ARHITECTURA SISTEMELOR DE CALCUL
– Seminar 1 –

1. Conversia numerelor între bazele de numerație 2, 10, 16
2. Reprezentarea numerelor întregi în memorie
3. Elementele limbajului de asamblare IA-32
4. Primul program în limbaj de asamblare

1. CONVERSIA NUMERELOR ÎNTRE BAZELE DE NUMERAȚIE 2, 10, 16

Un sistem de numeraţie este constituit din *totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri* denumite *cifre*.

Pentru orice sistem de numeraţie, numărul de cifre ale sistemului este egal cu baza *b*:

– dacă *b* = 2 (sistemul de numerație binar) cifrele sunt 0 şi 1;
– dacă *b* = 10 (sistemul de numerație zecimal) cifrele sunt 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
– dacă *b* = 16 (sistemul de numerație hexazecimal) cifrele sunt 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Se observă că pentru numerele scrise într-o bază mai mare decât baza 10 (zecimal) se folosesc şi alte simboluri (litere) pe lângă cifrele obişnuite din baza 10. Astfel, în cazul numerelor scrise în hexazecimal, literele A, B, C, D, E, F au ca şi valori asociate 10, 11, 12, 13, 14, 15.

1.a. Conversia numerelor din baza 10 într-o bază oarecare

Se realizează prin *împărţirea succesivă* a numărului scris în baza 10 la baza în care se doreşte conversia (se împarte numărul la bază, apoi se împarte câtul obţinut la bază ş.a.m.d. până când câtul obținut devine 0), după care se iau resturile obţinute în ordine inversă, care constituie valoarea numărului în baza cerută.

Exemple:

1. Să se convertească numărul 347 din baza 10 în baza 16.

– împărțim succesiv numărul 347 la baza 16:

347	16		
– 32	21	16	
= 27	– 16	1	16
– 16	= **5**	– 0	0
= **11**		= **1**	

– luăm resturile obținute în ordine inversă și ținând cont că 11 reprezintă cifra B în baza 16, obținem:

$$\mathbf{347_{(10)} = 15B_{(16)}}$$

2. Să se convertească numărul 57 din baza 10 în baza 2.

– împărțim succesiv numărul 57 la baza 2:

57	2					
– 4	28	2				
= 17	– 2	14	2			
– 16	= 8	– 14	7	2		
= **1**	– 8	= **0**	– 6	3	2	
	= **0**		= **1**	– 2	1	2
				= **1**	– 0	0
					= **1**	

– luăm resturile obținute în ordine inversă, obținem:

$$\mathbf{57_{(10)} = 111001_{(2)}}$$

1.b. Conversia numerelor dintr-o bază oarecare în baza 10

Dacă considerăm numărul *N* format din *n* cifre scrise baza *b*:

$$N_{(b)} = c_{n-1}c_{n-2}\ldots c_2c_1c_0$$

atunci valoarea sa în baza 10 se obține astfel:

$$N_{(10)} = c_{n-1} * b^{n-1} + c_{n-2} * b^{n-2} + \ldots + c_2 * b^2 + c_1 * b^1 + c_0 * b^0$$

Exemple:

1. Care e valoarea în baza 10 a numărului întreg hexazecimal $3A8_{(16)}$?

$$3A8_{(16)} = 3*16^2 + A*16^1 + 8*16^0 = 3*256 + 10*16 + 8*1 = 936_{(10)}$$

2. Care e valoarea în baza 10 a numărului întreg binar $1101101_{(2)}$?

$$1101101_{(2)} = 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 1*64 + 1*32 + 1*8 + 1*4 + 1 = 109_{(10)}$$

1.c. Conversia rapidă între bazele de numerație 2 și 16

Putem realiza conversii rapide între numere scrise în bazele de numerație 2 și 16, ținând cont de faptul că *fiecărei cifre hexazecimale îi corespund 4 (patru) cifre binare* și invers (vezi tabelul de mai jos).

Valoare zecimală	Cifra hexazecimală	Numărul binar corespondent
0	**0**	**0000**
1	**1**	**0001**
2	**2**	**0010**
3	**3**	**0011**
4	**4**	**0100**
5	**5**	**0101**
6	**6**	**0110**
7	**7**	**0111**
8	**8**	**1000**
9	**9**	**1001**
10	**A**	**1010**
11	**B**	**1011**
12	**C**	**1100**
13	**D**	**1101**
14	**E**	**1110**
15	**F**	**1111**

2. REPREZENTAREA NUMERELOR ÎNTREGI ÎN MEMORIE

Reprezentarea unui număr se referă la exprimarea/configurația sa în baza 2 în memoria calculatorului.

În arhitectura IA-32 un număr întreg poate fi reprezentat pe 8, 16, 32 sau 64 de biți (1, 2, 4 sau 8 octeți).

Există 2 convenții de reprezentare: *convenția de reprezentare fără semn* și *convenția de reprezentare cu semn*. Astfel, într-o locație de memorie de *n* biți se află:

- fie un număr natural cuprins între 0 și 2^n-1, în *convenția de reprezentare fără semn*;
- fie un număr întreg cuprins între -2^{n-1} și $2^{n-1}-1$, în *convenția de reprezentare cu semn*.

2.a. Bitul de semn

Dacă interpretăm o anumită configurație de biți ca întreg cu semn, atunci, prin convenție, pentru reprezentarea semnului unui număr se folosește un singur bit (numit *bit de semn*).

Bitul de semn = *bitul superior (high) din octetul superior (high)* al locației în care se reprezintă numărul.

Dacă valoarea acestui bit este 0, atunci numărul este POZITIV. Dacă valoarea acestui bit este 1, atunci numărul este NEGATIV.

2.b. Regula de reprezentare a numerelor întregi

Un număr întreg cuprins între $-2^n - 1$ și $2^{n-1} - 1$ se reprezintă într-o locaţie de n biţi astfel:
- dacă numărul este pozitiv, atunci în locaţie se reprezintă numărul respectiv scris în baza 2;
- dacă numărul este negativ, atunci în locaţie se înscrie complementul în baza 2 a numărului.

2.c. Complementul unui număr întreg

De-a lungul timpului au existat mai multe modalități de reprezentare cu semn a unui număr întreg:
- *cod direct*: se reprezintă valoarea absolută a numărului pe n-1 biți din cei n ai locației, iar în bitul cel mai semnificativ să se pună semnul. Deși această soluție e foarte apropiată de cea naturală, s-a dovedit a fi mai puțin eficientă decât altele. O problemă ar fi faptul că în această reprezentare $-7 + 7 \neq 0$.
- *cod invers (complement față de 1)*: se reprezintă valoarea absolută a numărului pe n-1 biți din cei n ai locației, iar în cazul în care numărul este negativ, se inversează toți cei n biți ai reprezentării. Și la această reprezentare s-a renunțat, deoarece s-a dovedit ineficientă (în plus, apare din nou problema $-7 + 7 \neq 0$).
- *cod complementar (complement față de 2)*

Numerele întregi cu semn se reprezintă în cod complementar (complementul față de 2).
Principalul avantaj oferit de codul complementar față de alte modalități de reprezentare este că bitul de semn participă la stabilirea valorii numărului.

Definiție *Pentru complementarea unui număr întreg reprezentat pe n biți, mai întâi se inversează valorile tuturor biților (valoarea 0 devine 1 și valoarea 1 devine 0) din locația de reprezentare, după care se adaugă 1 la valoarea obținută.*

Exemplu: Care este reprezentarea în baza 2 a numărului întreg reprezentat pe un octet -18?

Se consideră 18, valoarea absolută în baza 10 a numărului dat. Vom obține complementul acestuia.

Procedăm astfel:
- convertim numărul dat în baza 2:

  $18_{(10)} = 10010_{(2)}$

- calculăm complementul față de 2 al numărului dat conform definiției:

Reprezentarea inițială	00010010
După inversarea biților	11101101
Se adună 1	11101101 +
	00000001
Complementul față de 2	**11101110**

Astfel, complementul numărului $18_{(10)} = 12_{(16)} = 00010010_{(2)}$ este $\mathbf{11101110}_{(2)} = \mathbf{EE}_{(16)} = \mathbf{238}_{(10)}$.

Așadar, $-18_{(10)} = \mathbf{11101110}_{(2)} = \mathbf{EE}_{(16)}$.

Reguli alternative de complementare

1. *Se lasă neschimbați biții începând din dreapta reprezentării binare până la primul bit 1 inclusiv, iar restul biților se inversează până la bitul n-1 inclusiv.*
2. *Se scade binar conținutul (evident binar) al locației de complementat din 100...00, unde numărul de după cifra binară 1 are atâtea zerouri câți biți are locația de complementat.*
3. *Se scade hexazecimal conținutul (evident hexazecimal) al locației de complementat din 100...00, unde după cifra hexazecimală 1 apar atâtea zerouri câte cifre hexazecimale are locația de complementat.*

Exemplu:

Care este reprezentarea în baza 2 a numărului întreg reprezentat pe un octet -18?

Se consideră 18, valoarea absolută în baza 10 a numărului dat. Vom obține complementul acestuia.

Procedăm astfel:

- convertim numărul dat în baza 2:

  $18_{(10)} = 10010_{(2)}$

- aplicăm regula de scădere binară (regula nr. 2):

  100000000 –
  00010010
  11101110$_{(2)}$

sau

- convertim numărul dat în baza 16:

  $18_{(10)} = 12_{(16)}$

- aplicăm regula de scădere hexazecimală (regula nr. 3):

  100 –
  12
  EE$_{(16)}$

Așadar, $-18_{(10)}$ =**11101110**$_{(2)}$ **= EE**$_{(16)}$.

2.d. Dimensiune de reprezentare

Dimensiunea de reprezentare este numărul maxim de cifre binare (numărul de biți) din reprezentarea unui număr întreg.

2.e. Interval de reprezentare admisibil

Reprezentarea se referă la baza 2, iar interpretările la baza 10, fiind întotdeauna posibile 2 interpretări pentru o reprezentare.

Intervalul de reprezentare admisibil este intervalul de valori în baza 10 care sunt posibil a fi reprezentate în baza 2 pe 1 OCTET, 1 CUVÂNT, 1 DUBLUCUVÂNT etc.

Având în vedere că orice configurație binară are 2 interpretări posibile (cu semn și fără semn) rezultă că fiecare dimensiune de reprezentare a datelor în calculator are asociate 2 intervale de reprezentare admisibile:

- intervalul de reprezentare admisibil în reprezentarea cu semn;
- intervalul de reprezentare admisibil în reprezentarea fără semn.

Pentru valorile cele mai frecvent utilizate la nivelul programelor, acestea sunt:

[0, +255]	interval de reprezentare admisibil pentru „întreg fără semn reprezentat pe 1 octet”
[–128, +127]	interval de reprezentare admisibil pentru „întreg cu semn reprezentat pe 1 octet”
[0, +65535]	interval de reprezentare admisibil pentru „întreg fără semn reprezentat pe 2 octeți”
[–32768, +32767]	interval de reprezentare admisibil pentru „întreg cu semn reprezentat pe 2 octeți”

3. ELEMENTELE LIMBAJULUI DE ASAMBLARE IA-32

3.a. Tipuri fundamentale de date

În limbaj de asamblare, **tip de dată** = **dimensiune de reprezentare**.

- bit (**bi**nary digi**t**): este unitatea elementară de informație
- octet (byte): o succesiune de 8 biți, numerotată, prin convenție, astfel:

Octet (8 biți)							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

- cuvânt (word): o succesiune de 16 biți (2 octeți), numerotată, prin convenție, astfel:

Cuvânt (16 biți)															
Octetul superior (high)								Octetul inferior (low)							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

– dublucuvânt (doubleword): o succesiune de 32 biți (4 octeți), numerotată, prin convenție, astfel:

Dublucuvânt (32 biți)																															
Cuvântul superior (16 biți)																Cuvântul inferior (16 biți)															
Octetul superior al cuvântului superior								Octetul inferior al cuvântului superior								Octetul superior al cuvântului inferior								Octetul inferior al cuvântului inferior							
31							**24**	**23**							**16**	**15**							**8**	**7**							**0**

– quadword: o succesiune de 64 biți (8 octeți), numerotată, prin convenție, astfel:

Quadword (64 biți)																															
Dublucuvântul superior (32 biți)																Dublucuvântul inferior (32 biți)															
Cuvântul superior al dublucuvântului superior								Cuvântul inferior al dublucuvântului superior								Cuvântul superior al dublucuvântului inferior								Cuvântul inferior al dublucuvântului inferior							
63			**...**			**48**	**47**			**...**			**32**	**31**			**...**			**16**	**15**			**...**			**0**					

Exemplu: Se da quadword-ul 1A2B3C4D95847362h. Care este valoarea octetului inferior al cuvantului superior al dublucuvantului inferior din quadword-ul dat?

Qword-ul dat: 1A2B3C4D95847362h

Dublucuvantul inferior la qword-ului dat: 95847362h

Cuvantul superior al dublucuvantului inferior la qword-ului dat: 9584h

Valoarea octetului inferior al cuvantului superior al dublucuvantului inferior din quadword-ul dat: 84h

3.b. Directive pentru definirea datelor

– definire date = declarare (specificarea atributelor) + alocare (rezervarea spațiului de memorie necesar)
– tipul de dată = dimensiunea de reprezentare: octet, cuvânt, dublucuvânt, quadword

– definirea unei variabile (declarare + inițializarea valorii acesteia)

[nume] ***tip_data*** ***lista_expresii*** **[;comentariu]**

unde:

– ***tip_data*** este o directivă de definire a datelor, una din următoarele:
DB – date de tip octet (byte – 8 biţi)
DW – date de tip cuvânt (word – 16 biţi)
DD – date de tip dublucuvânt (dword – 32 biţi)
DQ – date de tip 8 octeţi (qword – 64 biţi)
DT – date de tip 10 octeţi (tword – 80 biţi)

– declararea unei variabile (alocare fără inițializare)

[nume] ***tip_alocare*** ***factor*** **[;comentariu]**

unde:

– ***tip_alocare*** este o directivă de rezervare de date neiniţializate, una din următoarele:
RESB – date de tip octet (byte – 8 biţi)
RESW – date de tip cuvânt (word – 16 biţi)
RESD – date de tip dublucuvânt (dword – 32 biţi)
RESQ – date de tip 8 octeţi (qword – 64 biţi)
REST – date de tip 10 octeţi (tword – 80 biţi)
– ***factor*** este un număr care indică câți bytes/words/dwords/quadwords/twords vor fi alocați

– definirea unei constante

nume EQU ***valoare***

Exemplu:

```
; segmentul de date
segment data use32 class=data
    a db 7       ; 1 byte
    b dw 101b    ; 2 bytes (1 word)
    c dd 2bfh    ; 4 bytes (1 doubleword)
    d dq 307o    ; 8 bytes (1 quadword)
    e dt 1024    ; 10 bytes
    zece EQU 10
```

3.c. Regiștri în arhitectura IA-32

Arhitectura IA-32 pune la dispoziția programatorilor 16 regiștri. Acești regiștri pot fi grupați astfel:

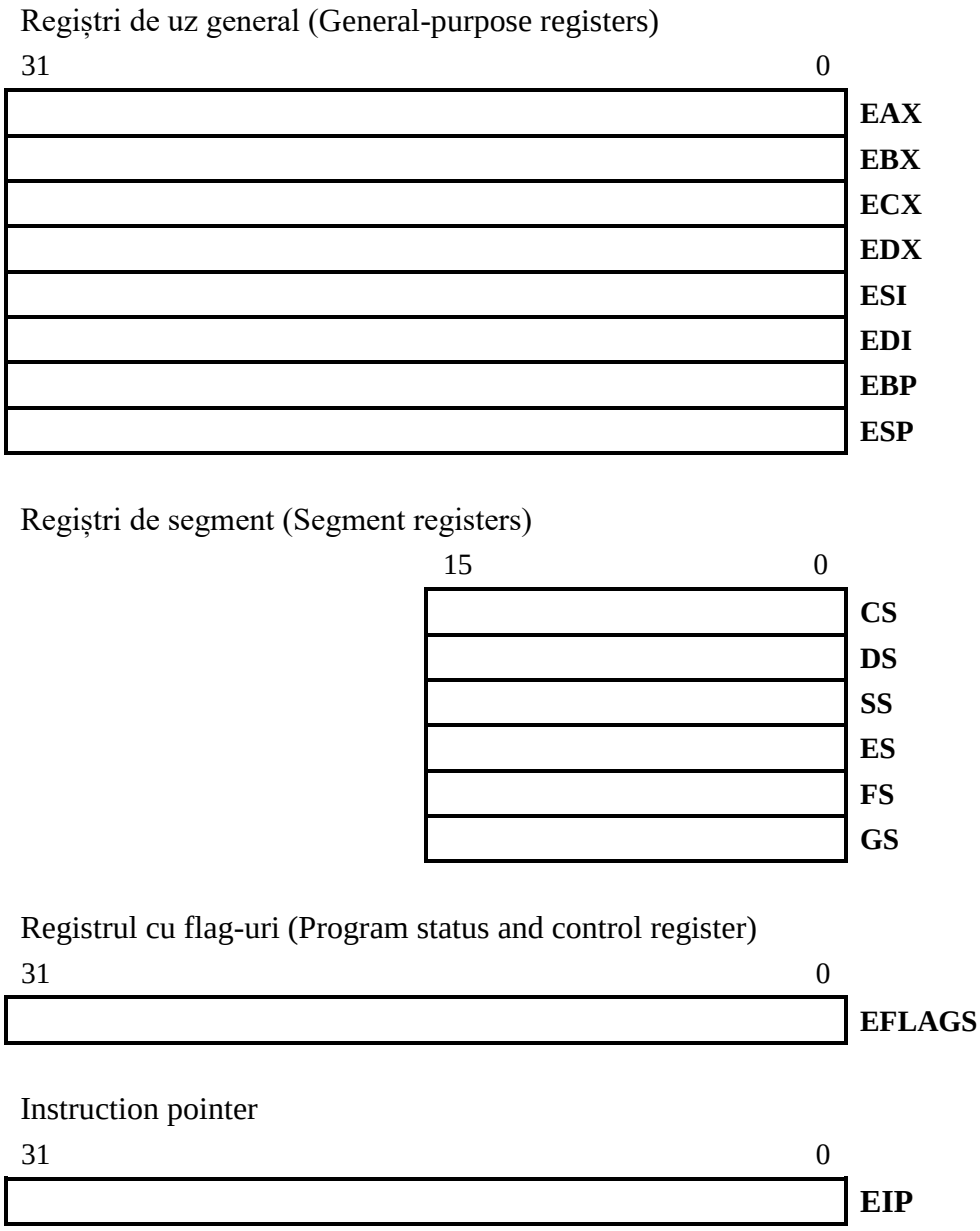
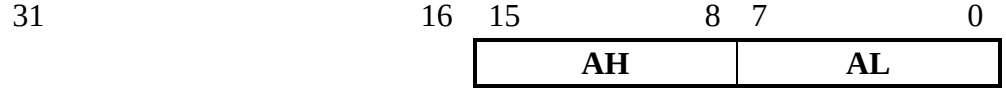


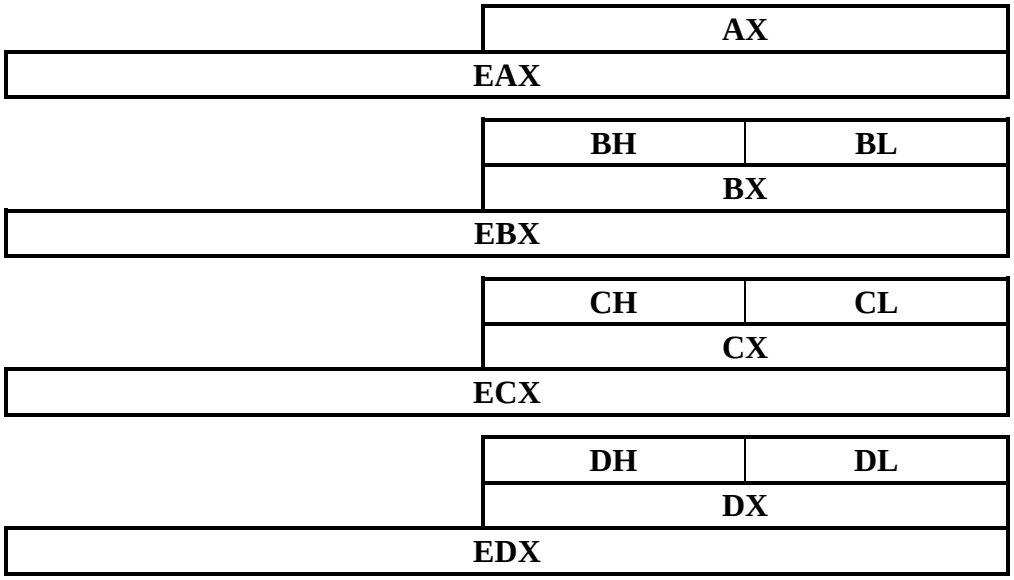
Fig. 1 Regiștri existenți în arhitectura IA-32

Regiștri de uz general (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP) sunt regiștri pe 32 de biți și sunt destinați pentru a stoca: operanzi/rezultate ale operațiilor aritmetice sau logice, operanzi folosiți în calcule de adresă sau pointeri spre locații de memorie.

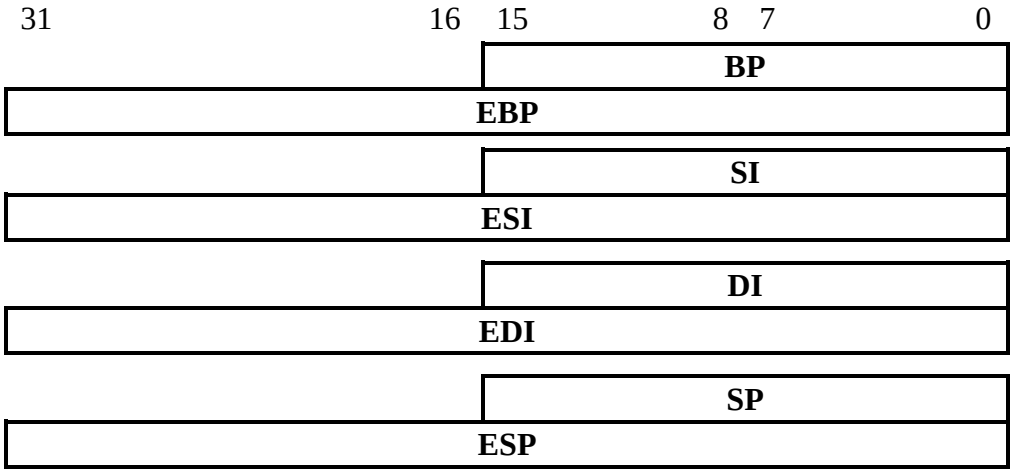
Regiștri EAX, EBX, ECX și EDX (pe 32 de biți) pot fi accesați și:

- pe 8 biți: AH | AL, BH | BL, CH | CL, DH | DL;
- pe 16 biți: AX, BX, CX, DX.





Regiștri EBP, ESI, EDI și ESP (pe 32 de biți) pot fi accesați DOAR pe 16 biți: BP, SI, DI, SP.



3.d. Formatul unei linii sursă

Formatul unei linii sursă în limbajul de asamblare este următorul:

[*etichetă*[*:*]] [*prefixe*] [*instrucțiune*] [*operanzi*] [*;comentariu*]

- *eticheta*: un nume scris de utilizator cu ajutorul căreia se pot referi date sau locații de memorie
- *instrucțiune*: indică simbolic (printr-o mnemonică) acțiunea ce va fi executată de către microprocesor
- *operanzi*: pot fi valori imediate, regiștri sau locații de memorie
- *comentariu*: orice text care începe cu caracterul `;`

3.e. Exemple de instrucțiuni în limbaj de asamblare

MOV

Sintaxa: **`mov`** `d, s`

unde:
- `d` poate fi un registru sau o locație de memorie
- `s` poate fi o valoare imediată (constantă), un registru sau o locație de memorie

Efect: `d ← s`

Cei doi operanzi `d` și `s` trebuie să aibă aceiași dimensiune de reprezentare (ambii sunt fie OCTEȚI, fie CUVINTE, fie DUBLUCUVINTE).

Ambii operanzi pot fi regiștri, însă CEL MULT un operand poate fi o locație de memorie.

Exemple:

```
mov al, 2
mov ax, bx
mov bl, [a]
mov byte [a], 22
mov [b], dl
```

ADD

Sintaxa: **add** d, s

unde:
- d poate fi un registru sau o locație de memorie
- s poate fi o valoare imediată (constantă), un registru sau o locație de memorie

Efect: d ← d + s

Cei doi operanzi d și s trebuie să aibă aceiași dimensiune de reprezentare (ambii sunt fie OCTEȚI, fie CUVINTE, fie DUBLUCUVINTE).

Ambii operanzi pot fi regiștri, însă CEL MULT un operand poate fi o locație de memorie.

Exemple:

```
add al, 10b
add ax, bx
add bl, [a]
add word [a], 10h
add [b], dx
```

SUB

Sintaxa: **sub** d, s

unde:
- d poate fi un registru sau o locație de memorie
- s poate fi o valoare imediată (constantă), un registru sau o locație de memorie

Efect: d ← d - s

Cei doi operanzi d și s trebuie să aibă aceiași dimensiune de reprezentare (ambii sunt fie OCTEȚI, fie CUVINTE, fie DUBLUCUVINTE).

Ambii operanzi pot fi regiștri, însă CEL MULT un operand poate fi o locație de memorie.

Exemple:

```
sub al, 0fh
sub ax, bx
sub bl, [a]
sub byte [a], 1010b
sub [b], edx
```

4. PRIMUL PROGRAM ÎN LIMBAJ DE ASAMBLARE

```
; vom programa pe 32 de biti
bits 32

; declar punctul de intrare in program
; (start este o eticheta care indica prima instructiune din program)
global start

; declar functiile externe pe care le voi folosi
extern exit
import exit msvcrt.dll

; segmentul de date
; aici vom declara datele (constante, variabile etc.)
segment data use32 class=data
```

```
; segmentul de cod
; aici vom scrie programul
segment code use32 class=code
    start:
      ; începând de aici vom scrie instructiunile care compun programul

      ...

      ; inchei executia programului
      push dword 0    ; pun pe stiva argumentul functiei
      call [exit]     ; apelez functia exit()
```

EXERCIȚII

Scrieți un program în limbaj de asamblare care să calculeze expresia aritmetică, considerând domeniile de definiție ale variabilelor:

a. 1 + 2

b. 1 – 2

c. 1 + 255

d. a + b, unde a, b – byte

e. a – b, unde a, b – byte

f. a + b, unde a, b – word

g. a – b, unde a, b – word

h. a + b, unde a, b – dword

i. a – b, unde a, b – dword

j. (a + b) – (c + 10), unde a, b, c – byte

k. (a + b) – (c + 10), unde a, b, c – word

l. (a + b) – (c + 10), unde a, b, c – dword