L10. FUNCŢIONAREA STIVEI HARDWARE ŞI LUCRUL CU SUBRUTINE. APLICAŢII DE SCRIERE A UNOR MESAJE PE AFIŞAJUL ALFA-NUMERIC ŞI DE GENERARE A UNOR MELODII, PRIN APELURI DE SUBRUTINE ROM.

1. Obiective

Prin parcurgerea acestei ședințe de laborator studenții vor fi capabili:

- Să rezerve spațiu în memorie pentru stivă;
- Să inițializeze vârful stivei;
- Să utilizeze instrucțiunile de depunere și extragere din stivă, precum și cele de apel și revenire;
- Să definească și să folosească subrutine;
- Să utilizeze serviciile oferite de programul monitor.

2. Stiva microprocesorului 8085

Stiva este o zonă de memorie liniară, formată din locații amplasate la adrese consecutive, cu acces indirect pe la un sigur capăt (vârful stivei), prin intermediul registrului SP (Stack Pointer) care este indicatorul vârfului stivei.

În cazul microprocesorului 8085, stiva crește în sensul descrescător al adreselor locațiilor de memorie. Deoarece funcționarea stivei implică atât operații de citire (extragere din stivă) sau de scriere (depunere în stivă), aceasta trebuie organizată într-o zonă de memorie de tip RAM.

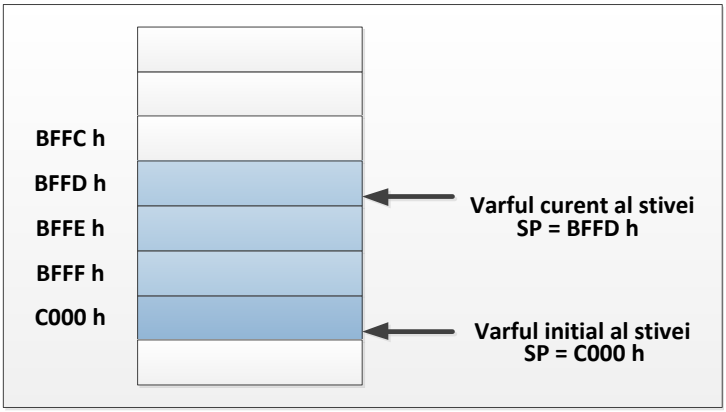


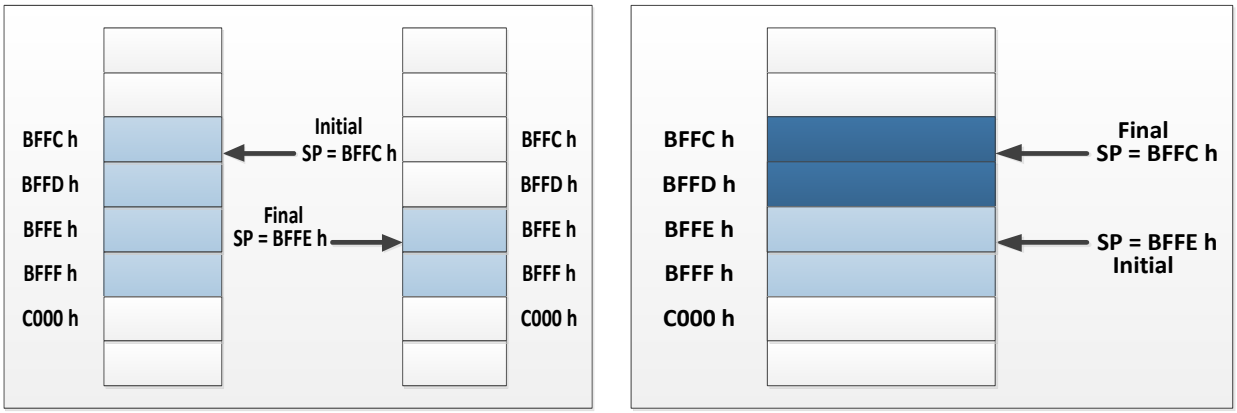
Figura 1. Organizarea stivei

Rolul stivei este de a facilita stocarea temporară a informațiilor (date sau adrese) în memorie fără a reține pentru fiecare adresa exactă la care sunt stocate. Se va reține doar ordinea de stocare a informațiilor, acestea urmând a fi extrase în ordinea inversă a depunerii în

stivă. Stiva are o organizare de tip LIFO (Last In First Out) – primul element extras este ultimul care a fost depus.

Operațiile de depunere și de extragere sunt efectuate cu elemente formate din doi octeți, indiferent dacă acestea sunt date sau adrese. Depunerea unui element în memorie se face prin două scrieri succesive în stivă, iar extragerea unui element prin două citiri succesive din memorie. De fiecare dată, adresa locației înscrise sau citite este furnizată de registrul SP, care este actualizat corespunzător.

SP conține în orice moment adresa ultimei locații ocupate din stivă. De aceea, după fiecare octet citit din stiva, SP este incrementat, iar înaintea scrierii unui octet în stivă SP este decrementat.



a) b)

Figura 2. a)Extragerea din stivă; b)Depunerea în stivă.

2.1. Instrucțiuni de lucru cu stiva

Operațiile de depunere și de extragere sunt realizate la execuția unor instrucțiuni specifice:

- PUSH/POP pentru conținutul registrelor pereche (B, D și H) și a cuvântului de stare al programului (PSW);
- CALL/RET pentru conținutul registrului PC. Informațiile din registrele pereche B, D și H pot fi date pe 16 biți sau adrese ale altor date amplasate în memorie. PSW conține date (registrul A) și condiții (flagurile). Registrul PC conține adresa instrucțiunii următoare.

PUSH rp	PUSH register pair on stack top	$SP - 1 \leftarrow rph$ $SP - 2 \leftarrow rpl$ $SP \leftarrow SP - 2$
PUSH PSW	PUSH PSW on stack top	$SP - 1 \leftarrow A$ $SP - 2 \leftarrow F$ $SP \leftarrow SP - 2$
POP rp	POP into register pair from stack top	$rpl \leftarrow SP$ $rph \leftarrow SP + 1$ $SP \leftarrow SP + 2$

POP PSW POP into PSW from stack top

$$F \leftarrow SP$$
$$A \leftarrow SP + 1$$
$$SP \leftarrow SP + 2$$

Stiva este o zonă de memorie care poate fi definită folosind aceleași directive de asamblare ca și în cazul altor variabile. În cazul variabilelor obișnuite se marchează printr-o etichetă adresa de început a zonei de memorie rezervate (adresa cea mai mică). În cazul stivei se marchează adresa de după cea a ultimei locații rezervate, care va constitui vârful stivei și va inițializa registrul SP, deoarece stiva crește în sensul descrescător al adreselor.

2.2. Instrucțiuni de apel și de revenire

Pentru a evita repetarea în cadrul programului a unei secvențe de cod, se preferă ca respectiva secvență sa apară în memorie o singură dată, urmând ca atunci când se dorește execuția ei, în programul principal să se efectueze un salt la adresa respectivă.

În cazul în care se dorește continuarea programului principal din punctul în care a fost părăsit, acest lucru nu mai este posibil deoarece instrucțiunea de salt distruge vechiul conținut al registrului PC. Adresa de revenire nu este cunoscută nici de secvența de instrucțiuni, dat fiind că nu este constantă. Revenirea va trebui să se facă de fiecare dată într-un alt punct, deoarece saltul din programul principal poate să apară în orice punct al său.

Dacă se dorește revenirea, este necesar ca programul principal să transmită secvenței la care sare o informație pe care aceasta să o folosească pentru a relua execuția programului principal din punctul în care a fost părăsit. Informația transmisă este adresa instrucțiunii de după cea care a determinat părăsirea programului principal. Acest lucru se realizează prin intermediul stivei.

Instrucțiunea care realizează un salt la o adresă după ce în prealabil a depus pe stivă adresa instrucțiunii imediat următoare din programul principal se numește **instrucțiune de apel**. La 8085 este codificată în limbaj de asamblare cu mnemonica CALL.

Instrucțiunea care realizează revenire în programul principal, deoarece reîncarcă în PC adresa din vârful stivei (cea salvată de CALL) se numește **instrucțiune de revenire**. În cazul microprocesorului 8085, instrucțiunea de revenire este codificată cu mnemonica RET.

CALL addr Call

$$SP - 1 \leftarrow HIGH(addr)$$
$$SP - 2 \leftarrow LOW(addr)$$
$$SP \leftarrow SP - 2$$
$$PC \leftarrow addr$$

Ccondition addr Condition call

$$if\ condition = TRUE$$
$$CALL\ addr$$
$$else$$
$$PC \leftarrow PC + 3$$

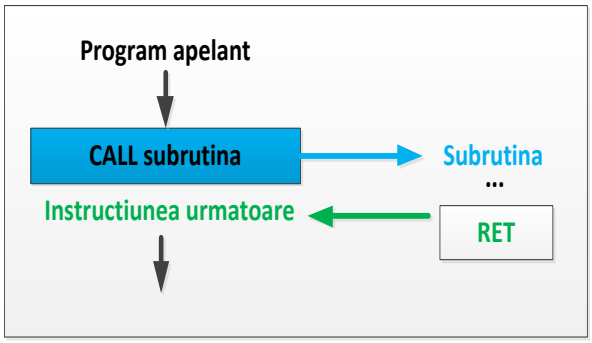
RET	Return	$PC_L \leftarrow SP$ $PC_H \leftarrow SP + 1$ $SP \leftarrow SP + 2$
Rcondition	Conditional return	$if\ condition = TRUE$ RET $else$ $PC \leftarrow PC + 1$

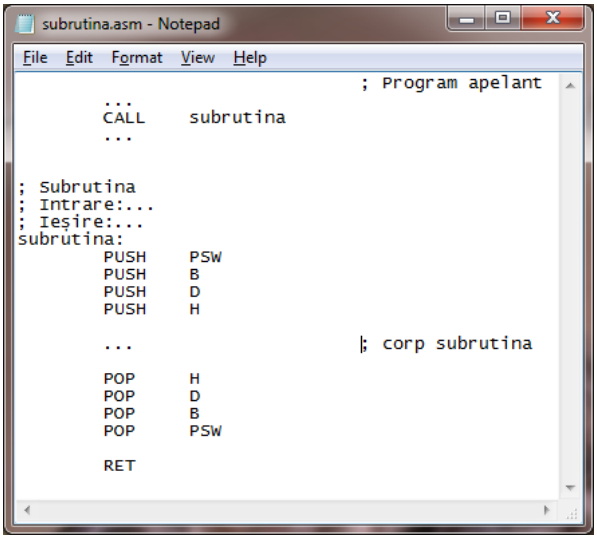
Similar instrucțiunilor de salt, există instrucțiuni de apel și revenire necondiționate, precum și instrucțiuni de apel și revenire condiționate.

Condiția	Descriere	Condiția este adevarată dacă	CCC
NZ	**Not Zero** – rezultatul este diferit de 0;	$Z = 0$	000
Z	**Zero** – rezultatul este egal cu 0;	$Z = 1$	001
NC	**Not Carry** – operația nu a produs transport/împrumut;	$CY = 0$	010
C	**Carry** – operația a produs transport/împrumut.	$CY = 1$	011

Secvența de instrucțiuni care este apelată se numește **subrutină**. Programul principal care apelează subrutina se numește **program apelant**.

Exemplu:





Principalele caracteristici ale subrutinelor:

- Este o succesiune de instrucțiuni cu unul sau mai multe puncte de intrare etichetate și cu unul sau mai multe puncte de ieșire;

- Realizează o anumită funcție: operații I/O, conversii ale formatului datelor, operații matematice în virgulă fixă sau mobilă, etc.;
- Poate fi apelată ori de câte ori este nevoie;
- I se alocă memorie numai o singură dată;
- La terminare trebuie să se întoarcă în programul apelant indiferent de locul de unde este apelată (terminare cu instrucțiune de revenire);
- Poate primi parametri din programul apelant și îi poate furniza rezultate acestuia prin registre, prin stivă sau prin locații fixe de memorie;
- Registrele modificate de subrutină trebuie să ia măsurile necesare pentru salvarea informațiilor care trebuie păstrate.

Este recomandat ca la scrierea codului unei subrutine, aceasta să fie documentată, precizându-se:

- Numele și funcția îndeplinită;
- Modul în care sunt primiți parametrii de intrare;
- Modul în care sunt returnate rezultatele către programul apelant;
- Registrele modificate în cadrul subrutinei.

3. Utilizarea serviciilor programului monitor

Programul monitor al sistemului cu microprocesor 8085 EMAC Universal Trainer conţine o serie de servicii, pe care programul utilizatorului le poate apela pentru a realiza operaţii de I/O de bază. Fiecare serviciu are un cod, care, pentru această versiune de monitor, este cuprins între **0** şi **27** (**1Bh**). Pentru a apela un anume serviciu, se încarcă în registrul C codul serviciului şi apoi se utilizează o instrucţiune CALL BIOS, unde $\boldsymbol{BIOS = 0200h}$ este punctul de intrare unic pentru toate serviciile. Principalele servicii BIOS care pot fi apelate din programul utilizatorului sunt prezentate în cele ce urmează:

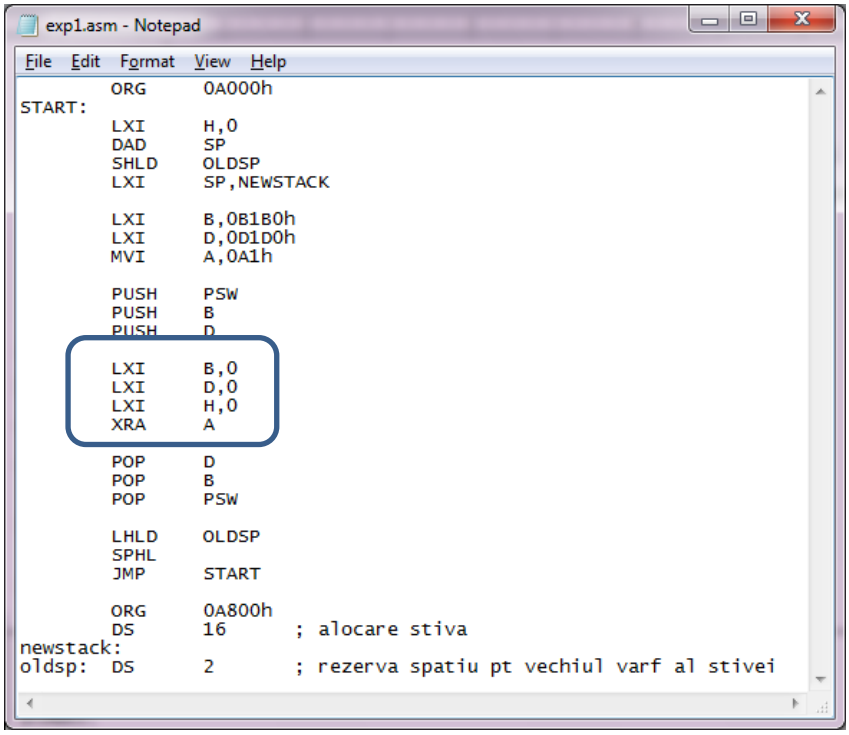
Cod	Denumire	Parametrii de intrare	Descrierea serviciului
0h	**DEMO**		Program demonstrativ.
1h	**CONIN**	C = 1 B = portul de comunicație 1 sau 2	Citește de la consolă un caracter pe portul specificat de registrul B. Caracterul ASCII returnat va fi stocat în registrul L.
2h	**CONSTAT**	C = 2 B = portul de comunicație 1 sau 2	Inspectează portul de comunicație selectat de B și returneazp 0FFh dacă un caracter este gata și 00h în caz contrar în registrul L.
3h	**CONOUT**	C = 3 B = portul de comunicație 1 sau 2 E = caracterul ASCII	Transferă un caracter ASCII pe portul de comunicație indicat de registrul B.

4h	**PSTRING**	C = 4 B = portul de comunicație 1 sau 2 DE = pointer la șirul de caractere	Transmite un șir de caractere terminat cu $ pe portul selectat de registrul B. $. Semnul $ nu este afișat, iar în DE pointerul se plasează după semnul $
5h	**UPRINT**	C = 5 B = portul de comunicație 1 sau 2 DE = numărul fără semn pe 16 biți	Afișare fără semn a unui număr pe 16 biți în sistemul de numerație zecimal fără a folosi semnul pe portul de comunicație selectat în registrul B.
6h	**SPRINT**	C = 6 B = portul de comunicație 1 sau 2 DE = numărul fără semn pe 16 biți	Afișare cu semn a unui număr pe 16 biți în sistemul de numerație zecimal folosind reprezentarea în cod complementar, pe portul de comunicație selectat de registrul B.
7h	**MULT**	C = 7 DE = primul număr pe 16 biți HL = al doilea număr pe 16 biți	Înmulțește conținutul lui HL cu conținutul lui DE și stochează rezultatul în HL (partea HIGH) și DE (partea LOW).
8h	**DIV**	C = 8 HL = deîmpărțitul pe 16 biți DE = împărțitorul pe 16 biți	Împarte HL prin DE. Câtul împărțirii va fi stocat în HL, iar restul în DE.
9h	**ADCIN**	E = canalul selectat	Intrare A/D pentru citirea unei valori pe 8 biți de pe canalul A/D selectat. Rezultatul conversiei va fi stocat în registrul L.
Ah	**DIPSWIN**	C = A	Citește poziția curentă a fiecăruia dintre cele 8 micro-comutatoare. Poziția celor 8 micro-comutatoare va fi stocată în registrul L.
Bh	**PTBIN**	C = B	Citește conținutul portului digital de intrare portB și returnează complementul acestuia în registrul L.
Ch	**PTAOUT**	E = valoarea pe 8 biți	Scrie pe portul de ieșire digital portA o valoare pe 8 biți.
Dh	**HEXPRINT**	C = D B = portul de comunicație 1 sau 2 DE = numarul fara semn pe 16 biți	Transmite pe portul de comunicație selectat de registrul B, valoarea din registrul extins DE în hexazecimal (4 cifre).
Eh	**DACOUT**	C = E E = valoarea pe 8	Acest serviciu este o ieșire de tip DAC care transferă un număr pe 8 biți convertorului digital analogic.

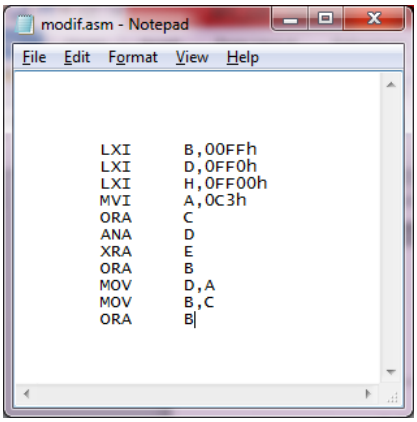
		biți	
10h	**PITCH**	C = 10 DE = valoarea pe 16 biți a diviziunii	Serviciul trimite numărul pe 16 biți stocat în registrul DE către timerul difuzorului. Cu cât acest număr este mai mare, ci atât va fi mai mică diviziunea. Dacă DE = 0, atunci difuzorul este dezactivat.
11h	**LEDOUT**	E = caracterul ASCII D = poziția	Afişează un caracter ASCII din E pe afişajul alfanumeric în poziţia specificată de D.
12h	**LEDHEX**	DE = numărul	Afişează numărul din DE în hexazecimal pe afişajul alfanumeric pe cele 4 celule din stânga.
13h	**LEDDEC**	DE = numărul	Afişează numărul din DE în zecimal pe afişajul alfanumeric pe cele 4 celule din stânga.
14h	**DELAY**	HL = valoarea întârzierii	Întârziere proporţională cu valoarea din HL
15h	**TUNE**	B = durata tonului DE = secvența de tonuri	Cântă o secvenţă de tonuri terminată cu 0 şi adresată de DE, în ritmul constant specificat de B (Cu cât B este mai mare, cu atât este mai mare durata tonului).
16h	**PRNOUT**	E = caracterul	Transferă caracterul din E pe portul paralel.
17h	**OUT422**	E = caracterul	Se transferă octetul din registrul E pe portul RS422.
18h	**IN422**	C = 18	Citește un byte de pe portul RS422. Dacă a fost recepționat un caracter, H = 1 și registrul L va conține caracterul. Dacă nu s-a recepționat nici un caracter, HL va fi 0.
19h	**KEYIN**		Aşteaptă apăsarea unei taste şi returnează codul ei în L. Codificarea este între 0÷0Fh pentru primele 16 taste, de sus în jos şi de la stânga la dreapta. Următoarele 3 rânduri returnează coduri între 14h÷1Fh.
1Ah	**WRSCL**	C = 18 DE = adresa primilor 8 bytes ce trebuie transferați ceasului de timp real.	Scrie 8 bytes în memorie începând de la adresa specificată în registrul DE, după ceasul de timp real opțional. Ceasul oferă informații de temporizare în BCD inclusiv informații ca sute de secunde, secunde, minute, ore, zi, dată, lună sau an. Data de la sfârșitul lunii este ajustată automat în funcție de lună și de ani bisecți sau nu. Ceasul de timp real operează în formatul cu 24 de ore sau cu 12 ore cu indicator AM/PM.
1Bh	**RDSCL**	C = 19 DE = adresa de start a zonei în care sunt stocați cei 8 bytes citiți de la ceasul de timp real.	Citește 8 bytes de date de la ceasul de timp real și îi stochează consecutiv, începând cu adresa din registrul pereche DE. Cei 8 bytes de date sunt formatați similar cu datele de intrare transmise serviciului WRSCL.

4. Aplicații propuse

4.1. Să se scrie un fișier cu numele **exp1.asm** care să conțină următoarea secvență de instrucțiuni. Să se asambleze cu ASM85 și să se încarce în memoria microsistemului. Să se execute instrucțiunile în regim pas cu pas folosind NoICE85, urmărind conținutul registrului SP și al celorlalte registre pereche, precum și conținutul stivei.



Se considera fragmentul selectat modificat după cum urmează:



Să se precizeze valorile registrilor după execuția fiecărei instrucțiuni.

4.2. Să se scrie codul unui program în limbaj de asamblare, care:

a) Execută programul demonstrativ;

exp3.asm - Notepad

File Edit Format View Help

```
        MVI     C,DEMO
        CALL    BIOS
```

b) Afișează un mesaj;

exp3.asm - Notepad

File Edit Format View Help

```
        LXI     H,MESAJ1
        CALL    LDISPLAY
```

c) Cântă un mesaj;

exp3.asm - Notepad

File Edit Format View Help

```
        MVI     B,0FFH
        LXI     D,MESAJ1
        MVI     C,TUNE
        CALL    BIOS
```

d) Așteaptă apăsarea unei taste.

exp3.asm - Notepad

File Edit Format View Help

```
        MVI     C,KEYIN
        CALL    BIOS
```

exp3.asm - Notepad

File Edit Format View Help

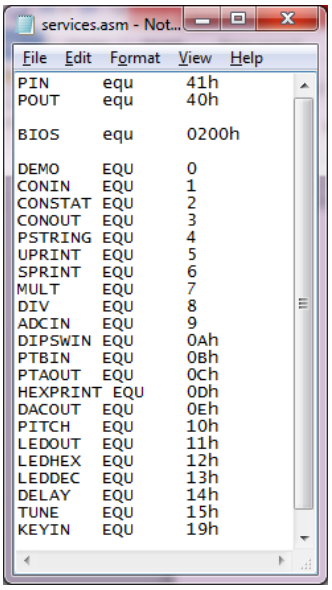
```
        include "services.asm"
        ORG     0A000h
LOOP:
        LXI     H,0             ; incarca 0 in HL
        DAD     SP              ; incarca SP in HL
        SHLD    OLDSP           ; memoreaza varful stivei
        LXI     SP,NEWSTACK     ; initializeaza SP cu noul varf al stivei

        ...

        LHLD    OLDSP           ; incarca in HL vechiul varf al stivei
        SPHL                    ; incarca in SP din HL vechiul varf al stivei

        JMP     LOOP


LDISPLAY:
        MVI     D,0             ; in D - pozitia initiala de afisare
        MVI     B,8             ; in BC - numarul de celule
NEXT:
        MOV     A,M             ; citeste caracterul din sir
        ORA     A               ; verificare terminator sir
        JZ      CLEAR           ; sterge celulele urmatoare
        MOV     E,A             ; altfel afiseaza caracterul citit
        MVI     C,LEDOUT        ; serviciul pentru afisare (11)
        CALL    BIOS            ; apel BIOS
        INX     H               ; incrementeaza pointerul pe sir
        INR     D               ; incrementeaza pozitia de afisare
        DCR     B               ; decrementeaza numarul de celule
        JNZ     NEXT            ; daca mai sunt celule, le afiseaza
        JMP     ENDDISPLAY      ; mesajul a fost afisat!

CLEAR:
        MVI     E,0             ; au ramas celule libere
        MVI     C,LEDOUT        ; sterge celula curenta
        CALL    BIOS            ; apel BIOS
        INR     D               ; incrementeaza pozitia de afisare
        DCR     B               ; decrementeaza numarul de celule
        JNZ     CLEAR           ; continua daca au mai ramas celule

ENDDISPLAY:
        RET                     ;revine in programul apelant

        ORG     0A800H
LNG_NEWSTACK    EQU     16      ; dimensiunea stivei
        DS      LNG_NEWSTACK    ; rezerva spatiu pt stiva
NEWSTACK:
OLDSP:  DS      2               ; rezerva spatiu pt vechiul varf
MESAJ1: STR     "MESAJUL1"      ; defineste un sir ASCII
        DB      0               ; terminatorul de sir
```

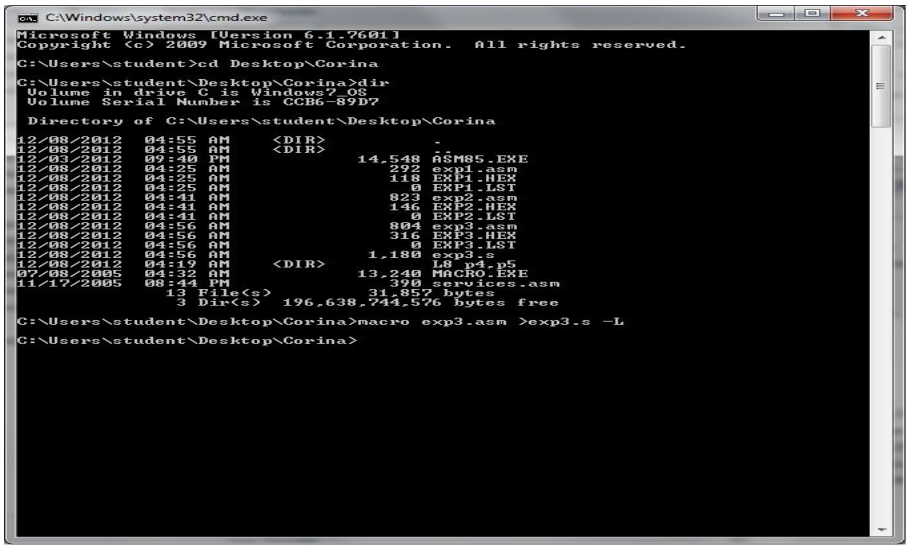
Codurile serviciilor sunt definite într-un fişier separat (*services.asm*), care trebuie să existe în directorul în care se află şi fişierul principal şi care poate fi inclus în fişierul sursă în limbaj de asamblare cu directiva: **INCLUDE ”services.asm”.**

Această directivă se adresează unui program denumit macroprocesor (Macro.exe), care trebuie executat înainte de asamblarea propriu-zisă şi care generează un fişier pentru asamblor cu extensia **„.s”** prin includerea tuturor fişierelor specificate de fişierul sursă.

Programul **Macro.exe** va fi folosit pentru pre-procesare astfel: ***macro fisier.asm >fisier.s –L*** din **cmd**.

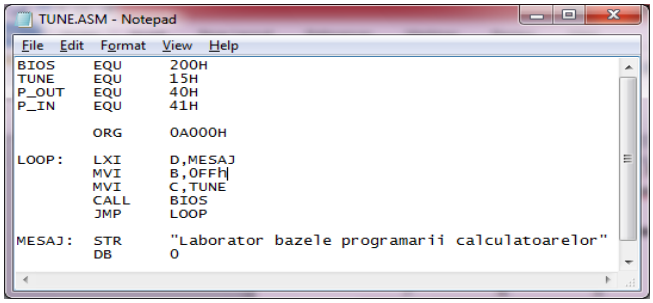
Macro.exe va prelucra fişierul **fisier.asm** şi va genera **fisier.s**, cu toate fişierele specificate incluse, iar asamblorul va trebui să prelucreze fişierul generat de Macro.exe, adică **fisier.s**, pentru a genera codul executabil **fisier.hex**.





Care este diferența între fișierul .asm scris și fișierul .s generat? Ce semnificație au liniile de cod suplimentare?

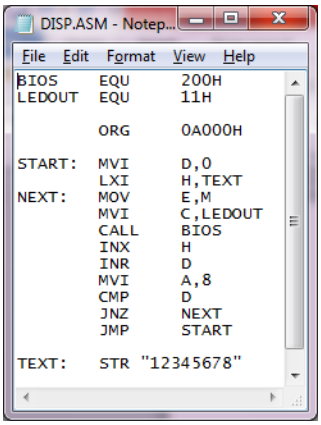
4.3. Să se scrie un program în limbaj de asamblare care cântă o melodie folosind mesajul ”Laborator bazele programării calculatoare”.



Să se modifice programul anterior astfel încât durata melodiei să fie dată cu ajutorul microcomutatoarelor. Să se afișeze valoarea duratei pe portul de ieșire.

4.4. Să se realizeze un program în limbaj de asamblare care să cânte o melodie specifică unei alarme folosind coduri ASCII.

4.5. Să se scrie codul unui program în limbaj de asamblare care să afișeze un text pe display.



Să se explice fiecare linie de cod din program.

5. Referințe bibliografice

[1] C.Huțanu, M.Postolache, *Sisteme cu microprocesoare*, Editura Academică, Iași, 2001.
[2] Gh.Toacșe, *Introducere în microprocesoare*, Editura Științifică și Enciclopedică, București, 1985.
[3] ***, Universal Trainer, Lab Manual for Board Revisions R1 and R2, EMAC INC, 1993.
[4] ***, Universal Trainer, Reference Manual, EMAC INC, 1993.
[5] ***, Universal Trainer, Self Instruction Manual, EMAC INC, 1992.