Lucrarea de laborator numarul 5

Descrierea simulatorului de calculator didactic.

Acest simulator încearca să pună la dispoziția studenților o unealta care să îi ajute să înțeleagă procesele logice interne ale unui procesor şi însuşirea bazelor programării în limbajul de asamablare. El simulează un calculator didactic. Acesta este un model abstract, realizat pentru a ilustra principiile de baza ale functionarii procesorului. Procesoarele existente în calculatoarele nostre sunt particularizări ale acestui model.

Fereastra de început conține mai multe zone importanate. Prima este o fereastra în care este reprezentată memoria principală. Aceasta este împărțită în 16x16=256 locații de memorie fiecare locație de memorie având 8 biti. Pentru a uşura înțelegerea, continuul memoriei este reprezentat în hexadecimal. Ex $A0_{(16)} = 1001000_{(2)}$. La fel şi adresele, pe verticală fiind reprezentată jumătatea cea mai semnificativă (prima jumatate) a adresei, iar pe orizontala este reprezentata jumătatea cea mai puțin semnificativă aparținând adresei. De asemenea in coltul dreapta jos, în bara de stare, apare adresa locației de memorie în care se afla cursorul în acel moment. Memoria poate fii editată într-o maniera asemănătoare celei utilizate în editoarele de text folosind Ctrl-C, Ctrl-V etc. Aceste funcții pot fi găsite şi în meniul edit.

Simulatorul dispune de asemenea de 16 registre pentru uz general de 8 biti, numerotate de la R0 la RF ultimul dintre ele, RF, actionând ca şi ieşire. De asemenea in coltul dreapata sus se afla cele doua registre sistem, contorul de programe (Program Counter-PC) şi registrul de instrucțiuni (Instruction Register-IR). Lângă contorul de programe se află un buton care permite setarea la zero a contorul de programe. Acesata echivaleză cu repornirea executării programului. Tot în aceasta zonă se afla şi atotputernicul buton de help.

Sub contorul de programe şi registrul de instructiuni se afla două butoane care permit deschiderea şi salvarea fisierelor de tip PRG si ASM. Primul tip de fisier este un format propietar iar cel de al doilea este un tip de fişier sursă în limbajul de asamblare. Aceleaşi funcții pot fi găsite şi în meniu “File”.

Mai jos sunt patru butoane care folosesc la depanarea programelor. Primul, etichetat “Run” demarează execuția programului din memorie până

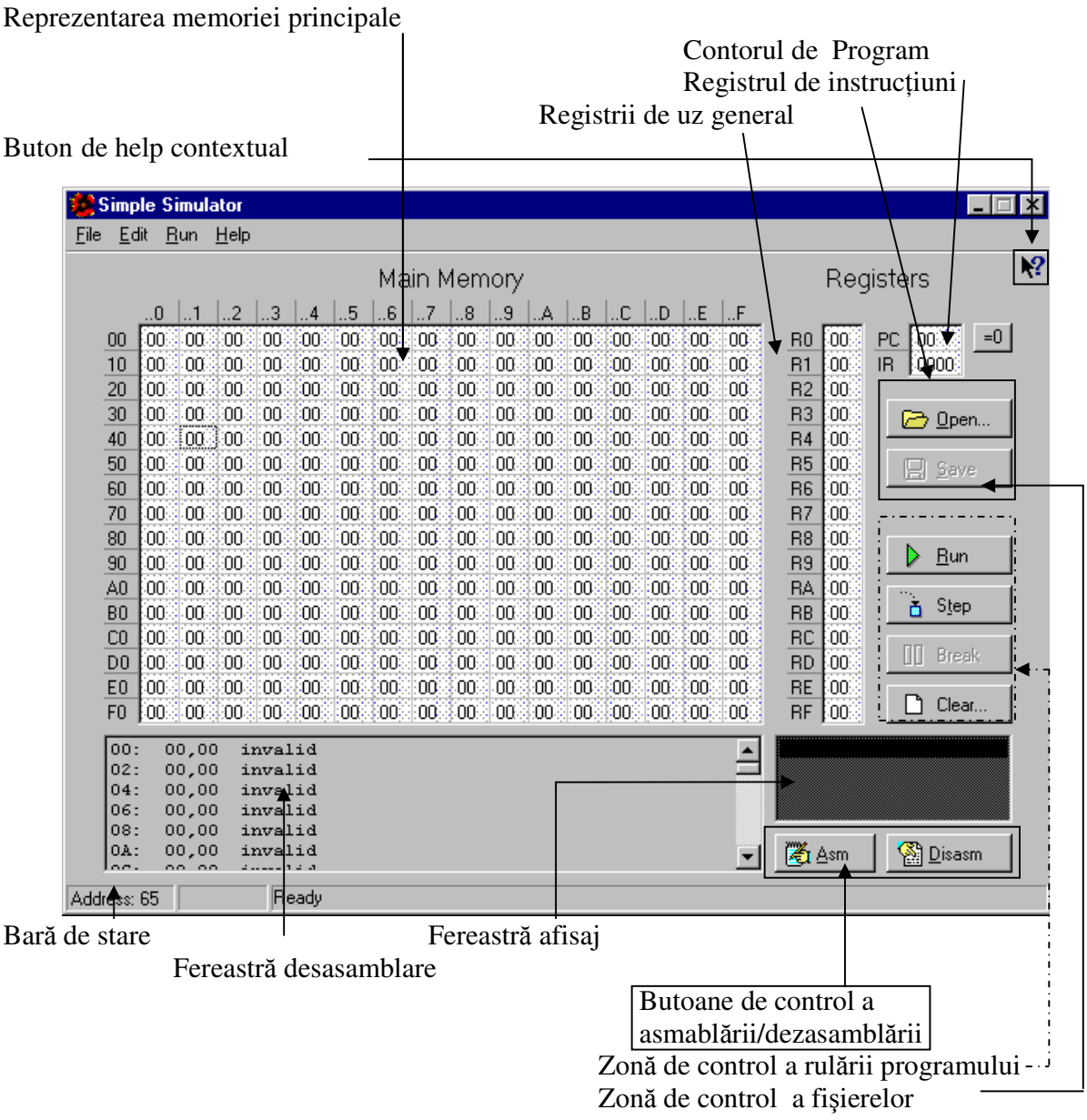
când se apasă butonul etichetat “Break”, se întâlneşte instrucţiunea “halt” sau se întâlneşte o instruţiune invalidă.

Butonul etichetat “Step” porneşte execuţia pas cu pas a programului din memorie. Adică executa instrucţiunea a cărei adresă se afla in PC si il incrementeza pe acesta. Daca programul ruleaza va trebui sa apasati prima data Break. Numai dupa aceasta efectul butonului Step va fi vizibil.

Butonul etichetat “Break” va opri executia programului din memorie.

Butonul etichetat “Clear” foloseste la resetarea memoriei principale, regştrilor şi/sau a contorului de programe sau a registrului de instrucţiuni.

De asemenea meniul Run conţine aproape aceleaşi funcţii, excepţie făcând funcţia Break.



Lucrarea de laborator numarul 5

Freastră pentru afişare ne permite afişarea sub formă caracterelor ASCII a conţinutului registrului RF. Afişarea conţinutului se face în momentul scrierii registrului. Adică dacă se scrie în registru, atunci se tipăreşte. Daca nu, nu se intâmplă nimic.

Bară de stare este împărţită în trei parţi. Prima parte afisează adresa locaţiei de memorie curente, a doua daca memoria a fost sau nu modificată, iar a treia starea programului sau intervalul de memorie selectată. Zonele de memorie modificate apar colorate in rosu.

Fereastra de desamblare arata continutul memorie sub forma instrucţiunilor limbajului de asamblare.

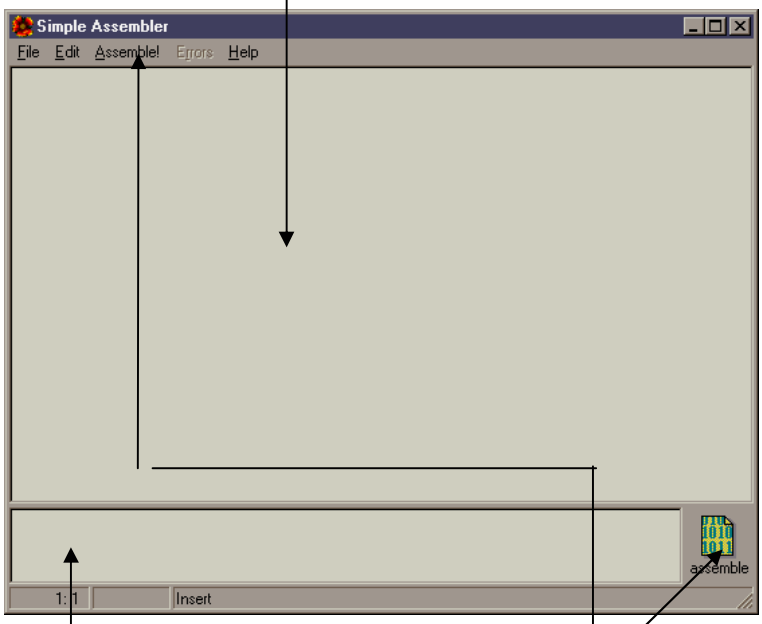
Limbajul de asamblare a fost creat pentru a uşura scrierea programelor în cod maşină. Scrierea direct în cod maşină a programelor este o modalitate prea abstractă pentru a avea o raspândire mare în rândul programatorilor.

Pentru acesta s-au realizat interpretoare de limbaj de asamblare. Ideea de baza a acestui limbaj este reprezentarea grupurilor de biti care reprezinta intrucţiunile în cod maşină cu cuvinte cheie numite "mnemonice". De exemplu intrucţiunea de oprire a execuţiei programului este 1100 0000 0000 0000 (C0 $00_{(16)}$). Aceata intrucţiune este reprezentată în limbajul de asamblare prin mnemonicul *halt*. Bineinteles că şi pentru celelate istrucţiuni exista mnemonice care vor fi prezentate mai târziu.

Pentru a uşura înţelegerea şi scrierea programelor în cod maşina

Butonul "ASM" porneşte mediul de programare în limbajul de asamblare. Butonul "Disasm" deschide un fişier text cu sursa în asamblare a programului aflat în memorie. Chiar daca el a fost scris "de mână" direct în memorie.

Fereastra de editare



Fereastra de prezentare a erorilor

Buton start asamblare

Lucrarea de laborator numarul 5

Mediul de programare în asamablare este împarţit în două parţi fereastra de editare şi fereastra de erori, în care se afiseză ultima eroare de asambare. De asemenea se află şi butonul de start a asamblorului.

În partea a doua a acestui laborator se vor explica câteva exemple simple de programe in ASM.

Formatul general al liniilor de program este:

[eticheta:][instructiune][;comentarii]

- eticheta este un sir de caractere alfanumeric care nu poate să înceapa cu o cifra. Eticheta trebuie sa se termină cu caracterul ":".Etichetele sunt folosite pentru a determina destinaţiile salturilor şi reprezintă o adresa din memorie.
- instrucţiunea este formata dintr-un mnemonic şi operanzi
- comentariile sunt siruri de caractere alfanumerice care încep cu ";". Comentariile ţin până la sfârşitul rândului.

Toate cele trei părţi sunt opţionale. Deci putem avea orice combinatie. (etichetă – comentariu, instrucţiune-comentariu, etc.)

Instrucţiunile sunt în marea majoritate pe doi bytes. Prima jumătate a primului byte reprezintă opcode-ul. Restul grupurilor de cate 4 biţi reprezinta operanzii.

Tipurile de date folosite

Acest simulator foloseste doua categorii de tipuri de date numerele şi şirurile de caractere.

Numere sunt reprezentate în trei baze – binar, decimal si hexadecimal.

Numerele binare sunt reprezentate prin şiruri de '1' si '0' terminate cu litera "b"

Numerele decimale sunt reprezentate prin siruri de cifre de la '0' la '9' terminate cu d. Pentru a se arata ca numarul este negativ se poate pune în faţa semnul '-'. De asemenea folosirea literei 'd' nu este obligatorie.

Numerele hexadecimale sunt reprezentate prin siruri de cifre de la '0' la '9' si litere de la 'A'la 'F'. Pot fi reprezentate in trei modalitati:

- *Stil C*- numarul începe cu *"0x"* şi continuat cu un şir de cifre hexadecimale *'0'-'9'* şi *'A'-'F'*. Ex: *0x1F*.
- *Stil Pascal* –numarul începe cu caracterul *'$'* urmat de cifre hexadecimale. Ex: *$1F*.

Lucrarea de laborator numarul 5

- *Stil Asamblor*- numărul este format dintr-un şir de caractere hexadecimale care nu poat începe cu o "litera" şi este terminat cu litera *'h'*. În cazul în care numărul trebuie inceape cu o "litera" se adauga cifra *'0'*. Ex *1Fh* sau *0AFh*.

Remarci:

- Semnul "-" se poate folosi numai pentru reprezentările decimale ale numerelor negative. Daca doriti sa folositi un numar negativ reprezenat in lata baza decat 10 va trebui sa il reprezenati in complement fata de 2. Ex: -10 (*00001010b*) va fi reprezentat *0F6h* sau *11110110b.*
- Nu sunt permise spatii in interiorul numerelor.

Tipul *string* este format din şiruri de aproape orice caractere execepție facând ghilimelele şi apostroafele. Pentru a reprezenta un singur caracter de obicei se folosec apostroafele. Ex *'a'*. Pentru reprezentarea mai multor caractere acestea sunt încadrate de ghilimele. Ex *"abcd!#"*. Caracterele vor fi reprezentate în memorie prin codul lor ascii. Ex "*abcd*" va fi reprezentat în memorie în felul urmator 61 62 63 64 sau în binar 0110 0001 0110 0011 0110 0100.

În continuare se vor prezenta câteva instrucțiuni simple şi uzuale.

data byte - Scrie în memorie la adresa curentă

Instructiunea nu apare în memorie si deci nu are opcode. Are mnemonicul *db* iar datele pot fi siruri de caractere sau numere.

direct load - *Încarcare directa* **-** Încarca registrul cu date de la o adresa din memoria principala.
Are mnemonicul *load,* op-code-ul *1* si formatul *load reg, [adresa]* sau *load reg, [eticheta].*

Exemplu:

load R4, [0Ah] –încarcă în registrul R4 valoarea de la adresa $0A_{(16)}$

load R4, [label] - încarca în registrul R4 octetul de la adresa determinata de eticheta "*label*"

În cod hexadecimal exemplul va arăta astfel:

14 12

Semnificatia acestei instructiuni este:

1(*opcode*) 4 (*registrul destinatie*) 12 (*adresa sursa*)

Sau în binar

00010100 00010010

În cazul în care al doilea operand este o eticheta, la scrierea în memorie pe locul etichetei va apare adresa de memorie determinată de aceasta.

immediate load – *Încărcare imediata*- Încărca registrul cu date în modul imediat

Are mnemonicul *load,* op-code-ul *2* şi formatul *load reg, numar* sau *load reg, eticheta*

Exemplu:

load R4, 12h –încărca în registrul R4 valoarea $12_{(16)}$ reprezentată în baza 16.

load R4,label – încărcă în registrul R4 cu adresa locației determinate de eticheta "*label*"

În cod hexadecimal exemplul va arăta astfel:

24 12

Semnificația acestei instrucțiuni este:

2(*opcode*) 4 (*registrul destinație*) 12 (*data ce trebuie încarcata*)

Sau în binar:

00100100 00010010

La fel ca în cazul de mai sus eticheta va apărea ca o adresă în memoria calculatorului.

indirect load – *Încarcare indirecta* – Încarcarea registrul cu date de la o adresa aflată în alt registru.

Are mnemonicul *load*, op-code-ul D şi formatul *load reg_destinatie, [reg_adresa_sursa]*

Exemplu:

load R4, [R5] –încarcă în registrul R4 valoarea cu datele aflate la adresa din registrul 5.

În cod hexadecimal exemplul va arată astfel:

D0 45

Semnificatia acestei instrucțiuni este:

Lucrarea de laborator numarul 5

D(*opcode*) 0(*biti nefolositi*) 4 (*registrul destinatie*) 5 (*registrul in care se afla adresa sursa*)

Sau în binar:
11010000 01000101

move-*copiaza între registrii***-**mnemonicul *move*, op-code-ul şi formatul *move reg1,reg2*. Copiaza conținutul registrului reg2 in registrul reg1.
Exemplu:
move R1, R2 - copiaza contnutul registrului 2 în registrul 1
În cod hexdecimal exemplu va arăta astfel:
40 12 move(40) R1(1),R2(2)
Sau în binar:
0100 0000 0001 0010

jmpEQ *- salt daca egal* – salt daca conținutul registrului reg este egal cu cel al registrului R0
Are mnemonicul *jmpEQ,* op-codul-ul B şi formatul *jmpEQ reg=R0 adresa_destinatie,* sau *jmpEQ reg=R0 eticheta_salt.*

Exemplu:
jmpEQ R0=R1 0xAA - încarcă în PC instructiunea de la adresa $AA_{(16)}$
jmpEQ R0=R1 salt - încarcă în PC instructiunea de la adresa la care se afla eticheta salt.
În cod hexadecimal exemplul va arata astfel:
B1 45
Sau în binar:
10110001 01000101

Aceste date se pot interpreta în felul urmator:
B(se compara registrul 0) 1 (cu registrul 1) 45 (şi daca sunt egale se încarcă în PC a adresa 45)

jmp *- salt neconditionat* - încarcă în PC adresa instrucțiunii care trebuie executate.
Are mnemonicul *jmp*, op-code-ul B si formatul *jmp adresa_destinatie* sau *jmp eticheta.*

Exemplu:
jmp 0x45 - încarcă în PC instrucțiunea de la adresa AAh

Lucrarea de laborator numarul 5

jmp salt - încarcă în PC instrucțiunea de la adresa la care se află eticheta *salt*.

În cod hexadecimal exemplul va arata astfel:
B0 45
Sau în binar:
10110000 01000101

Aceste date se pot interpreta în felul urmator:
B(se compara registrul 0) 0 (cu registrul 0) 45 (şi dacă sunt egale se încarcă în PC a adresa 45)

Acesta instrucțiune apare în memorie ca un caz particular al instructiunii *jmpEQ* deoarece tot timpul R0=R0.

Exemplu:
jmp 0x45 - încarcă în PC instrucțiunea de la adresa AAh
jmp salt - încarcă în PC instrucțiunea de la adresa la care se află eticheta salt.

addi- *adunarea a doi intregi* **–** adună conținutul a doi registrii în complement fața de 2 şi stocheaz rezultatul în un al treilea registru.

Are mnemonicul *addi*, op-code-ul 5 si formatul *addi reg3, reg1, reg2.* Unde reg1 si reg2 sunt sursele adunarii iar reg3 este destinația.
Exemplu:

addi RF,R1,R2

În cod hexadecimal exemplul va arata astfel:
5F 12
Sau în binar:
1011111 00010010
Semnificația acestei instrucțiuni este:
5(*opcode*) F (*registrul destinatie*) 1 (*registru sursa*) 2 (*registru sursa*)

halt – *stop* - Opreste executia programului
Are mnemonicul *halt,* op-codul C si nu are operanzi.

Lucrarea de laborator numarul 5

În continuare vom prezenta doua exemple. Primul exemplu va tipări la nesfârşit în fereastra de afişare toate caracterele ascii. Al doilea va afişa un text aflat în memoria calculatorului.

```
        load   RF,0       ;initialize to 0
        load   R7,1       ;increase step
NextChar: addi  RF,RF,R7  ;increase (is written to RF(=screen))
        jmp    NextChar   ;repetat
```

În continuare se va descrie funcţionarea programului:

**se încarcă în registrul RF 0* – se iniţializeaza contorul

**se încarcă în registrul R7 1* – se iniţializeaza pasul de crestere a contorului

**se adună la registrul RF conţinutul registrului R1* – se incrementeaza contorul. De fiecare data când registrul se incrementează contorul RF se afisează pe ecran caracterul ASCII cu codul corespunzător conţinutului registrului RF.

**salt necondiţionat la adresa determinată de eticheta* NextChar – în felul acesta se creaza un cilclu infinit.

În acest simulator daca continutul unui registru este FF adunarea unui numar va da rezultatul truncat la ultimele doua cifre hexadecimale. Această operaţiune nu va genera un mesaj de eroare.

Echivalentul în C++ al programului ar fi

```
include <stdio.h>
int i;

main(){
i=0;
while(1){
        i=i+1;
        putc(i);
        }
}
```

Programul va apărea în memorie sub forma
Iniţial contorul de programe va avea valoarea 0 la fel ca şi registrul de instrucţiuni.

Lucrarea de laborator numarul 5

00: 2F 00 - load (2) RF (F) ,0 (00)

Unitatatea de comanda începe faza de extragere de la adresa *00* plasând instrucțiunea *2F 00* în registrul de instrucțiuni. Unitatea de comanda este proiectată în asemenea fel încât să extragă din memorie cate doi octeți, marimea registrului de instrucțiuni. După aceasata se creste număratorul de program cu doi. Astfel la sfârsitul acestui ciclu *fetch* contorul de programe şi registrul de instrucțiuni vor arăta astfel.

PC: 02 0000 0010
IR: 2F 00 0010 0111 0000 0001

Din acest moment urmează decodificarea instrucțiunii şi execuția ei în paralel cu următorul ciclu *fetch***.** Instrucțiunea este decodificată şi Unitatea de Comandă trage concluzia că trebuie să încarce în registrul *RF* valoarea *0.*

02: 27 01 - 0010 0111 0000 0001 - load (2) R7 (7) ,1 (01)

Asemănător, urmează procesarea instrucțiuni de la adresa *02h.* Ea este încarcată în registrul de instrucțiuni dupa care contorul de programe este incrementat cu *2*. Urmează decodificarea intructiunii. În acest moment Unitatea de Comanda setează registrul de *R7* cu valoarea *1*.
Regiştrii sistem vor arăta astfel:

PC: 04 0000 0100
IR: 27 01 0010 0111 0000 0001

În pasul următor se încarcă în registrul de instrucțiuni, instrucțiunea de la adresa *04h*.

04: 5F F7 - 0101 1111 1111 0111 - addi (5) RF (F), RF (F), R7,(7)

Contorul de programe este incrementat cu 2 şi se decodează datele din registru şi Unitatea de Comandă constată că trebuie efectuată o adunare. În acest moment Unitatea de Comandă activează circuitele logice pentru a efectua o adunare având ca surse regiştrii *RF* şi *R7*, iar ca destinație registrul *RF*. În momentul în care registrul *RF* este scris, conținutul lui este afişat pe ecran. Cele doua registre sistem arată în acest moment asfel.

PC: 06 0000 0110
IR: 5F F7 0101 1111 1111 0111

Lucrarea de laborator numarul 5

Se încarcă următoarea instrucțiune, cea de la adresa *06h*.

06: B0 04 -1011 0000 0000 0100-jmp (B) la adresa 04h (04)

Dupa cum se observă eticheta *NextChar* a fost substituită, de interpretorul limbajului de asamblare, cu adresa instrucțiunii pe care o indică. Instructiunea este decodificată şi Unitatea de Comandă o interpretează ca istrucțiune de salt la adresa *04h*. Contorul de programe este setat la valoarea instrușiunii la care trebui efectuat saltul.

În continuare se va descrie al doilea exemplu.

```
            load   R1,Text     ;the start of the string
            load   R2,1        ;increase step
            load   R0,0        ;string-terminator
NextChar:   load   RF,[R1]     ;get character and print it on screen
            addi   R1,R1,R2    ;increase address
            jmpEQ  RF=R0,Ready;when string-terminator, then ready
            jmp    NextChar    ;next character
Ready:      halt

Text:       db     10
            db     "Hello world !!",10
            db     "    from the",10
            db     "  Simple Simulator",10
            db     0           ;string-terminator
```

Programul funcționează în felul urmator:

load R1,Text **se încarcă în registrul R1 adresa de început a stringului* – adresa de început este obtinută prin înlocuirea etichetei *Text* cu adresa pe care o indică.
load R2,1 **se încarcă în registrul R2 1* – se inițializează pasul de crestere a contorului
load R0,0 **se încarcă în registrul R0 0* – se inițializează R0 cu caracterul de sfârşit de string
NextChar: **load** RF,[R1] **se încarcă în registrul RF caracterul aflat la adresa conținută în registrul R1* – se afişează caracterul aflat la adresa conținută de registrul R1.

Lucrarea de laborator numarul 5

addi R1,R1,R2 **se adună conţinutul R1 la conţinutul R2 iar rezultatul se stochează în R1*- se setează in R1 adresa urmatorului caracter care trebuie afişat.
jmpEQ RF=R0,Ready **se compară conţinutul registrului RF cu conţinutul registrului R1 si daca sunt egale se încarcă în PC adresa instrucţiunii indicate de eticheta* Ready –se verifică dacă s-a ajuns la caracterul care indică sfârşitul şirului. Eticheta "*Ready*" indicând instructiunea "*Halt*" aceasta are ca efect oprirea programului.
jmp NextChar **salt necondiţionat la adresa determinată de eticheta* NextChar – în felul acesta se continua ciclul atât timp cât nu se întâlneşte caracterul de sfârşit de şir.

```
db      10
db      "Hello world !!",10
db      "    from the",10
db      "  Simple Simulator",10
db      0           ;string-terminator
```

Această secţiune din program încarcă în memorie şirul de caractere care trebuie afişat. Încărcarea în memorie începe de la locaţia imediat urmatore instrucţiunii "*halt*". Dupa intructiune "*halt"* apare un spaţiu liber. Aceasta se datorează faptului ca în memorie se alocă spaţiu pentru operanzi, chiar dacă acestia nu sunt folosiţi.

Echivalentul în C++ al programului ar fi

```
include <stdio.h>;
include <string.h>;

int i;
char *Text;

main(){
i=0;
Text ="\n Hello world !! \n from the\n Simple Simulator\n ";
        do{
        putc(Text[i]);
        i=i+1;
        } while(Text[i]!=0)
}
```

Lucrarea de laborator numarul 5

În memorie programul apare astfel:

Registrii sistem:
PC: 00 0000 0000
IR: 00 00 0000 0000 0000 0000

00: 21 10 load (2) R1 (1), Text (10)-eticheta este înlocuită cu o adresa
Reprezentare în binar. 0010 0001 0001 0000

La începutul execuţiei programului registrul de instrucţiuni si registrul de programe sunt au valoarea zero. În prima fază se încarcă în registrul de programe instrucţiunea de la adresa *00h* şi se incrementeză cu 2 contorul de programe. Datele aflate în registrul de instrucţiuni sunt interpretate de Unitatea de Comandă care dă ordinele necesare. În paralel cu executarea instrucţiunii se încarcă registrul de instrucţiuni instrucţiunea de la adresa *02h*.
Registrii sistem:
PC: 02 0000 0010
IR: 21 10 0010 0001 0001 0000

02: 22 01 load (2) R2 (2), 1 (01)
0010 0010 0000 0001

În mod similar se acţionea la pasul următor. Se incrementeză contorul de program şi apoi se decodifică instrucţiunea. Unitatea de Comandă interpretează instrucţiunea ca o comanda *load* şi seteză registrul *R2* la valoarea *01h*.
Registrii sistem vor arăta astfel:

PC: 04 0000 0100
IR: 22 01 0010 0010 0000 0001

04: 20 00 load (2) R0 (0), 0 (00)
0010 0000 0000 0000

În timpul execuţiei de la adresa *02h* în registrul de instrucţiuni se încarcă instrucţiunea de la adresa *04h*. Dupa terminarea execuţiei de la la adresa *02h* se incrementează contorul de programe cu 2. Instrucţiunea este decodificată iar UC (Unitatea de Comandă) se conformează şi încarcă în registrul *R0* date ce reprezintă valoarea *00h*.
Registrii sistem arată astfel:

Lucrarea de laborator numarul 5

PC: 06 0000 0110
IR: 20 00 0010 0000 0000 0000

NextChar:06: D0 F1 load (D0) RF (F), [R1](1)-adresa aflată în registrul R1
1101 0000 1111 0001

În timpul execuției, în registrul de instrucțiuni s-a încărcat instrucțiunea de la adresa *06h*. Acum contorul de programe este incrementat cu 2 şi instrucțiunea este decodificată. UC execută instrucțiunea şi încarcă în registrul *RF* datele de la adresa conținută în registrul *R1*.

Regiştrii sistem:
PC: 08 0000 1000
IR: D0 F1 1101 0000 1111 0001

08: 51 12 addi (5) R1(1),R1(1),R2(2)
0101 0001 0001 0010

În urma operației *fetch* în registrul de instrucțiuni se află instrucțiunea indicată de contorul de program. Contorul de program este incrmentat cu valoarea 2, indicând instrucțiunea de la adresa *0Ah*. Instrucțiunea este decodificată de către UC, iar aceasta trimite unitații aritmetico-logice conținuturile rgistrilor *R1* şi *R2* si comanda adunarea lor în complement față de 2, iar rezultatul este stocat în registrul *R1*.

Regiştrii sistem:
PC: 0A 0000 1010
IR: 51 12 0101 0001 0001 0010

0A: BF 0E jmpEQ(B) RF(F)=R0(0) , Ready-eticheta este înlocuită
1011 1111 0000 1110

În timpul execuției instrucțiunea indicată de contorul de instrucțiuni este încarcată în registrul de instrucțiuni. După terminarea execuției instrucțiunii precendente contorul de programe este incrementat cu 2. Instrucțiunea *BF 0E* este decodificată, interpretată ca instrucțiune de salt la adresa *0Eh* dacă registrele R0 şi RF au acelaşi coținut. Pentru aceasta contorul de programe este setat la valoarea *0Eh*. Dacă condiția aceasta nu este îndeplinită contorul de programe este incrementată cu 2.

Lucrarea de laborator numarul 5

Registrii sistem:

PC: 0C-0000 1011 daca condiția nu ester îndeplinită sau 0E-0000 1110 dacă condiția de salt este îndeplinită
IR: BF 0E 1011 1111 0000 1110

0C: B0 06 jmp(B0) NextChar (06)
1011 0000 0000 0110

Instrucțiunea care este încarcată în registrul de programe în următorul ciclu fetch, dacă condiția de la instrucțiunea nu este îndeplinită este o instrucțiune de salt necondiționat. Ea este decodata de UC, iar în urma decodării UC încarcă in PC adresa instrucțiunii care va trebui să fie executată.

Registrii sistem arată astfel:

PC: 06 0000 0110
IR: B0 06 1011 0000 0000 0110

Ready: 0E: C0 00 halt
1100 0000 0000 0000

Ultima instrucțiune din program este o instrucține de oprire a execuției programului. Ea este executată în acest program numai dacă condiția de salt de la instrucțiunea de la adresa *0Ah* a fost îndeplinită.

În următoarea parte se prezintă felul în care sunt reprezentate în memorie caracterle care vor fi afişate pe ecran. Se observă ca în fiecare celula de memorie este scrisa reprezentarea în hexadecimal a codului ASCII corespunzător fiecărui caracter.

10: 0A \n
11: 48 *H*
12: 65 *e*
13: 6C *l*
14: 6C *l*
15: 6F *o*
16: 20
17: 77 *w*
18: 6F *o*
19: 72 *r*
1A: 6C *l*
1B: 64 *d*
1C: 20
1D: 21 *!*
1E: 21 *!*
1F: 0A \n
20: 20
21: 20

Lucrarea de laborator numarul 5

22: 20
23: 20
24: 66 *f*
25: 72 *r*
26: 6F *o*
27: 6D *m*
28: 20
29: 74 *t*
2A: 68 *h*
2B: 65 *e*
2C: 0A \n
2D: 20
2E: 20
2F: 53 *S*
30: 69 *i*
31: 6D *m*
32: 70 *p*
33: 6C *l*
34: 65 *e*
35: 20
36: 53 *S*
37: 75 *i*
38: 6D *m*
39: 75 *u*
3A: 6C *l*
3B: 61 *a*
3C: 74 *t*
3D: 6F *o*
3E: 72 *r*
3F: 0A \n

Lucrarea de laborator numarul 5

Exercitii 1:

1. Sa se scrie programul urmator în formă hexadecimala şi apoi în formă binară.

   00:load R1,[0x0A]
   02:load R2,0x0B
   04:addi R3,R1,R2
   06:halt

2. Se da urmatorul program scris în formă hexadecimală. Să se converetească in limbaj de asamblare.
   00:21 01
   02:40 12
   03:5F 12
   04:C0 00
3. Sa se scrie un program in limbaj de asamablare care sa adune numerele 1d,2d,3d,4d,10h,ABh, 0011 1010b. Rezulatul va fi plasat in registrul R7.
4. Sa se scrie un program in limbaj de asamblare care sa afiseze de la inceput la inceput si de la sfarsit la inceput un sirul de caractere ascii aflate in memorie. Sirul este descris mai jos.Se cunosc adresa de inceput si caracterul care termina sirul.
   start_sir db "luna"
   db 0h
5. Sa se scrie un program in limbaj de asamblare care sa numere de cate ori apare caracterul 'a' in urmatorul sir de caractere aflat in memorie*."zdreanta cel cu ochii de faianta"*.
6. Sa se scrie un program in limbaj de asamblare care sa numere cuvintele din urmatorul sir de caractere. Cuvintele sunt subsiruri de caractere despartite de unul sau mai multe caractere ' '.
   "Să vedem cum ar trebui codificate instrucţiunile unui calculator obişnuit."

Lucrarea de laborator numarul 5

Exercitii 2:

1. Sa se scrie programul urmator în formă hexadecimala şi apoi în formă binară.

```
        00:load R1,string
        02:load R2,-1
        04:addi RF,R1,R2
        06:halt
string: 08:db "ABC"
```

2. Se da urmatorul program scris în formă hexadecimală. Să se convereteasc̆ă in limbaj de asamblare.

```
00:20 13
02:21 10
04:22 01
06:D0 F1
08:51 21
0A:B1 0E
0C:B0 06
0E:C0 00
```

3. Sa se scrie un program in limbaj de asamablare care sa adune numerele 31h, 3h, 12h, 0101b, 36h. Rezulatul va fi plasat in registrul R7.
4. Sa se scrie un program in limbaj de asamblare care sa afiseze de la inceput la inceput si de la sfarsit la inceput un sirul de caractere ascii aflate in memorie. Sirul este descris mai jos.Se cunosc adresa de inceput si caracterul care termina sirul.

   start_sir: db "acesta este un program care demonstreaza afiserea sirurilor – srevin sircs etse lujasem muca"
   db 0h
5. Sa se scrie un program in limbaj de asamblare care sa numere de cate ori apare caracterul 'c' in urmatorul sir de caractere aflat in memorie.*" Poate veţi fi surprinşi să aflaţi că lista instrucţiunilor în cod maşină este destul de scurtă.".*
6. Sa se scrie un program in limbaj de asamblare care sa numere cuvintele din urmatorul sir de caractere. Cuvintele sunt subsiruri de caractere despartite de unul sau mai multe caractere ' '.

   "Primele calculatoare nu excelau în flexibilitate, deoarece programul executat de fiecare dispozitiv era cablat în unitatea de comandă ca o parte a sistemului."

Lucrarea de laborator numarul 5

Exercitii 3:

1. Sa se scrie programul urmator în formă hexadecimala şi apoi în formă binară.

```
        00:load R0,10
        02:load R1,1
        04:load R2,1
        08:load R3,0
salt:   0A:addi R1,R1,R2
        0C:addi R3,R3,R2
        0E:jmpEQ R0=R3 stop
        10:jmp salt
stop:   12:halt
```

2. Se da urmatorul program scris în formă hexadecimală. Să se converetească in limbaj de asamblare.
   00:21 01
   02:22 F9
   04:5F 12
   06:C0 00
3. Sa se scrie un program in limbaj de asamablare care sa adune numerele 35h, 2Bh, 21h ,00010100,-1d. Rezulatul va fi plasat in registrul R7.
4. Sa se scrie un program in limbaj de asamblare care sa afiseze de la inceput la inceput si de la sfarsit la inceput un sirul de caractere ascii aflate in memorie. Sirul este descris mai jos.Se cunosc adresa de inceput si caracterul care termina sirul.
   start_sir: db " sir citit de la inceput - tirfas al ed titic ris"
   db 0h
5. Sa se scrie un program in limbaj de asamblare care sa numere de cate ori apare caracterul 'c' in urmatorul sir de caractere aflat in memorie.*" Poate veţi fi surprinşi să aflaţi că lista instrucţiunilor în cod maşină este destul de scurtă.".*
6. Sa se scrie un program in limbaj de asamblare care sa numere cuvintele din urmatorul sir de caractere. Cuvintele sunt subsiruri de caractere despartite de unul sau mai multe caractere ' '.
   "Conceptul de program stocat în memorie a devenit în prezent soluţia standard de lucru"

Lucrarea de laborator numarul 5

Exercitii 4:

1. Sa se scrie programul urmator în formă hexadecimala şi apoi în formă binară.

```
        00:load R0,10
        02:load R1,0
        04:load R2,1
        08:load R3,1
salt:   0A:addi R3,R3,R2
        0C:addi R4,R4,R3
        0E:addi R1,R1,R2
        10:jmpEQ R0=R1 stop
        12:jmp salt
stop:   14:halt
```

2. Se da urmatorul program scris în formă hexadecimală. Să se converetească in limbaj de asamblare.

   00:20 0E
   02:21 0F
   04:B1 0A
   06:5F 10
   08:B0 0C
   0A:50 10
   0C:C0 00

3. Sa se scrie un program in limbaj de asamablare care sa adune numerele 00010000b, 2Bh, 27h, 03d, A5h, -3d. Rezulatul va fi plasat in registrul R7.
4. Sa se scrie un program in limbaj de asamblare care sa afiseze de la inceput la inceput si de la sfarsit la inceput un sirul de caractere ascii aflate in memorie. Sirul este descris mai jos.Se cunosc adresa de inceput si caracterul care termina sirul.

   start_sir: db "acesta este un program demonstrativ – srevni sircs"
   db 0h
5. Sa se scrie un program in limbaj de asamblare care sa numere de cate ori apare caracterul 'c' in urmatorul sir de caractere aflat in memorie*." Poate veţi fi surprinşi să aflaţi că lista instrucţiunilor în cod maşină este destul de scurtă.".*
6. Sa se scrie un program in limbaj de asamblare care sa numere cuvintele din urmatorul sir de caractere. Cuvintele sunt subsiruri de caractere despartite de unul sau mai multe caractere ' '.

   "După cum am menţionat anterior, de obicei se utilizează celule care au opt biţi, aşa că vom considera că şi celulele de memorie ale calculatorului nostru au această mărime."

Lucrarea de laborator numarul 5

Exercitii 5:

1. Sa se scrie programul urmator în formă hexadecimala şi apoi în formă binară.

```
        00:load R0,stop_string
        02:load R1,start_string
        04:load R2,1
        06:load R3,[R1]
        08:move RF,R3
salt:   0A:addi R1,R1,R2
        0C:addi R1,R1,R2
        0E:jmpEQ R0=R1 stop
        10:jmp salt
stop:   12:halt
start_string:db “sir de caractere”
stop_string:db 0x0
```

2. Se da urmatorul program scris în formă hexadecimală. Să se converetească in limbaj de asamblare.

```
00:20 0A
02:21 01
04:22 01
06:23 18
08:24 19
0A:54 43
0C:53 32
0E:51 12
10:40 1F
12:B1 16
14:B0 0A
16:C0 00
```

3. Sa se scrie un program in limbaj de asamablare care sa adune numerele 21h, 45h, 34h, 10h. Rezulatul va fi plasat in registrul R7.
4. Sa se scrie un program in limbaj de asamblare care sa afiseze de la inceput la inceput si de la sfarsit la inceput un sirul de caractere ascii aflate in memorie. Sirul este descris mai jos.Se cunosc adresa de inceput si caracterul care termina sirul.

   start_sir: db “acesta este un program care demonstreaza afiserea sirurilor – srevin sircs etse lujasem muca”

   db 0h
5. Sa se scrie un program in limbaj de asamblare care sa numere de cate ori apare caracterul ‘c’ in urmatorul sir de caractere aflat in memorie.*” Poate veţi fi surprinşi să aflaţi că lista instrucţiunilor în cod maşină este destul de scurtă.”*.
6. Sa se scrie un program in limbaj de asamblare care sa gaseasca de cate ori substringul “in” apare in urmatorul string.

“Apoi, unitatea de comandă analizează instrucțiunea din registrul său de la instrucțiuni şi trage concluzia că trebuie să încarce în registrul 5 conținutul celulei de memorie de la adresa 6C.”