Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Курсовой проект

по дисциплине

“Архитектура ЭВМ и периферийные устройства”

на тему

Программная модель SIMPLECOMPUTER.

Реализация транслятора с языка SIMPLEASSEMBLER.

Выполнила студентка Копытина Татьяна Алексеевна
Ф.И.О.

Группы ИП-013

Работу принял _____ Ю.С. Майданов
подпись

Оценка_____

Новосибирск – 2022

СОДЕРЖАНИЕ:

Задание……..….3

Алгоритм……4

Транслятор с языка Simple Assembler……..…5

Тестирование программы…….6

Листинг программы……..9

- Файл **main.c**……9
- Файл **sat.c**……..20

Задание:

В рамках курсовой работы, необходимо доработать модель SimpleComputer так, чтобы она обрабатывала команды, записанные в оперативной памяти.

Необходимо реализовать две функции:

int ALU (intcommand, intoperand) – реализует алгоритм работы арифметико-логического устройства. Если возникает ошибка при выполнении функции, которая не позволяет дальше выполнять программу, то функция возвращает -1, иначе 0;

int CU (void) – обеспечивает работу устройства управления.

Алгоритм.

Сначала из оперативной памяти считывается ячейка, адрес которой находится в регистре **instructionCounter**, полученное значение декодируется как команда. Если декодирование невозможно, то устанавливается флаги: «указана неверная команда» и «игнорирование тактовых импульсов», и работа функции прекращается. Если получена арифметическая или логическая операция, то вызывается функция ALU, иначе команда выполняется самим устройством управления. Затем определяется, какая команда должна быть выполнена следующей и адрес её ячейки памяти заносится в регистр **instructionCounter**.

Транслятор с языка Simple Assembler.

Чтобы программа смогла быть обработана SimpleComputer, для этого был реализован транслятор, который переводит текст SimpleAssembler в бинарный формат, который потом считывается консолью управления. Мною была реализована следующая программа на SimpleAssembler:

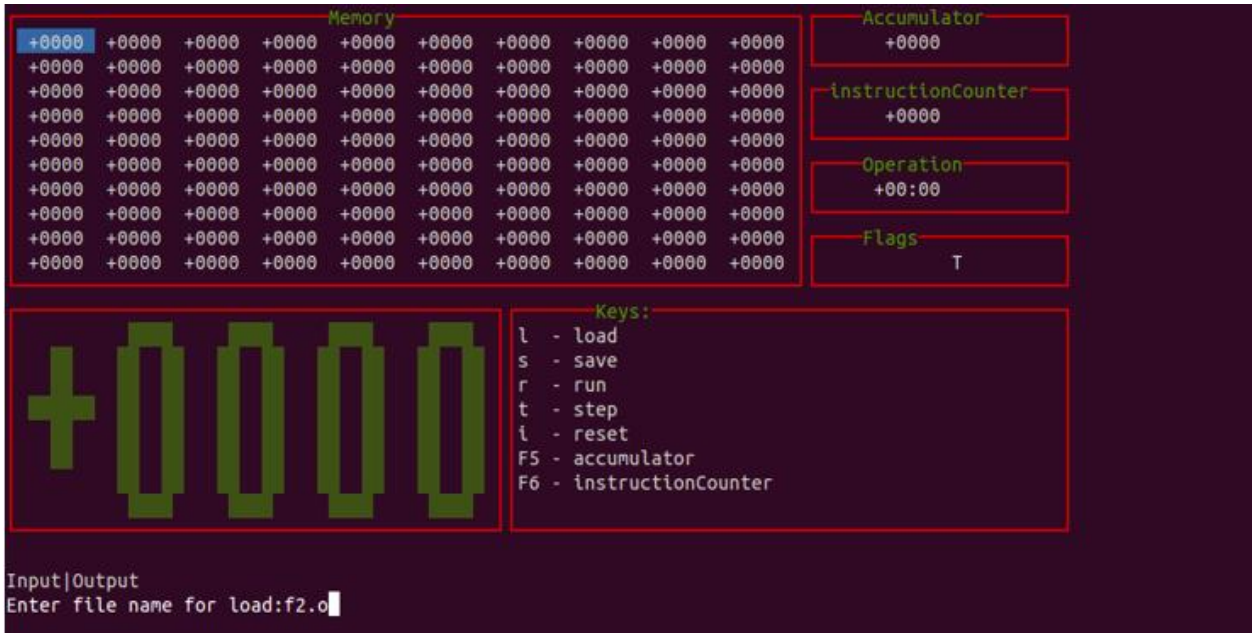
Программа на SimpleAssembler (Нахождение факториала):

```
00 READ 15        //Вводим с терминала в указанную ячейку пам. Знач.
01 LOAD 16        //Загружаем в аккумулятор знач. из указанной ячейки памяти.
02 MUL15          //Вычисляем произведения знач. в аккум. и знач. данной ячей.
03 STORE 16       //Выгружаем значение из аккумулятора в указанную ячейку.
04 LOAD 15        //Загружаем в аккумулятор знач. из указанной ячейки памяти.
05 SUB17          //Вычитаем из значения в аккумуляторе значение данной ячей.
06 JZ09           //Переход к указанной ячейки, если в аккумуляторе 0.
07 STORE 15       //Выгружаем значение из аккумулятора в указанную ячейку.
08 JUMP 01        //переход к указанному адресу памяти
09 WRITE 16       //Вывод из терминала знач. указанной ячейки памяти.
10 HALT 00        //Остановка программы.
15 = +0000        //Заносим в 15-ю ячейку 0
16 = +0001        //Заносим в 16-ю ячейку 1
17 = +0001        //Заносим в 17-ю ячейку 1
```

Тестирование программы.

Консоль управления моделью SimpleComputer:

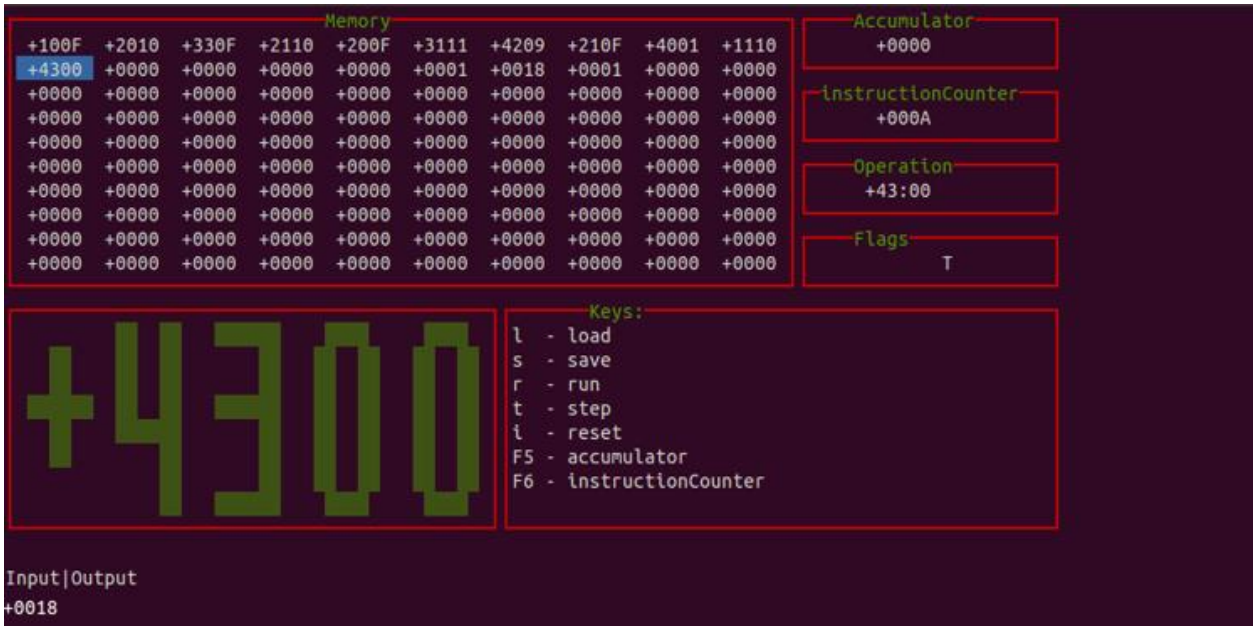
При нажатии на клавишу l, программа попросит ввести имя файла, в котором содержится информация переведенного с языка Assembler в бинарный формат.



Программа считывает информацию и заносит ее в память SimpleComputer.

При нажатии на r, мы запускаем нашу программу, теперь нужно ввести число, факториал которого нам нужно вычислить.

После нажатия на enter программа начинает вычислять факториал.



Проверка:

4! = 1*2*3*4 = 24. Программа вывела число 18 (число в шестнадцатеричной системе счисления), при переводе в десятичную систему счисления мы получаем число 24.

Листинг программы.

Файл main.c:

```
#include<math.h>
#include <stdlib.h>
#include <stdio.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include "mySimpleComputer.h"
#include "myTerm.h"
#include "myBigChars.h"
#include "myReadKey.h"


intBigChars[18][2];
intinsCounter;
intcur_cell;
intqwe=0;
intfd;
int accumulator;
enum keys button=-1;
structitimervaloldTimerVal;

voidcolorType(int type)
{
  switch(type)
  {
            case 1 : mt_setfgcolor (White);mt_setbgcolor (Violet); break;
            case 2 : mt_setbgcolor(Blue);mt_setfgcolor (White); break;
            default :mt_setfgcolor (White);mt_setbgcolor (Violet); break;
  }
}

intPrintMemory()
{
inti,j=1,k;
  int command = 0, operand = 0, value = 0;
  for(i=0;i <100; i++)
  {
            if(i==insCounter) colorType(2);
            if(i%10==0) j++;

            sc_memoryGet(i, &value);
            k = value >> 14;

            sc_commandDecode(value, &command, &operand);
            mt_gotoXY(j,2+(i%10)*7);

            printf(" %c%02d%02X ",(k == 0)?'+':'-',command,operand);
            colorType(1);
  }
```

```
  mt_gotoXY(25, 0);

  return 0;
}


voidDrawAcc()
{
int command = 0, operand = 0;

sc_commandDecode(accumulator, &command, &operand);

mt_gotoXY(2, 80);
printf("%s%02d%02X", ((accumulator >> 14) & 1 == 1 )? "-": "+",command, operand);

mt_gotoXY(25, 0);
}

voidDrawInsCounter()
{
mt_gotoXY(5, 80);
printf("%c%04X",((insCounter>> 14) & 1 == 1 )? '-': '+',insCounter);

mt_gotoXY(25, 0);


voidDrawOperationField()
{

int value = 0, command = 0, operand = 0;
sc_memoryGet(insCounter, &value);
sc_commandDecode(value, &command, &operand);
mt_gotoXY(8, 79);
printf("%s%02d:%02X", ((value >> 14)== 1 ? "-": "+"), command, operand);

mt_gotoXY(25, 0);
}


voidDrawFlags()
{
int flag = 0;

mt_gotoXY(11,80); sc_regGet( P ,&flag); if ( flag == 1 ) printf("P"); else printf(" ");
  mt_gotoXY(11,82); sc_regGet( O ,&flag); if ( flag == 1 ) printf("O"); else printf(" ");
  mt_gotoXY(11,84); sc_regGet( M ,&flag); if ( flag == 1 ) printf("M"); else printf(" ");
  mt_gotoXY(11,86); sc_regGet( T ,&flag); if ( flag == 1 ) printf("T"); else printf(" ");
  mt_gotoXY(11,88); sc_regGet( E ,&flag); if ( flag == 1 ) printf("E"); else printf(" ");

mt_gotoXY(25, 0);
}

voidDrawBigChars()
{

int value = 0, command = 0, operand = 0;

sc_memoryGet(insCounter, &value);
sc_commandDecode(value, &command, &operand);

if ((value >> 14) == 0)bc_printbigchar(BigChars[16],13,1,Green,Violet);
```

```
  elsebc_printbigchar(BigChars[17],13,1,Green,Violet);

  bc_printbigchar(BigChars[command/10],13,9,Green,Violet);
  bc_printbigchar(BigChars[command%10],13,18,Green,Violet);
  bc_printbigchar(BigChars[operand/16],13,27,Green,Violet);
  bc_printbigchar(BigChars[operand%16],13,36,Green,Violet);

mt_gotoXY(25, 0);

}

void CLR()
{
  mt_gotoXY(25,0); printf("                                                    \n");
  mt_gotoXY(26,0); printf("                                                    \n");

  mt_gotoXY(25, 0);
}


voidDrawFields()
{

PrintMemory();
DrawAcc();
DrawInsCounter();
DrawOperationField();
DrawFlags();
DrawBigChars();


}

voidDrawConsole()
{
  colorType(1);
  mt_clrscr ();

  mt_setfgcolor(Red);
  bc_box(46, 13, 96,23);
  mt_gotoXY (13, 54);
  mt_setfgcolor(Green);
  printf("Keys:");

  mt_setfgcolor(White);
  mt_gotoXY (14, 47);
  printf("l  - load");
  mt_gotoXY(15,47);
  printf("s  - save");
  mt_gotoXY(16,47);
  printf("r  - run");
  mt_gotoXY(17,47);
  printf("t  - step");
  mt_gotoXY(18,47);
  printf("i  - reset");
  mt_gotoXY(19,47);
  printf("F5 - accumulator");
  mt_gotoXY(20,47);
  printf("F6 - instructionCounter");

  mt_setfgcolor(Red);
```

```
  bc_box(0, 1, 72, 13);
  mt_gotoXY (0, 30);
  mt_setfgcolor(Green);
  printf("Memory");
  mt_setfgcolor(Red);
  bc_box(73, 1, 96, 4);
  mt_gotoXY (1, 78);
  mt_setfgcolor(Green);
  printf("Accumulator");
  mt_setfgcolor(Red);
  bc_box(73, 4, 96,7);
  mt_gotoXY (4, 75);
  mt_setfgcolor(Green);
  printf("instructionCounter");
  mt_setfgcolor(Red);
  bc_box(73, 7, 96,10);
  mt_gotoXY (7, 78);
  mt_setfgcolor(Green);
  printf("Operation");
  mt_setfgcolor(Red);
  bc_box(73, 10, 96,13);
  mt_gotoXY (10, 78);
  mt_setfgcolor(Green);
  printf("Flags");
  mt_setfgcolor(Red);
  bc_box(0, 13,45,23);

  mt_setfgcolor(White);
  mt_gotoXY(24,0);
  printf("Input|Output");

  DrawFields();
mt_gotoXY(25, 0);
}

int ALU(int command, int operand)
{
intflagE, flagO, flagM;
int value = 0, value1 = 0, value2 = 0;

switch (command)
  {
case 30:
if (sc_memoryGet(operand, &value) == 0)
        {
                        accumulator = accumulator + value;
if (accumulator > 0x7FFF)
           {
accumulator = accumulator & 0x7FFF;
sc_regSet(P, 1);
           }
        }
break;

case 31:
if (sc_memoryGet(operand, &value) == 0)
        {
accumulator = accumulator - value;
if (accumulator > 0x7FFF) {
accumulator = accumulator & 0x7FFF;
sc_regSet(P, 1);
```

```
            }
        }
break;

  case 61:
if (sc_memoryGet(operand, &value) == 0)
        {
if (value != 0) {
accumulator = sqrt(value);
if (accumulator > 0x7FFF) {
accumulator = accumulator & 0x7FFF;
sc_regSet(P, 1);
              }
            }
elsesc_regSet(O, 1);
        }
break;

case 32:

if (sc_memoryGet(operand, &value) == 0)
        {
if (value != 0) {
accumulator = accumulator / value;
if (accumulator > 0x7FFF) {
accumulator = accumulator & 0x7FFF;
sc_regSet(P, 1);
              }
            }
elsesc_regSet(O, 1);
        }
break;

case 33:
if (sc_memoryGet(operand, &value) == 0)
        {
accumulator = accumulator * value;
if (accumulator > 0x7FFF) {
accumulator = accumulator & 0x7FFF;
sc_regSet(P, 1);
            }
        }
break;


default:
sc_regSet(E, 1);
break;
  }

sc_regGet(E, &flagE);
sc_regGet(O, &flagO);
sc_regGet(M, &flagM);

if ((flagE == 1) || (flagO == 1) || (flagM == 1)) return -1;
else return 0;
}

int CU()
{
charbuf[5];
```

```
        intflagE, flagO, flagM;
        int value = 0, value1 = 0, command = 0, operand = 0, flagT;

        if ((sc_memoryGet(insCounter, &value) == 0) && (sc_commandDecode(value, &command, &operand)
== 0))
          {
        switch (command)
                {
        case 10:
        sc_regGet(T, &flagT);
        if (flagT == 0) sc_regSet(T, 1);
        rk_mytermrestore();
        CLR();
        printf("memory value:");
                                        scanf("%X", &value);
        rk_mytermregime(0, 1, 1, 0, 1);
        CLR();
        if (sc_memorySet(operand, value) == 0)
                        {
        PrintMemory();
        DrawBigChars();
        DrawOperationField();
        insCounter++;
                        }
        else write(1, "Error memory value", 30);

        if (flagT == 0) sc_regSet(T, 0);
        break;

        case 11:
        if ((sc_memoryGet(operand, &value) == 0) && (sc_commandDecode(value, &command, &operand) ==
0))
                        {
        CLR();
        sprintf(buf, "%s%02X%02X", ((value >> 14) & 1 == 1 ? "-": "+"), command, operand);
        write(1, buf, sizeof(buf));
        insCounter++;
                        }
        break;

        case 20:
        if (sc_memoryGet(operand, &value) == 0)
                        {
        accumulator = value;
        insCounter++;
                        }
        break;

        case 21:
        sc_memorySet(operand, accumulator);

        insCounter++;
        break;

        case 40:
        insCounter=operand;
        break;
                                        case 41:
        if (((accumulator >> 14) & 1) == 1) insCounter=operand;
        elseinsCounter++;
        break;
```

```
case 42:
if (accumulator == 0) insCounter=operand;
elseinsCounter++;
break;

case 43:
sc_regSet(T, 1);
break;

case 56:
sc_regGet(P, &value);
                                        if (value == 1) insCounter=operand;
elseinsCounter++;
break;

default:
if (ALU(command, operand) == 0)
insCounter++;
break;
        }
   }

sc_regGet(E, &flagE);
sc_regGet(O, &flagO);
sc_regGet(M, &flagM);


if ((flagE == 1) || (flagO == 1) || (flagM == 1))
   {
sc_regSet(T, 1);
return -1;
   }
else return 0;
}


voidConsoleDefault()
{
  sc_memoryInit();
sc_regInit();
sc_regSet(T, 1);
accumulator = 0;
insCounter = 0;
}


voidtimerHandler(intsigno)
{
intflagT;
  sc_regGet(T, &flagT);
  if ((signo == SIGALRM) && (flagT == 0))
  {
            CU();
DrawFields();
mt_gotoXY (25, 0);
  }
 }

intrunTimer()
{
```

```
structitimervalnval,oval;
structsigactionnvec;
  signal(SIGALRM, timerHandler);
  nval.it_value.tv_sec = 0;
nval.it_value.tv_usec = 20;
  nval.it_interval.tv_sec = 0;
nval.it_interval.tv_usec = 20;
  if (setitimer(ITIMER_REAL, &nval, &oldTimerVal) != 0) return -1;
  return 0;
}

intReadBC()
{
  int count;
  fd = open ("BC.bin", O_RDONLY );
  bc_bigcharread (fd, BigChars ,18, &count);
  close(fd);
  return 0;
}

int main()
{
  intflagT = 0, value = 0, command = 0, operand = 0, iC = 0;
  introws,cols;
  char filename[20];

  mt_clrscr();
  mt_getscreensize (&rows, &cols);
  if ((rows<24)||(cols<97))
  {
        printf("Small size of terminal\n");
        return -1;
  }

  if (rk_mytermsave() == -1) return -1;

  ReadBC();

  ConsoleDefault();
  runTimer();
  DrawConsole();

  rk_mytermregime (0, 0, 0, 0, 1);
  while(1)
  {
        mt_gotoXY (25, 0);
        rk_mytermregime (0, 0, 0, 0, 1);
        sc_regGet(T,&flagT);

        rk_readkey(&button);

        switch(button)
        {
              case K_ESC:
              {
                    if (rk_mytermrestore() == -1) return -1;
                    return 0;
              }break;
              case K_RIGHT:
              {
                    if (flagT == 0) continue;
```

```
                        if ((insCounter+1)>=100 ) insCounter=0;
                        elseinsCounter++;
                        DrawFields();
                }break;
                case K_UP:
                {
                        if (flagT == 0) continue;

                        if ( (insCounter-10)<0) insCounter+=90;
                        elseinsCounter-=10;
                        DrawFields();
                }break;
                case K_DOWN:
                {
                        if (flagT == 0) continue;

                        if ( (insCounter+10)>=100 ) insCounter-=90;
                        elseinsCounter+=10;
                        DrawFields();
                }break;
                case K_LEFT:
                {
                        if (flagT == 0) continue;

                        if ( (insCounter-1)<0 ) insCounter=99;
                        elseinsCounter--;
                        DrawFields();
                }break;
                case K_L:
                {
                        if (flagT == 0) continue;

                        rk_mytermrestore();
CLR();
printf("Enter file name for load:");
scanf("%s",filename);
rk_mytermregime(0, 0, 0, 0, 1);
CLR();
if (sc_memoryLoad(filename) == 0)
        {
insCounter = 0;
sc_regInit();
sc_regSet(T, 1);
accumulator = 0;
DrawFields();
printf("File loaded");
        }
elseprintf("Error load file");

                }break;
                case K_S:
                {
                        if (flagT == 0) continue;

                        rk_mytermrestore();
CLR();
printf("Enter file name for save:");
scanf("%s",filename);

rk_mytermregime(0, 0, 0, 0, 1);
```

```
CLR();
if (sc_memorySave(filename) == 0) printf("File saved");
elseprintf("Error save file");
                        }break;
                        case K_I:
                        {
                                if (flagT == 0) continue;

                                ConsoleDefault();
                                DrawFields();
                        }break;
                        case K_F5:
                        {
                                if (flagT == 0) continue;

                                rk_mytermrestore();
CLR();
                                printf("Accumulator:");
                                scanf("%d", &accumulator);
DrawAcc();
CLR();

                                rk_mytermregime(0, 0, 0, 0, 1);

                        }break;

                        case K_F6:
                        {
                                if (flagT == 0) continue;

                                rk_mytermrestore();
CLR();
                                printf("instructionCounter:");
                                scanf("%d", &iC);
                                CLR();
            if(iC<0 || iC>99) { printf("Error value of insCounter"); insCounter = 0; sc_regSet(M,1);}
                                else {insCounter = iC; sc_regSet(M,0);}
DrawFields();

                                rk_mytermregime(0, 0, 0, 0, 1);
                        }break;

                        case K_ENTER:
                        {
                                if (flagT == 0) continue;

                                rk_mytermrestore();
                                CLR();
                                mt_gotoXY(25,0);
                                printf("Command: ");
                                scanf("%d",&command);
                                CLR();
                                mt_gotoXY(25,0);
                                printf("Operand: ");
                                scanf("%d",&operand);

rk_mytermregime(0, 0, 0, 0, 1);
                                CLR();

                                if (sc_commandEncode(command, operand, &value) == 0)
{
```

```
sc_memorySet(insCounter, value);
DrawFields();
            }
else {mt_gotoXY(25,0);printf("Error command or operand value");}
                        }break;

                        case K_T:
                        {
if (flagT == 0) continue;

CLR();
                                    CU();
DrawFields();
                        }break;

                        case K_R:
                        {
if (flagT == 0) continue;

CLR();
insCounter = 0;
accumulator = 0;
sc_regInit();
DrawFields();
                        }break;
                }

  }
  rk_mytermrestore();
  return 0;
}
```

Файл sat.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "mySimpleComputer.h"

intasm_analis(char *comm)
{
if(strcmp(comm,"READ") == 0)return 10;
if(strcmp(comm,"WRITE") == 0)return 11;
if(strcmp(comm,"LOAD") == 0)return 20;
if(strcmp(comm,"STORE") == 0)return 21;
if(strcmp(comm,"ADD") == 0)return 30;
if(strcmp(comm,"SUB") == 0)return 31;
if(strcmp(comm,"DIVIDE") == 0)return 32;
if(strcmp(comm,"MUL") == 0)return 33;
if(strcmp(comm,"JUMP") == 0)return 40;
if(strcmp(comm,"JNEG") == 0)return 41;
if(strcmp(comm,"JZ") == 0)return 42;
if(strcmp(comm,"HALT") == 0)return 43;
if(strcmp(comm,"JC") == 0)return 56;
if(strcmp(comm,"KOR") == 0)return 61;

if(comm[0] == '=')return 1;
return -1;
}
int main(intargc, char** argv)
{

  char line[100],ch;
  charstrComm[10],operand_str[10];
  intvalue,command,operand,address;
  FILE *file, *file1;


if(argc != 3)
   {
printf("Usage: sat file.sa file.o\n ");
return -1;
   }
else if ((file = fopen (argv[1], "rb" )) <= 0)
   {
printf("Can`t open '%s' file.\n", argv[1]);
return -1;
   }


sc_memoryInit();
do{
```

```
  fgets(line,sizeof(line),file);
  if(sscanf(line,"%d %s %s",&address,strComm,&operand_str)<3)
  {
printf("Translation error1\n");
return -1;
  }

  command = asm_analis(strComm);

  if(command != -1)
  {
        if(command != 1)
  {
        if(sscanf(operand_str, "%d", &operand) !=1 ||
sc_commandEncode(command,operand,&value)==-1 || sc_memorySet(address,value)==-1)
  {
        printf("Translation error2\n");
        return -1;
  }
  }
  else
  {
        int a=0,b=0;
        sscanf(operand_str, "%c%02d%02X", &ch, &command, &operand);

        if (ch == '-') a |=(1 << 14);


        value=a + (command << 7)+operand;

        if(sc_memorySet(address,value)==-1)
  {
        printf("Translation error3\n");
        return -1;
  }

  }
    }
else
   {
  printf("Translation error4\n");
return -1;
    };
  }
  while(!feof(file));

  fclose(file);

  if (sc_memorySave(argv[2]) == -1)
   {
printf("Can`t create '%s' file.\n", argv[2]);
return -1;
   }

printf("Translation status: OK.\n");
```

```
  return 0;
}
```