specifica segmentul de date pentru calculul adresei adresei fizice din adresa logica *segment (DS) : offset (TABW + BP).*

Asocierile implicite (subintelese) ale registrelor utilizate pentru adresare: BX, SI, DI, SP, BP si IP cu registrele segment, ca si *posibilitatile utilizarii operatorului : pentru a modifica explicit aceste asocieri* sunt date in tabelul 1.

Tab.1. Asocierile implicite si explicite ale registrelor segment si offset

Tip operatie	Registru segment implicit	Registre segment utilizabile explicit	Sursa offset-ului
Incarcarea codurilor instructiunilor in coada Q	CS	-	IP
Operatii cu stiva	SS	-	SP
Transferuri date, cu exceptiile:	DS	ES, SS, CS	EA (memorie)
- Siruri sursa	DS	ES, SS, CS	SI
- Siruri destinatie	ES	-	DI
- Registrul BP	SS	DS, ES, CS	EA (memorie)

1.5. Directivele asamblorului TASM

Un program scris in limbaj de asamblare contine alaturi de instructiunile propriu-zise si *pseudo-instructiuni (directive)*, care se adreseaza programului asamblor modificindu-i modul de lucru sau permitind specificarea datelor.

1. Directivele care specifica *segmentele* utilizate sunt:

a. *Directiva SEGMENT* - folosita pentru a marca inceputul unui segment (ex. DSEG, CSEG, STIVA);

Exemple:

DSEG SEGMENT

defineste un segment de date numit DSEG,

STIVA SEGMENT

defineste un segment de stiva (neutilizat aici) numit STIVA,

CSEG SEGMENT

defineste un segment de cod (de program) numit CSEG.

b. *Directiva ENDS* - folosita pentru a marca sfirsitul segmentului;

Exemple:

DSEG ENDS

declara sfirsitul segmentului de date DSEG,

STIVA ENDS

declara sfirsitul segmentului de stiva STIVA,

CSEG ENDS

declara sfirsitul segmentului de cod CSEG.

c. *Directiva ASSUME* - informeaza TASM asupra registrelor de segment prin care vor fi accesate segmentele logice definite anterior prin directiva SEGMENT (adresele de baza ale segmentelor logice).

Exemplu:

ASSUME CS: CSEG, DS: DSEG, SS: STIVA, ES: DSEG

informeaza asamblorul ca segmentele CSEG, STIVA si DSEG vor avea adresele de baza incarcate in CS, SS, DS si ES.

2. *Directiva EQU* permite declararea constantelor. *Constantele* astfel declarate sunt inlocuite cu numere propriu-zise in momentul asamblarii. Pentru ele nu se aloca spatiu de memorie.

Exemplu:

ZERO EQU 0

Asamblorul inlocuieste in program constanta ZERO cu valoarea numerica 0.

3. Pentru *declararea variabilelor* se utilizeaza urmatoarele directive:

a. *Directiva DB* (Define Byte) declara octeti sau siruri de octeti.

Exemplu:

REZ DB ? , ? , ? , ?

declara o variabila REZ de tip sir de octeti formata din 4 octeti neinitializati

b. *Directiva DW* (Define Word) declara cuvinte sau siruri de cuvinte.

Exemple:

TABW DW 5 DUP(7000H)

declara o variabila TABW de tip sir de cuvinte formata din 5 cuvinte identice initializate cu 7000H

DW 40 DUP(?)

declara un sir de 40 de cuvinte neinitializate fara nume (simpla alocare)

c. *Directiva DD* (Define Double-word) declara dublu-cuvinte sau siruri de dublu-cuvinte.

Variabilele sint definite ca rezidente la o anumita adresa relativa (*offset*) in cadrul unui anumit *segment* si sunt caracterizate de tipul datelor.

Se observa ca pentru *rezervarea memoriei variabilelor neinitializate* se utilizeaza operatorul:

? = rezervare zona de memorie pentru variabila neinitializata

iar pentru *precizarea valorilor multiple* ale variabilelor se utilizeaza operatorul:

DUP = precizare numar replici (DUPlicate).

4. Declararea *etichetelor* utilizate pentru referirea la date din segmentele de date sau stiva (etichete adrese de variabile) se face cu **directiva LABEL**, eventual precizand si atributul.

Exemple:

TAB LABEL BYTE

defineste o eticheta cu numele TAB inaintea variabilei TABW, permitand accesul la variabila TABW octet cu octet (varianta pentru BYTE PTR !),

VIRF LABEL WORD

defineste eticheta VIRF.

Etichetele sunt (ca si variabilele) nume simbolice de adrese. Ele sunt caracterizate de un anumit *offset* in cadul unui *segment*.

In general etichetele identifica instructiunile. In acest caz etichetele pot fi referite in alte instructiuni pentru executarea salturilor in program. Daca referirile la o eticheta sunt facute in cadrul segmentului in care ea este definita ("home segment") atunci se spune ca ea are atributul NEAR. O eticheta poate fi referita intr-o instructiune a altui segment logic daca poarta atributul FAR. Atributele etichetelor se stabilesc la definirea acestora.

Pentru declararea etichetelor in segmentul de program (etichete adrese de instructiuni) se utilizeaza si operatorul:

:

Exemple:

START:

declara eticheta de inceput a codului executabil

NEXT:

declara eticheta NEXT utilizata pentru iteratii

5. Directiva END, utilizata pentru declararea sfarsitului de program (cod).

Exemplu:

END START

indica asamblorului ca programul inceput la eticheta START se sfarseste.

6. Directiva RECORD, utilizata pentru definirea *inregistrarilor de date*. Formatul declaratiei de definire a unei inregistrari este:

nume_inreg RECORD *nume_camp* : *expresie* | = *expresie'* | |,..|

unde:

numele campurilor (*nume_camp*, ...) sunt unice;

expresia *expresie* este evaluata la o constanta intreaga intre 1 si 16 si specifica numarul de biti (lungimea) ai campului; suma lungimilor trebuie sa fie mai mica sau egala cu 16, respectiv 8;

expresie' este optionala si serveste ca valoare initiala a campului;

|,... | indica repetarea optionala a lui *nume_camp* : *expresie* | = *expresie'* |

Exemplul 1:

CHIPS RECORD RAM:7, EPROM:4, ROM:5

defineste o inregistrare cu 16 biti formata din 3 campuri cu numele RAM, EPROM si ROM avand lungimile de 7, 4, si respectiv 5 biti.

Exemplul 2:

CHIPS RECORD RAM:7=4, EPROM:4=2, ROM:5=0

defineste o inregistrare cu 16 biti formata din 3 campuri cu numele RAM, EPROM si ROM avand lungimile de 7, 4, si respectiv 5 biti, cu precizarea valorilor initiale ale campurilor.

7. Directiva STRUCT, utilizata pentru definirea *structurilor*. Formatul declaratiei de definire a unei structuri este:

nume_structura STRUCT

| *nume_camp* / {DB | DW | DD} *expresie* | |, ..|

nume_structura ENDS

unde:

numele campurilor (*nume_camp*, ...) sunt unice;
|,... | indica repetarea optionala a lui *nume_camp* {DB | DW | DD} *expresie*

Exemplu:

```
PROCES      STRUCT
        STARE       DB      0
        VAL_CRT     DW      ?
PROCES      ENDS
```

2. Instructiuni de transfer

Instructiunile de transfer intre registre sau intre un registru si memorie realizeaza operatiile de atribuire (copiere) si de permutare. Operatii de transfer pot fi realizate si intre registre si porturi.

1. Instructiunea de atribuire (copiere) are forma:

MOV *operand1, operand2*

si efectul:

operand1 = operand2

Se observa ca sensul in care se face atribuirea este de la dreapta la stanga (sens utilizat in general pentru scrierea functiilor si a expresiilor matematice). Astfel, se poate spune ca forma instructiunii este:

MOV *destinatie, sursa*

Exemplu:

MOV AX, BX

copiaza in registrul AX valoarea continuta in registrul BX

2. Instructiunea de permutare (inter-schimbare) are forma:

XCHG *operand1, operand2*

si efectul:

temp = operand1
operand1= operand2
operand2 = temp

Astfel, cei doi operanzi isi schimba continutul intre ei, operanzii fiind si sursa si destinatie.

Exemplu:

MOV CX, BX

copiaza in registrul CX valoarea continuta in registrul BX, si in registrul BX valoarea initiala a lui CX

3. Instructiunea de citire dintr-un port are forma:

IN acumulator, port

si efectul:

acumulator = port

4. Instructiunea de scriere intr-un port are forma:

OUT port, acumulator

si efectul:

port = acumulator

3. Instructiuni aritmetice

Instructiunile aritmetice ale microprocesorului Intel 8086 utilizeaza 1 sau 2 operanzi.

3.1. Instructiuni aritmetice care utilizeaza 2 operanzi

Instructiunile care utilizeaza 2 operanzi sunt: adunari (cu sau fara bitul Carry de la operatia anterioara), scaderi (cu sau fara bitul Carry de la operatia anterioara), inmultiri, impartiri. In cele ce urmeaza *operand1* este sursa si destinatie iar *operand2* este doar sursa.

1. Instructiunea de adunare are forma:

ADD *operand1, operand2*

si efectul: *operand1 = operand1 + operand2*

Exemplu (initial *AX = 7FFFH* si *BX = 8000H*):

ADD BX, DX

Efect:

```
                1000 0000 0000 0000 B      =  8000 H +
                1100 0000 0000 0000 B      = C000 H
                ---------------------------------------------
(CF = 1)        0100 0000 0000 0000 B      =  4000 H
```

Valori in urma executiei instructiunii: *BX = 4000H, CF = 1 (transport).*

2. Instructiunea de adunare cu bitul Carry (transport) de la operatia anterioara are forma:

ADC *operand1, operand2*

si efectul: *operand1 = operand1 + operand2 +* CF

Exemplu (initial *AX = 7FFFH, CX = 4000H* si *CF = 1*):

ADD AX, CX

Efect:

```
                0111 1111 1111 1111 B      = 7FFF H +
                0100 0000 0000 0000 B      =  4000 H +
                                   1 B     =      1 H
                ---------------------------------------------
(CF = 0)        1100 0000 0000 0000 B      = C000 H
```

Valori in urma executiei instructiunii: *AX = C000H, CF = 0.*

3. Instructiunea de scadere are forma:

SUB *operand1, operand2*

si efectul: *operand1 = operand1 - operand2*

Exemplu (initial *BX = 8000H* si *DX = 0C000H*):

SUB BX, DX

Efect:

```
                1000 0000 0000 0000 B      =  8000 H -
                1100 0000 0000 0000 B      = C000 H
                ---------------------------------------------
(CF = 1)        1100 0000 0000 0000 B      = C000 H
```

Valori in urma executiei instructiunii: *BX = C000H, CF = 1 (imprumut).*

4. Instructiunea de scadere cu bitul Carry (imprumut) de la operatia anterioara are forma:

SBB *operand1, operand2*

si efectul: *operand1 = operand1 - operand2* - CF

Exemplu (initial *AX = 7FFFH, CX = 4000H,* si *CF = 1*):

SBB AX, CX

```
Efect:          0111 1111 1111 1111 B      = 7FFF H -
                0100 0000 0000 0000 B      =  4000 H -
                                   1 B     =      1 H
                ---------------------------------------------
      (CF = 0)  0011 1111 1111 1110 B      = 3FFE H
```

Valori in urma executiei instructiunii: *AX = 3FFEH, CF = 0.*

5. Instructiunea de inmultire ***intre numere fara semn*** are forma:

MUL *operand*

si efectul:
- pentru operanzi de 8 biti: AX = AL * operand
- pentru operanzi de 16 biti: DX, AX = AX * operand

6. Instructiunea de inmultire ***intre numere cu semn*** are forma:

IMUL *operand*

si efectul:
- pentru operanzi de 8 biti: AX = AL * operand
- pentru operanzi de 16 biti: DX, AX = AX * operand

7. Instructiunea de impartire ***intre numere fara semn*** are forma:

DIV *operand*

si efectul:
- pentru operanzi de 8 biti: AL = AX / *operand* *(catul)*
  AH = AX MOD *operand* *(restul)*
- pentru operanzi de 16 biti: AX = DX, AX / *operand*
  DX = DX, AX MOD *operand*

8. Instructiunea de impartire ***intre numere cu semn*** are forma:

IDIV *operand*

si efectul: - pentru operanzi de 8 biti: AL = AX / *operand*
AH = AX MOD *operand*
- pentru operanzi de 16 biti: AX = DX, AX / *operand*
DX = DX, AX MOD *operand*

3.2. Instructiuni aritmetice care utilizeaza 1 operand

Instructiunile care utilizeaza 1 operand (sursa si destinatie) sunt: incrementare, decrementare, negare (aritmetica, in complement fata de 2).

1. Instructiunea de incrementare are forma:

INC *operand*

si efectul: *operand = operand* + 1

Exemplu (initial DI = 0FFFFH):

INC DI

Efect:

```
                1111 1111 1111 1111 B     =  FFFF H +
                0000 0000 0000 0001 B     =       1 H
                ---------------------------------------
        (CF = 1) 0000 0000 0000 0000 B    =   0000 H
```

Valori in urma executiei instructiunii: DI *= 0000H, CF = 1 (transport).*

2. Instructiunea de decrementare are forma:

DEC *operand*

si efectul: *operand = operand* - 1

Exemplu (initial SI = 0001H):

DEC SI

Efect:

```
                0000 0000 0000 0001 B     =  0001 H -
                0000 0000 0000 0001 B     =       1 H
                ---------------------------------------
        (CF = 0) 0000 0000 0000 0000 B    =  0000 H
```

Valori in urma executiei instructiunii: SI *= 0000H, CF = 0.*

3. Instructiunea de negare are forma:

NEG *operand*

si efectul: *operand* = (0) - *operand*

Exemplul 1 (initial SI = 0001H):

NEG SI

Efect:

```
                (0000 0000 0000 0000 B      =  0000 H)
              - 0000 0000 0000 0001 B      =  0001 H
                ---------------------------------------
(CF = 1)        1111 1111 1111 1111 B      =  FFFF H
```

Valori in urma executiei instructiunii: SI = *FFFFH, CF = 1 (transport).*

Exemplul 2 (initial DI = 0FFFFH):

NEG DI

Efect:

```
                (0000 0000 0000 0000 B      =  0000 H)
              - 1111 1111 1111 1111 B      =  FFFF H
                ---------------------------------------
(CF = 1)        0000 0000 0000 0001 B      =  0001 H
```

Valori in urma executiei instructiunii: DI = *0001H, CF = 1 (transport).*

Exemplul 3 (initial AX = 7FFFH):

NEG AX

Efect:

```
                (0000 0000 0000 0000 B      =  0000 H)
              - 0111 1111 1111 1111 B      =  7FFF H
                ---------------------------------------
(CF = 1)        1000 0000 0000 0001 B      =  8001 H
```

Valori in urma executiei instructiunii: *AX = 8001H, CF = 1 (transport).*

Exemplul 4 (initial BX = 8000H):

NEG BX

Efect:

```
                (0000 0000 0000 0000 B      =  0000 H)
              - 1000 0000 0000 0000 B      =  8000 H
                ---------------------------------------
(CF = 1)        1000 0000 0000 0000 B      =  8000 H
```

Valori in urma executiei instructiunii: *BX = 8000H, CF = 1 (transport).*

Exemplul 5 (initial CX = 4000H):

```
                NEG  CX
Efect:
              (0000 0000 0000 0000 B      =  0000 H)
             - 0100 0000 0000 0000 B      =  4000 H
              ---------------------------------------
      (CF = 1) 1100 0000 0000 0000 B      =  C000 H
```

Valori in urma executiei instructiunii: *CX = C000H, CF = 1 (transport).*

Exemplul 6 (initial DX = 0C000H):

```
                NEG  DX
Efect:
              (0000 0000 0000 0000 B      =  0000 H)
             - 1100 0000 0000 0000 B      =  C000 H
              ---------------------------------------
      (CF = 1) 0100 0000 0000 0000 B      =  4000 H
```

Valori in urma executiei instructiunii: *DX = 4000H, CF = 1 (transport).*

4. Exemple de programe

4.1. Calcule in dubla precizie

Rolul principal al instructiunilor de adunare cu transport si scadere cu imprumut este acela de a permite *efectuarea calculelor in dubla precizie (pe 32 biti).*

1. Adunarea a doua valori de 32 biti - perechile de registre (AX, BX), respectiv (CX, DX):

Initial: *AX = 7FFFH, BX = 8000H, CX = 4000H* si *DX = 0C000H.*

```
ADD  BX, DX;  adunarea LSW
ADC  AX, CX           ;  adunarea MSW
```

Efectul secventei este:

```
AX BX +
```

```
                    CX DX
                    ------------
                    AX BX

adica:
                    0111 1111 1111 1111        1000 0000 0000 0000 B     = 7FFF  8000 H +
                    0100 0000 0000 0000        1100 0000 0000 0000 B     =  4000  C000 H
                                    (CF = 1)
                    ------------------------------------------------------------------------------------
          (CF = 0)  1100 0000 0000 0000        0100 0000 0000 0000 B     =  C000 4000 H
```

In urma executiei secventei: *AX = C000H, BX = 4000H, CF = 0 (rezultat C000 4000 H).*

2. Scaderea a doua valori de 32 biti - perechile de registre (AX, BX), respectiv (CX, DX):

Initial: *AX = 7FFFH, BX = 8000H, CX = 4000H* si *DX = 0C000H.*

```
SUB  BX, DX ;  scaderea LSW
SBB  AX, CX ;  scaderea MSW
```

Efectul secventei este:

```
                    AX BX -
                    CX DX
                    ------------
                    AX BX

adica:              0111 1111 1111 1111        1000 0000 0000 0000 B     = 7FFF  8000 H -
                    0100 0000 0000 0000        1100 0000 0000 0000 B     =  4000 C000 H
                                    (CF = 1)
                    ------------------------------------------------------------------------------------
          (CF = 0)  0011 1111 1111 1110        1100 0000 0000 0000 B     =  3FFE C000 H
```

Valori in urma executiei instructiunii: *AX = 3FFEH, BX = C000H, CF = 0 (rezultat 3FFE C000 H).*

3. Calculul sumei in dubla precizie a variabilelor de tip cuvant a si b. Rezultatul este plasat in varibila c.

```
DAT   SEGMENT
      a dw 0a46fh
      b dw 0dc89h
      c dw ?, ?
```

```
DAT    ENDS

ASSUME CS: COD, DS: DAT
COD    SEGMENT

START:
        MOV  AX, DAT
        MOV  DS, AX

        mov  ax, a                      ; initializari
        mov  dx, 0

        add  ax, b                      ; suma in dubla precizie
        adc  dx, 0

        mov  c, ax                      ; memorare rezultat
        mov  [c+2], dx

        MOV  AH, 4CH
        INT  21H

 COD    ENDS
        END  START
```

4.2. Alte programe

1. Program de transformare a unui caracter litera mica citit de la tastatura in caracter litera mare afisat pe ecran.

```
data  segment
numeprog                db 25 dup(0ah),'Transformare caracter$'
citire            db 2 dup(0ah),0dh,' Introduceti litera mica $'
afisare           db 2 dup(0ah),0dh,' Litera mare este: $'
data  ends

assume cs:cod, ds:data
cod    segment
start:
        mov     ax, data
        mov     ds, ax
        mov     dx, offset numeprog   ; afisare sir caractere
        mov     ah, 9                         ; (nume program)
        int     21h

        mov     dx, offset citire     ; afisare sir caractere
        mov     ah, 9                         ; (mesaj citire)
```

```
        int     21h

        mov     ah, 1                           ; citire caracter cu ecou pe
        int     21h                             ; ecran (litera mica)

        sub     al, 20h         ; conversie litera mica -> litera mare
        mov     bl, al

        mov     dx, offset afisare      ; afisare sir caractere
        mov     ah, 9                           ; (mesaj afisare)
        int     21h

        mov     dl, bl
        mov     ah, 2                           ; afisare caracter
        int     21h                             ; (litera mare)

        mov     ah, 8                           ; citire caracter fara ecou pe
        int     21h                             ; ecran (Enter)
        mov     ah, 4ch                 ; exit
        int     21h
        cod     ends
        end     start
```

2. Program de transformare a unui caracter litera mare citit de la tastatura in caracter litera mica afisat pe ecran.

```
data  segment
numeprog                db 25 dup(0ah),'Transformare caracter$'
citire          db 2 dup(0ah),0dh,' Introduceti litera mare $'
afisare         db 2 dup(0ah),0dh,' Litera mica este: $'
data  ends

assume cs:cod, ds:data
cod    segment
start:
        mov     ax, data
        mov     ds, ax
        mov     dx, offset numeprog     ; afisare sir caractere
        mov     ah, 9                           ; (nume program)
        int     21h

        mov     dx, offset citire       ; afisare sir caractere
        mov     ah, 9                           ; (mesaj citire)
        int     21h

        mov     ah, 1                           ; citire caracter cu ecou pe
```

```
int     21h                             ; ecran (litera mare)

add     al, 20h         ; conversie litera mare -> litera mica
mov     bl, al
mov     dx, offset afisare      ; afisare sir caractere
mov     ah, 9                           ; (mesaj afisare)
int     21h

mov     dl, bl
mov     ah, 2                           ; afisare caracter
int     21h                             ; (litera mica)

mov     ah, 8                           ; citire caracter fara ecou pe
int     21h                             ; ecran (Enter)

mov     ah, 4ch                 ; exit
int     21h
cod     ends
end     start
```

5. Desfasurarea lucrarii

1. *Se deschide un editor de texte* (Borland C cu comanda *BC*, Turbo Pascal cu comanda *TURBO*, etc.).

a) *Se editeaza urmatorul textul urmatorului program (exceptand comentariile):*

```
DATA SEGMENT                            ; nu sunt date de declarat

DATA ENDS

ASSUME CS: COD, DS: DATA
COD   SEGMENT

START:
        MOV  AX, DATA                   ; initializare DS
        MOV  DS, AX

        MOV  AX, 7FFFH                  ; initializari registre de date
        MOV  BX, 8000H
        MOV  CX, 4000H
        MOV  DX, C000H

        ADD  BX, DX                     ; adunarea LSW
```

```
        ADC  AX, CX                     ; adunarea MSW

        MOV  AH, 4CH                    ; exit
        INT  21H

COD     ENDS
        END  START
```

b) *Se salveaza fisierul editat* cu numele *AP21.ASM*. Se iese din editor (cu comanda *[Alt] X*) si se da comanda:

*DIR AP21.**

urmarindu-se efectul.

c) *Se realizeaza asamblarea* fisierului editat cu comanda:

TASM AP21

Se urmaresc mesajele de pe ecran si se corecteaza eventualele erori (reintrand in editor).
Se da comanda:

*DIR AP21.**

si se urmareste efectul.

d) *Se realizeaza editarea de legaturi* cu comanda:

TLINK AP21

Se urmaresc mesajele de pe ecran si se corecteaza eventualele erori.
Se da comanda:

*DIR AP21.**

si se urmareste efectul.

e) *Se dau succesiv comenzile:*

TYPE AP21.ASM
TYPE AP21.MAP
TYPE AP21.OBJ
TYPE AP21.EXE

Se analizeaza efectele.

f) *Se lanseaza in executie depanatorul simbolic* cu comanda:

TD AP21

Se executa pas cu pas programul (apasand tasta *F7*) urmarindu-se in special continuturile registrelor AX, BX, CX, DX si a flagului Carry (CF).

2. Se editeaza *programul de transformare a unui caracter litera mica citit de la tastatura in caracter litera mare afisat pe ecran* (vezi $4.2.) intr-un fisier cu numele *AP22.ASM.*

Se parcurg etapele 1.a) ... 1.f) pentru acest program si se lanseaza in executie programul *AP22.EXE.*

3. Se editeaza *programul de transformare a unui caracter litera mare citit de la tastatura in caracter litera mica afisat pe ecran* (vezi $4.2.) intr-un fisier cu numele *AP23.ASM.*

Se parcurg etapele 1.a) ... 1.f) pentru acest program si se lanseaza in executie programul *AP23.EXE.*

6. Teme si exercitii

1. Sa se scrie un *program care sa utilizeze calculul in dubla precizie pentru a calcula suma a doua cuvinte, aflate in registrele AX si DX, folosind numai registrele de tip octet (AH, AL, BH, BL, CH, CL, DH, DL).*

2. Sa se scrie un *program care sa transforme caracterele 'a', ..., 'f' citite de la tastatura in valorile 10, ..., 15.*

3. Sa se scrie un *program care sa transforme caracterele 'A', ..., 'F' citite de la tastatura in valorile 10, ..., 15.*

4. Sa se scrie un *program care sa citeasca de la tastatura valorile 10, ..., 15 (doua cifre succesive), sa le transforme in caracterele 'A', ..., 'F' si sa le afiseze pe ecran.*

7. Intrebari

1. Care sunt tipurile operanzilor instructiunilor microprocesorului Intel 8086 ?

2. Care sunt modurile de adresare a memoriei ?
3. Care sunt modurile de adresare a porturilor ?

4. Care este diferenta intre negarea aritmetica (NEG) si negarea logica (NOT) ?
5. Care este diferenta intre deplasarea aritmetica la dreapta (SAR) si deplasarea logica la dreapta (SHR) ?
6. Care este diferenta intre deplasarea logica si rotatia fara CF ?

5. Ce sunt directivele asamblorului TASM ?
6. Care sunt directivele pentru specificarea segmentelor ?

7. Care sunt directivele pentru declararea constantelor si variabilelor?
8. Care sunt diferentele intre constante si variabile ?
9. Cum se definesc inregistrarile de date ?
10. Cum se definesc structurile ?