6. Setul de instrucţiuni 8086 (I)

Setul complet de comenzi pe care un microprocesor le înţelege şi le poate executa este cunoscut sub numele de **set de instrucţiuni**.
În următoarele două lucrări sunt prezentate instrucţiunile din setul de bază al familiei de procesoare Intel. Acestea dispun de un set puternic şi diversificat de instrucţiuni specifice procesoarelor de tip CISC (Complex Instruction Set Computer). Instrucţiunile pot fi clasificate în funcţie de mai multe criterii:

a)După **numărul de operanzi:**
- instrucţiuni fără operanzi
- instrucţiuni cu un operand
- instrucţiuni cu doi operanzi

b)După **lungimea instrucţiunii**:
- 1-6 octeţi

c)După **funcţia instrucţiunii**:
- instrucţiuni de transfer: MOV, PUSH, POP, IN, OUT — datele sunt copiate între memorie sau porturi I/O şi regiştrii procesorului, fără a fi prelucrate
- instrucţiuni aritmetice şi logice: ADD, INC, AND, CMP— datele sunt prelucrate în format numeric
- instrucţiuni de manipulare a şirurilor: MOVS, CMPS, REP
- instrucţiuni specifice întreruperilor: INT
- instrucţiuni de ramificare: CALL, JMP
- instrucţiuni de control al microprocesorului: CLC, STC, NOP, HLT

Se vor folosi următoarele **notaţii:**

reg – conţinutul unui registru oarecare de 8 sau 16 biţi, exceptând regiştrii segment
reg8 – conţinutul unui registru de 8 biţi
reg16 – conţinutul unui registru de 16 biţi
sreg – conţinutul unui registru segment
acc – conţinutul acumulatorului
mem – conţinutul unei locaţii de memorie pe unul sau mai mulţi octeţi, adresată cu unul dintre modurile de adresare permise pentru memoria de date, cu excepţia adresării imediate
mem8 – conţinutul unei locaţii de memorie pe un octet
[reg] – conţinutul unei locaţii de memorie a cărei adresă se află într-un registru
[mem] – conţinutul unei locaţii de memorie a cărei adresă se află într-o altă locaţie de memorie

port – adresa, numărul de ordine al unui port de intrare sau de ieşire
imed – operandul este o valoare constantă, ce se adresează în mod imediat
(X) – conţinutul locaţiei X
((X)) – conţinutul locaţiei de memorie adresate de X

6.1 Instrucţiuni de transfer

Aceste instrucţiuni permit copierea/transferul datelor (pe octet sau pe cuvânt) de la o sursă la o destinaţie, sursa fiind întotdeauna operandul al doilea.

Sursa	**Destinaţia**
reg	reg
mem	mem
imed	-
port (intrare)	port (ieşire)

Instrucţiunea MOV (*Data movement* - Transfer de date)

MOV — este instrucţiunea cel mai des folosită.

MOV destinaţie, sursă; (destinaţie) ← sursă

Operanzi: reg,imed
mem,imed
reg,reg
reg,mem
mem,reg

Observaţii:

- Operanzii trebuie să aibă dimensiune egală;
- Este interzis ca ambii operanzi să fie locaţii de memorie;
- Nu sunt folosiţi regiştrii IP şi FLAGS;
- Registrul CS nu poate fi destinaţie;
- Nu afectează nici un flag.

Exemplu:

```
MOV   AL,    BH        ;AL=BH – transfer pe octet
MOV   AX,   [ADR]      ;AX = (DS:ADR), DS implicit
                       ;transfer pe cuvant
MOV   AX,   ES:[BX]    ;AX = (ES:(BX))
```

Cuvântul pe 16 biţi aflat în memorie în segmentul suplimentar, indicat prin registru segment ES la un offset dat de conţinutul lui BX faţă de începutul acestui segment este transferat în registrul AX.

```
MOV   byte ptr [BX+100],  15  ;in memorie, la octetul de
                              la locatia data de segment DS,
                              offset BX+100 se depune valoarea 15
```

Operatorul **ptr** permite modificarea atributului unei valori. Fără utilizarea acestuia, ultima instrucţiune ar putea fi interpretată în două moduri: se depune valoarea 15 la octetul de la adresa DS:BX+100 sau la cuvântul de la adresa DS:BX+100. Forma *byte ptr* precizează că este vorba de un transfer pe octet.

Instrucţiuni incorecte:

```
MOV   AX,BH      ;lungime operanzi diferită
MOV   [BX],[SI]  ;ambii operanzi în memorie
MOV   CS,BX      ;reg CS apare la destinaţie
```

Instrucţiuni specifice stivei

Stiva reprezintă un concept abstract de structură de date, o listă de tip LIFO- "Last in first out" şi se găseşte în segmentul SS. Adresa curentă, numită şi adresa vârfului stivei, se găseşte în registrul intern SP (Stack Pointer).

Instrucţiunile PUSH (Push Value onto Stack) şi **POP (Pop Value off Stack)** sunt folosite pentru salvarea unor date şi respectiv refacerea lor în ordine inversă.

PUSH sursă	*POP destinaţie*
SP ← SP – 2	*SS : ((SP) + 1) → high (destinaţie)*
SS : ((SP) + 1) ← high (sursă)	*SS : ((SP)) → low (destinaţie)*
SS : ((SP)) ← low (sursă)	*SP ← SP + 2*

Operanzi:

imed, reg16, sreg, mem sreg, reg16, mem

În urma utilizării instrucţiunii PUSH registrul SP este decrementat cu 2, după care operandul sursă este salvat în octeţii de la adresele (SP)+1→octetul high, respectiv (SP)→octetul low.

Instrucţiunea POP copiază conţinutul vârfului stivei, adică octeţii de la adresele (SP)+1 şi (SP) în operandul destinaţie, mecanism urmat de incrementarea lui SP cu 2. În urma unei secvenţe de refacere, indicatorul SP trebuie adus la valoarea sa de dinaintea secvenţei de salvare (numărul operaţiilor POP coincide cu cel al instrucţiunilor PUSH).

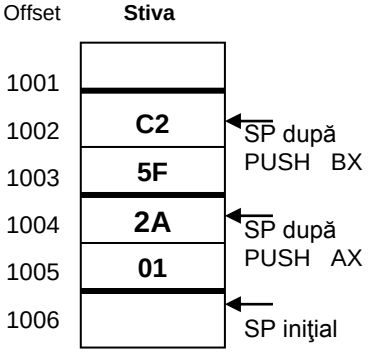
Observaţii:

- Registrul CS nu poate apărea ca destinaţie;
- Sursa şi destinaţia sunt operanzi pe 16 biţi

Exemplu:

```
SP=1006h; AX=012Ah; BX=5FC2h;

        PUSH   AX;
        PUSH   BX;
        POP    BX;
        POP    AX;
```



Instrucţiuni incorecte:

```
        PUSH    AH; AH are doar 8 biţi, ar altera stiva
        POP     CS;
```

Transferuri bilaterale XCHG (Exchange)

Instrucţiunea **XCHG** constă în interschimbarea conţinutului celor doi operanzi.

XCHG destinaţie, sursă

Operanzi: reg,reg
reg,mem
mem,reg

Observaţie:

- Regiştrii segment nu pot apărea ca şi operanzi
- Operanzii trebuie sa aibă dimensiune egală
- Cel puţin un operand trebuie să fie registru

Exemplu:

Dacă ambii operanzi sunt aflaţi în memorie la locaţiile `mem1` şi `mem2` şi trebuie interschimbaţi, se va folosi următoarea secvenţă:

```
MOV      reg,  mem1
XCHG     reg,  mem2
MOV      mem1,  reg
```

Instrucţiunea XLAT

Nu are operanzi. Se încarcă în AL conţinutul locaţiei din segmentul de date de la offset-ul lui BX adunat cu conţinutul lui AL. Este utilă în conversia unor tipuri de date şi se foloseşte împreună cu tabele de translatare.

AL ← (DS : (BX + AL))

Exemplu:

```
Var db 'ABCDEF'; Var este un sir de octeti
               ;initializati cu codurile Ascii ale
               ; literelor A...F
LEA  BX, Var   ;incarca in BX adresa efectiva a lui Var
MOV  AL, 2     ; AL=2
XLAT           ; AL=43h, adica codul Ascii al lui 'C'
```

Instrucţiuni de transfer cu porturile: IN (Input Byte/ Word)
OUT (Output to Port)

Oferă posibilitatea de a schimba informaţii cu perifericele prin intermediul porturilor de intrare/ieşire.

IN destinaţie, port *OUT port, sursă*

Transferurile sunt făcute din sau în acumulator, pe cuvânt sau pe octet (AX, AL). Adresa portului poate fi specificată explicit pentru primele 256 de porturi (00h—FFh) sau prin intermediul registrului DX.

Exemplu:

```
IN    AL,70H    ;AL = (port 70H)
MOV   DX,3ECH
OUT   DX,AX     ;(port 3ECH) =AX
```

Instrucţiuni de transfer specifice adreselor

Instrucţiunea **LEA (Load Effective Address)**

LEA destinaţie, sursă;

Operanzi: reg, mem

Permite copierea adresei efective a sursei (operand aflat în memorie) în registrul general specificat. Operaţia se face în faza de execuţie.

Exemplu:

```
LEA   BX,VAL1          ;BX = offset  VAL1
LEA   DI,[AX][CX]      ;se adună conţinuturile lui AX şi
                       ;CX şi rezultatul se depune în DI
```

În mod asemănător se poate obţine adresa efectivă folosind operatorul **OFFSET** şi instrucţiunea MOV. Atribuirea se face la asamblare.

```
MOV   BX,OFFSET VAL1
```

Instrucţiunile **LDS (Load Data Segment)** şi **LES (Load Extra Segment)**

LDS registru, sursă *LES registru, sursă*

Exemplu:

```
x db 2468h          ;în segmentul de date se definesc
y dw 1357h          ;variabilele

lds si, x         ; încarcă SI=2468h şi DS=1357h
```

Aceste instrucţiuni transferă o adresă completă într-o pereche de regiştri. Perechea de regiştri DS:registru / ES:registru este încărcată cu adresa completă de 32 de biţi conţinută în sursă, definită ca operand double-word în memorie.

Instrucţiuni specifice flagurilor

Instrucţiunile **LAHF (Load AH with Flags)** şi **SAHF (Store AH into Flags)**

$(AH) \leftarrow (PSW)_L$ $(PSW)_L \leftarrow (AH)$

Nu au operanzi. Se încarcă registrul AH cu octetul low al registrului PSW, respectiv se depune conţinutul registrului AH în octetul low al PSW.

Exemplu:

```
LAHF
SHL     AH,7      ;deplasare logică stânga cu 7 poziţii
AND     AH,80H    ;AH conţine flagul CF
```

Instrucţiunile **PUSHF (Push Flags)** şi **POPF (Pop Flags)**

SP ← SP – 2 *PSW ← SS : ((SP) + 1 : (SP))*
SS : ((SP) + 1: (SP)) ← PSW *SP ← SP + 2*

Nu au operanzi. Efectul lor este salvarea registrului PSW pe stivă, respectiv refacerea acestuia de pe stivă.

Exemplu:

```
PUSHF           ;încarcă stiva cu valorile Flags
POP AX          ;sunt luate de pe stivă şi depuse în AX
OR AX,01        ;se setează Carry,
PUSH AX         ;se depune în stivă cu Carry modificat
POPF            ;se ia din stivă înapoi în Flags, cu CF=1
```

6.2 Instrucţiuni aritmetice şi logice

Instrucţiuni specifice adunării

Sunt instrucţiuni cu doi operanzi iar rezultatul se depune în primul operand. Se modifică flag-urile C, S, Z, P, O, A, de unde şi denumirea de flaguri aritmetice. Semnificaţia tuturor flagurilor a fost prezentată în lucrarea anterioară.

Instrucţiunea **ADD (Integer Addition)**

ADD destinaţie, sursă; (destinaţie) = (destinaţie) + (sursă)

Operanzi: reg,imed
mem,imed
reg,reg
reg,mem
mem,reg

Se adună conţinutul sursei la destinaţie şi rezultatul se depune în operandul destinaţie, vechea valoare pierzându-se. Flag-urile S, Z, P se modifică conform rezultatului operaţiei.

Observaţii:

- Operanzii trebuie să aibă dimensiuni egale;
- Este interzis ca ambii operanzi să fie locaţii de memorie.

Exemplu:

```
ADD AX,CX;
ADD byte ptr [DI],4    ;destinaţia în memorie pe octet,
                       ;sursa imediată
```

Exemplu:

```
MOV AX, 8FFEh      ; AX=8FFEh
MOV BX, 0C001h     ; BX=0C001h
ADD AX,BX          ; AX=4FFFh şi CF=1
```

Instrucţiunea **ADC (Add with Carry)** este asemănătoare cu ADD, singura diferenţă este că se adună şi bitul de transport. Este utilă în cazul execuţiei unor adunări pe lungimi mai mari decât cuvântul. Toate flagurile aritmetice sunt afectate.

(destinaţie) = (destinaţie) + (sursă) + C

Exemplu:

```
MOV AX, 8FFEh      ; AX=8FFEh
MOV BX, 0C001h     ; BX=0C001h
ADC AX,BX          ; AX=5000h şi CF=0
```

Instrucţiunea **INC (Increment)** are ca efect incrementarea cu valoarea 1 a destinaţiei. Se modifică toate flag-urile aritmetice, mai puţin Carry.

INC destinaţie; (destinaţie) = (destinaţie) + 1

Exemplu:

```
mov AX, 0FFFFh ; AX=0FFFFh
inc AX         ; AX=0000h, iar CF=0 şi ZF=1
inc byte [7]   ; se incrementează octetul din
               ; memorie de la adresa DS:7
```

Instrucţiunile **DAA (Decimal Adjust for Addition)** şi **AAA (Ascii Adjust for Addition)**

dacă ($AL_{0,3}$) > 9 sau A = 1
  atunci (AL) ← (AL) + 6
    A ← 1,
  altfel A ← 0

dacă ($AL_{4,7}$) > 9 sau C = 1
  atunci (AL) ← (AL) + 60h
    C ← 1,
  altfel C ← 0

dacă ($AL_{0,3}$) > 9 sau A = 1
  atunci (AL) ← (AL) + 6
    (AH) ← (AH) + 1
    A ← 1
    C ← 1
    (AL) ← (AL) AND 0Fh
  altfel A ← 0, C ← 0

Nu au operanzi. Se execută corecţia zecimală a acumulatorului AL, respectiv AX după o operaţie de adunare cu operanzi BCD împachetaţi (2 cifre pe un octet), respectiv despachetaţi (o cifră pe octet).

Instrucţiuni specifice scăderii

Instrucţiunea **SUB (Subtraction)** poate fi interpretată ca şi o adunare a destinaţiei cu complementul faţă de 2 al sursei. Se inversează rolul bistabilului Carry. Toate flagurile aritmetice sunt afectate.

SUB destinaţie, sursă; (destinaţie) = (destinaţie) – (sursă)

Exemplu:
```
MOV AX,4000h   ; AX= 4000h
MOV BX, 0B000h ; BX=0B000h
SUB AX, BX     ; AX= 9000h şi CF=1
```

În cazul instrucţiunii ***SBB*** **(Subtract with Borrow)** se ţine cont de un împrumut anterior. Se foloseşte la scăderi de operanzi pe mai multe cuvinte.

SBB destinaţie, sursă; (destinaţie) = (destinaţie) – (sursă) - C

Exemplu:
```
MOV AX,4000h   ; AX= 4000h
MOV BX, 0B000h ; BX=0B000h
SUB AX, BX     ; AX= 9000h şi CF=1
SBB AX,1       ; AX=8FFEh şi CF=0
```

Instrucţiunea **DEC** are ca efect decrementarea cu valoarea 1 a destinaţiei. Se modifică toate flag-urile aritmetice mai putin Carry.

DEC destinaţie; (destinaţie) = (destinaţie) – 1

Exemplu:
```
MOV AX, 01h    ; AX=0001h
DEC AX         ; AX=0000h şi CF=0, iar ZF=1
```

```
MOV AX, 00h     ; AX= 0000h
DEC AX          ;AX=0FFFFh şi CF=0 (instrucţ. echivalentă
                ;SUB AX,1 ar fi setat CF), OF=0, iar ZF=0
```

Instrucţiunea **NEG** realizează complementul faţă de 2 al destinaţiei. Toate flagurile aritmetice sunt afectate.

NEG destinaţie; (destinaţie) = 0 – (destinaţie)

Exemplu:
```
MOV AX, 0F0Fh  ;AX=0F0Fh
NEG AX         ;AX=F0F1h şi CF=1 (borrow)
```

Exemplu:
```
;următoarea secvenţă calculeaza modulul unei valori
OR AX,AX       ;se testează semnul
JNS et         ;salt dacă numărul e pozitiv
NEG AX         ;negarea numărului negativ
et:...
```

Semnificaţia instrucţiunii **CMP** este execuţia unei scăderi fictive, fără modificarea operanzilor dar cu poziţionarea tuturor flagurilor aritmetice.

CMP destinaţie, sursă;
dacă destinaţie = sursă → Z=1
destinaţie < sursă → C=1
destinaţie > sursă → C+Z=0 → C=Z=0

Exemplu:
```
MOV AX, 2000h  ;AX=2000h
MOV BX, 400h   ;BX=400h
CMP AX, BX     ;AX=2000h, BX=400h şi OF=0, SF=0, CF=0
```

Instrucţiunile **DAS (Decimal Adjust for Subtraction)** şi **AAS (Ascii Adjust for Subtraction)**

dacă ($AL_{0,3}$) > 9 sau A = 1
  atunci (AL) ← (AL) - 6
         A ← 1,
  altfel A ← 0

dacă ($AL_{4,7}$) > 9 sau C = 1
  atunci (AL) ← (AL) - 60h
         C ← 1 ,
  altfel C ← 0

dacă ($AL_{0,3}$) > 9 sau A = 1
  atunci (AL) ← (AL) - 6
         (AH) ← (AH) - 1
         A ← 1
         C ← 1
         (AL) ← (AL) AND 0Fh
  altfel A ← 0, C ← 0

Nu au operanzi. Se execută corecţia zecimală a acumulatorului AL, respectiv AX după o operaţie de scădere cu operanzi BCD impachetaţi, respectiv despachetaţi.

Instrucţiuni specifice înmulţirii

Operaţiile specifice înmulţirii se fac între acumulator şi un alt operand; rezultatul obţinut este pe 16 sau 32 de biţi.

Instrucţiunea **CBW (Convert Byte to Word)** converteşte octetul la cuvânt, adică bitul de semn din AL se extinde la tot registrul AH. Efectul instrucţiunii este echivalent cu reprezentarea registrului AL în complement faţă de doi pe un număr dublu de biţi. Nu este afectat nici un flag.

dacă $AL_7 = 1 \rightarrow AH = 0FFh$
altfel AH = 0

Instrucţiunea **CWD (Convert Word to Doubleword)** converteşte cuvântul la dublu-cuvânt, adică bitul de semn din AX se extinde la tot registrul DX. Nu este afectat nici un flag.

dacă $AX_{15} = 1 \rightarrow DX = 0FFFFh$
altfel DX= 0

Prin cele două instrucţiuni se face extensia semnului acumulatorului în acumulatorul extins.

Exemplu:

```
MOV AL, 54h     ; AL=54h
CBW             ; AX=0054h
CWD             ; DX:AX=0000 0054h
```

Instrucţiunile **MUL**—înmulţire fără semn şi **IMUL**—înmulţire cu semn

MUL sursă; *IMUL sursă;*
*(acumulator extins) = (acumulator) * (sursă)*

AX = AL * (reg8/mem8)
DX:AX = AX * (reg/mem)

Destinaţia este implicit acumulatorul iar sursa un registru sau o locaţie de memorie. Ambii operanzi sunt fără semn, respectiv cu semn. Înmulţirea NU duce la depăşiri. Sunt afectate flagurile O şi C.

Exemplu:

```
A db 5h
B dw 300h
MOV AL,10h
MUL A           ;( AX)  ← (AL) * A
MOV AX,100h
MUL B           ;( DX:AX) ← (AX ) * B
```

Exemplu:
```
mov AL,-16      ; AL=0F0h
mov BL,2        ; BL=2
mul BL          ;AX=01E0h=480, 0F0h=240 nr fara semn
```

Exemplu:
```
mov AL,-16      ; AL=0F0h
mov BL,2        ; BL=2
imul BL         ;AX=FFE0h=-32, 0F0h=-16 nr cu semn
```

Instrucţiunea **AAM (Ascii Adjust for Multiply)** nu are operanzi şi realizează o corecţie a acumulatorului AX, după o operaţie de înmulţire pe 8 biţi cu operanzi BCD despachetaţi.

(AH) ← (AX) / 10;
(AL) ← (AX) MOD 10;

Exemplu:
```
MOV AL,8
MOV BL,7
MUL BL           ;( AX ) = 38h
AAM              ;AX = 0506h
```

Instrucţiuni specifice împărţirii

Instrucţiunile **DIV şi IDIV**

DIV sursă;
(acumulator) = (acumulator extins) / (sursă)
(extensia acumulatorului) = (acumulator extins) MOD (sursă)

AL = AX / (reg8/mem8)
AH = AX MOD (reg/mem8)

AX = (DX:AX) / (reg/mem)
DX = (DX:AX) MOD (reg/mem)

Destinaţia este implicit acumulatorul sau acumulatorul extins iar sursa un registru general sau o locaţie de memorie. Ambii operanzi sunt fără semn, respectiv cu semn. Împărţirea presupune că lungimea deîmpărţitului este dublă faţă de cea a împărţitorului. Poate duce la depăşiri când împărţitorul este zero sau când câtul depăşeşte dimensiunea rezervată rezultatului. În urma unei depăşiri se iniţiază o întrerupere de nivel 0. Flagurile nu sunt afectate.

Exemplu:
```
A db 5
mov AX,51h      ; AX=51h=5h*10h+01h=81_10=5_10*16_10+1_10
div A           ; AL=10h (câtul), iar AH=01h (restul)
```

Exemplu:

```
B dw 300h
mov AX, 14h     ;AX=0014h =20₁₀
mov DX, 03h     ;DX=0003h = 3₁₀
div B           ;DX:AX=3₁₀*16⁴+20₁₀=196628₁₀ se împarte
                ;la 300h=768₁₀ => AX=0100h=256₁₀(cât)
                ;şi DX=0014h=20₁₀ (rest)
```

Instrucţiunea **AAD (ASCII Adjust for Division)** nu are operanzi şi realizează o corecţie a acumulatorului AX, interpretat ca două cifre BCD despachetate (o cifră pe octet). Efectul este următorul:

AL = AH * 10 + AL
AH = 0

6.3 Exerciţii şi teme

6.3.1. Analizaţi secventele de mai jos şi apoi executaţi-le manual pentru a completa valorile în tabelele alăturate. Apoi introduceţi-le pe calculator şi executaţi-le cu emulatorul *emu8086* sau cu Turbo Debugger-ul pentru a verifica corectitudinea rezultatelor.

Exemplu:

```
MOV     AX, BBBBH       ;AX=BBBBH
ADD     AX, 4445H       ;AX=0000H

 BBBB    1011 1011 1011 1011
+4445    0100 0100 0100 0101
-----------------------------
 0000    0000 0000 0000 0000
```

AX	0000H
Sign flag	0
Carry flag	1
Parity flag	1
Zero flag	1

a.
```
mov al,64h
mov bl,0A6h
add al,bl
```

AL	
Sign flag	
Carry flag	
Overflow flag	
Zero flag	

b.
```
mov al,38h
mov bl,62h
sub al,bl
```

AL	
Sign flag	
Carry flag	
Overflow flag	
Zero flag	

c.
```
mov al,57h
mov bl,39h
sub al,bl
```

AL	
Sign flag	
Carry flag	
Overflow flag	
Zero flag	

d.
```
mov ax,2B54h
mov bx,100h
mov [bx],ax
```

AX	
[DS:100h]	
[DS:101h]	
Overflow flag	
Zero flag	

e.
```
mov ax,2B54h
mov bx,100h
mov 20h[bx],ax
```

AX	
[100h]	
[120h]	
[121h]	
Zero flag	

f.
```
mov al,'a'
mov ah,'A'
```

AL	
AH	
Carry flag	
Overflow flag	
Zero flag	

g.
```
mov bx,2863h
mov sp,0102h
push bx
```

BX	
SP	
[SS:101h]	
[SS:100h]	
Zero flag	

h.
```
mov ax,2B54h
mov bx,100h
mov dx, 1234h
mov [bx+20h], dx
mov ax,[bx+20h]
```

AX	
[100h]	
[120h]	
[121h]	
Zero flag	

6.3.2. Adunări şi scăderi

- Să se evalueze expresia *r=(x+y)-(z+15)-(t-10)*. Variabilele sunt reprezentate pe 16 biţi cu semn.

```
Date:
        x dw 100
        y dw 500
        z dw 1000
        t dw -200
        r dw ?

Cod:
        mov ax, x
        add ax, y               ;ax = x+y
        mov bx, z
        add bx, 15              ;bx = z+15
        sub ax, bx              ;ax =(x+y) - (z+15)
        mov bx, t
        sub bx, 10              ;bx = t-10
        sub ax, bx              ;ax = =(x+y)-(z+15)-(t-10)
        mov r, ax
```

6.3.3. Inmulţiri şi împărţiri

- Să se evalueze *r=(x-y*z)/t*, unde *x, y, z, t* şi *r* vor fi reprezentate pe 16 biţi cu semn.

```
Date:
        x dw 2000
        y dw -500
        z dw 200
        t dw 300
        r dw ?
```

```
Cod:
        mov ax, y
        imul z                  ;dx:ax = y*z
        mov cx, dx
        mov bx, ax              ;cx:bx = y*z , avem nevoie de dx:ax
                                ;pt. conversia lui x
        mov ax, x
        cwd                     ; dx:ax = a
        sub ax, bx
        sbb dx, cx              ;dx:ax = x-y*z
        idiv t                  ;ax = (x-y*z)/t
        mov r, ax
```

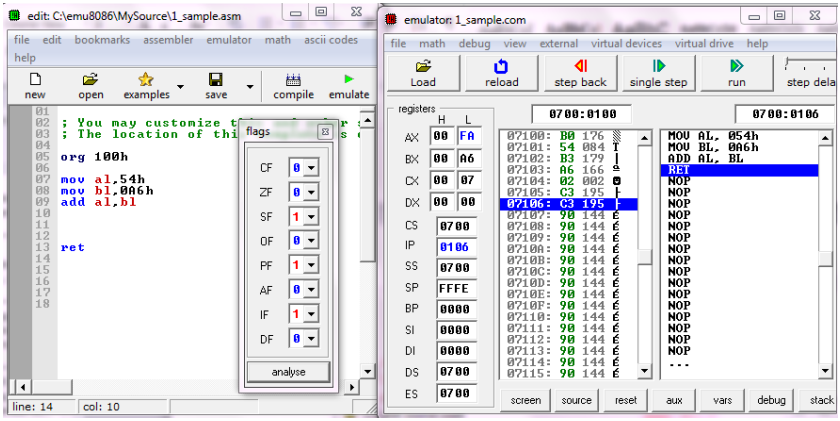


Figura 6.1. Verificarea corectitudinii rezultatelor cu EMU8086

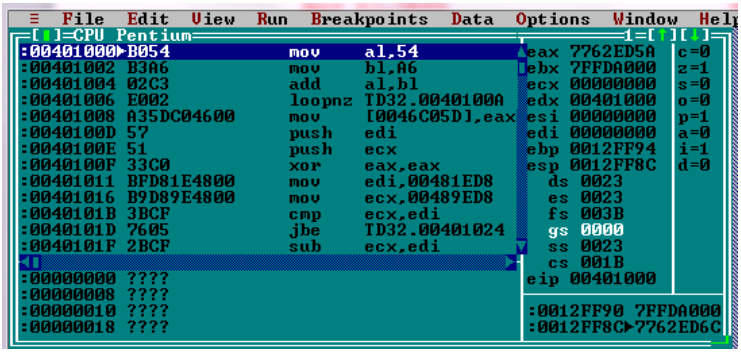


Figura 6.2. Verificarea corectitudinii rezultatelor cu TD32