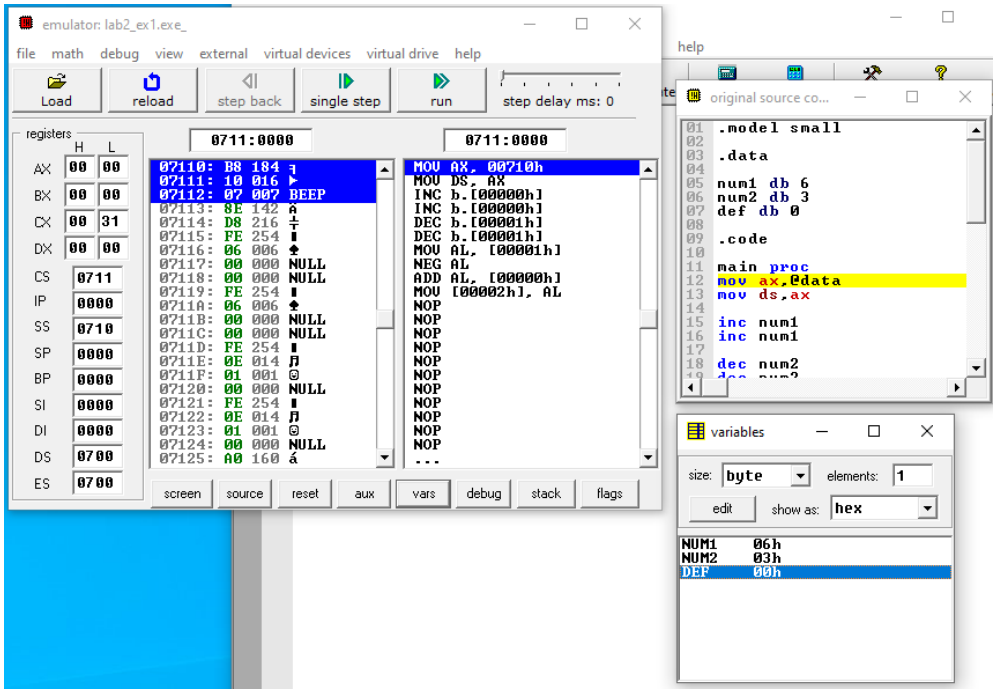
TEMA 1

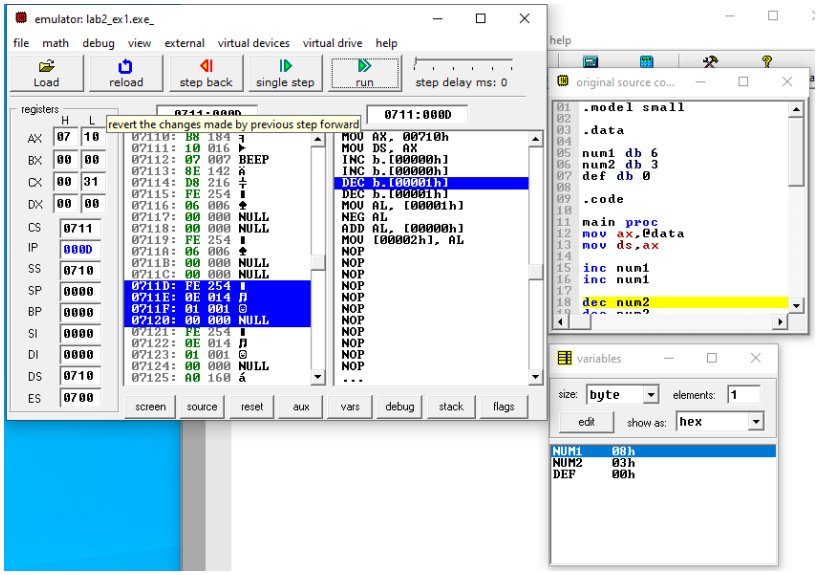
PMD Ex1

```
.model small

.data

num1 db 6
num2 db 3
def db 0

.code

main proc
    mov ax,@data
    mov ds,ax

    inc num1
    inc num1

    dec num2
    dec num2

    mov al,num2
    neg al
    add al, num1
    mov def,al

main endp
end
```

Aceste este programul propriu-zis, am declarat ca variabile num1 cu valoarea 6, num2 cu valoarea 3 si def cu valoarea 0, cea din urma variabila fiind pentru stocarea rezultatului final.

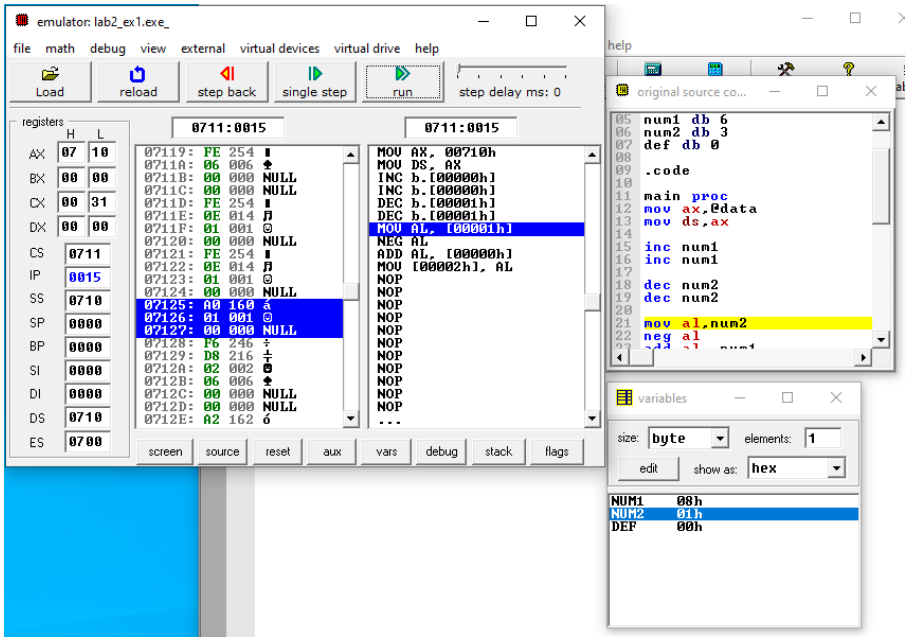


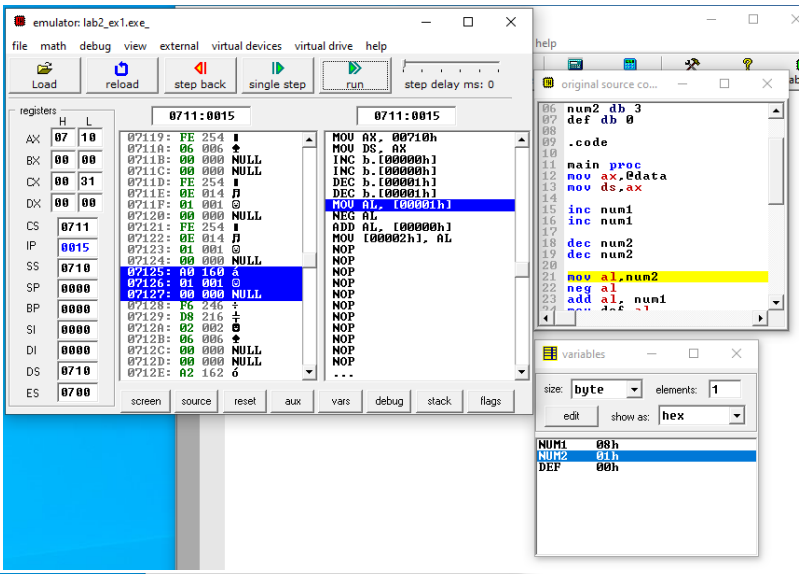
In aceasta secventa de rulare se incarca datele in registrul acumulator AX.



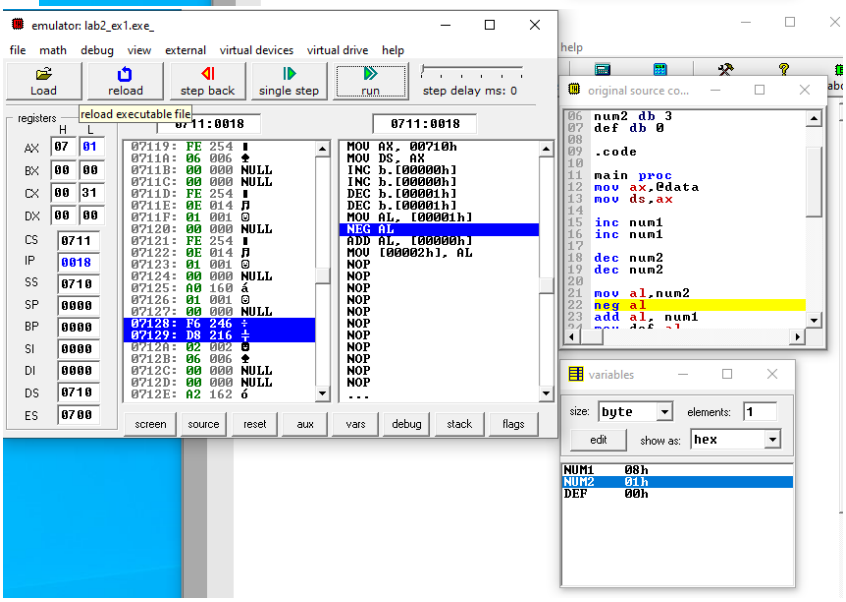
In aceasta secventa s-a facut de doua ori incrementarea variabilei num1, astfel ea are acum valoarea 8.

Aici s-a facut decrementarea variabilei num2 de 2 ori, astfel ea are acum valoarea 1.



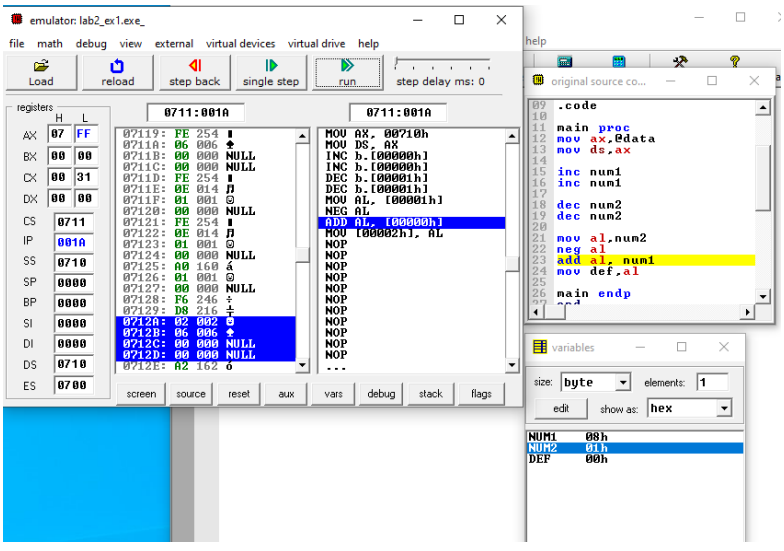


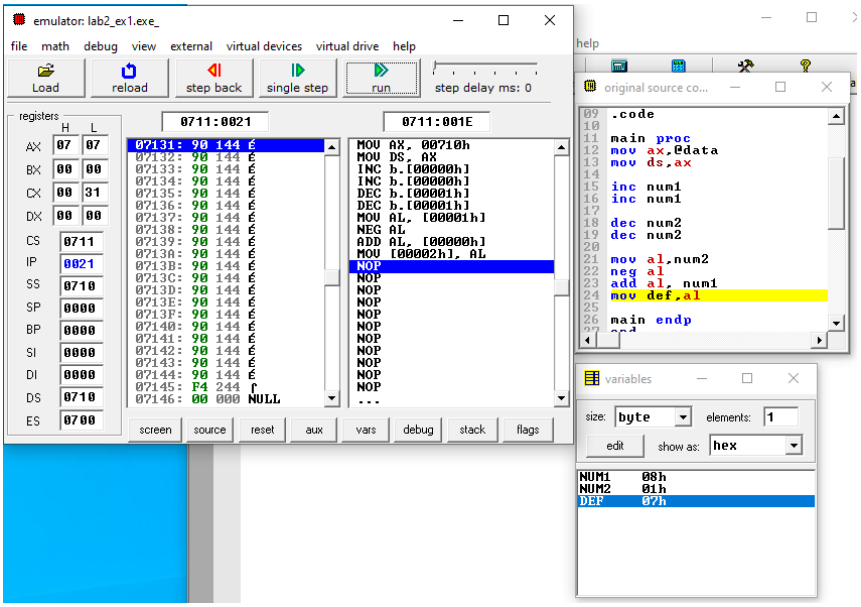
Mutam num2 in registrul AL (A low) pentru convertirea lui in numar negativ.



Negam registrul AL.

Adunam la registrul AL si variabila num1.





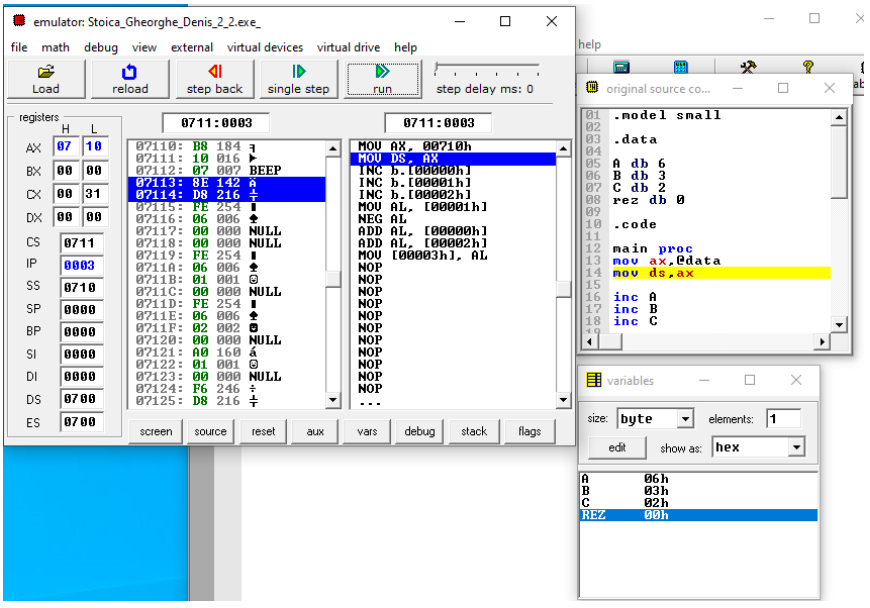
Si in final mutam din valoarea din registrul AL in variabila def si putem vedea in vars ca valoarea este 7 pentru respectivele valori date pentru num1 si num2.

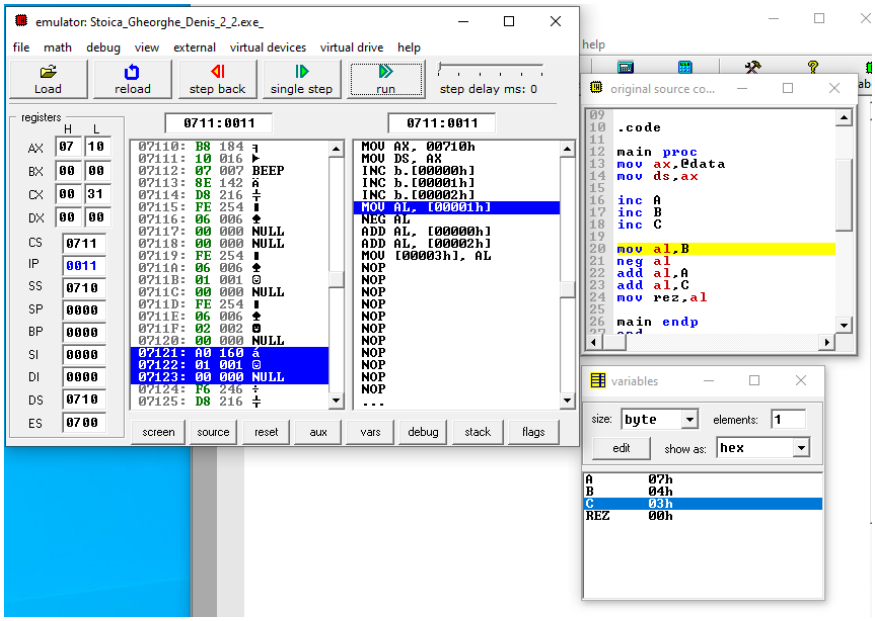
PMD Ex2

```
.model small

.data

A db 6
B db 3
C db 2
rez db 0

.code

main proc
    mov ax,@data
    mov ds,ax

    inc A
    inc B
    inc C

    mov al,B
    neg al
    add al,A
    add al,C
    mov rez,al

main endp
end
```

Aceste este programul propriu-zis, am declarat ca variabile A cu valoarea 6, B cu valoarea 3, C cu valoarea 2 si rez cu valoarea 0, cea din urma variabila fiind pentru stocarea rezultatului final.

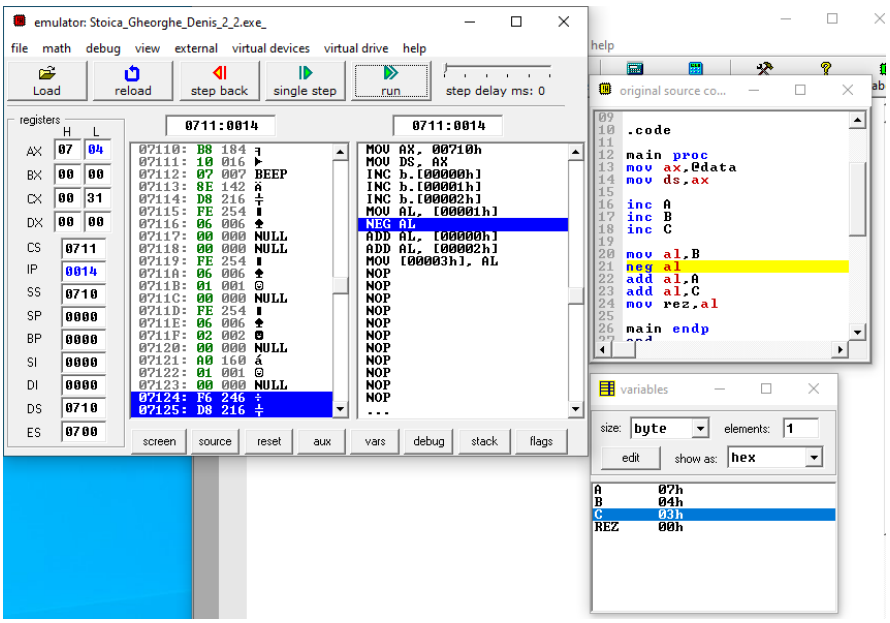
In aceasta secventa de rulare se incarca datele in registrul acumulator AX.

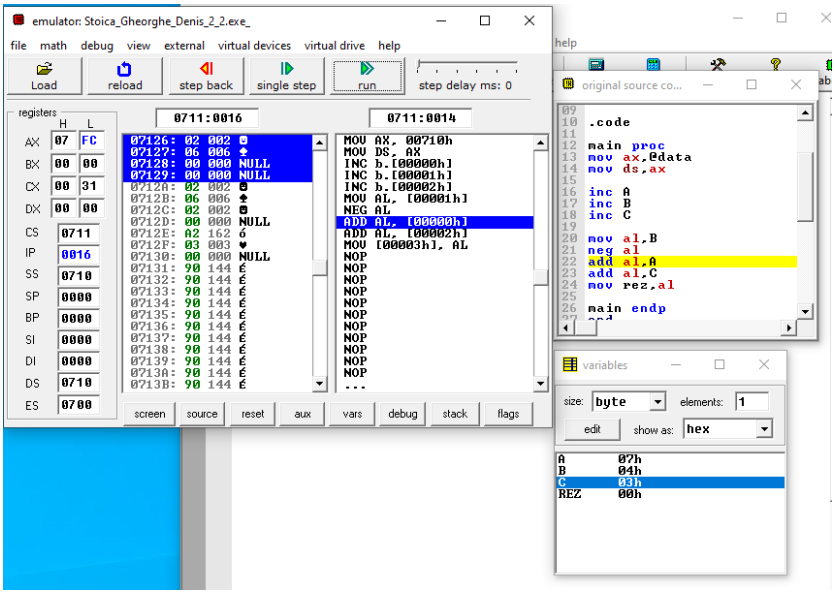




In aceasta secventa se poate observa ca s-au facut 3 incrementari la A, B si respectiv la C, unde variabilei A i se modifica valoarea in 7, variabilei B in 4, iar variabilei C in 3.

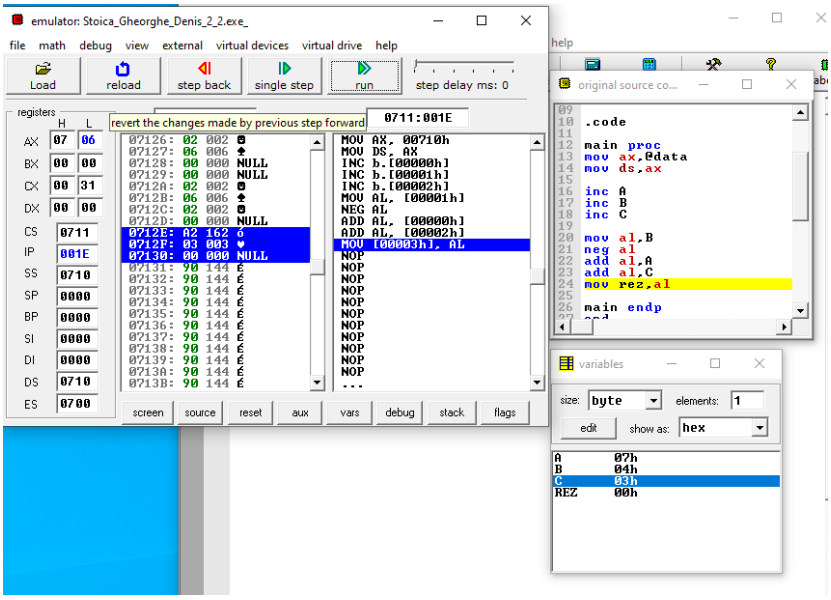
Aici mutam variabila B in registrul AL pentru ca ulterior sa il negam pentru indeplinirea cerintei.

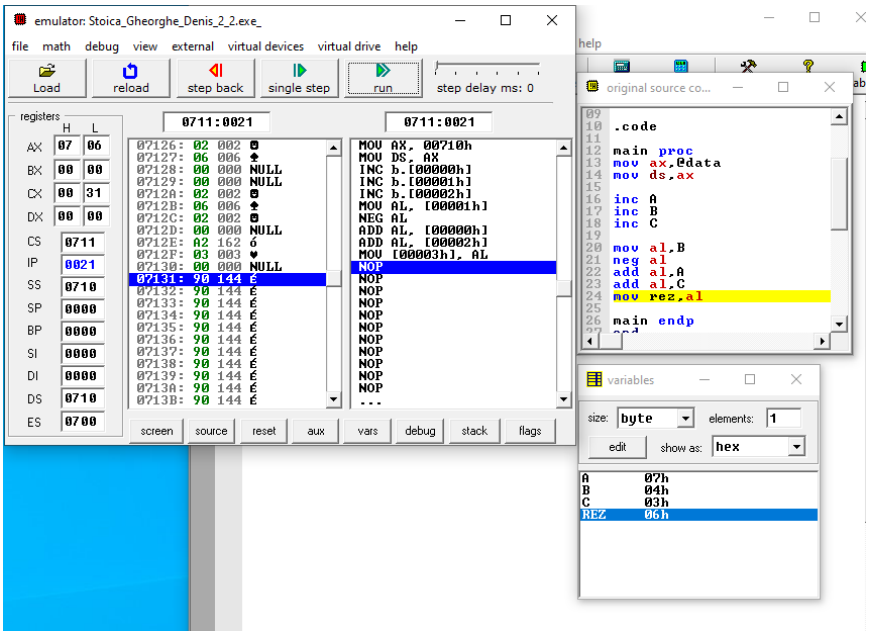




Secventa aceasta de rulare a negat registrul AL, astfel acesta are valoarea -4.

In penultima imagine, se observa ca s-au executat 2 single step-uri astfel ca s-a adunat in registrul AL variabila A si variabila C.





In final, la executarea ultimului single step se muta valoarea din registrul AL in variabila rez, si dupa cum se observa in fereastra vars valoarea lui rez este 6 fiind cea corecta.

TEMA 2

```
.model small

.data
;initiem variabilele
A db 6
B db 3
C db 2

.code

main proc
mov AX,@data   ;incarcam registrul cu datele
mov ds,AX     ;incrementam cu +1 fiecare variabila
inc A
inc B
inc C

mov AL,B   ;mutam variabila B in registrul AL
neg AL     ;Negam registrul AL, implicit valoarea lui adica -4
add AL,A   ;Adunam la registrul AL, var A si C
add AL,C

mov AH,2   ;Setam in AH valoarea 2 corespondenta pentru afisare
mov DL,AL  ;Mutam valoarea din AL in registrul DL raspunzator pentru afisarea variabilelor
INT 21H    ;Initiem intreruperea pentru afisare

main endp
end
```

Temă Lab 1 PMS

1. Registre generale: → AX - acumulator
→ BX - de bază
→ CX - counter
→ DX - de date

2. AX - 16 biți
CL - 8 biți
BH - 8 biți
AL - 8 biți
DX - 16 biți

3. Δ dix) 1

4. X db ?

5. Nu se poate aduna direct, însă putem muta var. X în reg AL, apoi să adunăm reg AL cu var. B astfel va func. adunarea.

6. CS - Code Segment, conține adresa de început a segmentului de cod unde se află codurile instrucțiunilor

7. SS - Stack Segment

8. ADD AL, BH;
MOV AL, A; → Nu poz. ind. de condiție
SUB AL, Y;
→ Poziționează indicatori de condiție

9. Codul tastei citite se va stoca în AL și este un registru pe 8 biți.

10. Data ce se dorește a fi afișată se stochează în registrul DL, fiind un registru pe 8 biți.

11. Acronimul pentru numele registrului care conține adresa instrucțiunii următoare celei care este ~~execută~~ de către microprocesor este ~~CS (Code segment)~~.
IP (Instruction pointer).

1. Decodificator de memorie, - începând cu 00000H, C1 capacitate 64k
- începând cu adresa 40000H, C2, capacitate 32k
- începând cu 70000H, C3, capacitate 32k

1) Determinarea adreselor de început și sfârșit

$64k = 2^6 * 2^{10} = 2^{16}$ → A_{16} ... → 1 1

$32k = 2^5 * 2^{10} = 2^{15}$ → A_{15} ... → 1 1

C1 adresă început: 00000H; Adresa fin.: 1FFFFH

C2 adr. inc.: 40000H; Adr. fin.: 47FFFH

C3 adr. inc.: 70000H; Adr. fin.: 7FFFFH

2) Harta memoriei:

		A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	.....	A_3	A_2	A_1	A_0
C1	A.i.	0	0	0	0	0	...	0	0	0	0
	A.f.	0	0	0	1	1	...	1	1	1	1
C2	A.i.	0	1	0	0	0	.....	0	0	0	0
	A.f.	0	1	0	0	1	...	1	1	1	1
C3	A.i.	0	1	1	1	0	...	0	0	0	0
	A.f.	0	1	1	1	1	...	1	1	1	1

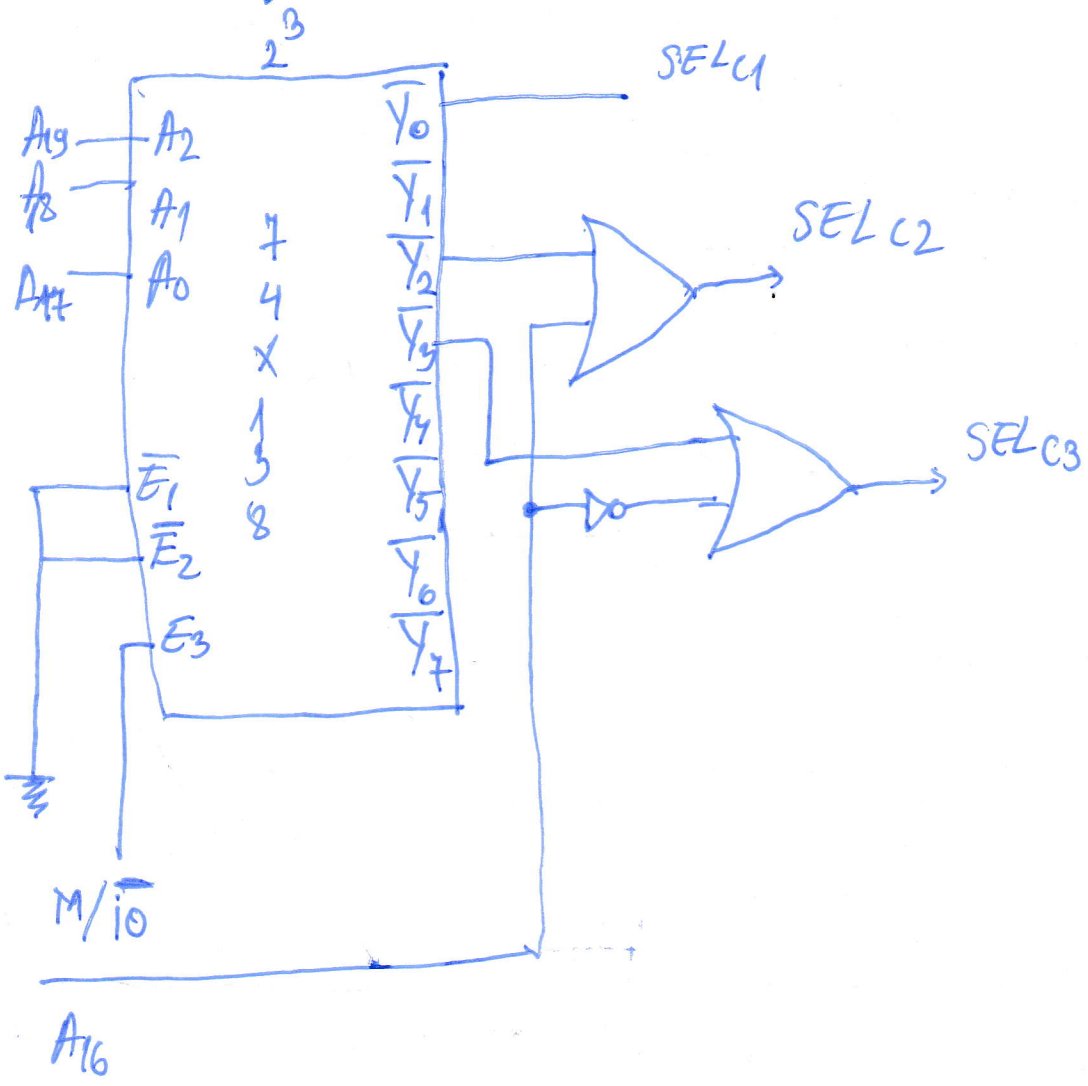
3) Ecuațiile semnalelor de selecție

$$SEL_{C1} = \overline{A_{19}} + \overline{A_{18}} + \overline{A_{17}}$$

$$SEL_{C2} = \overline{A_{19}} + A_{18} + \overline{A_{17}} + \overline{A_{16}}$$

$$SEL_{C3} = \overline{A_{19}} + A_{18} + A_{17} + A_{16}$$

4) Decodificatorul memoriei



② Să se conecteze 64 ko EPROM, 27256 (capacitate 32 ko) 64 ko SRAM, 62256 (capac. 32 ko)

1) Det. adrese început și sfârșit

$64k = 2^6 * 2^{10} = 2^{16} \rightarrow A_{16} \dots \rightarrow 1/1$

A.î. : 00000H ; A.f : 1FFFFH

SRAM → A.î. : 40000H ; A.f : 5FFFFH

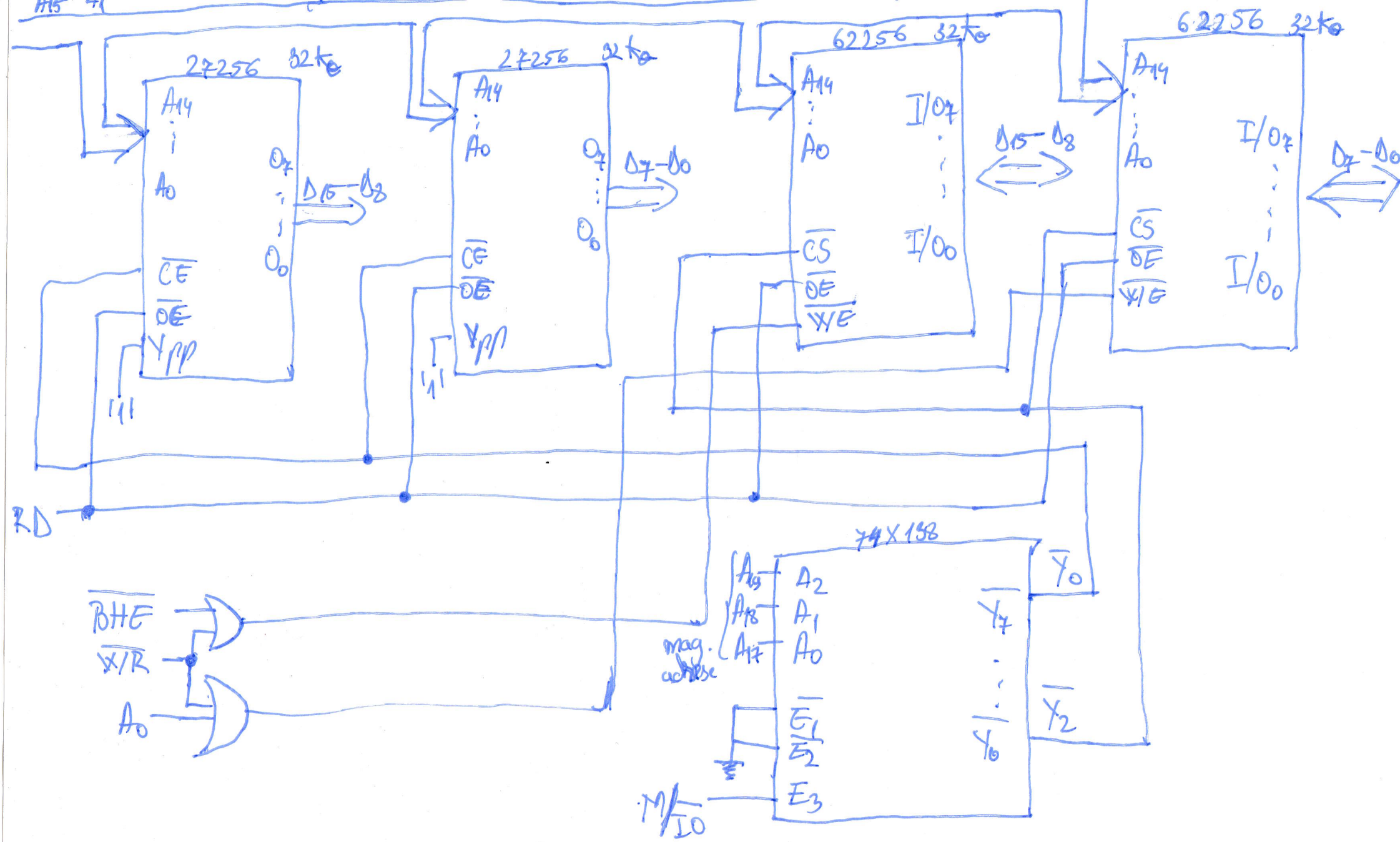
2) Harta memoriei

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	.....	A_4	A_3	A_2	A_1	A_0
A.î.	0	0	0	0	0	---	0	0	0	0	0
A.f	0	0	0	1	1	---	1	1	1	1	1
A.î.	0	1	0	0	0	---	0	0	0	0	0
A.f	0	1	0	1	1	---	1	1	1	1	1

3) Ecuațiile semnalelor de selecție

$SEL_{EPROM} = \overline{A_{19}} + \overline{A_{18}} + \overline{A_{17}}$

$SEL_{SRAM} = \overline{A_{19}} + A_{18} + \overline{A_{17}}$


27256
32 kB
A14
A0
O7
O0
D15–D8
CE
OE
Vpp
'1'
27256
32 kB
A14
A0
O7
O0
D7–D0
CE
OE
Vpp
'1'
62256
32 kB
A14
A0
I/O7
I/O0
D15–D8
CS
OE
WE
62256
32 kB
A14
A0
I/O7
I/O0
D7–D0
CS
OE
WE
RD
74X138
A2
A1
A0
E1
E2
E3
Y0
Y7
Y2
mag. achse
M/IO
BHE
W/R
A0

TEMA 4

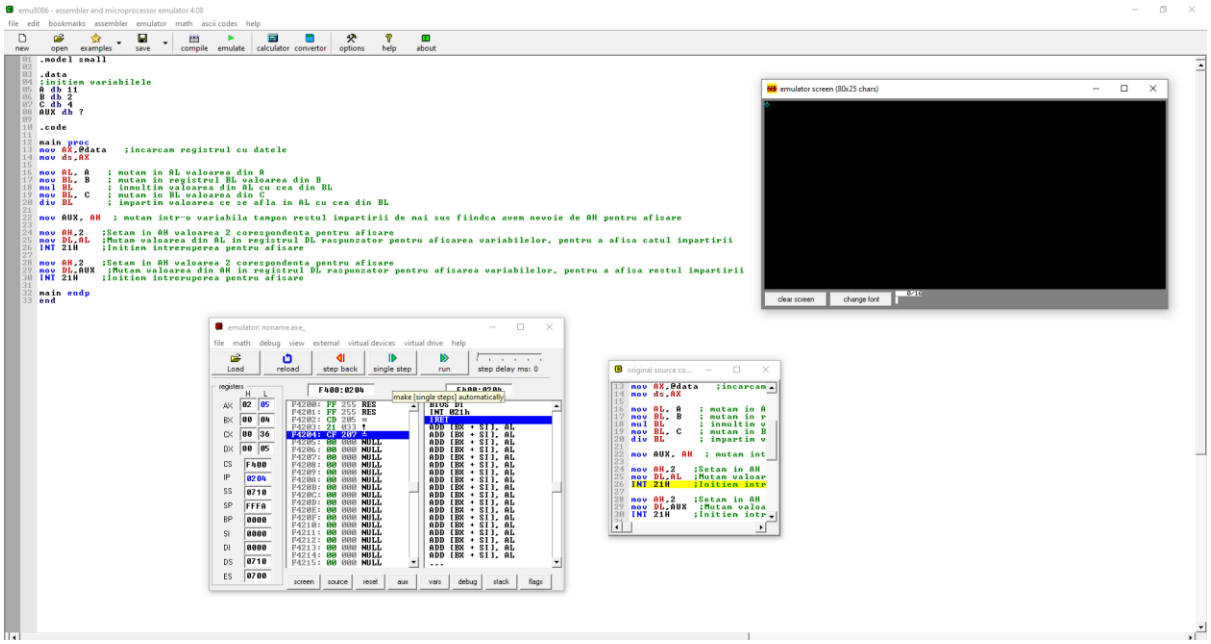
```
.model small

.data
;initiem variabilele
A db 11
B db 2
C db 4
AUX db ?

.code

main proc
mov AX,@data   ;incarcam registrul cu datele
mov ds,AX

mov AL, A   ; mutam in AL valoarea din A
mov BL, B   ; mutam in registrul BL valoarea din B
mul BL      ; inmultim valoarea din AL cu cea din BL
mov BL, C   ; mutam in BL valoarea din C
div BL      ; impartim valoarea ce se afla in AL cu cea din BL

mov AUX, AH  ; mutam intr-o variabila tampon restul impartirii de mai sus fiindca avem nevoie de AH pentru afisare

mov AH,2   ;Setam in AH valoarea 2 corespondenta pentru afisare
mov DL,AL  ;Mutam valoarea din AL in registrul DL raspunzator pentru afisarea variabilelor, pentru a afisa catul impartirii
INT 21H    ;Initiem intreruperea pentru afisare

mov AH,2   ;Setam in AH valoarea 2 corespondenta pentru afisare
```

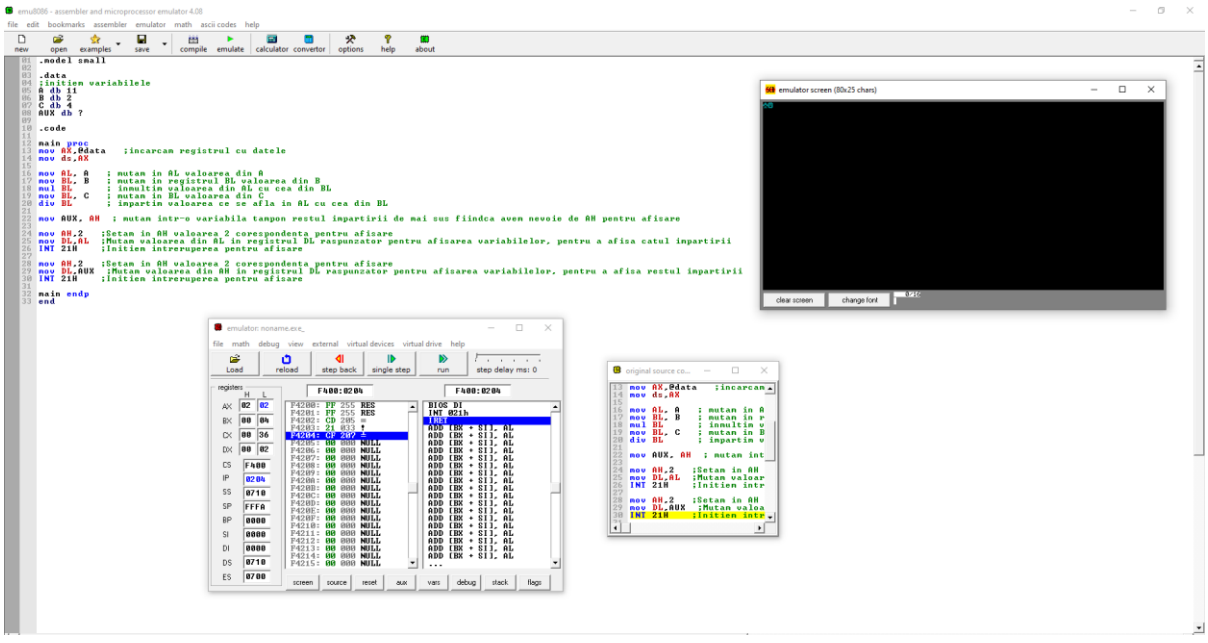
mov DL,AUX ;Mutam valoarea din AH in registrul DL raspunzator pentru afisarea variabilelor, pentru a afisa restul impartirii

INT 21H ;Initiem intreruperea pentru afisare

main endp

end





2.4.2

MOV AL, N //muta valoarea lui N in registrul AL

LBL_WHILE: CMP AL, 0 // Instructiunea while urmata de intructiunea de comparare pentru a se vedea daca AL = 0 sau nu, in caz afirmatie se apeleaza JZ FINAL, care este conditia iesirii din bucla

JZ FINAL ;condiția de ieșire din buclă este N = 0 (AL = 0 în acest caz)

DEC N // decrementam pe N

MOV AL, N // mutam noua valoare a lui N in AL

JMP LBL_WHILE // Salt din nou la while pentru a repeta procedeul

FINAL: NOP ; no operation - instrucțiune dummy, fără effect

2.4.3

MOV CX, N // se muta in registrul contor CX valoarea N pentru a stii cat sa mearga loop-ul

LBL_FOR: MOV AL, X // se executa loop-ul apoi se muta in AL valoarea lui X

SHR AL, 1 ;împărțire la 2 realizată prin deplasare la dreapta cu o poziție

MOV X, AL //se muta in X rezultatul impartirii la 2 din AL

LOOP LBL_FOR // Se reia loopul de la inceput

2.5 Intrebari

1. Pentru a ridica la patrat numarul salvat in AL trebuie apelata instructiunea in forma aceasta:

MUL AL

2. MOV AL, X ; mutam in AL valoarea X

MUL AL ; facem patratul valorii respective

MUL X ; inmultim inca o data cu X valoarea din AL pentru a calcula cubul

3. SHL AL, 3 ; inmultire cu 8 realizati prin deplasare la stanga cu 3 pozitii

4.

Pentru rezultatul -5:

- ZF este 0 deoarece rezultatul nu e 0
- SF este 1 pentru ca bitul cel mai semnificativ e 1

Pentru rezultatul 0:

- ZF este 1 pentru ca rezultatul e 0
- SF este 0 pentru ca bitul cel mai semnificativ e 0

5. Este instructiunea de salt conditionat LOOP care decrementeaza automat pe CX daca si numai daca CX != 0

6. In urma executiei intructiunii daca cele doua valori din registre sunt egale atunci flag-ul ZF va fi pe 1 deoarece rezultatul ultimei intructiuni aritmetice executate de procesor e 0.

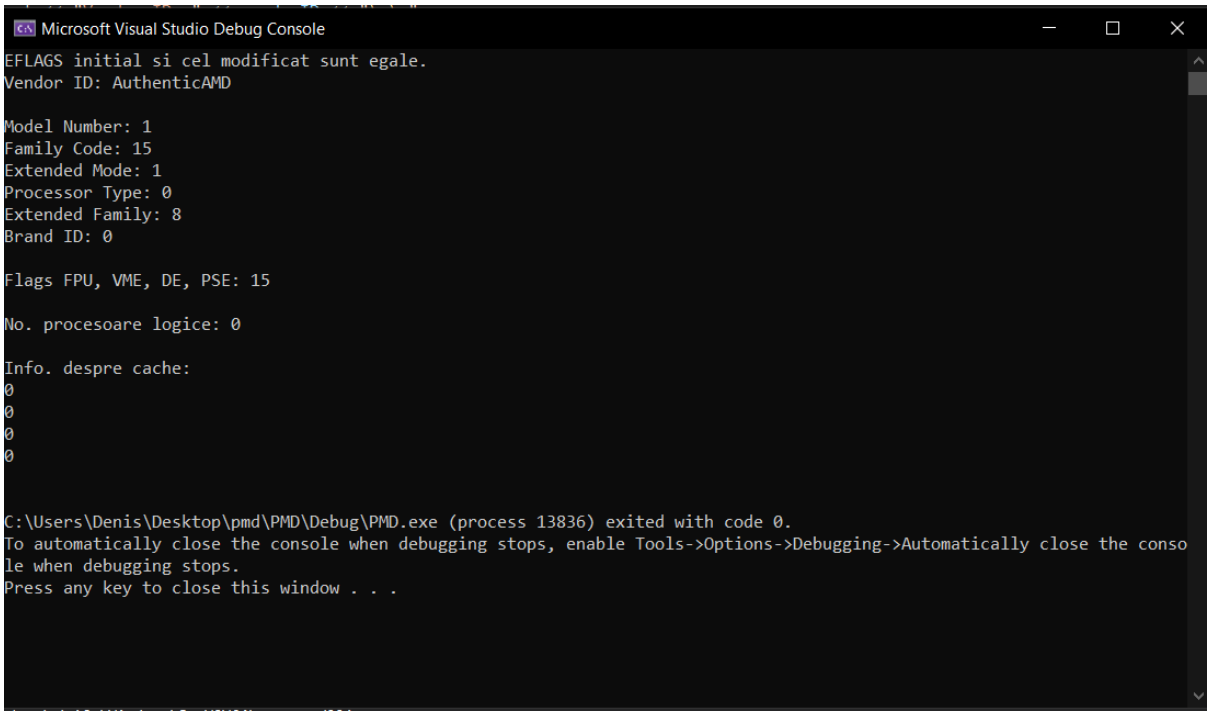
In cazul in care AL < BL rezultatul ultimei instructiuni aritmetici executate de procesor va fi mai mica ca < 0 astfel flag-ul SF va fi pe 1.

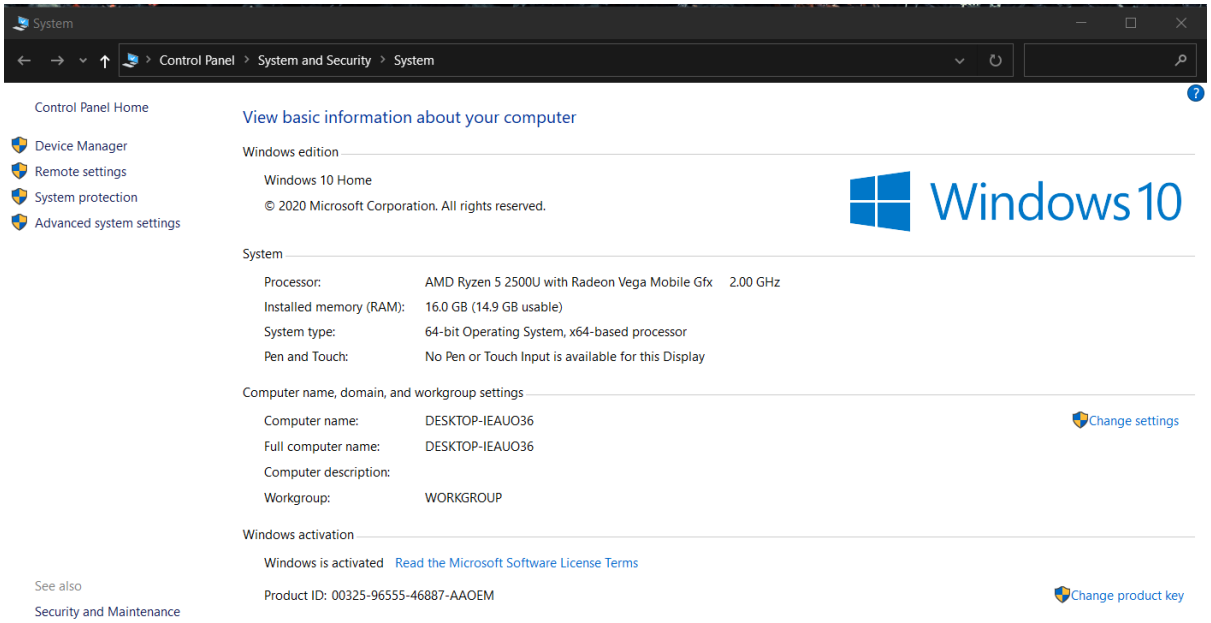
7. Se folosestea instructiunea de salt JZ.

Exemplu: JZ E0

TEMA 6

PRINTSCREEN REZULTAT PROGRAM LUCRARE





RASPUNSURI INTREBARI

1. **Folosind documentația Intel, explicați de ce se aplică următoarele operații pe biți, constând în deplasarea spre dreapta cu 4, 8, 12, 16, respectiv 20 de poziții, a variabilelor din porțiunea de cod de mai jos:**

```
vendorID[12] = '\0';

cout << "Vendor ID: " << vendorID << "\n\n";

 modelNum >>= 4;

FamilyCODE >>= 8;

procTYPE >>= 12;

ExtMODE >>= 16;

extFam >>= 20;

cout << "Model Number: " << modelNum << "\n";

 cout << "Family Code: " << FamilyCODE << "\n";

 cout << "Extended Mode: " << ExtMODE << "\n";

 cout << "Processor Type: " << procTYPE << "\n";

cout << "Extended Family: " << extFam << "\n";
```

Deplasarea spre dreapta cu 4, 8, 12, 16, 20 de pozitii se foloseste pentru a afla informatia din pozitia registrului EAX. Ca exemplu putem da variabila FamilyCODE, care pentru a fi afisata corect trebuie deplasata cu 8 pozitii fiindca se afla de la pozitia 8 pana la pozitia 11 inclusiv, asemanator facandu-se si pentru celelalte variabile.

2. **Explicați care este rolul instrucțiunilor: pushfd si pop eax.**

   Rolul instructiunii PUSHFD este de a apela flags din reg. EFLAGS si de a salva continutul EFLAGS in stiva. Operatiile sunt pe 32 biti.
   Rolul instructiunii POP EAX este de a copia din varful stivei si de a salva informatia in registrul EAX.

3. **Folosind documentația Intel furnizată, scrieți care este registrul procesorului care va conține informațiile Extended Family și Extended Model în urma apelării instrucțiunii CPUID și care sunt pozițiile binare revendicate de fiecare dintre acestea.**

   Registrul procesorului care va contine informatiile Extended Family si Extended Model in urma apelarii instructiunii CPUID este registrul EAX, adica Extended AX, iar

pozitiile binare pentru Extended Family sunt intre 20 si 27, respectiv pentru Extended Model intre 16 si 19.

4. **Folosind documentația Intel furnizată, scrieți care este registrul procesorului care va contine informatiile APIC ID si Count în urma apelării instrucțiunii CPUID și care sunt pozițiile binare revendicate de fiecare dintre acestea.**
   Registrul procesorului care va contine informatiile APIC ID si Count in urma apelarii CPUID este registrul EBX, iar pozitiile binare pentru Count sunt intre 16 si 23, respectiv pentru APIC ID intre 24 si 31.

5. **Folosind documentația Intel furnizată, scrieți care ar trebui să fie continutul binar al registrului EAX in urma apelului instrucțiunii CPUID pentru procesoarele Intel 486 SX.**

   Continutul registrului EAX in urma apelului instructiunii CPUID pentru procesoarele Intel 486 SX sunt:
   Stepping ID: xxxx
   Model: 0010
   Family: 0100
   Processor type: 00
   Extended Model: 0000
   Extended Family: 00000000

6. **Folosind documentația Intel furnizată, scrieți care ar trebui să fie continutul binar al registrului EAX in urma apelului instrucțiunii CPUID pentru procesoarele Intel Pentium Pro, precum și pentru procesoarele Intel Core i7.**

   Continutul registrului EAX in urma apelului instructiunii CPUID pentru Intel Pentium Pro este:

00000000	0000	00	0110	0001	xxxx (2)	Pentium Pro processor

   Continutul registrului EAX in urma apelului instructiunii CPUID pentru Intel Core i7 este:

00000000	0001	00	0110	1010	xxxx (2)	Intel Core i7 processor and Intel Xeon processor. All processors are manufactured using the 45 nm process.

7. **Explicați la ce este folosită variabila unsigned long int brandID din codul exemplu. Care biți, din care registru, vor fi salvați în această variabilă?**

   Variabila brandID este folosita pentru a salva informatie despre procesor. Pozitiile bitilor folositi sunt intre 0 si 7, iar registrul folosit este EBX.

   Processors that implement the Brand ID feature return the Brand ID in bits 7 through 0 of the EBX register when the CPUID instruction is executed with EAX=1 (see Table 4-1). Processors that do not support the feature return a value of 0 in EBX bits 7 through 0.