Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»
Кафедра ВС

Курсовая работа
по дисциплине ««Архитектура ЭВМ»
Вариант 10

Выполнила: студентка группы ИП-013 Даськова Любовь Сергеевна

Проверил: доцент кафедры ВС
Майданов Юрий Сергеевич

Новосибирск 2022

# Оглавление

Условие задачи……2
Описание алгоритмов……3
Блок-схема……10
Результат……10
Листинг……12
Вывод……26
Список литературы……26

# 1. Условие задачи

В рамках курсовой работы необходимо доработать модель Simple Computer так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице 1. Из пользовательских функций необходимо реализовать команду CHL. Для разработки программ требуется создать транслятор с языка Simple Assembler.

Таблица 1. Команды центрального процессора Simple Computer

| Операция | | Значение |
|---|---|---|
| **Обозначение** | **Код** | |
| **Операции ввода/вывода** | | |
| READ | 10 | Ввод с терминала в указанную ячейку памяти с контролем переполнения |
| WRITE | 11 | Вывод на терминал значение указанной ячейки памяти |
| **Операции загрузки/выгрузки в аккумулятор** | | |
| LOAD | 20 | Загрузка в аккумулятор значения из указанного адреса памяти |
| STORE | 21 | Выгружает значение из аккумулятора по указанному адресу памяти |
| **Арифметические операции** | | |
| ADD | 30 | Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе) |
| SUB | 31 | Вычитает из слова в аккумуляторе слово из указанной ячейки памяти (результат в аккумуляторе) |
| DIVIDE | 32 | Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе) |
| MUL | 33 | Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе) |
| **Операции передачи управления** | | |
| JUMP | 40 | Переход к указанному адресу памяти |
| JNEG | 41 | Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число |
| JZ | 42 | Переход к указанному адресу памяти, если в аккумуляторе находится ноль |
| HALT | 43 | Останов, выполняется при завершении работы программы |
| CHL | 60 | Логический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе) |

# 2. Описание алгоритмов

## 2.1. Обработка тактовых импульсов

Отключить флаг игнорирования тактовых импульсов и начать выполнения программы, можно только с помощью кнопки “R”.

```
else if(key==KEY_R)
{
    rk_mytermsave();
    sc_regSet(IGNORING_CLOCK_PULSES, 0);
    sig();
}
```

За нажатием следует запуск системного таймера.

```
void sig()
{
    struct itimerval nval,oval;
    nval.it_interval.tv_sec=1;
    nval.it_interval.tv_usec=0;
    nval.it_value.tv_sec=1;
    nval.it_value.tv_usec=0;
    setitimer(ITIMER_REAL,&nval,&oval);
}
```

Объявление сигнала в главной функции.

```
signal(SIGALRM,signalhandler);
signal(SIGUSR1,signalhandler);
```

Затем обрабатывается сам сигнал.

```
if(signo==SIGALRM&&value==0)
{
    f6(count_t);
    sleep(1);
    int value;
    sc_memoryGet(count_t, &value);
    int result=CU(value,count_t);
    if(result==1)
    {
        alarm(0);
        sc_regSet(IGNORING_CLOCK_PULSES, 1);
        menu_flag();
        mt_gotoXY(23,0);
    }
    else
    {
        sc_regSet(IGNORING_CLOCK_PULSES, 0);
        menu();
        if(result!=2)
            count_t++;
    }

}
```

Для прерывания используется пользовательский сигнал.

```
else if(signo==SIGUSR1)
{
    count_t=0;
    sc_regInit();
    alarm(0);
    menu();
}
```

## 2.2. Обработка команд

Обработка команд происходит в функции устройства управления.

```
int CU(int value_m,int cell_num)
```

Для начала команда загружается в дешифратор

```
void decoder(int value,int *number,int *operand)
{
    if(value|0x3fff==0x3fff)
    {
        int num,oper;
        num=value;
        num>>=7;
        *number=num;
        oper=value;
        oper&=0x7F;
        *operand=oper;
    }
    else
    {
        *number=0;
        *operand=0;
    }
}
```

в которой выделяется код команды и операнд.

Потом в функции CU происходит выполнение команд и контроль операнд. Если опернад больше допустимого значения, то программа останавливается и тактовые импульсы игнорируются.

Выполнение арифметических и логических операций происходит в блоке АЛУ контролируется переполнение аккумулятора и если такое происходит программа останавливается.

```
switch( number)
{
case 0x10:
        mt_gotoXY(23,0);
        printf("input value\n");
        rk_mytermsave2();
        rk_mytermrestore();
        scanf("%x",&value);
        rk_mytermrestore2();
        if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value<=0xffffffff))
        {
            sc_memorySet(operand,value);
        }
        else
        {
            sc_regSet(OVERFLOW, 1);
            return 1;
        }
        break;
```

```
int ALU(int operand,int command)
{
    int result,value;
    sc_memoryGet(operand,&value);
    switch(command)
    {
        case 0x30:
            result=value+accum;
            break;

        case 0x31:
            result=accum-value;
            break;
        case 0x32:
            if(value!=0)
                result=accum/value;
            else
                return 0xffff1;
            break;
        case 0x33:
            result=accum*value;
            break;
```

## 2.3. Обработка кнопок

```
else if(key==KEY_F6)
{
    int value;
    scanf("%d",&value);
    menu();
    if(value>-1&&value<100)
    {
        menu_iC(value);
        int x=value%10+2;
        int y=value/10*6+2;
        sw(0,x,y,value);
        mt_gotoXY(23,0);
        rk_readkey(&key);
        key_enter(key,value);
    }
}
else if(key==KEY_F5)
{
    int value;
    scanf("%x",&value);
    if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value<
    {
```

Каждая кнопка обрабатывается в своем блоке.

Блок для обработки прерывания тактовых импульсов.

```
 if(value==0)
 {
        raise(SIGUSR1);
        sc_regSet(IGNORING_CLOCK_PULSES, 1);
        menu();
 }
```

Передвижение по полю с памятью реализуется с помощью отдельной функции.

```
while(key==KEY_UP||key==KEY_DOWN||key==KEY_RIGHT||key==KEY_LEFT)
{
    if(key==KEY_UP)
    {
        if(X-1>1)
        {
            sw(1,X,Y,count);
            X-=1;
            count--;
            sw(0,X,Y,count);
        }
    }
    else if(key==KEY_DOWN)
    {
        if((X+1)<12)
        {
            sw(1,X,Y,count);
            X+=1;
```

в конце передвижения определяется не нажата ли кнопка “e” (эквивалентно enter)

```
    menu_iC(count);
    mt_gotoXY(23,0);
    rk_readkey(&key);
```

```
void key_enter(enum keys key, int count)//Pë C,
{
    if(key==KEY_ENTER)
    {
        int value;
        scanf("%x",&value);
        if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value<=0xffffffff))
        {
            sc_memorySet(count,value);
        }
    }
    else if(key==KEY_T)
    {
        count_t=count;
        menu_iC(count_t);
        rk_mytermsave();
        int memory;
        sc_memoryGet(count, &memory);
        if(CU(memory,count)==1||CU(memory,count)==2)
        {
            menu_flag();
            mt_gotoXY(23,0);
```

В этой же функции проверяется не нажата ли кнопка “t” которая выполняет одну команду.

## 2.4. Графика

Обновление консоли может происходить в общей функции

```
void menu()
{
    mt_clrscr();
    menu_memory();
    menu_accum();
    menu_iC();
    menu_Oper();
    menu_flag();
    menu_key();
    BChar();
    printf("\nInput/Output\n");
    mt_gotoXY(23,0);

}
```

или отдельно, если требуется перерисовать только одну часть консоли.

Функция для отрисовки больших чисел

```
void BChar(int value=0x0)
{
    int X=13;
    int Y=1;
    int big[2];
    if(!(value&0x80000000))
    {
        big[0]=BC[16][0];
        big[1]=BC[16][1];
        bc_printbigchar(big,X,Y,Standart,Standart);
    }
    else
    {
        big[0]=BC[17][0];
        big[1]=BC[17][1];
        bc_printbigchar(big,X,Y,Standart,Standart);
        value=neg(value);
```

```
void sw(int fl_color,int X,int Y,int count)
{
    int value;
    if(fl_color==0)
        mt_setbgcolor(White);
    else
        mt_setbgcolor(Standart);
    mt_gotoXY(X,Y);
    sc_memoryGet(count, &value);
    if(value&0x80000000)
            printf("-%04x",neg(value));
        else
            printf("+%04x",value);
    BChar(value);
```

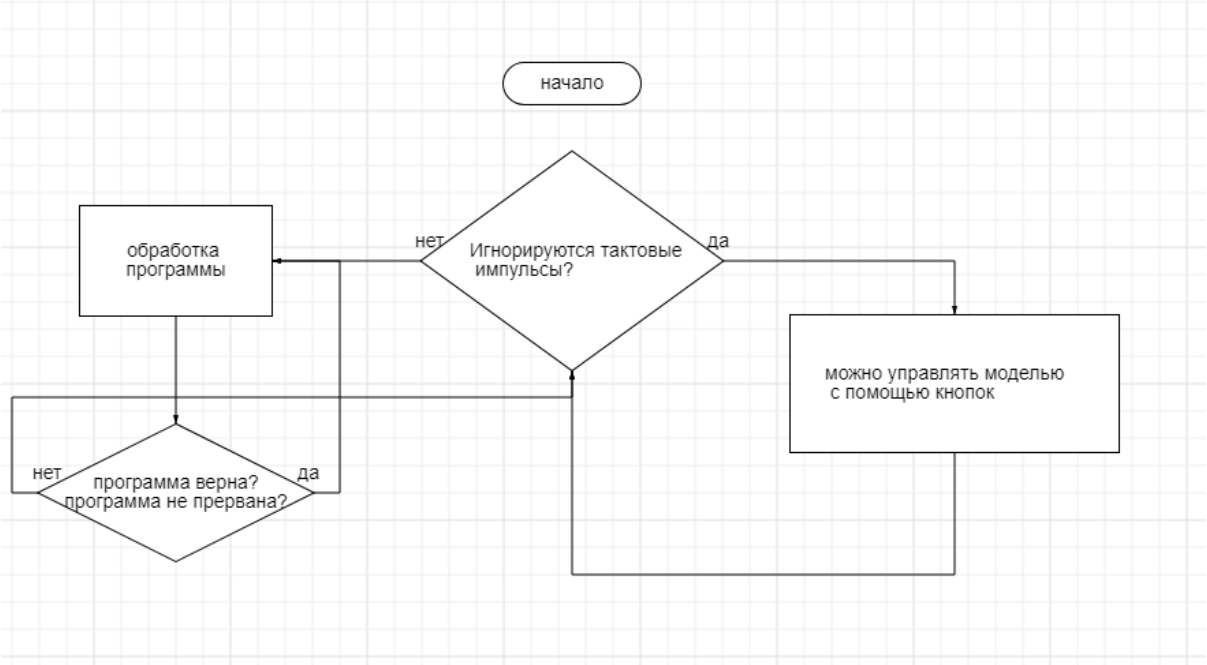
Функция для подсвечивания выбранного пункта меню.
Она стирает предыдущую выбранную клетку и рисует белым новую.

## 2.4. Simple Assembler

Сначала функция считывает одну строчку и отправляют ее в обработчик.
Функция убирает лишние пробелы до первого найденного символа, после этого функция считывает два символа в массив. Если массив остался пустым программа выдает ошибку и указывает строку с ошибкой. Затем используются строки. Пока в строке большие символы английского алфавита или = программа собирает символы в строку. Потом она отправляет их в отдельную функцию, где с помощью map ищется соответствующий номер команды. Если такого нет выдается ошибка. Потом программа ищет операнд, но если команда =, то программа ищет строку. Строка или операнд обрабатывается и потом все данные передаются в функцию которая шифрует машинную команду и записывает ее в массив. Массив сохраняется. Это и есть программа.

```
int checkCommand(string str)
{
    map <string,int> Command={
                        {"READ",0x10},
                        {"WRITE",0x11},
                        {"LOAD",0x20},
                        {"STORE",0x21},
                        {"ADD",0x30},
                        {"SUB",0x31},
                        {"DIVIDE",0x32},
                        {"MUL",0x33},
                        {"JUMP",0x40},
                        {"JNEG",0x41},
                        {"JZ",0x42},
                        {"HALT",0x43},
                        {"CHL",0x60},
                        {"=",0x1},
                      };
    map <string,int>::iterator i=Command.begin();
```

## 3. Блок-схема

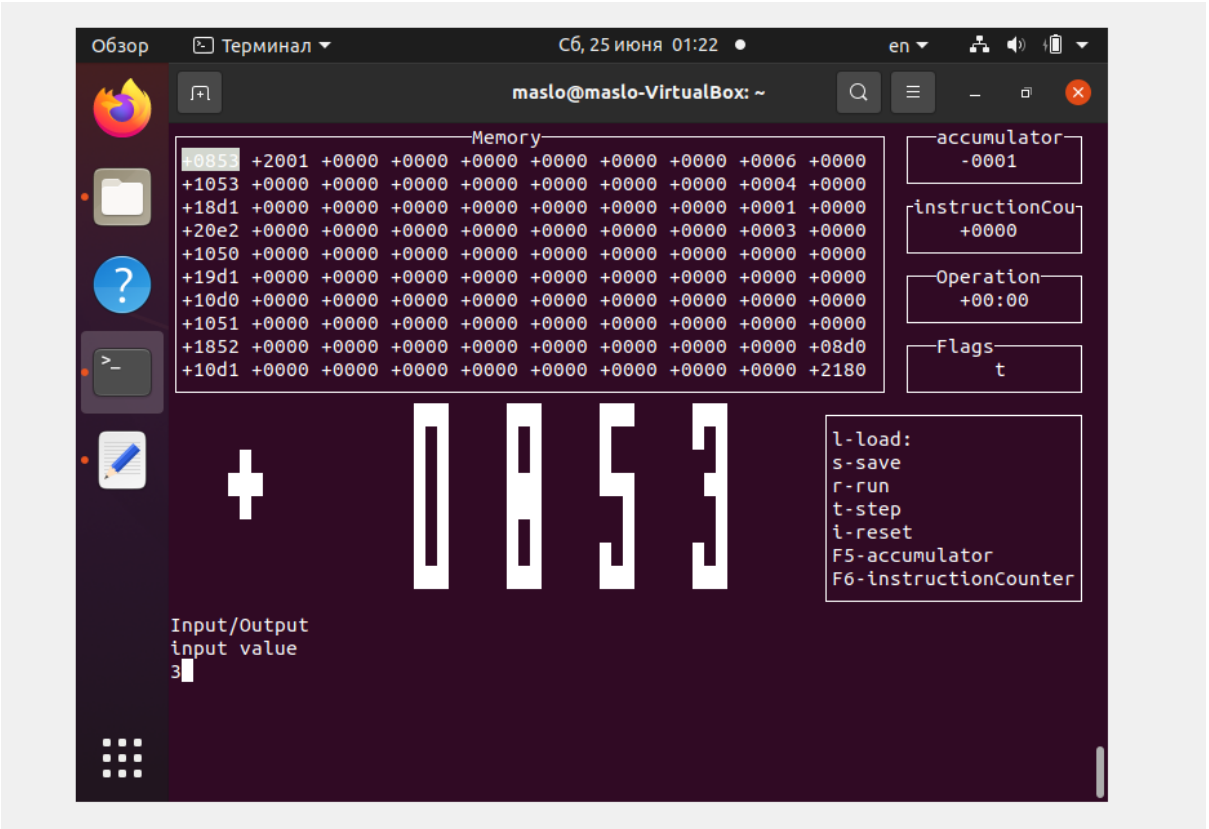



## 4. Результаты тестирования

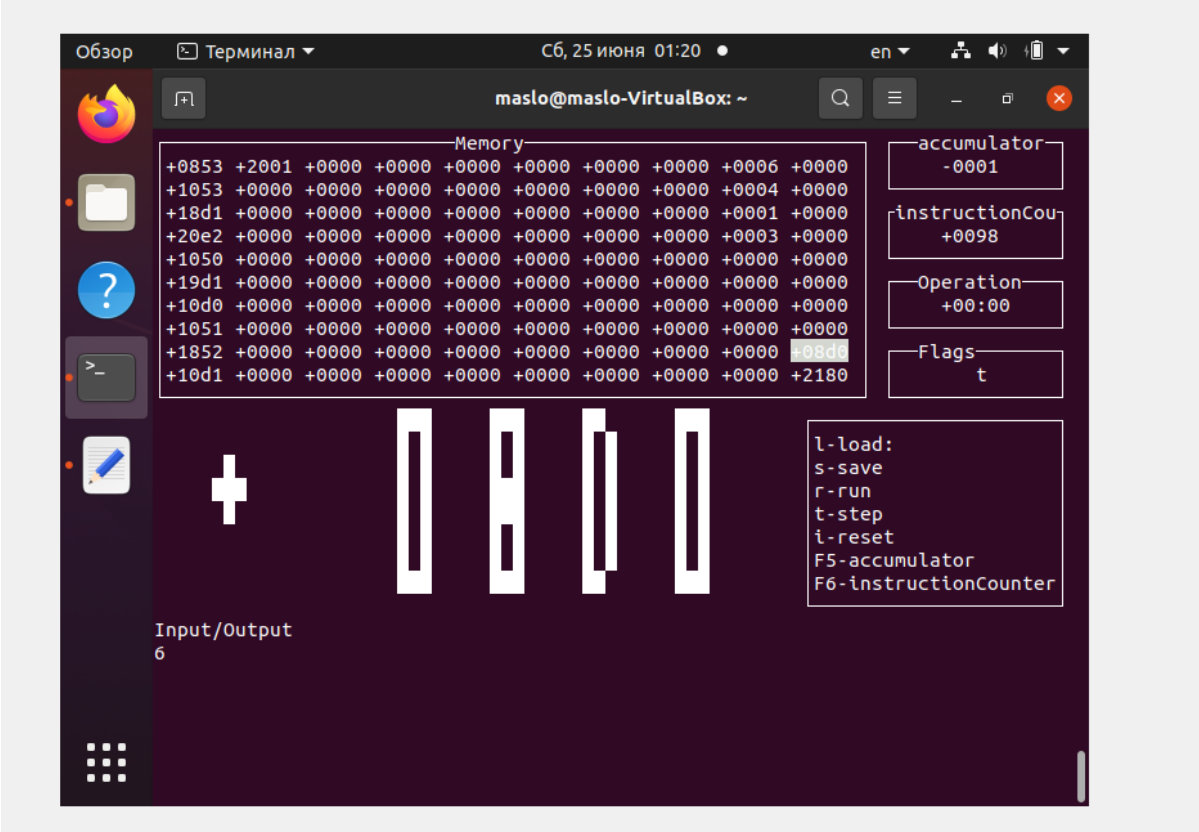
Программа на ассемблере

```
00 READ 83 ;n
80 = +0001;s
81 = +0001; i
82 = +0001;1
01 LOAD 83; load n in acc
02 SUB 81;n-i
03 JNEG 98;if n-i<0
98 WRITE 80;print s
99 HALT 00
04 LOAD 80;if n-i>=0 load s in acc
05 MUL 81;s*i
06 STORE 80;s=s*i
07 LOAD 81;load i in acc
08 ADD 82;i+1
09 STORE 81;i=i+1
10 JUMP 01;
```

Программа осуществляет поиск факториала.
Ввод числа




Результат




# 5. Листинг кода

Main.cpp

```
#include "MBC.h"
#include "MRK.h"
#include <stdlib.h>
#include <stdio.h>
#include<string.h>
#include<signal.h>
#include<sys/time.h>
const int cursor=23;
void menu();
void menu_flag();
void key_enter(enum keys key, int count);
void f6(int value);
void I();
int CU(int value,int cell_num);
int count_t=0;
void signalhandler(int signo)
{
	int value;
	sc_regGet(IGNORING_CLOCK_PULSES, &value);
	if(count_t==100)
	{
		count_t=0;
	}
	if(signo==SIGALRM&&value==0)
	{
		f6(count_t);
		sleep(1);
		int value;
		sc_memoryGet(count_t, &value);
		int result=CU(value,count_t);
		if(result==1)
		{
			alarm(0);

sc_regSet(IGNORING_CLOCK_PULSES, 1);
			menu_flag();
			mt_gotoXY(23,0);
		}
		else
		{

sc_regSet(IGNORING_CLOCK_PULSES, 0);
			menu();
			if(result!=2)
				count_t++;
		}

	}
	else if(signo==SIGUSR1)
	{
		count_t=0;
		sc_regInit();
```

```
                alarm(0);
                menu();
        }
}
void sig()
{
        struct itimerval nval,oval;
        nval.it_interval.tv_sec=1;
        nval.it_interval.tv_usec=0;
        nval.it_value.tv_sec=1;
        nval.it_value.tv_usec=0;
        setitimer(ITIMER_REAL,&nval,&oval);
}
int neg(int value)
{
        return ~value+1;
}
void menu_memory()
{

        bc_box(1,1,12,62);
        mt_gotoXY(1,27);
        printf("Memory");

        int value;
        int size=M_SIZE/10;
        int count=0;
        int j1=0;
        for (int i=0;i<size;i++)
        {
        for(int j=0;j<size;j++)
        {
                mt_gotoXY(i+2,j1+2);
                sc_memoryGet(count, &value);
                if(value&0x80000000)
                        printf("-%04x",neg(value));
                else
                        printf("+%04x",value);
                count+=10;
                j1+=6;
        }
        j1=0;
        count=i+1;
        }
        printf("'\n");



}
int accum=0x0;
void menu_accum()
{
        bc_box(1,64,3,79);
        mt_gotoXY(1,67);
        printf("accumulator");
        mt_gotoXY(2,69);
        if(accum&0x80000000)
                printf("-%04x",neg(accum));
        else
                printf("+%04x",accum);
}
void menu_iC(int value=0)
{
        mt_setbgcolor(Standart);
        bc_box(4,64,6,79);
        mt_gotoXY(4,65);
        printf("instructionCou");
        mt_gotoXY(5,69);
        printf("%+05d",value);
}
void menu_Oper()
{
        bc_box(7,64,9,79);
        mt_gotoXY(7,67);
        printf("Operation");
        mt_gotoXY(8,69);
        printf("+00:00");
}
void menu_flag()
{
        bc_box(10,64,12,79);
        mt_gotoXY(10,67);
        printf("Flags");
        int value;
        int y=69;
        int x=11;
        mt_gotoXY(x,y);
        for(int i=0;i<REG_SIZE;i++)
        {
                sc_regGet(i, &value);
                if(value==1&&i==0)
                        printf("p");
                else if(value==1&&i==1)
                        printf("0");
                else if(value==1&&i==2)
                        printf("m");
                else if(value==1&&i==3)
                        printf("t");
                else if(value==1&&i==4)
                        printf("e");
                y++;
                mt_gotoXY(x,y);
        }


}
void menu_key()
{
        bc_box(13,57,21,79);
        int x=14,y=58;
        mt_gotoXY(x,y+2);
        printf("keys:");
        mt_gotoXY(x,y);
        printf("l-load");
        x++;
        mt_gotoXY(x,y);
        printf("s-save");
        x++;
        mt_gotoXY(x,y);
        printf("r-run");
        x++;
        mt_gotoXY(x,y);
        printf("t-step");
        x++;
        mt_gotoXY(x,y);
        printf("i-reset");
        x++;
        mt_gotoXY(x,y);
        printf("F5-accumulator");
        x++;
        mt_gotoXY(x,y);
        printf("F6-instructionCounter");
```

```
		x++;
		mt_gotoXY(x,y);
}
void BChar(int value=0x0)
{
		int X=13;
		int Y=1;
		int big[2];
		if(!(value&0x80000000))
		{
			big[0]=BC[16][0];
			big[1]=BC[16][1];

bc_printbigchar(big,X,Y,Standart,Standart);
		}
		else
		{
			big[0]=BC[17][0];
			big[1]=BC[17][1];

bc_printbigchar(big,X,Y,Standart,Standart);
			value=neg(value);
		}
		Y+=8*5;
		int rem=0x10;
		for(int i=0;i<4;i++)
		{
			int value0=value%rem;
			value/=0x10;
			switch(value0)
			{
				case 0x0:
					big[0]=BC[0][0];
					big[1]=BC[0][1];
					break;
				case 0x1:
					big[0]=BC[1][0];
					big[1]=BC[1][1];
					break;
				case 0x2:
					big[0]=BC[2][0];
					big[1]=BC[2][1];
					break;
				case 0x3:
					big[0]=BC[3][0];
					big[1]=BC[3][1];
					break;
				case 0x4:
					big[0]=BC[4][0];
					big[1]=BC[4][1];
					break;
				case 0x5:
					big[0]=BC[5][0];
					big[1]=BC[5][1];
					break;
				case 0x6:
					big[0]=BC[6][0];
					big[1]=BC[6][1];
					break;
				case 0x7:
					big[0]=BC[7][0];
					big[1]=BC[7][1];
					break;
				case 0x8:
					big[0]=BC[8][0];
					big[1]=BC[8][1];
					break;
				case 0x9:
					big[0]=BC[9][0];
					big[1]=BC[9][1];
					break;
				case 0xa:
					big[0]=BC[10][0];
					big[1]=BC[10][1];
					break;
				case 0xb:
					big[0]=BC[11][0];
					big[1]=BC[11][1];
					break;
				case 0xc:
					big[0]=BC[12][0];
					big[1]=BC[12][1];
					break;
				case 0xd:
					big[0]=BC[13][0];
					big[1]=BC[13][1];
					break;
				case 0xe:
					big[0]=BC[14][0];
					big[1]=BC[14][1];
					break;
				case 0xf:
					big[0]=BC[15][0];
					big[1]=BC[15][1];
					break;
			}

bc_printbigchar(big,X,Y,Standart,Standart);
			Y-=8;
		}
}
void sw(int fl_color,int X,int Y,int count)
{
		int value;
		if(fl_color==0)
			mt_setbgcolor(White);
		else
			mt_setbgcolor(Standart);
		mt_gotoXY(X,Y);
		sc_memoryGet(count, &value);
		if(value&0x80000000)
			printf("-%04x",neg(value));
		else
			printf("+%04x",value);
		BChar(value);
}
void memory_p()
{
		int X=2,Y=2;
		int count=0;
		sw(0,X,Y,count);
		keys key;
		mt_gotoXY(23,0);
		rk_readkey(&key);

while(key==KEY_UP||key==KEY_DOWN||key==KEY_RIGH
T||key==KEY_LEFT)
		{
			if(key==KEY_UP)
			{
				if(X-1>1)
				{
```

```
                                        sw(1,X,Y,count);
                                        X-=1;
                                        count--;
                                        sw(0,X,Y,count);
                                }
                        }
                        else if(key==KEY_DOWN)
                        {
                                if((X+1)<12)
                                {
                                        sw(1,X,Y,count);
                                        X+=1;
                                        count++;
                                        sw(0,X,Y,count);
                                }
                        }
                        else if(key==KEY_RIGHT)
                        {
                                if(Y+6<62)
                                {
                                        sw(1,X,Y,count);
                                        Y+=6;
                                        count+=10;
                                        sw(0,X,Y,count);
                                }
                        }
                        else if(key==KEY_LEFT)
                        {
                                if(Y-6>0)
                                {
                                        sw(1,X,Y,count);
                                        Y-=6;
                                        count-=10;
                                        sw(0,X,Y,count);
                                }
                        }
                        //keys key;
                        menu_iC(count);
                        mt_gotoXY(23,0);
                        rk_readkey(&key);

                }
                key_enter(key,count);
}
void I()
{
                sc_memoryInit();
                sc_regInit();
                sc_regSet(IGNORING_CLOCK_PULSES, 1);
                accum=0;
}
void f6(int value)//PsPєCЂP°CЃPєP° PєPSPsPїPєPë P±PµP·
PєPsPsCЂPrPëPSP°C,
{
                menu();
                if(value>-1&&value<100)
                {
                        menu_iC(value);
                        int x=value%10+2;
                        int y=value/10*6+2;
                        sw(0,x,y,value);
                        mt_gotoXY(23,0);
                }
}
void inp()
{
                int value;
                sc_regGet(IGNORING_CLOCK_PULSES, &value);
                keys key;
                        rk_readkey(&key);
                if(value==0)
                {
                                raise(SIGUSR1);

sc_regSet(IGNORING_CLOCK_PULSES, 1);
                                menu();
                }
                else
                {

if(key==KEY_UP||key==KEY_DOWN||key==KEY_RIGHT||ke
y==KEY_LEFT)
                {
                        memory_p();
                }
                else if(key==KEY_F6)
                {
                        int value;
                        scanf("%d",&value);
                        menu();
                        if(value>-1&&value<100)
                        {
                                menu_iC(value);
                                int x=value%10+2;
                                int y=value/10*6+2;
                                sw(0,x,y,value);
                                mt_gotoXY(23,0);
                                rk_readkey(&key);
                                key_enter(key,value);
                        }
                }
                else if(key==KEY_F5)
                {
                        int value;
                        scanf("%x",&value);

if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value
<=0xffffffff))
                        {
                                accum=value;
                        }
                }
                else if(key==KEY_I)
                {
                        I();
                }
                else if(key==KEY_S)
                {
                        char fname[128];
                        for(int i=0;i<128;i++)
                                fname[i]=' ';
                        printf("Input filename\n");
                        //scanf("%s",&fname);
                        fgets(fname,128,stdin);
                        sc_memorySave(fname);
                }
                else if(key==KEY_L)
                {
                        char fname[128];
                        for(int i=0;i<128;i++)
                                fname[i]=' ';
                        printf("Input filename\n");
```

```
                        //scanf("%s",&fname);
                        fgets(fname,128,stdin);
                        sc_memoryLoad(fname);
                }
                else if(key==KEY_R)
                {
                        rk_mytermsave();

sc_regSet(IGNORING_CLOCK_PULSES, 0);
                        sig();
                }
                menu();
                }
}
void key_enter(enum keys key, int count)//Pë C,
{
                if(key==KEY_ENTER)
                {
                        int value;
                        scanf("%x",&value);

if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value
<=0xffffffff))
                        {
                                sc_memorySet(count,value);
                        }
                }
                else if(key==KEY_T)
                {
                        count_t=count;
                        menu_iC(count_t);
                        rk_mytermsave();
                        int memory;
                        sc_memoryGet(count, &memory);

if(CU(memory,count)==1||CU(memory,count)==2)
                        {
                                menu_flag();
                                mt_gotoXY(23,0);
                                sleep(1);
                                sc_regInit();

sc_regSet(IGNORING_CLOCK_PULSES, 1);
                                menu_flag();
                        }
                        count_t=0;
                        menu_iC(count_t);
                }

}
void menu()
{
                mt_clrscr();
                menu_memory();
                menu_accum();
                menu_iC();
                menu_Oper();
                menu_flag();
                menu_key();
                BChar();
                printf("\nInput/Output\n");
                mt_gotoXY(23,0);

}
```

```
void decoder(int value,int *number,int *operand)
{
                if(value|0x3fff==0x3fff)
                {
                        int num,oper;
                        num=value;
                        num>>=7;
                        *number=num;
                        oper=value;
                        oper&=0x7F;
                        *operand=oper;
                }
                else
                {
                        *number=0;
                        *operand=0;
                }
}
int ALU(int operand,int command)
{
                int result,value;
                sc_memoryGet(operand,&value);
                switch(command)
                {
                        case 0x30:
                                result=value+accum;
                                break;

                        case 0x31:
                                result=accum-value;
                                break;
                        case 0x32:
                                if(value!=0)

result=accum/value;
                                else
                                        return 0xffff1;
                                break;
                        case 0x33:
                                result=accum*value;
                                break;
                        case 0x60:
                                result=value<<1;
                                break;
                        default:
                                result=accum;
                }

if((result>=0x0&&result<=0xffff)||(result>=0xffff0001&&result
<=0xffffffff))
                                                return result;
                else
                                                return 0xfffff;



}
int CU(int value_m,int cell_num)
{
                int number,operand;
                decoder(value_m,&number,&operand);
                if(number==0)
                {
                        sc_regSet(INCORRECT_COMMAND,
1);
                        return 1;
                }
```

```
        sc_regSet(IGNORING_CLOCK_PULSES, 1);
        menu_flag();
        mt_gotoXY(23,0);
        int value;
        if(operand<0||operand>99)
        {
                sc_regSet(OUT_OF_MEMORY, 1);
                return 1;
        }

if(number==0x30||number==0x31||number==0x32||number==0x
33||number==0x60)
        {
                        int
result=ALU(operand,number);
                        if(result==0xfffff)
                        {

sc_regSet(OUT_OF_MEMORY, 1);
                                return 1;
                        }
                        else if(result==0xffff1)
                        {

sc_regSet(DIVISION_ERR_BY_ZERO, 1);
                                return 1;
                        }
                        else
                        {
                                accum=result;
                                menu_accum();
                                return 0;
                        }
        }
        switch( number)
        {
        case 0x10:
                        mt_gotoXY(23,0);
                        printf("input value\n");
                        rk_mytermsave2();
                        rk_mytermrestore();
                        scanf("%x",&value);
                        rk_mytermrestore2();

if((value>=0x0&&value<=0xffff)||(value>=0xffff0001&&value
<=0xffffffff))
                        {

sc_memorySet(operand,value);
                        }
                        else
                        {

sc_regSet(OVERFLOW, 1);
                                return 1;
                        }
                        break;
        case 0x11:
                        mt_gotoXY(cursor,0);

sc_memoryGet(operand,&value);
                        printf("%x",value);
                        sleep(2);
                        break;
        case 0x20:

sc_memoryGet(operand,&value);
                        accum=value;
                        menu_accum();
                        break;
        case 0x21:

sc_memorySet(operand,accum);
                        menu_memory();
                        break;
        case 0x40:
                        count_t=operand;
                        f6(count_t);
                        sleep(1);
                        return 2;
        case 0x41:

if(accum>=0xffff0001&&accum<=0xffffffff)
                        {
                                count_t=operand;
                                f6(count_t);
                                sleep(1);
                                return 2;
                        }
                        break;
        case 0x42:
                        if(accum==0)
                        {
                                count_t=operand;
                                f6(count_t);
                                sleep(1);
                                return 2;
                        }
                        break;
        case 0x43:
                        return 1;
        default:

sc_regSet(INCORRECT_COMMAND,
1);//PSPµPIPµCЂPSP°CЦ PєPsPjP°PSPrP°
                        return 1;
        }
        menu();
        //sleep(1);
        return 0;
}
int main ()
{
        sc_memoryInit();
        sc_regInit();
        signal(SIGALRM,signalhandler);
        signal(SIGUSR1,signalhandler);
        sc_regSet(IGNORING_CLOCK_PULSES, 1);
        menu();
        setbuf(stdout,NULL);
        while(1)
        {
                inp();
        }
        return 0;
}
```

## Assembler.cpp

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <fstream>
#include <map>
using namespace std;
#define  M_SIZE 100
int OPmemory[M_SIZE];
void coder(int memory_cell,int command,int operand);
void memory_clear()
{
	for (int i = 0; i < M_SIZE; i++)
		OPmemory[i] = 0;

}
int sc_memorySave(char* filename)
{
	ofstream addressData(filename, ios::out |
ios::binary);
	addressData.write((char *)OPmemory,
sizeof(OPmemory));
	addressData.close();
	return 0;
}
void str_clear(char *str,int n)
{
	for(int i=0;i<n;i++)
		str[i]=' ';
}
void str_print(char *str,int n)
{
	for(int i=0;i<n;i++)
		printf(" %d ",str[i]);
}
int check_char(char lit)
{
	if(lit==' ')
		return 0;
	else if(lit>='0'&&lit<='9')
		return 1;
	else if(lit=='='||(lit>='A'&&lit<='Z'))
		return 2;
	else if(lit==';')
		return 3;
	else
		return 4;
}
int counter_space(int j,char *str,int n)
{
	while(str[j]==' '&&j<n)
		j++;
	return j;
}
int checkCommand(string str)
{
	map <string,int> Command={

{"READ",0x10},

{"WRITE",0x11},

{"LOAD",0x20},

{"STORE",0x21},

{"ADD",0x30},

{"SUB",0x31},

{"DIVIDE",0x32},

{"MUL",0x33},

{"JUMP",0x40},

{"JNEG",0x41},

{"JZ",0x42},

{"HALT",0x43},

{"CHL",0x60},

{"=",0x1},
                                                };
	map <string,int>::iterator i=Command.begin();

	i=Command.find(str);
	//cout<<i->second<<endl;
	if(i!=Command.end())
		return i->second;
	return -1;
}
int treat (char *str,int n)
{
	int memory_cell;
	char Arr_mc[2];
	str_clear(Arr_mc,2);
	int j=0,i=0;
	j=counter_space(j,str,n);
	//printf("l%d \n",j);

	//int result=check_char(str[j]);

	while(i<2&&j<n)
	{
		int result=check_char(str[j]);
		if(result==1)
			Arr_mc[i]=str[j];
		j++;
		i++;
	}
```

```
		if(Arr_mc[0]==' '||Arr_mc[1]==' ')
				return 1;



memory_cell=(Arr_mc[0]-'0')*10+(Arr_mc[1]-'0');
		j=counter_space(j,str,n);

		////////////////////////////////////////////////
		string str1;
		int result=check_char(str[j]);

//while(result!=1&&result!=0&&str[j]!=0&&j<n)
		while(result==2&&j<n)
		{
			//printf("l%d \n",str[j]);
			str1.push_back(str[j]);
			j++;
			result=check_char(str[j]);
		}
		int command=checkCommand(str1);
		//printf("%d \n",command);
		if(command==-1)
			return 2;
		j=counter_space(j,str,n);
		/////////////////////////////////////////
		str_clear(Arr_mc,2);
		char stOperand[5];
		str_clear(Arr_mc,5);
		i=0;
		int operand=0;

		if(command==0x1)
		{
			if(str[j]=='+'||str[j]=='-')
				stOperand[i]=str[j];
			else
				return 3;
			j++;
			i++;
			while(i<5&&j<n)
			{
				result=check_char(str[j]);
				if(result==1||result==2)

stOperand[i]=str[j];
				else
					return 3;
				i++;
				j++;
			}

		    int dop=16*16*16;
		    for(int k=1;k<5;k++)
		    {
			if(check_char(stOperand[k])==1)
				stOperand[k]-='0';
			else
				stOperand[k]-=55;
			//printf("llll%d \n",stOperand[k]);
			operand+=stOperand[k]*dop;
			//printf("%x \n",stOperand[k]*dop);
			dop/=16;
		    }

		    if(stOperand[0]=='-')
		    {
			operand--;
			operand=~operand;
		    }

		}
		else
		{
			while(i<2&&j<n)
			{
				result=check_char(str[j]);
				if(result==1)
					Arr_mc[i]=str[j];
				j++;
				i++;
			}
			if(Arr_mc[0]==' '||Arr_mc[1]==' ')
				return 3;


operand=(Arr_mc[0]-'0')*10+(Arr_mc[1]-'0');
		}
		//printf("ll%x \n",operand);
		//str_print(stOperand,5);
		j=counter_space(j,str,n);
		////////////////////////////////////////
		result=check_char(str[j]);
		if(result!=3&&str[j]!=0&&j<n)
			return 4;
		coder(memory_cell,command,operand);
		//str_print(ComArr,s_ComArr);
		//printf("%d \n",memory_cell);

		return 0;
}
void coder(int memory_cell,int command,int operand)
{
		if (command==0x1)
		{
			OPmemory[memory_cell]=operand;
			return;
		}
		int MC=command<<7;
		//printf("%x ",MC);
		MC|=operand;
		//printf("%x ",MC);
		OPmemory[memory_cell]=MC;
}

int main()
```

```
{
	memory_clear();
	ifstream file("assembler.sa");
	int n=64;
	if(!file)
		printf("hhh");
	char buffer[n];
	str_clear(buffer,n);
	int i=1;
	while(!file.eof())
	{
		str_clear(buffer,n);
		file.getline(buffer,n);
		if(buffer[0]==0)
			break;
		int result=treat (buffer,n);
		if(result>=1&&result<=4)
		{
			printf("number %d\n",i);
			memory_clear();
			if(result==1)
				printf("ERROR
memory cell\n");
			else if(result==2)
				printf("ERROR
command\n");
			else if(result==3)
				printf("ERROR
operand\n");
			else if(result==4)
				printf("ERROR
comment\n");
			return 0;
		}
		i++;
	}
	//printf("%c",s[2]);
	file.close();
	char fname[128];
	printf("Input filename\n");
	fgets(fname,128,stdin);
	sc_memorySave(fname);
	return 0;
}
```

## MBC.h

```
#include "MSC.h"
#include "MT.h"
#include <stdlib.h>
#include <stdio.h>

const int BC[18][2]=
{
	{0x5050507,0x7050505},//0
	{0x1010101,0x1010101},//1
	{0x1050507,0x7040402},//2
	{0x3010507,0x7050101},//3
	{0x7050505,0x1010101},//4
	{0x7040407,0x7050101},//5
	{0x7040507,0x7050505},//6
	{0x1010107,0x1010101},//7
	{0x7050507,0x7050507},//8
	{0x7050507,0x7050101},//9
	{0x7010507,0x7050505},//a
	{0x6050507,0x7050505},//b
	{0x4040507,0x7050504},//c
	{0x5050506,0x6050505},//d
	{0x7040407,0x7040404},//e
	{0x4070407,0x4040404},//f
	{0x7020000,0x2},//+
	{0x3000000,0x0}//-
};

int bc_printA(char *str);
int bc_box(int x1,int y1,int x2,int y2);
int bc_printbigchar(int *big,int x,int y, enum colors
tC,enum colors bgC);
int bc_getbigcharpos(int *big,int x,int y, int *value);
int bc_setbigcharpos(int *big,int x,int y, int value);
```

MBC.CPP

```
#include "MBC.h"


int bc_printA(char *str)
{
	printf("\E[11m%s\E[10m",str);
	return 0;
}
int bc_box(int x1,int y1,int x2,int y2)
{
	if(x1<0||y1<0||x2<0||y2<0)
		return -1;
	int rows,cols;
	/*mt_getscreensize(&rows,&cols);
	if(rows<x1+x2||cols<y1+y2)
		return -1;*/
	for(int i=x1+1;i<x2;i++)
	{
		mt_gotoXY(i, y1);
		printf("\u2502");
	}
	mt_gotoXY(x2, y1);
	printf("\u2514");
	for(int j=y1+1;j<y2;j++)
	{
		mt_gotoXY(x2, j);
		printf("\u2500");
	}
	mt_gotoXY(x2, y2);
	printf("\u2518");
	for(int i=x2-1;i>x1;i--)
	{
```

```
                mt_gotoXY(i, y2);
                printf("\u2502");
        }
        mt_gotoXY(x1, y2);
        printf("\u2510");
        for(int j=y2-1;j>y1;j--)
        {
                mt_gotoXY(x1, j);
                printf("\u2500");
        }
        mt_gotoXY(x1, y1);
        printf("\u250C");
        mt_gotoXY(x2+2, 0);
        return 0;
}
int bc_printbigchar(int *big,int x,int y, enum colors
tC,enum colors bgC)
{
        mt_setfgcolor(tC);
        mt_setbgcolor(bgC);
        y+=7;
        mt_gotoXY(x,y);
        for(int i=0;i<4;i++)
        {
                for(int j=0;j<8;j++)
                {
                        if(((big[0] >> (i*8+j)) &
0x1)==1)
                                printf("\u2588");
                        else
                                printf(" ");
                        y--;
                        mt_gotoXY(x, y);
                }
                y+=8;
                x++;
                mt_gotoXY(x, y);
        }
        for(int i=0;i<4;i++)
        {
                for(int j=0;j<8;j++)
                {
                        if(((big[1] >> (i*8+j)) &
0x1)==1)
                                printf("\u2588");
                        else
                                printf(" ");
                        y--;
                        mt_gotoXY(x, y);
                }
                y+=8;
                x++;
                mt_gotoXY(x, y);
        }
        mt_setfgcolor(Standart);
        mt_setbgcolor(Standart);
        return 0;
}
int bc_getbigcharpos(int *big,int x,int y, int *value)
{
        if(x<0||x>8||y<0||y>8)
        {
                return -1;
        }
        else if(y<4)
        {
                *value=((big[0] >> (x*8+7-x)) &
0x1);
        }
        else if(y>=4)
        {
                *value=((big[1] >> (x*8+7-x)) &
0x1);
        }
        return 0;
}
int bc_setbigcharpos(int *big,int x,int y, int value)
{
        if(x<0||x>8||y<0||y>8||(value!=0&&value!=1))
        {
                return -1;
        }
        else if(y<4&&value==0)
        {
                big[0]&=(~(big[0] << (y*8+7-x)));
        }
        else if(y<4&&value==1)
        {
                big[0]|=((big[0] << (y*8+7-x)));
        }
        else if(y>=4&&value==0)
        {
                big[1]&=(~(big[1] << (y*8+7-x)));
        }
        else if(y>=4&&value==1)
        {
                big[1]|=((big[1] << (y*8+7-x)));
        }
        return 0;
}
```

MRK.cpp

```
#include "MRK.h"
struct termios buff;
struct termios buff2;
int rk_mytermregime (int regime, int vtime, int vmin, int
echo, int sigint)
{
        if
((regime!=0&&regime!=1)||(vtime!=0&&vtime!=1)||(vm
in!=0&&vmin!=1)||(echo!=0&&echo!=1)||(sigint!=0&&s
igint!=1))
                return -1;
        struct termios ws;
        tcgetattr(0,&ws);//Ñ‚ÐµÐºÑƒÑ‰Ð¸Ðµ
Ð½Ð°ÑÑ‚Ñ€Ð¾Ð¹ÐºÐ¸
```

```
        if(regime)
                ws.c_lflag|=ICANON;
        else
        {
                ws.c_lflag&=~ICANON;
                ws.c_cc[VMIN]=vmin;
                ws.c_cc[VTIME]=vtime;
        }
        if(echo)//ÑÐ¸Ð¼Ð²Ð¾Ð»Ñ‹
Ð²Ñ‹Ð²Ð¾Ð´Ð´ÑÑ‚ÑÑ Ð² ÐºÐ¾Ð½Ñ‚Ñ€Ð¾Ð»Ð¬
                ws.c_lflag|=ECHO;
        else
                ws.c_lflag&=~ECHO;
        if(sigint)
                ws.c_lflag|=ISIG;
        else
                ws.c_lflag&=~ISIG;
        tcsetattr(0,TCSANOW,&ws);
        return 0;
}
int rk_mytermsave()
{
        tcgetattr(0,&buff);
        return 0;
}
int rk_mytermrestore()
{
        tcsetattr(0,TCSANOW,&buff);
        return 0;
}
int rk_mytermsave2()
{
        tcgetattr(0,&buff2);
        return 0;
}
int rk_mytermrestore2()
{
        tcsetattr(0,TCSANOW,&buff2);
        return 0;
}
int rk_readkey(enum keys* key)
{
        char temp[8];
        rk_mytermsave();
        //printf("1");
        rk_mytermregime(0,0,1,0,1);
        read(0,&temp,8);
        switch(temp[0])
        {
                case 'e':
                        *key=KEY_ENTER;
                        break;
                case 'l':
                        *key=KEY_L;
                        break;
                case 's':
                        *key=KEY_S;
                        break;
                case 'r':
                        *key=KEY_R;
                        break;
                case 't':
                        *key=KEY_T;
                        break;
                case 'i':
                        *key=KEY_I;
                        break;
                case '\E':
                        //read(0,&temp,1);
                        //read(0,&temp,1);
                        switch((int)temp[2])
                        {
                                case 65:
*key=KEY_UP;
                                        break;
                                case 66:
*key=KEY_DOWN;
                                        break;
                                case 67:
*key=KEY_RIGHT;
                                        break;
                                case 68:
*key=KEY_LEFT;
                                        break;
                                case 49:
//read(0,&temp,1);
switch((int)temp[3])
                                        {
        case 53:
                *key=KEY_F5;
                break;
        case 55:
                *key=KEY_F6;
                break;
        default:
                *key=ERROR;
                break;
                                        }
                                        break;
                                default:
```

```
*key=ERROR;
                                                    break;
                        }
                        break;
                default:

                        *key=ERROR;
        }
        /*for(int i=1;i<4;i++)
                printf(" %d %d",i,temp[i]);*/
        rk_mytermrestore();
        return 0;
}
/*int main()
{

        setbuf(stdout,NULL);
        //rk_readkey(&key);
        for(int i=1;i>0;i++)
        {
        printf("n");
        printf("n");
        printf("n");
        keys key;
        rk_readkey(&key);
        if(key==KEY_ENTER)
                printf("enter");
        else if(key==KEY_F5)
                printf("f5");
        else if(key==KEY_F6)
                printf("f6");
        else if(key==KEY_UP)
                printf("up");
        else if(key==KEY_DOWN)
                printf("down");
        else if(key==KEY_LEFT)
                printf("left");
        else if(key==KEY_RIGHT)
                printf("right");
        else if(key==KEY_R)
                printf("r");
        else if(key==ERROR)
                printf("er");
        }

/*      printf("1");
        //rk_mytermsave();
        printf("1");
        rk_mytermregime(0,0,1,0,1);
        int temp[4];
        read(0,&temp,4);
        printf("er");
        for(int i=0;i<4;i++)
                printf("%d %d ",i,temp[i]);
        //rk_mytermrestore();
*//*
        return 0;
}*/
```

MRK.h

```
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <string.h>
enum keys
{
        KEY_I,
        KEY_L,
        KEY_R,
        KEY_S,
        KEY_T,
        KEY_F5,
        KEY_F6,
        KEY_UP,
        KEY_DOWN,
        KEY_RIGHT,
        KEY_LEFT,
        KEY_ENTER,
        ERROR
};
extern struct termios buff;
int rk_mytermregime (int regime, int vtime, int vmin, int
echo, int sigint);
int rk_mytermsave();
int rk_mytermrestore();
int rk_mytermsave2();
int rk_mytermrestore2();
int rk_readkey(enum keys* key);
```

MSC.cpp

```
#include"MSC.h"
unsigned char regFLAGS;
int OPmemory[M_SIZE];
//Đ¸Đ½Đ¸Ñ†Đ¸Đ°Đ»Đ¸Đ·Đ¸ÑĐƒĐµÑ, Đ¾Đ¿
int sc_memoryInit()
{
        for (int i = 0; i < M_SIZE; i++)
        {
                OPmemory[i] = 0;
        }
        return 0;
}
//Đ·Đ°Đ´Đ°Đ¼Ñ, Đ·Đ½Đ°Ñ‡Đ¼Đ½Đ¸Đ¼
ÑÑ‡Đ¼Đ¹ĐºĐ¸ Đ¸Đ»Đ¸ ÑÑ‚Đ°Đ²Đ¸Ñ, Ñ„Đ»Đ°Đ³ 1
int sc_memorySet(int address, int value)
{
        if (address < M_SIZE && address >= 0)
        {
                OPmemory[address] = value;
        }
        else
        {
                sc_regSet(OUT_OF_MEMORY, 1);
                printf("Error
OUT_OF_MEMORY");
```

```
                return -1;
        }
        return 0;
}
//Ð²Ð¾Ð·Ð²Ñ€Ð°Ñ‰Ð°ÐµÑ‚ Ð·Ð½Ð°Ñ‡ÐµÐ½Ð¸Ðµ,
ÑÑ‡ÐµÐ¹ÐºÐ¸ Ð¿Ð°Ð¼ÑÑ‚Ð¸ Ñ Ð·Ð°Ð´Ð°Ð½Ð½Ñ‹Ð¼
Ð°Ð´Ñ€ÐµÑÐ¾Ð¼ Ð² Ð¾Ð±Ñ‰ÑƒÑŽ Ð²Ð¸Ð´Ð¸Ð¼ÑƒÑŽ
Ð¿Ð°Ð¼ÑÑ‚ÑŒ
int sc_memoryGet(int address, int* value)
{
        if (address < M_SIZE && address >= 0)
        {
                *value = OPmemory[address];
                return 0;
        }
        else
        {
                sc_regSet(OUT_OF_MEMORY, 1);
                printf("Error
OUT_OF_MEMORY");
                return -1;
        }
}
//ÑÐ¾Ñ…Ñ€Ð°Ð½ÑÐµÑ‚
ÑÐ¾Ð´ÐµÑ€Ð¶Ð¸Ð¼Ð¾Ðµ Ð¿Ð°Ð¼ÑÑ‚Ð¸ Ð²
Ñ„Ð°Ð¹Ð» Ð² Ð±Ð¸Ð½Ð°Ñ€Ð½Ð¾Ð¼ Ð²Ð¸Ð´Ðµ
int sc_memorySave(char* filename)
{
        ofstream addressData(filename, ios::out |
ios::binary);
        addressData.write((char *)OPmemory,
sizeof(OPmemory));
        addressData.close();
        return 0;
}
//Ð·Ð°Ð³Ñ€ÑƒÐ¶Ð°ÐµÑ‚
ÑÐ¾Ð´ÐµÑ€Ð¶Ð¸Ð¼Ð¾Ðµ Ð¿Ð°Ð¼ÑÑ‚Ð¸ Ð¸Ð·
Ñ„Ð°Ð¹Ð»Ð°
int sc_memoryLoad(char* filename)
{
         ifstream addressData(filename, ios::in |
ios::binary);
        addressData.read((char *)&OPmemory,
sizeof(OPmemory));
        return 0;
}
//Ð¸Ð½Ð¸Ñ†Ð¸Ð°Ð»Ð¸Ð·Ð¸Ñ€ÑƒÐµÑ‚
Ñ€ÐµÐ³Ð¸ÑÑ‚Ñ€ Ñ„Ð»Ð°Ð³Ð¾Ð²
int sc_regInit()
{
        regFLAGS = 0;
        return 0;
}
// Ñƒ ÑÑ‚Ð°Ð½Ð°Ð²Ð»Ð¸Ð²Ð°ÐµÑ‚
Ð·Ð½Ð°Ñ‡ÐµÐ½Ð¸Ðµ Ñƒ Ñ„Ð»Ð°Ð³Ð¾Ð² Ð¿Ð°Ñ€Ð°Ð¼ÐµÑ‚Ñ€Ð°
Ñ€ÐµÐ³Ð¸ÑÑ‚Ñ€Ð° Ñ„Ð»Ð°Ð³Ð¾Ð²
int sc_regSet(int reg, int value)
{
        if (reg >= 0 && reg < REG_SIZE)
        {
                if (value == 0)
                {
                        regFLAGS = regFLAGS &
(~(1 << (reg)));
                }
                else if (value == 1)
                {
                        regFLAGS = regFLAGS |
(1 << (reg));
                }
                else
                {
                        printf("Error incorrect
value");
                        return -1;
                }
        }
        else
        {
                printf("Error incorrect registr");
                return -1;
        }
        return 0;
}
//Ð²Ð¾Ð·Ð²Ñ€Ð°Ñ‰Ð°ÐµÑ‚ Ð·Ð½Ð°Ñ‡ÐµÐ½Ð¸Ðµ,
Ñƒ Ñ„Ð»Ð°Ð³Ð¾Ð²Ð¾Ð³Ð¾ Ñ€ÐµÐ³Ð¸ÑÑ‚Ñ€Ð°
Ñ„Ð»Ð°Ð³Ð¾Ð²
int sc_regGet(int reg, int* value)
{
        if (reg >= 0 && reg < REG_SIZE)
                *value = (regFLAGS >> (reg )) &
0x1;
        else
        {
                printf("Error incorrect registr");
                return -1;
        }
        return 0;
}
//ÐºÐ¾Ð´Ð¸Ñ€ÑƒÐµÑ‚ Ñ
Ñ„Ð°Ð·Ð¾Ð²Ñ‹Ð¹ Ð½Ð°Ð±Ð¾Ñ€ Ð¸
Ð¾Ð¿Ñ€ÐµÐ´ÐµÐ»ÑÐµÑ‚ Ð¸
Ð¿Ñ€Ð¾Ð²ÐµÑ€ÑÐµÑ‚ Ñ€ÐµÐ·ÑƒÐ»ÑŒÑ‚Ð°Ñ‚, Ð²
value
int sc_commandEncode(int command, int operand, int*
value)
{
        if (command < 10 || (command > 11 &&
command < 20) || (command > 21 && command < 30) ||
                (command > 33 && command < 40)
|| (command > 43 && command < 51) || command > 76)
        {
                printf("Error incorrect command");
                return -1;
        }
        if (operand < 0 || operand >127)
        {
                printf("Error incorrect operand");
```

```
                return -1;
        }
        *value = operand;
        *value |= command << 7;
        return 0;
}
int sc_commandDecode(int value, int* command, int*
operand)
{
        int command1, operand1;
        int i = 0;
        if ((value & 0x4000) == 1)
        {

sc_regSet(INCORRECT_COMMAND, 1);
                printf("Error
INCORRECT_COMMAND");
                return -1;
        }
        operand1 = value & 0x7F;
        command1 = (value >> 7) & 0x7F;

        if ((command1 < 10 || (command1 > 11 &&
command1 < 20) || (command1 > 21 && command1 <
30) ||
                (command1 > 33 && command1 <
40) || (command1 > 43 && command1 < 51) ||
command1 > 76) ||
                (operand1 < 0 || operand1 >127))

        {

sc_regSet(INCORRECT_COMMAND, 1);
                printf("Error
INCORRECT_COMMAND");
                return -1;
        }
        else
        {
                *operand = operand1;
                *command = command1;
                return 0;
        }
}
```

MSC.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include<fstream>
using namespace std;

#define OVERFLOW 0
#define DIVISION_ERR_BY_ZERO 1
#define OUT_OF_MEMORY 2
#define IGNORING_CLOCK_PULSES 3
#define INCORRECT_COMMAND 4
#define REG_SIZE 5
#define  M_SIZE 100

int sc_memoryInit();
int sc_memorySet(int address, int value);
int sc_memoryGet(int address, int* value);
int sc_memorySave(char* filename);
int sc_memoryLoad(char* filename);
int sc_regInit();
int sc_regSet(int reg, int value);
int sc_regGet(int reg, int* value);
int sc_commandEncode(int command, int operand, int*
value);
int sc_commandDecode(int value, int* command, int*
operand);
```

MT.h

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include<termios.h>
enum colors {Black, Red, Green, Yellow, Blue, Dark,
Cyan, White, Standart = 9};
int mt_clrscr();
int mt_gotoXY(int X, int Y);
int mt_getscreensize(int *rows, int *cols);
int mt_setfgcolor(enum colors color);
int mt_setbgcolor(enum colors color);
```

MT.cpp

```
#include "MT.h"

int mt_clrscr()
{
  printf("\E[H\E[2J");
  return 0;
}

int mt_gotoXY(int X, int Y)
{
  printf("\E[%d;%dH",X,Y);
  return 0;
}

int mt_getscreensize(int *rows, int *cols)
{
  struct winsize ws;
  if(!ioctl(1, TIOCGWINSZ, &ws))
  {
    (*rows) = ws.ws_row;
    (*cols) = ws.ws_col;
    return 0;
  }
  else
    return -1;
}

int mt_setfgcolor(enum colors color)
{
  printf("\E[3%dm",color);
```

```
    return 0;
}

int mt_setbgcolor(enum colors color)
{
    printf("\E[4%dm",color);
    return 0;
}
```

# 6. Вывод

Подводя итоги, в ходе проделанной работы мною были приобретены навыки:

1. Проектирования архитектуры Simple Computer
2. Программный перевод Simple Assembler в машинный код

# 7. Список литературы


https://purecodecpp.com/archives/2751

https://snipp.ru/handbk/table-ascii

https://server.179.ru/tasks/cpp/total/161.html

https://metanit.com/cpp/tutorial/2.16.php