LAB 4 - Instrucțiunile procesorului 8086 (partea întâi)

1. Lucrul cu datele definite în memorie: Directivele de fortare a tipului: byte ptr si word ptr

Problema 1-1

a) Realizaţi un desen care să ilustreze conţinutul memoriei dacă se vor depune în memorie (după convenţia *Little Endian*) următoarele date:

```
var     db      14h, 8, 33, 10101111b   ; p1_01_a
v2      dw      1234h, 0ff00h   ; multi-octet-> Conventia Little END-ian
lit     db      'E'             ; caracter 'E' = 45h (Cod Ascii) , 'A'=41h, 'B'=42h, …  'a'=61h, 'b'=62h, …
msg     db      "ABCD"          ; similar cu: 'A', 'B', 'C', 'D'
v3      dd      12345678h       ; multi-octet-> Conventia Little END-ian
```

Rezolvare: Corespunzător directivelor, în memorie se vor regăsi următoarele date:

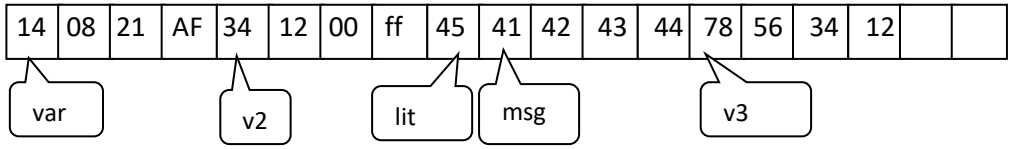

Figura 1. Reprezentarea datelor (în hexazecimal) în memorie pentru Problema ***1***

b) Folosind datele definite anterior, specificaţi conţinutul regiştrilor după execuţia instrucţiunilor: b1) mov AL, var[2]; b2) mov AX, var[3]; b3) mov BL, v2[1];

Rezolvare: *; p1_1_b2*

```
b1) mov AL, var[2]      ; AL va conţine octetul din memorie de la adresa dată de offsetul lui var +2; astfel, AL=33=21h
b2) mov AX, var[3]      ; această instrucţiune este ilegală, întrucât AX are dimensiunea de 16 biţi, iar variabila var are tipul "byte"
                        ; instrucţiunea poate fi folosită: mov AX, word ptr var[3], caz în care AX = 34AFh (aplicarea conv. Little Endian)
b3) mov BL, v2[1]       ; această instrucţiune este ilegală, întrucât BL are dimensiunea de 8 biţi, iar variabila v2 are tipul "word"
                        ; instrucţiunea poate fi folosită: mov BL, byte ptr v2[1], caz în care BL = 12h
```

Problema 1-2

Analizaţi următoarele date definite în memorie în segmentul de date:

```
org 100h  ; p1_02_a
.data
a db 12h
b db 34h
c db 56h
d db 78h
e db 12h,34h,56h,78h
f dw 1234h,5678h
g dd 12345678h
```

```
a db 12h                a     07102: 12
b db 34h                b     07103: 34
c db 56h                c     07104: 56
d db 78h                d     07105: 78
e db 12h,34h,56h,78h    e +0  07106: 12
                            +1  07107: 34
                            +2  07108: 56
                            +3  07109: 78
f dw 1234h,5678h        f +0  0710A: 34
                            +1  0710B: 12
                            +2  0710C: 78
                            +3  0710D: 56
g dd 12345678h          g +0  0710E: 78
                            +1  0710F: 56
                            +2  07110: 34
                            +3  07111: 12
```

Figura 2. Reprezentarea variabilelor în memorie

Rezolvare:

Despre **a**, **b**, **c**, **d** spunem că sunt variabile de tip octet;

despre **e** spunem că este vector sau şir de octeţi cu 4 elemente;

variabila **f** este vector sau şir de cuvinte, cu 2 elemente,

iar **g** este variabilă de tip dublucuvânt, cu 1 element.

b) Considerând datele definite anterior, şi folosind ***o singură instrucţiune pt fiecare cerinţă***, realizaţi următoarele:

a) încărcaţi în AL pe a, în AH pe b
b1) în BL pe c şi în BH pe d
b2) în BX pe e+0 şi e+1
b3) în CX pe f+2 si f+3
c1) în DL pe f+1
c2) în DH pe g+0
d1) în DX partea superioară a variabilei g din memorie; prin "parte superioară" ne referim la partea c.m.s. a valorii, deci la cuvântul 1234h;
d2) iar în AX partea inferioară a variabilei g din memorie

Rezolvare:

a) mov AX, word ptr a *; p1_02_b;* *AX = | b | a | = | 34 | 12 | h, deci AX=3412h*
b1) mov BX, word ptr c *; sau* *mov BX, word ptr a+2*
b2) mov BX, word ptr e *; BX=3412h*
b3) mov CX, f+2 *; CX=5678h*
c1) mov DL, byte ptr f+1 *; DL=12h*
c2) mov DH, byte ptr g+0 *; DH=78h*
d1) mov DX, word ptr g+2 *; DX=1234h – partea superioară a lui g*
d2) mov AX, word ptr g *; AX=5678h – partea inferioară a lui g*

(se subliniază că g e de tip doubleword şi se încarcă într-o pereche de regiştri, De exemplu, DX:AX – dacă ne pregătim să o împărţim la o valoare de tip word)

Observație: În cadrul simulatorului, la folosirea unui *șablon de program de tip* ***.com*** (unde codul este compact, toate valorile – reprezentând datele, codul, stiva sunt "înghesuite" în cadrul unui singur segment – se observă aceasta simplu, pentru că DS=SS=CS=ES=0700h), apare specificată ***directiva org 100h***. Această directivă va determina așezarea programului nostru în memorie începând de la adresa care are offsetul 100h. Primele două locații de memorie sunt ocupate de simulator pentru a depune codificarea unei instrucțiuni de salt (jmp adresă) la prima instrucțiune a programului scris de noi. Astfel, în simulator se sare peste zona în care sunt depuse datele programului nostru (cele definite cu directivele *db, dw, dd*). Prima locație din această zonă are offsetul 102h; spunem că ***prima variabilă din memorie începe întotdeauna de la adresa cu offsetul 102h*** (așa cum se poate observa în Figura 3).

Problema 1-3:

a) Folosiţi directivele potrivite astfel încât să definiţi în memorie un șir de octeţi (în hexazecimal) care să conţină numerele pare de la 0h la 10h, iar apoi un șir de 5 cuvinte care în convenţia cu semn să denote atât numere pozitive cât și numere negative (alternativ).

Rezolvare:

```
A db 0, 2, 4, 6, 8, 0Ah, 0Ch, 0Eh        ; p1_03_a
B dw 210h, 8765h, 7654h, 9ABCh, 9ABh      ; cifrele hexazecimale dacă lipsesc, se consideră zerouri.
```

Dacă nu s-ar fi impus ca definirea elementelor şirului să se realizeze în hexazecimal, am fi putut folosi şi definirea elementelor şirului direct cu semn, sub forma: B dw 528, -30875, 30292, -25924, 2475;
ar fi fost mult mai uşor la definire, iar în memorie s-ar fi obţinut aceleaşi valori: 210h=528, 8765h= -30875, etc.

b) Fie următoarea secvenţă scrisă în simulator:

```
org 100h                ; p1_03_b
.data
a db 7, 6, 5, 4, 3, 2, 1, 0
.code
mov AL, a+x             ; AL = _____
mov BL, [a+x]           ; BL = _____
mov CL, a[x]            ; CL = _____
 ret
```

unde x e un număr astfel încât să fie accesat al treilea element al lui a.
a) Înlocuiţi pe x cu valoarea potrivită şi apoi comentaţi fiecare instrucţiune, scriind rezultatul obţinut în registrul corespunzător.
b) Adăugaţi o singură linie de cod astfel încât programul să funcţioneze fără a înlocui în mod explicit pe x cu 2 în fiecare din cele 3 instrucţiuni de la pct a).

Rezolvare:

a) Pentru accesarea celui de-al III-lea element de tip octet (byte) al şirului, trebuie ca x=2:

```
mov AL,a+2          ; AL=5
mov BL,[a+2]        ; BL=5
mov CL,a[2]         ; CL=5
```

adresarea se realizează asupra aceluiaşi element din şir, al treilea, adică 5.

b) se poate adăuga directiva de definire a constantei x cu EQU, chiar înainte de directiva *.code*:

~~x db 2 ; nu putem aduna 2 variabile din memorie asa, ca a+x, eventual cu instructiunea add (dar nici aceasta nu permite 2 operanzi din memorie)~~

```
x EQU 2     ; constanta numită x de valoare 2
            ; se va obţine acelaşi efect ca la punctul a).
```

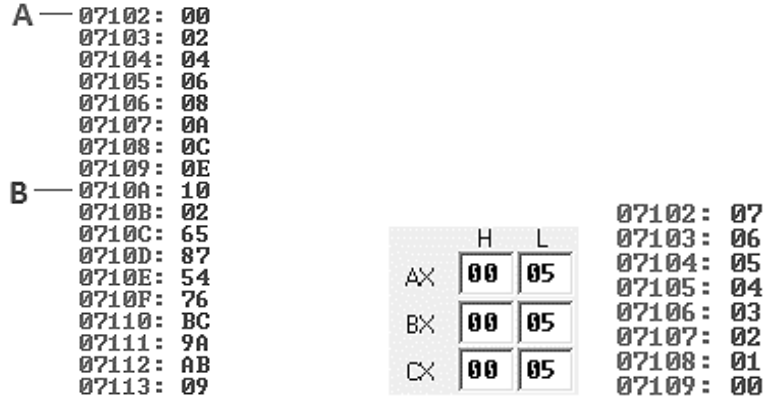

Figura 3. a) Reprezentarea variabilelor în memorie pentru problema ***3 a)*** (stânga); b) Reprezentarea variabilelor în regiştri şi în memorie pentru problema ***3 b)*** (dreapta);

Problema 1-4

Introduceţi în simulator următoarea secvenţă de program:

```
org 100h        ; p1_04
.data
  a db 1, 2, 3, '1'
  b dw 1, 2, 3, '1'
sirAscii db 'ABC 123'
sirAscii2 db 'A', 'B', 'C', '1', '2', '3'
   ; codurile Ascii: '0' =30h, '1'=31h, …, '9'=39h
.code
  mov AL, [a+2]
  mov AX, [b+4]
ret
```

Observaţi modificarea valorilor din regiştri odată cu fiecare comandă Single Step efectuată şi apoi:

a) Vizualizaţi zona din memorie a datelor;
- specificaţi adresa logică şi fizică a variabilelor;
- specificaţi cum "se execută" directivele de definire a datelor;

b) Adăugaţi şi instrucţiunile de mai jos, la sfârşit, dar înainte de *ret*:

```
mov BL, [b+4]
mov BX, [a+10]
```

c) În caz că nu funcţionează instrucţiunile de la punctul b), modificaţi-le a.î. să fie "legale".

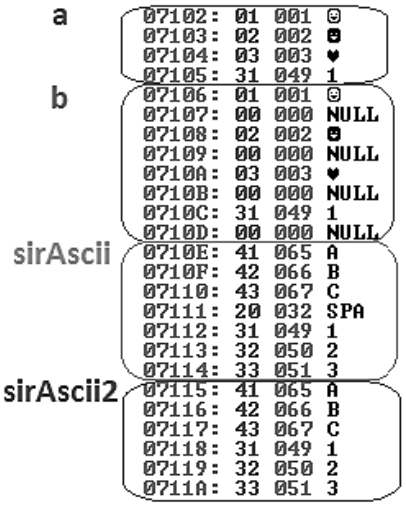

Figura 4. Reprezentarea în simulator a datelor din memorie pentru Probl. **PR4**

Rezolvare:

a) Datele definite în memorie au fost depuse aşa cum sugerează Figura 4, unde dinspre stânga spre dreapta, se observă, înainte de simbolul "**:**" locaţia sau *adresa fizică (pe cei 20 biţi)*, iar după "**:**" este *conţinutul exprimat ca valoare în hexazecimal*, apoi *în zecimal* şi respectiv ca şi *cod Ascii*. Adresa fizică a variabilei *a* se observă că este 07102h, care în formă logică se scrie 0700h:0102h; adresa lui *b* este 07106h sau în formă logică 0700h:0106h; adresa unde începe *sirAscii* este 0710Eh sau 0700h:010Eh; iar adresa unde începe *sirAscii2* este 07115h sau 0700h:0115h.
Directivele de definire a datelor nu "se execută", se poate observa la execuţie că CPU sare peste acea "zonă a datelor", ajungând direct la prima instrucţiune din „zona de cod";

b) se adaugă cele 2 instrucţiuni înainte de ret, dar acestea nu vor funcţiona în simulator; apare eroarea *"wrong parameters: operands do not match"* care sugerează faptul că e posibil ca un operand să fie văzut de CPU ca fiind pe 8 biţi şi un altul ca fiind pe 16 biţi. Rezolvarea constă în forţarea tipului unui operand a.î. să corespundă cu tipul celuilalt operand; în acest caz se forţează tipul operandului preluat din memorie, întrucât registrul are o dimensiune fixă;
c) Modificarea constă în potrivirea dimensiunii operanzilor:

```
mov BL, byte ptr [b+4]      ; variabila b a fost definită în memorie ca word, dar ne vom referi la ea ca byte
mov BX, word ptr [a+10]     ; variabila a a fost definită în memorie ca byte, dar ne vom referi la ea ca word.
```

Problema 1-5 a) Definiţi un vector "x" cu elemente pe cuvânt, iniţializat cu valorile impare cuprinse între 0 şi 10h;
b) Vizualizaţi acest şir şi explicaţi cât este: lungimea, tipul şi dimensiunea lui;
c) Scrieţi o secvenţă de program astfel încât primul, al treilea şi ultimul element al vectorului să fie transferate în regiştrii AX, BX, CX şi apoi depuşi pe stivă, în această ordine;
d) Scrieţi instrucţiunile potrivite pt a obţine şi în memorie, de la adresa cu offsetul 130h, folosind instrucţiuni MOV, aceeaşi structură ca şi cea obţinută la pct c) în stivă.

Rezolvare:

a) x dw 01h, 03h, 05h, 07h, 09h, 0Bh, 0Dh, 0Fh; *; p1_05*
b) lungime = 8 – vectorul are 8 elemente;
tip = 2 deoarece vectorul are elemente de tip cuvânt (adică 2 octeţi);
dimensiune = 16 (adică numărul de octeţi ocupaţi în memorie de vector).

c) pentru accesarea celui de-a treilea şi respectiv ultimului element, se va folosi indexul 4 şi resp. 14, indexul fiind calculat ca 2*(ordinElement-1), unde ordinElement = 1,2,3,4,...,8 (primul, al doilea, al treilea,...).

```
mov AX, x          ; AX=0001h
mov BX, x+4        ; BX=0005h
mov CX, x+14       ; CX=000Fh
push AX            ; se depune în stivă 0001h
push BX            ; se depune în stivă 0005h
push CX            ; se depune în stivă 000Fh
```

```
16FF7:
16FF8: 0F 015 ☼
16FF9: 00 000 NULL
16FFA: 05 005 ♣
16FFB: 00 000 NULL
16FFC: 01 001 ☺
16FFD: 00 000 NULL
16FFE:
```

Figura 5. Conţinutul stivei

d) pentru a obţine aceeaşi structură ca şi cea obţinută la pct c) în stivă şi în memorie, începând de la adresa 130h, vom scrie următoarea secvenţă:

```
mov AX, x          ; AX=0001h
mov BX, x+4        ; BX=0005h
mov CX, x+14       ; CX=000Fh
mov [130h+4], AX   ; se depune 0001h
mov [130h+2], BX   ; se depune 0005h
mov [130h+0], CX   ; se depune 000Fh
```

```
07130: 0F 015 ☼
07131: 00 000 NULL
07132: 05 005 ♣
07133: 00 000 NULL
07134: 01 001 ☺
07135: 00 000 NULL
```

Figura 6. Conţinutul memoriei

Problema 1-6

Fie următoarea secvenţă de instrucţiuni:

```
org 100h        ; p1_06
.data
a db 5,4,3,2,1,0
.code
lea BX, a              ; BX = _____
mov SI, _____          ; SI = _____
_____                  ; AL = _____
ret
```

a1) Comentaţi prima instrucţiune (şi specificaţi o instrucţ. echivalentă), scriind rezultatul obţinut în registrul corespunzător; a2) Completaţi cu o valoare ce trebuie scrisă în reg. SI a.î. în AL să se încarce elementul de pe poziţia a III-a din sir;
b) Înlocuiţi prima instrucţiune cu *mov bx,0* și ultima instrucţ. cu *mov AL,a[BX][SI]*. Comentaţi şi apoi executaţi noua secvenţă de program obţinută.
* elementele șirului sunt indexate de la poziţia 0 (prima din șir)

Rezolvare: Se vor analiza prin comparaţie cele 2 secvenţe de mai jos:

```
org 100h      ; p1_06_a
.data
a db 7,6,5,4,3,2,1,0
.code
lea BX, a         ; BX = 0102h,          ; instrucţiune echivalentă:  mov BX, offset a
mov SI, 2         ; SI = 2
mov AL, [BX][SI]
; AL = [0102h+2]= [0104h]= 05h,
; adică în AL se va încărca
; al III-lea element al vectorului
```

```
org 100h     ; p1_06_b
.data
a db 7,6,5,4,3,2,1,0
.code
mov BX, 0             ; BX = 0
mov SI, 2             ; SI = 2
mov AL, a [BX][SI]
; AL = [0102h+0+2]=[0104h]=05h,
; adică în AL se va încărca
; al III-lea element al vectorului
```

2. Instrucțiunile XCHG și XLAT

Instrucţiunea xchg interschimbă conţinutul a doi operanzi; cei doi operanzi pot fi ambii regiștri, sau un registru și o variabilă din memorie, dar nu pot fi ambii operanzi din memorie.
De exemplu, dacă dorim să interschimbăm conţinutul regiştrilor AX şi BX, avem două posibilităţi:

Problema 2-1

a) Scrieţi o secvenţă de instrucţiuni prin care să interschimbaţi conţinutul a 2 regiştri folosind a1) doar instrucţiuni mov; a2) o instrucţiune specifică;
b) Scrieţi o secvenţă de instrucţiuni prin care să interschimbaţi conţinutul a 2 variabile de tip cuvânt din memorie.

Răspuns:

a1) fără instrucţiunea xchg:
-folosim o variabilă temporară:

```
.data
temp dw 0
.code
mov temp, AX
mov AX, BX
mov BX, temp
```

a2) cu instrucţiunea xchg:

```
xchg AX, BX
```

b) Dacă dorim să interschimbăm conţinutul a două locaţii de memorie, de exemplu *mem1* şi *mem2*, putem scrie următoarea secvenţă:

```
; considerăm initial [mem1]=00h, [mem2]=FFh
; nu putem scrie xchg mem1, mem2;

mov AL, mem1 ; AL=00h
xchg AL, mem2 ; AL=FFh, [mem2]=00h
mov mem1, AL  ; [mem1]=FFh
```

Instrucţiunea xlat încarcă în registrul AL conţinutul locaţiei din segmentul de date de la offsetul indicat de BX, la care se adaugă deplasamentul dat de AL: **AL** = [**DS:** (**BX+AL**)] . Instrucţiunea xlat se foloseşte pentru conversia/translatarea sau codificarea datelor, utilizând tabele de conversie.

Problema 2-2 Explicaţi efectul instrucţiunii xlat din următoarea secvenţă:

```
org 100h
.data
tabela db 0,5,1,7,3,4,2,6,11,8,9,10,12,14,15   ; variabila tabela
.code
lea BX, tabela      ; se încarcă în BX adresa de offset a tabelei ; instrucțiunea e echivalentă cu: mov BX, offset tabela
mov AL, 5           ; se vrea valoarea de pe poziţia dată de indexul 5 din tabela
xlat                ; rezultă AL = 4, valoarea returnată de xlat
```

Răspuns:

3. Instrucțiunile CBW și CWD

Extensii de valori pentru numere cu semn:

Instructiunea CBW (*convert byte to word*) converteşte o valoare pe octet (din registrul AL) într-o valoare pe cuvânt (în registrul AX), păstrând semnul valorii prin extensia bitului de semn.
Instructiunea CWD (*convert word to doubleword*) converteşte o valoare pe cuvânt (din registrul AX) într-o valoare pe dublucuvânt (în perechea de regiştri DX:AX), păstrând semnul valorii prin extensia bitului de semn.

Problema 3-1
Analizaţi instrucţiunea *cbw* şi explicaţi efectul ei în următoarea secvenţă de program:

```
.data                  Răspuns:
val  db -100
.code
mov al, val     ; AL = 9Ch
cbw             ; AX = FF9Ch
```

în exemplul de mai sus, atât AL, cât şi AX, reprezintă aceeaşi valoare (-100d), în convenţia Complement faţă de 2.

Problema 3-2
Analizaţi efectul secvenţei de instrucţiuni de mai jos şi explicaţi:

```
org 100h                Răspuns:
.data
val1 db 100             ; val1=100=64h
val2 db 125             ; val2=125=7Dh
val3 db 50              ; val3=50=32h
.code
mov BX, 0               ; iniţializare sumă cu 0
mov AL, val1            ; AL=64h=100
cbw                     ; AX=val1 =0064h=100
mov BX, AX              ; BX=100=64h
mov AL, val2            ; AL=7Dh=125
cbw                     ; AX=125=007Dh
add BX, AX              ; BX=225=00E1h
mov AL, val3            ; AL=32h=50
cbw                     ; AX=50=0032h
add BX, AX              ; BX=275=0113h
```

Secvenţa de instrucţiuni de mai sus realizează suma a 3 valori reprezentate pe octet, fără a testa o posibilă depăşire de capacitate a valorii rezultate. Operaţia de adunare se va realiza pe cuvânt (în registrul BX).

Problema 3-3 Analizaţi efectul secvenţei de instrucţiuni de mai jos şi explicaţi:

```
org 100h            Răspuns:
.data
val  dw -100        ; -100 = FF9Ch
.code
mov ax, val         ; AX = FF9Ch = -100
cwd                 ; DX:AX = FFFFh:FF9Ch = -100
neg AX              ; AX = 0064h = +100
cwd                 ; DX:AX = 0000h: 0064h = +100
```

Problema 3-4 Analizaţi efectul secvenţei de instrucţiuni de mai jos şi explicaţi:

```
org 100h            Răspuns:
.data
val  db -100        ; -100=9Ch
.code
mov AL, val         ; AL = 9Ch=-100
cbw                 ; AX =FF9Ch= -100
cwd                 ; DX:AX = FFFFh:FF9Ch= -100
```

Extensii de valori pentru numere fără semn:

Nu există instrucțiuni speciale pentru extensia numerelor fără semn. Soluția este să înscriem noi valoarea 0 în partea superioară a registrului în care se dorește obținerea valorii extinse.

1) Fie nr **fără semn** scris pe octet în registrul AL. Extindeți acest nr la registrul AX.
mov AH, 0 ; AX = nr cerut, extins pe 16 biți
2) Fie nr **fără semn** scris pe cuvânt în registrul AX. Extindeți acest nr la 32 biti.
mov DX, 0 ; DX:AX = nr cerut, extins pe 32 biți.

Aici trebuie să avem grijă pt că nu se permite accesarea regiștrilor în perechi, de exemplu *mov DX:AX, 2* *este ilegală*.

4. Instrucțiunile LEA, LDS și LES

Instrucţiunea ***LEA*** încarcă adresa de offset a variabilei specificate ca operand sursă în registrul destinaţie. Mai precis, instrucţiunea *lea BX, a* va încărca adresa de offset a variabilei *a* din segmentul dat de valoarea lui DS, în BX. Echivalentă cu: *mov BX, offset a*

Problema 4-1

Explicaţi efectul următoarei secvenţe de instrucţiuni. Ce valori vor conţine regiştrii BX, SI, DI, DS, ES ?

Răspuns:

```
org 100h
.data
a dw 1234h      ; în EMU, a începe la adresa cu offsetul 102h
b dw 5678h      ; în EMU, b începe la adresa cu offsetul 104h,        DS : [104h]
c dw 9ABCh      ; în EMU, c începe la adresa cu offsetul 106h
.code
lea BX, a                   ; BX = 0102h = adresa efectivă (offset) a variabilei a
lds SI, a                   ; SI = 1234h, DS = 5678h
sau daca am fi folosit:
les DI, b                   ; DI = 5678h, ES = 9ABCh
ret
```

Instrucţiunea ***LDS*** (sau ***LES***) încarcă o pereche de regiştri: *DS* (respectiv *ES*) şi *registrul specificat ca destinaţie* cu perechea de cuvinte preluate din memorie de la adresa cu offsetul dat de operandul sursă din segmentul dat de valoarea din DS. În secvenţa de mai sus, a doua instrucţiune (*lds SI, a*) a modificat valoarea lui DS şi din această cauză, dacă imediat apoi am executa instrucţiunea *les DI,b* se va căuta conţinutul din memorie de la offsetul dat de b, dar în segmentul cu adresa de început 5678h (iar noi nu ştim ce se află acolo).

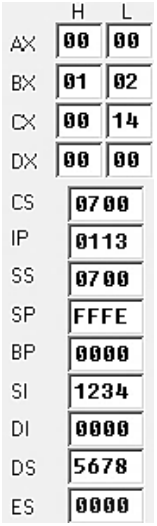

Figura 7. Valorile finale ale regiştrilor pentru **Pr4-1**

5. Instrucțiuni pentru adunare și scădere: ADD, SUB, ADC, SBB

Instrucţiunea ADD adună doi operanzi (), rezultatul fiind apoi plasat în primul operand.
Instrucţiunea ADC adună doi operanzi, împreună cu indicatorul de transport CF (*Carry flag*) rezultat anterior.
Instrucţiunea SUB scade doi operanzi, rezultatul fiind apoi plasat în primul operand.
Instrucţiunea SBB scade doi operanzi, împreună cu indicatorul de transport CF (*Carry flag*) rezultat anterior.
Operanzii pot fi: 1) doi regiștri, 2) un registru și o variabilă din memorie, 3) un registru sau o variabilă din memorie și o valoare imediată.
În aceste cazuri, va trebui să fim atenţi la dimensiunea operanzilor şi la valorile acestora, pentru a nu rezulta depăşiri de capacitate.

Concret, la realizarea operaţiilor aritmetice de adunare şi scădere, vom putea lucra astfel:

- ***se pot aduna 2 numere*** **scrise pe n biţi**, rezultatul obţinut va fi tot pe n biţi, însă trebuie interpretate flagurile întrucât ar putea apărea depășire de capacitate (practic *unele rezultate ar putea avea nevoie de scrierea pe n+1 biți*);

- ***se pot scădea 2 numere*** **scrise pe n biţi**, rezultatul obţinut va fi tot pe n biţi, însă trebuie interpretate flagurile întrucât la scădere ar putea apărea depășire de capacitate în sensul nevoii de împrumut - borrow (practic *unele rezultate ar putea avea nevoie de scrierea pe n+1 biți* pentru a marca faptul că e nevoie de un împrumut pentru a putea realiza scăderea);

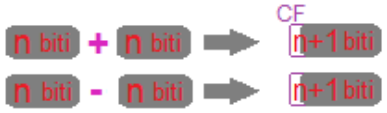

Figura 8. Reprezentarea dimensiunii rezultatului obţinut la adunarea şi scăderea a 2 numere pe n biţi

FARA semn: CF=1 !!! => rezultatul nu e bun stocat doar pe n biți => să citim / interpretăm rezultatul pe n+1 biți : CF|rez pe n biti
CU semn: OF=1 !!! => rezultatul nu e bun stocat doar pe n biți => să citim / interpretăm rezultatul pe n+1 biți : CF|rez pe n biti

Analizați modul de lucru cu instrucțiunile ADC și SBB în cazul numerelor fără semn, respectiv cu semn

Problema 5-1 Transformaţi secvenţa de cod astfel încât variabilele a, b şi c (ca nr fără semn) să fie definite ca o singură variabilă de tip octet denumită "sir". Cum se poate adresa acum *variabila a* din memorie? Dar *variabilele b, c* ? Realizaţi suma lor. Observați modul de adresare.

Rezolvare:

Se dă:

```
org 100h  ; p5_01_a
a db 82h ; 82h=130
b db 9Bh ; 9Bh=155
c db 3Ah ; 3Ah=58
suma dw ?
.code
clc                ; (Clear CF) CF=0
mov AX, 0          ; AX=AH  AL = 00 00h
mov AL, a          ; AL= 82h
add AL, b          ; AL= 82h+9Bh= 1Dh, CF=1
adc AH, 0          ; AH = 01h
add AL, c          ; AL = 1Dh+3Ah= 57h, CF=0
adc AH, 0          ; AH = 01h+0= 01h
mov suma, AX       ; suma=0157h=343
```

Adresare directă

```
org 100h   ; p5_01_b
sir db 82h,9Bh,3Ah

suma dw ?
.code
clc                ; (Clear CF) CF=0
mov AX, 0          ; AX=AH  AL = 00 00h
mov AL, sir        ; AL= 82h
add AL, sir+1      ; AL= 82h+9Bh= 1Dh, CF=1
adc AH, 0          ; AH = 01h
add AL, sir+2      ; AL = 1Dh+3Ah= 57h, CF=0
adc AH, 0          ; AH = 01h+0= 01h
mov suma, AX       ; suma=0157h=343
```

Adresare indirectă

```
org 100h   ; p5_01_c
sir db 82h,9Bh,3Ah

suma dw ?
.code
clc                ; (Clear CF) CF=0
mov AX, 0          ; AX=AH  AL = 00 00h
mov AL, sir[0]     ; AL= 82h
add AL, sir[1]     ; AL= 82h+9Bh= 1Dh, CF=1
adc AH,0           ; AH = 01h
add AL, sir[2]     ; AL = 1Dh+3Ah= 57h, CF=0
adc AH, 0          ; AH = 01h+0= 01h
mov suma, AX       ; suma=0157h=343
```

Problema 5-2. Să se scrie un program care să adune două numere; cele 2 valori *a* şi *b* vor fi definite în memorie ca octeţi (numerele vor fi considerate în convenţia fără semn). Suma acestora ar putea fi mai mare decât valoarea 0FFh şi astfel, se propune stocarea ei în cadrul unei variabile de tip cuvânt.

Rezolvare:

Varianta A) Folosind ***adunare de regiştri pe 16 biţi*** (cu ***ADD***)

```
org 100h   ; p5_02_a
.data
a db 130    ; se defineşte variabila a =82h
b db 150    ; se defineşte variabila b =96h
suma dw ?  ; se defineşte var. suma
            ; neiniţializată, pe cuvânt
.code
mov ah,0          ; AH=0
mov bh,0          ; BH=0
mov AL,a          ; AL=a=82h => AX=00 82h
mov BL,b          ; BL=b=96h => BX=00 96h
add AX,BX         ; AX=a+b=0118h
mov suma, AX   ; suma=AX=0118h=280
ret
```

Varianta B) Folosind ***adunare cu carry de regiştri pe 8 biţi*** - instrucţiunea ***ADC*** s-a aplicat asupra extensiei lui AL (în care s-a realiz adunarea) adică în AH.

```
org 100h    ; p5_02_b
.data
a db 130    ; se defineşte variabila a
b db 150    ; se defineşte variabila b
suma dw ?  ; se defineşte var. suma
            ; neiniţializată, pe cuvânt
.code
clc ;CF=0 (să nu rămână setat dinainte)
mov AX,0        ; AX=0
mov AL,a        ; AL=a=82h
add AL,b        ; AL=a+b=18h, CF=1
adc AH,0        ; AH=CF=01h
mov suma, AX  ; suma=AX=0118h=280
ret
```

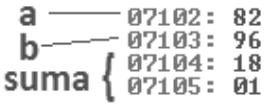

Figura 9. Ilustrarea variabilelor în memorie pentru problema ***PR5-2***

Observaţii: Valorile propuse în problemă au fost numerele *fără semn* 130 şi 150; acestea, reprezentate în hexazecimal sunt valorile 82h şi 96h, iar în memorie se vor găsi la adresele 07102h şi respectiv 07103h. Suma celor două numere este 0118h care în zecimal fără semn reprezintă numărul 280.
Observaţi formula prin care puteţi interpreta valoarea concatenată (fără a folosi regulile tradiţionale de conversie din hexa în zecimal şi fără a scrie puterile lui 16).
Formula este: A:B -> $A*2^{nr\ biti}+B$ care aplicată la problema va fi:

$$01h * 2^8 + 18h = 1* 2^8 + 24 = 256 + 24 = 280$$

Similar, se tratează cazul operației de scădere: sub se folosește prima dată pe partea inferioară, sbb se folosește a doua oară pe partea superioară.

Analizați instructiunile INC, DEC ȘI NEG – aceste instrucțiuni au doar un singur operand și au următorul efect:
INC – adună un 1 la operand;
DEC –scade un 1 din operand,
NEG – obține inversul unui număr sau realizează 0-nr

Problema 5-3. Să se scrie o secvenţă de program care realizează suma a 2 variabile reprezentate pe câte 2 cuvinte fiecare, de exemplu: 12345678h şi 9ABCDEF0h, numerele fiind exprimate în convenţia *fără semn*.

Rezolvare:

Se presupune că prima variabilă este 12345678h adică 305.419.896 în zecimal, iar cea de-a doua variabilă este 9ABCDEF0h, adică 2.596.069.104. Suma celor 2 numere va fi: 2.901.489.000=ACF13568h. Pentru obţ. rezultatului, deoarece în procesorul 8086 nu există regiştri de 32 biţi, se împarte fiecare variabilă în două grupe de cuvinte (adică doi octeți fiecare grupă), conform Figurii 10:

```
org 100h        ; p5_03
; se alocă în memorie câte 2 cuvinte pentru fiecare variabilă, astfel:
.data
A  dw  1234h        ; se defineşte variabila A în mem., pe cuvânt: A=1234h
B  dw  5678h        ; se defineşte variabila B în mem., pe cuvânt: B=5678h
C  dw  9ABCh        ; se defineşte variabila C în mem., pe cuvânt: C=9ABCh
D  dw  0DEF0h       ; se defineşte variabila D în mem., pe cuvânt: D=DEF0h
; se alocă în memorie 2 cuvinte și pentru rezultat:
E  dw  0            ; se alocă un cuvânt în memorie pentru variabila E=0
F  dw  0            ; se alocă un cuvânt în memorie pentru variabila F=0
.code
mov AX, B           ; AX=B=5678h
add AX, D           ; AX=B+D=3568h
mov F, AX           ; F=B+D=3568h
mov AX, A           ; AX=A=1234h, iar instrucţ. mov nu va afecta flagul CF
adc AX, C           ; AX=(A+C)+CF=ACF0h+1=ACF1h
mov E, AX           ; E=ACF1h
```

a) rulaţi programul de mai sus cu simulatorul, urmăriţi variabilele în zona de memorie şi apoi specificaţi valorile obţinute în zecimal (atât pentru A,B,C,D,E,F cât şi pentru A:B, C:D şi E:F); verificaţi suma atât în hexazecimal cât şi în zecimal;

b) Înlocuiţi în variabila A cu valoarea 7654h şi reanalizaţi punctul a);

c) Rescrieţi problema a.î. cele 2 variabile ce se doresc a fi însumate să fie scrise pe câte 4 octeţi fiecare. Comparaţi rezultatele obţinute cu cele de la pct a).

d) Urmăriţi calculul: ACF1h= 44273, iar 3568h= 13672; folosind formula propusă, se va obţine:

$44273 * 2^{16} + 13672 = 2.901.475.328 + 13.672 = 2.901.489.000$.

Figura 10. a) Logica programului; b) corespondenţa între zona de memorie şi variabilele folosite în problema ***PR5-3***

Problema 5-4. Să se scrie un program care să realizeze suma a două variabile din memorie reprezentate pe câte 6 octeţi fiecare. Suma obţinută se va considera tot în memorie, într-o variabilă scrisă de asemenea pe 6 octeţi.
Suma poate fi efectuată la nivel de byte sau la nivel de word, similar abordării de la ***problema precedentă.*** Se va rezolva în ambele variante.

Indicaţie: Se împarte fiecare variabilă în 3 grupe de câte 2 octeţi sau în 6 grupe de câte un octet, deci vom putea realiza însumarea intermediară prin adunare de cuvinte sau de octeţi, conform figurii:

Cele 2 variabile (de exemplu valorile 123456789ABCh şi 0ABCD87564321h) pot fi definite: a) la nivel de cuvinte, iar apoi preluate în regiştri de 16 biţi;
b) ca octet, iar apoi vor fi preluate în regiştri de 8 biţi;
(propusă) c) la nivel de cuvinte, iar apoi preluate în regiştri de 8 biţi;
(şi în simulator se pot defini date de tip dword (32 biţi),
deşi regiştrii au dimensiune maximă de 16 biţi).

Rezolvare:

a1) Se poate lucra folosind 3 variabile de tip cuvânt pentru fiecare variabilă, aşa cum se propune mai jos:

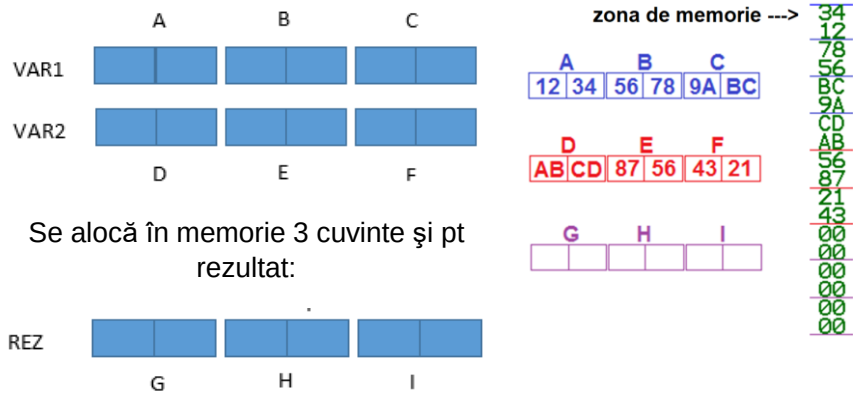

Figura 11. Ilustrarea variabilelor pentru ***PR5-4***, varianta a1)

Se realizează suma dintre C şi F, iar rezultatul se depune în I. Similar, se însumează B cu E (rezultatul se va depune în H) şi în final, se însumează A cu D (rezultatul se depune în G). Se ţine cont şi de transportul care poate apărea între două grupe consecutive, dinspre dreapta înspre stânga, deci se va folosi şi instrucţiunea *adc*.

```
mov AX, C       ; p5_4_a1                ; AX=9ABCh
add AX,F        ; AX=9ABCh+4321h=0DDDDh, CF se poate seta
mov I, AX       ; se depune rezultatul intermediar în memorie
mov AX, B       ; AX=5678h (instrucţiunea mov nu va afecta carry flag)
adc AX,E        ; AX=5678h+8756h=0DDCEh
mov H, AX       ; se depune rezultatul intermediar în memorie
mov AX, A       ; AX=1234h
adc AX,D        ; AX=1234h+0ABCDh=0BE01h
mov G, AX       ; ultimul posibil transport se va reflecta în CF
```

În cazul rezolvării de la punctul a1), resp. a2) variabilele programului au fost definite în memorie folosind directivele:

```
A dw 1234h
B dw 5678h
C dw 9ABCh
D dw 0ABCDh
E dw 8756h
F dw 4321h
G dw ?
H dw ?
I dw ?
```

```
var1 dw 1234h, 5678h, 9ABCh
var2 dw 0ABCDh, 8756h, 4321h
rez dw ?,?,?
```

a2) o altă posibilă variantă este cu folosirea unei variabile de tip şir sau vector cu 3 elemente de tip cuvânt:

```
mov AX, var1 [4]      ; p5_4_a2      ; AX=9ABCh
add AX,var2 [4]       ; AX=9ABCh+4321h=0DDDDh, CF se poate seta
mov rez [4], AX       ; se depune rezultatul intermediar în memorie
mov AX, var1 [2]      ; AX=5678h (instrucţiunea mov nu va afecta carry flag)
adc AX,var2 [2]       ; AX=5678h+8756h=0DDCEh
mov rez [2], AX       ; se depune rezultatul intermediar în memorie
mov AX, var1 [0]      ; AX=1234h
adc AX,var2 [0]       ; AX=1234h+0ABCDh=0BE01h
mov rez [0], AX       ; ultimul posibil transport se va reflecta în CF
```

b) cele 2 variante adaptate la nivel de 8 biţi se vor scrie (variabilele sunt acum pe 8 biţi şi au fost redefinte ca A1, B1, ... respectiv A2, B2, ...):

```
org 100h    ; p5_4_b1
.data
A1 db 12h
B1 db 34h
C1 db 56h
...
A2 db 0ABh
B2 db 0CDh
C2 db 87h
...
.code
mov AL, F1   ; AL=0BCh
add AL,F2    ; AL=BCh+21h=0DDh
mov F3, AL
mov AL, E1   ; AL=9Ah
adc AL,E2    ; AL=9Ah+43h=0DDh
mov E3, AL
...
```

```
org 100h    ; p5_4_b2
.data
var1 db 12h,34h,56h,78h,9Ah,0BCh
var2 db
0ABh,0CDh,87h,56h,43h,21h
rez db ?,?,?, ?,?,?

.code
mov AL, var1[5]   ; AL=0BCh
add AL,var2[5]    ; AL=BCh+21h=0DDh
mov rez[5], AL
mov AL, var1[4]   ; AL=9Ah
adc AL,var2[4]    ; AL=9Ah+43h=0DDh
mov rez[4], AL
...
```

Problema 5-5 Se consideră zona din memorie de 30 octeţi începând de la locaţia 1200h. Să se scrie o secvenţă de program care să umple această zonă din memorie cu valoarea AAh. Parcurgeţi această zonă cu ajutorul instrucţiunii inc; pentru implementarea operaţiei, consideraţi următoarea sugestie: se poate repeta un bloc de operaţii de atâtea ori cât arată registrul CX dacă se foloseşte construcţia:

```
mov CX, nr_de_repetări
eti:    ...          ; operaţia 1 ce se doreşte a fi repetată
        ...          ; operaţia 2 ce se doreşte a fi repetată
        ...          ; operaţia n ce se doreşte a fi repetată
        loop eti  ; asigură revenirea în program înapoi la eti de CX ori
```

Rezolvare:

```
...     ; definirea segm. de date, cod
mov BX,1200h
mov CX, 30
eti:
        mov [BX], 0AAh      ; la adresa dată de BX se depune valoarea AAh
        inc BX              ; se trece la următ. locaţie incrementând valoarea reg. BX
loop eti                    ; se revine la eti, parcurgând bucla de 30 de ori
```

La terminarea programului, în BX va fi adresa locaţiei următoare, adică 121Eh, iar adresa fizică este 0700h:121Eh=07000h+121Eh= 0821Eh.

Problema 5-6 . Se dă un şir cu 3 elemente pe octet. Scrieţi o secvenţă de program care să calculeze suma elementelor şirului folosind:
a) adresare bazată, b) adresare indexată; c) adresare bazată-indexată;
d) presupunând că se doreşte realizarea sumei unui şir de 10 elemente, numere mici (deci nu apare depăşire), repetaţi punctele b) şi c) folosind bucle.

Rezolvare:

a) ***Adresare bazată***

```
org 100h    ; p5_06_a
sir db 82h,9Bh,3Ah
suma dw ?
.code
clc
mov AX,0
mov BX, offset sir;
mov AL, [BX]

add AL, [BX+1]
adc AH,0

add AL, [BX+2]
adc AH,0
mov suma, AX
```

b) ***Adresare indexată***

```
org 100h  ; p5_06_b
sir db 82h,9Bh,3Ah
suma dw ?
.code
clc
mov AX,0
mov SI,0
mov AL, sir[SI]

inc SI
add AL, sir[SI]
adc AH,0
inc SI
add AL, sir[SI]
adc AH,0
mov suma, AX
```

c) ***Adresare bazată-indexată***

```
org 100h   ; p5_06_c
sir db 82h,9Bh,3Ah
suma dw ?
.code
clc
mov AX,0
mov SI,0
mov BX, offset sir
mov AL, [BX+SI]
inc SI
add AL, [BX+SI]
adc AH,0
inc SI
add AL, [BX+SI]
adc AH,0
mov suma, AX
```

d) **cu bucle**

```
org 100h
.data
sir db 3,5,4,2,7,0,1,1,2,1
suma db ?       ; la finalul execuţiei, suma trebuie să fie 26=1Ah
.code      ; p5_06_d1

mov SI,0

mov AL, sir[SI]

eti: inc SI
    add AL, sir [SI]
    cmp SI,10 ; nr de elemente ale şirului
  jl eti ; jump if SI is less than 10

mov suma, AL
```

```
.code     ; p5_06_d2
mov BX, offset sir
mov SI,0
mov CX, 10 ; nr de elem ale şirului
mov AL, [BX+SI]

eti: inc SI
     add AL, [BX+SI]
loop eti ; DEC CX and if it's not yet 0, repeat from eti

mov suma, AL
```

Problema 5-7. Scrieţi o secvenţă de program care să calculeze suma elementelor de pe poziţie pară (se consideră că suma nu intră în depăşire).

Rezolvare:

Cu adresare directă, fără indexare

```
org 100h    ; p5_07_a
.data
sir db 3,2,6,5,7,9,1,4,0,8
.code
mov AL,0        ; AL=0

add AL, sir     ; AL=0+3 = 3

add AL, sir+2   ; AL=3+6 = 9

add AL, sir+4   ; AL=9+7 = 16

add AL, sir+6   ; AL=16+1 = 17

add AL, sir+8   ; AL=17+0 = 17
ret
```

Accesare manuală

```
org 100h    ; p5_07_b
.data
sir db 3,2,6,5,7,9,1,4,0,8
.code
mov AL,0        ; AL=0
mov SI, 0
add AL, sir[SI]  ; AL=0+3 = 3
add SI,2
add AL, sir[SI]  ; AL=3+6 = 9
add SI,2
add AL, sir[SI]  ; AL=9+7 = 16
add SI,2
add AL, sir[SI]   ; AL=16+1 = 17
add SI,2
add AL, sir[SI]   ; AL=17+0 = 17
ret
```

Cu adresare indirectă, cu indexare

Accesare automată

```
org 100h    ; p5_07_c
.data
sir db 3,2,6,5,7,9,1,4,0,8
.code
mov AL,0     ; AL=0
mov SI, 0    ; SI=0
mov CX, 5    ; CX=5

iar:
   add AL, sir[SI]  ; AL=0+3=3, apoi 3+6=9, apoi 9+7=16, …

   add SI,2         ; SI=2,apoi 4, 6, …

   loop iar         ; CX=4, apoi 3, 2, …

ret
```

În registrul AL va fi aceeaşi valoare; sunt prezentate doar diferite modalităţi de accesare a elementelor şirului.

Problema 5-8. Să se definească în memorie 5 octeţi şi să realizeze suma acestora. Valoarea sumei se va depune în memorie în *suma*.

Rezolvare: O primă variantă este cu definirea celor 5 octeţi ca variabile separate, însă această soluţie nu este acceptabilă pentru un nr mare de elemente de prelucrat. Cea de-a doua variantă, cu definirea valorilor ca elemente ale unui şir sau vector şi mai ales cu folosirea unei bucle este mult mai potrivită.

Fol. adunare cu carry cu reg. pe 8 biţi

```
org 100h    ; p5_08_a
.data
a db 0E4h
b db 56h
c db 4Ah
d db 9Ch
e db 7Dh
suma dw ?
.code
mov AL, a
mov AH, 0
add AL, b
adc AH, 0
add AL, c
adc AH, 0
add AL, d
adc AH, 0
add AL, e
adc AH, 0
mov suma, AX
ret
```

Fol. adunare fără carry cu reg. pe 16 biţi

```
org 100h    ; p5_08_b
.data
a db 0E4h
b db 56h
c db 4Ah
d db 9Ch
e db 7Dh
suma dw ?
.code
mov BX, 0
mov AL, a
mov AH, 0
add BX, AX
mov AL, b
add BX, AX
mov AL, c
add BX, AX
mov AL, d
add BX, AX
mov AL, e
add BX, AX
mov suma, BX
ret
```

Fol. adunare cu carry cu reg. pe 8 biţi şi loop

```
org 100h    ; p5_08_c
.data
x db 0E4h, 56h, 4Ah, 9Ch, 7Dh
suma dw ?
.code
mov AL, x [0]


mov CX, 4
mov SI, 1
eti: add AL, x [SI]
     adc AH, 0
     inc SI
loop eti
mov suma, AX
ret
```

Valoarea sumei în oricare din cazuri

Fol. adunare fără carry cu reg. pe 16 biţi şi loop

```
org 100h    ; p5_08_d
.data
x db 0E4h, 56h, 4Ah, 9Ch, 7Dh
suma dw ?
.code
mov BX, 0
mov BL, x[0]
mov AH, 0
mov CX, 4
mov SI, 1
eti: mov AL, x [SI]
     add BX, AX
     inc SI
loop eti
mov suma, BX
ret
```

este 029Dh.

6. Instrucțiuni pentru înmulțire și împărțire: MUL, IMUL, DIV, IDIV

Instrucţiunea MUL înmulţeşte doi operanzi, dintre care primul e stocat implicit în acumulator, rezultatul fiind apoi plasat în acumulatorul extins. Instrucţiunea se aplică pentru numere ***fără semn***.
Instrucţiunea IMUL înmulţeşte doi operanzi, consideraţi ca numere întregi, ***cu semn***. Astfel, se va ţine cont de semnul rezultatului.
Unde "ACC" = AL sau AX, iar "Acc ext" = AX sau DX:AX

Obs:
Carry flag CF şi *Overflow flag* OF sunt setate dacă jumătatea superioară a rezultatului (AH în cazul operanzilor pe octet, sau DX pentru operanzi pe cuvânt) conține bitul cel mai semnificativ al produsului, în caz contrar sunt şterse.

În general, ***înmulţirea a 2 numere pe n biţi*** va produce un ***rezultat pe 2n biţi***.

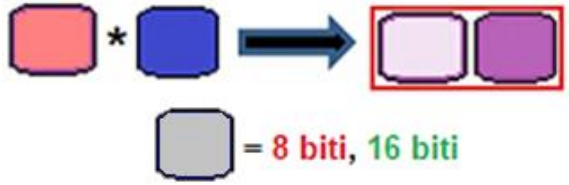

Figura 12. Ilustrarea modului de realizare al operaţiei de înmulţire

Exemplul 1:
; 82h = 130 (unsigned) = -126 (signed)
mov AL, 82h
mov BL, 2
mul BL ; 82h= 130 = -126 => **AL** * BL = 130 * 2 -> **AX** = rez =260 = 0104h
Dar daca scriem:
imul BL ; 82h= 130 = -126 => **AL** * BL = -126 * 2 -> **AX** = rez = - 252 = FF04h

Exemplul 2:
40000 =9C40h
100 = 64h
mov AX, 40000
mov BX, 100
mul **BX** ; **AX** * BX = 40 000 * 100 => **DX:AX** =rez= 4 000 000

Instrucţiunea DIV împarte doi operanzi, dintre care primul (deîmpărţitul) e stocat implicit în Acc extins, rezultatul fiind apoi plasat tot în Acc (aici se depune câtul) şi extensia acestuia (aici se depune restul). Instrucţiunea se aplică pentru numere ***fără semn***.
Instrucţiunea IDIV împarte doi operanzi ca numere întregi ***cu semn***.

Împărţirea a două numere în binar se efectuează invers operaţiei de înmulţire: va fi nevoie de scrierea ***deîmpărţitului ca un număr pe 2n biţi***,

de exemplu 157 = 10011101b; ***împărţitorul se va scrie folosind doar n biţi***, de exemplu 13=1101b. Realizând operaţia de împărţire, se va obţine câtul 12=1100b și restul 1=0001b, care așa cum se poate observa se vor scrie tot ***pe n biţi fiecare*** (atât ***câtul*** cât şi ***restul***).

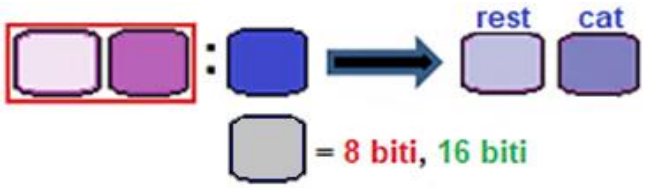

Figura 13. Ilustrarea modului de realizare al operaţiei de împărţire

9C40h = 40 000= -25536

Exemplul 1:
Pe 8 biti:
Mov AX, 9C40h
Mov BL, 200
DIV **BL** ; AX / BL = 40.000/200 -> REST=AH=00h CAT = AL=C8h=200,
Iar daca scriem:
IDIV **BL** ; AX / BL = -25536/200 -> REST=AH CAT = AL, dar aici ne dă o eroare “divide by zero” pt că de fapt încearcă să împartă -25536 la -56 (200=C8h=-56), iar calculul -25536 : (-56) furnizează un cât prea mare, care nu se poate scrie pe doar 8 biți cât are disponibili pentru a scrie câtul . Soluția în acest caz este să ducem operația pe dimensiunea următoare, deci să depunem deîmpărțitul într-un registru de 32 biți și să împărțim cu un nr scris într-un registru de 16 biți.

Exemplul 2:
Pe 16 biți:
mov AL, 200 ; nr -56 interpretat ca nr cu semn: AL=200=C8h=-56
cbw ; AX=200=FFC8h = -56
mov BX, AX ; BX=200=FFC8h = -56

mov AX, 9C40h ; AX=9C40h=40 000 = -25 536
cwd ; DX:AX=FFFF : 9C40h=-25536
IDIV BX ; (DX:AX) / BX = -25536/(-56) -> REST=DX=00h, CAT = AX=01C8h = 456,

Problema 6-1. Scrieţi directive sau instrucţiuni pentru următoarele:
a) Definiţi un vector “x” cu elemente pe octet, iniţializat cu valorile pare cuprinse între 0 şi 20h;
b) definiţi o constantă doi de valoare 2;
c) încărcaţi într-un registru adresa variabilei x;
d) realizaţi suma celui de-al treilea element al lui x cu nr. de la punctul b) şi depuneţi-o într-un registru;
e) obţineţi opusul valorii de la punctul d) şi apoi înmulţiţi acest număr cu ultimul element al vectorului;
f) modificaţi programul astfel încât să fie definită şi o zonă de memorie (numită *produs*) adecvată pentru a stoca rezultatul obţinut la punctul e);
g) realizaţi suma elementelor şirului folosind o buclă cu loop ca la problema anterioară şi depuneţi rezultatul obţinut în memorie într-o variabilă *suma*.

Rezolvare:

a) x db 0h, 2h, 4h, 6h, 8h, 0Ah, 0Ch, 0Eh, 10h, 12h, 14h, 16h, 18h, 1Ah, 1Ch, 1Eh;
b) doi EQU 2
c) lea BX, x ; *sau mov BX, offset x*
d) mov AL, x[2] ; *AL = 04h = al treilea element al lui x*
   add AL, doi ; *AL = 04h+02h=06h = 6 = suma cerută*
e) neg AL ; *AL = - 6 = FAh = valoarea negată sau opusul nr*
   mov AH, x[15] ; *AH = 1Eh = ultimul element al şirului*

; pentru înmulţire cu semn:
imul AH ; *se va înmulţi valoarea din AL cu cea din AH, ambele fiind considerate numere cu semn,*
; *se înmulţeşte -6*30=-180 şi se obţine AX=-180=FF4Ch*

; pentru înmulţire fără semn:
mul AH ; *se va înmulţi valoarea din AL cu cea din AH, ambele fiind considerate numere fără semn,*
; *se înmulţeşte -6=FAh care acum e văzut ca 250 şi se obţine AX=250*30=7500=1D4Ch*

f) se va defini în zona de date: *produs dw ?*
iar plasarea rezultatului se va realiza în zona de cod folosind instrucţiunea:
mov produs, AX deoarece produsul s-a obţinut în registrul AX
g) se va defini în zona de date: *suma dw ?*
Deşi elementele şirului sunt pe octet, dacă acestea sunt valori mari, suma poate intra în depăşire (e posibil să aibă nevoie de 9 biţi pentru a fi stocată), deci se foloseşte variabilă de tip cuvânt, în loc de una de tip octet.
În zona de cod se va scrie secvenţa care calculează suma elementelor şirului:

```
mov AL, x[0]        ; în AL se depune primul element al şirului
mov AH, 0           ; în AH se va vedea eventuala depăşire
mov BX, 1           ; BX=indexul 01h
mov CX, 15          ; nr de repetări ale buclei
eti:  add AL, x [BX] ; adună la AL pe x[1], apoi x[2], ... , x[15]
      adc AH, 0      ; dacă cumva apare depăşire, se va acumula la AH
      inc BX         ; se trece la următorul index
      loop eti       ; asigură revenirea în program înapoi la eti de CX ori
mov suma, AX        ; val. obţinută în AX = 00F0h, se depune în suma
```

Problema 6-2. Să se scrie un program care să realizeze produsul a trei variabile de tip octet definite în memorie. Rezultatul se va stoca tot în memorie. Rezolvaţi în ambele cazuri, considerând numerele fără semn, respectiv cu semn.

Rezolvare:

Realizarea produsului între a, b şi c, unde a,b,c sunt de tip octet se poate realiza astfel:

1) se înmulţeşte a cu b şi rezultatul se obţine în registrul AX;
2) se converteşte c la acelaşi tip al datei, respectiv cuvânt
3) se înmulţeşte rezultatul de la 1) cu cel de la 2) şi rezultatul final se obţine în perechea de regiştri DX:AX ca un dublucuvânt;

Pentru numere fără semn

```
org 100h     ; p6_2_a
.data
a db 10          ; 10
b db 0ABh        ; 10x16+11 =171
c db 100         ; 100
produs dd ?
.code
mov AL, a
mov BL, b
mul BL   ; rezultat în AX (pas 1)


mov BH, 0
mov BL, c  ; (pas 2)
mul BX   ; rezultat în DX:AX (pas 3)
mov produs, AX
mov produs +2, DX
ret
;realizaţi calculele şi verificaţi că
DX:AX=0002:9BF8h e corect
calculat în baza10=171000
```

Pentru numere cu semn

```
org 100h      ; p6_2_b
.data
a db 10          ; 10
b db 0ABh        ; -85
c db 100         ; 100
produs dd ?
.code
mov AL, a
mov BL, b
imul BL       ; rezultat în AX (pas 1)
mov BX, AX ; se copiază rez în BX
mov AL, c
cbw
xchg BX,AX  ; (pas 2)
imul BX   ; rezultat în DX:AX (pas 3)
mov produs, AX
mov produs +2, DX
ret
;realizaţi calculele şi verificaţi că
DX:AX=FFFE:B3F8h e corect
calculat în baza10 = -85000
```

Problema 6-3. Să se scrie un program care să realizeze următorul calcul: E = (2*a + b/c) * d,
unde a, b şi d sunt de tip byte, iar c este de tip word şi sunt definite în memorie. Rezultatul se va stoca tot în memorie. Rezolvaţi în ambele cazuri, considerând numerele fără semn, respectiv cu semn.

Rezolvare:

Obţinerea expresiei se poate realiza astfel:

1) se înmulţeşte 2 cu a (deci *byte*byte=> word*) şi rezultatul se obţine în registrul AX;
2) se realizează împărţirea **b/c**, dar aici apare o problemă: **b** e de tip octet, iar **c** e de tip cuvânt; astfel, prima dată se va converti **b** la dimensiunea dublă a lui **c** şi abia apoi se poate realiza împărţirea; rezultatul se va obţine sub formă de cât, de dimensiune *dword / word*, deci *word* în registrul AX.
3) se adună rezultatele de la 1) şi 2), ca *word + word*; astfel, s-a calculat toată paranteza în AX, de exemplu;
4) deoarece rezultatul de la 3) este pe *word*, va trebui convertită şi variabila **d** la *word* înainte de a realiza înmulţirea. Rezultatul final va fi de tip *word*word*, adică *doubleword* (obţ. în perechea DX:AX).

Pentru numere fără semn

```
org 100h      ; p6_3_a
.data
a db 10       ; 10
b db 0ABh     ; 10x16+11 =171
c dw 100      ; 100
d db 2        ; 2
calcul dd ?   ; rez
.code
mov AL, a     ; preluare a din memorie
mov BL, 2
mul BL        ; a*2->rezultat în AX (op 1)

mov CX, AX    ; copie a lui AX
mov AL, b     ; preluare b din memorie
mov AH, 0     ; ext fără semn
mov DX, 0     ; pregătire DX:AX
mov BX, c     ; preluare c din memorie

div BX        ; b/c ->rezultat în AX (op 2)

add AX, CX    ;(2*a + b/c)->rez. în AX (op 3)


mov BL, d     ; preluare d din memorie
mov BH, 0     ; ext fără semn
mul BX        ; (2*a + b/c) * d-> rezultat în DX:AX (op 4)

mov calcul, AX      ; depunem partea inf în memorie
mov calcul+2, DX    ; depunem partea sup în memorie
ret
; E = rezultat în DX:AX = 0000:002Ah=42
```

Pentru numere cu semn

```
org 100h      ; p6_3_b
.data
a db 10       ; 10
b db 0ABh     ; -85
c dw 100      ; 100
d db -2       ; -2
calcul dd ?   ; rez
.code
mov AL, a     ; preluare a din memorie
mov BL, 2
imul BL       ; a*2->rezultat în AX (op 1)

mov CX, AX    ; copie a lui AX
mov AL, b     ; preluare b din memorie
cbw           ; ext cu semn
cwd           ; pregătire DX:AX
mov BX, c     ; preluare c din memorie

idiv BX       ; b/c ->rezultat în AX (op 2)

add AX, CX    ; (2*a + b/c)->rez. în AX (op 3)

mov CX, AX    ; copie a lui AX
mov AL, d     ; preluare d din memorie
cbw           ; ext cu semn
imul CX       ; (2*a + b/c) * d-> rezultat în DX:AX (op 4)

mov calcul, AX      ; depunem partea inf în memorie
mov calcul+2, DX    ; depunem partea sup în memorie
ret
; E = rezultat în DX:AX = FFFF:FFD8h=-40
```