ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра ВС

Отчёт по курсовому проекту по дисциплине «Архитектура ЭВМ»

Вариант 4

Выполнил:
ст. гр. ИА-831
Ейбауер Д. А.

Проверил:
доцент кафедры ВС
Майданов Ю. С.

Новосибирск, 2020

Содержание:

1. Задание

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

Для управления модель (определения начальных состояний узлов Simple Computer, запуска программ на выполнение, отражения хода выполнения программ) требуется создать консоль. Для разработки программ требуется создать трансляторы с языков Simple Assembler и Simple Basic.

Оперативная память – это часть Simple Computer, где хранятся программа и данные. Память состоит из ячеек (массив), каждая из которых хранит 15 двоичных разрядов. Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является её адресом и задается 7-миразрядным числом. Предполагаем, что Simple Computer оборудован памятью из 100 ячеек (с адресами от 0 до 99).

Выполнение программ осуществляется центральным процессором Simple Computer. Процессор состоит из следующих функциональных блоков:

— регистры (аккумулятор, счетчик команд, регистр флагов);

— арифметико-логическое устройство (АЛУ);

— управляющее устройство (УУ);

— обработчик прерываний от внешних устройств (ОП);

— интерфейс доступа к оперативной памяти.

Регистры являются внутренней памятью процессора. Центральный процессор Simple Computer имеет: аккумулятор, используемый для временного хранения данных и результатов операций, счетчик команд, указывающий на адрес ячейки памяти, в которой хранится текущая выполняемая команда и регистр флагов,

сигнализирующий об определённых событиях. Аккумулятор имеет разрядность 15 бит, счетчика команд – 7 бит. Регистр флагов содержит 5 разрядов: переполнение при выполнении операции, ошибка деления на 0, ошибка выхода за границы памяти, игнорирование тактовых импульсов, указана неверная команда.

Арифметико-логическое устройство (англ. Arithmetic and logic unit, ALU) — блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде команды или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в аккумуляторе или может помещаться в оперативную память. В ходе выполнения операций АЛУ устанавливает значения флагов «деление на 0» и «переполнение». Управляющее устройство (англ. Control unit, CU) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. В его функции входит: чтение текущей команды из памяти, её декодирование, передача номера команды и операнда в АЛУ, определение следующей выполняемой команды и реализации взаимодействий с клавиатурой и монитором. Выбор очередной команды из оперативной памяти производится по сигналу от системного таймера. Если установлен флаг «игнорирование тактовых импульсов», то эти сигналы устройством управления игнорируются. В ходе выполнения операций устройство управления устанавливает значения флагов «указана неверная команда» и «игнорирование тактовых импульсов».

Обработчик прерываний реагирует на сигналы от системного таймера и кнопки «Reset». При поступлении сигнала от кнопки «Reset» состояние процессора сбрасывается в начальное (значения всех регистров обнуляется и устанавливается флаг «игнорирование сигналов от таймера»). При поступлении сигнала от системного таймера, работать начинает устройство управления.

Получив текущую команду из оперативной памяти, устройство управления декодирует её с целью определить номер функции, которую надо выполнить и операнд. Формат команды следующий старший разряд содержит признак команды (0 –

команда), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд.

Выполнение команд центральным процессором Simple Computer

Команды выполняются последовательно. Адрес ячейки памяти, в которой находится текущая выполняемая команда, задается в регистре «Счетчик команд». Устройство управления запрашивает содержимое указанной ячейки памяти и декодирует его согласно используемому формату команд. Получив код операции, устройство управления определяет, является ли эта операция арифметико-логической. Если да, то выполнение операции передается в АЛУ. В противном случае операция выполняется устройством управления. Процедура выполняется до тех пор, пока флаг «останов» не будет равен 1.

Консоль управления

Интерфейс консоли управления представлен на рисунке 1.

Он содержит следующие области:

— Memory – содержимое оперативной памяти Simple Computer;

— Accumulator – значение, находящееся в аккумуляторе;

— instructionCounter – значение регистра «счетчик команд»;

— Operation – результат декодирования операции;

— Flags – состояние регистра флагов («P» -переполнение при выполнении операции, «O» -ошибка деления на 0, «M» -ошибка выхода за границы памяти, «T» -игнорирование тактовых импульсов, «E» -указана неверная команда);

— Cell – значение выделенной ячейки памяти в области;

— Memory (используется для редактирования);

— Keys – подсказка по функциональным клавишам;

— Input/Otput – область, используемая Simple Computer в процессе выполнения программы для ввода информации с клавиатуры и вывода её на экран.

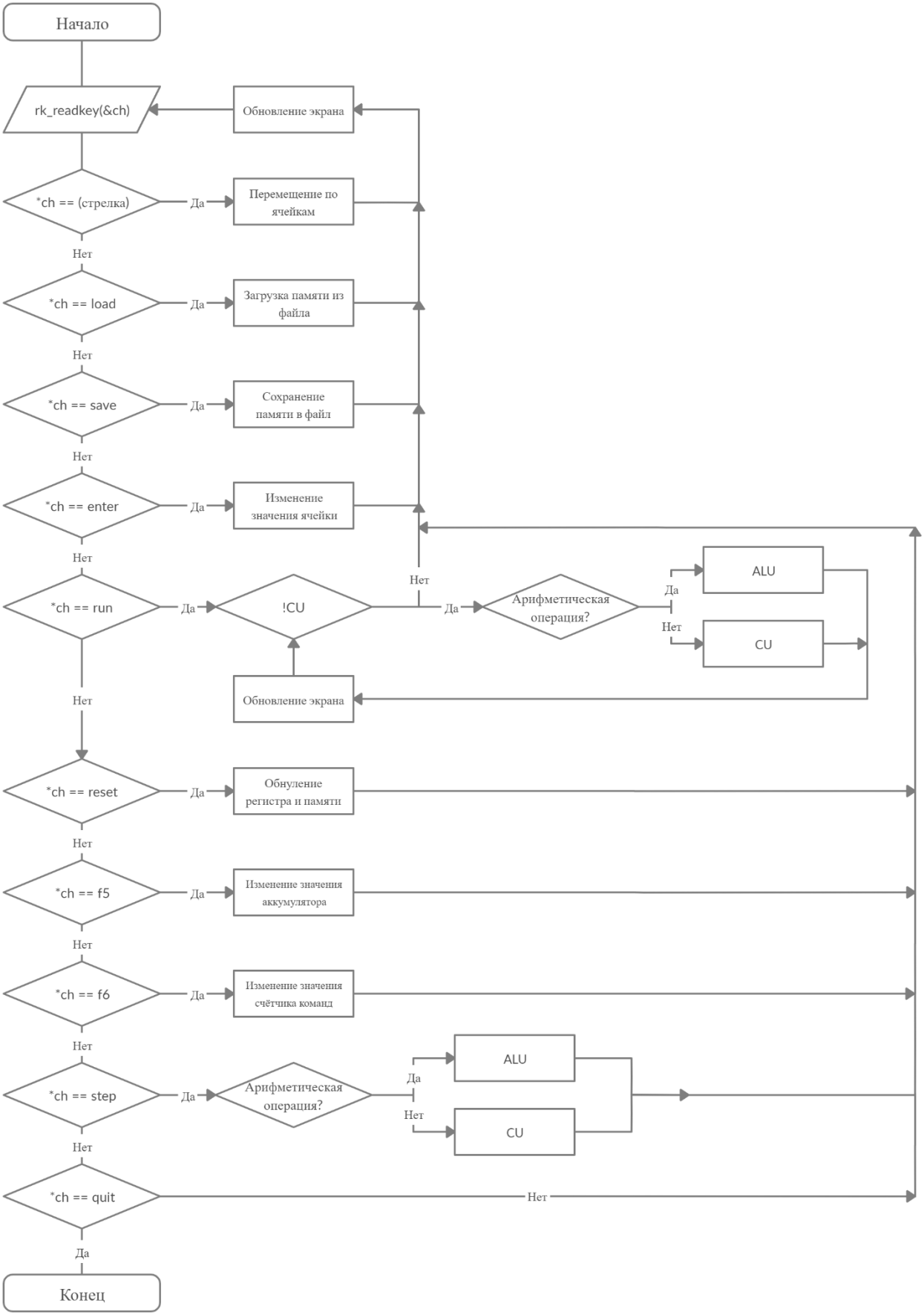
Содержимое ячеек памяти и регистров центрального процессора выводится в декодированном виде. При этом, знак «+» соответствует значению 0 в поле «признак команды», следующие две цифры – номер команды и затем операнд в шестнадцатеричной системе счисления.

Пользователь имеет возможность с помощью клавиш управления курсора выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу «F5», пользователь может задать значение аккумулятору, «F6» – регистру «счетчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может, нажав на клавиши «l», «s» соответственно (после нажатия в поле Input/Output пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорировать такты таймера» в 0) можно с помощью клавиши «r». В процессе выполнения программы, редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу «t». Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу «i».

В соответствии с вариантом, необходимо реализовать одну из пользовательских команд, а именно команду:

— XOR - 54 - Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)

2. Блок-схема алгоритма



3. Краткое описание работы программы

Описание работы Simple Computer

Инициализируем память и регистры Simple Computer. Затем программа начинает отслеживать нажатия пользователем клавиш до тех пор, пока не будет нажата клавиша q.

Клавиши:

— Стрелки вверх, вниз, влево, вправо - позволяют перемещать курсор по памяти компьютера;

— «l» - позволяет загрузить данные о памяти Simple Computer, после ввода названия файла;

— «s» - позволяет сохранить данные о памяти в файл;

— «Enter» - позволяет ввести собственное значение в ячейку памяти;

— «f5» - позволяет изменить значение, хранящееся в аккумуляторе;

— «f6» - позволяет изменить значение, хранящееся в счетчике команд;

— «t» - программа выполняет один такт управляющего устройства Control Unit;

— «i» - происходит очистка памяти и регистров;

При нажатии «t» производится вызов такта Control Unit. Происходит декодирование команды ячейки памяти, на которую указывает счетчик команд. Если эта команда является арифметико-логической командой, то Control Unit передает эту команду арифметико-логическому устройству (ALU), иначе выполняет данную команду самостоятельно.

4. Листинг

1. My_Simple_Computer.h

```
#ifndef MY_SIMPLE_COMPUTER
#define MY_SIMPLE_COMPUTER

#include <stdio.h>
#include <stdlib.h>

#define SIZE 100

//FLAGS:
#define M 0x00000001
#define E 0x00000010
#define T 0x00000100
#define O 0x00001000
#define P 0x00010000

int memory[SIZE];
extern int correct_command[38];
int registr;

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 flag, int value);
int sc_regGet(int flag, int *value);
int sc_commandEncode(int command, int operand, int *value);
int sc_commandDecode(int value, int *command, int *operand);

#endif
```

2. My_Simple_Computer.c

```
#include "My_Simple_Computer.h"

int correct_command[] = {0x10, 0x11, 0x20, 0x21, 0x30, 0x31, 0x32, 0x33, 0x40, 0x41, 0x42,
0x43, 0x51, 0x52, 0x53, 0x54,
                         0x55, 0x56, 0x57, 0x58, 0x59, 0x60, 0x61, 0x62, 0x63, 0x64, 0x65,
0x66, 0x67, 0x68, 0x69, 0x70,
                         0x71, 0x72, 0x73, 0x74, 0x75, 0x76};

int sc_memoryInit() {
    for (int i = 0; i < SIZE; i++) memory[i] = 0;
    return 0;
}

int sc_memorySet(int address, int value) {
    if ((address >= SIZE) || (address < 0)) {
        sc_regSet(M, 1);
        return 1;
    }
    if (value > 0xFFFF) return 1;
    sc_regSet(M, 0);
    memory[address] = value;
    return 0;
}

int sc_memoryGet(int address, int *value) {
    if ((address > SIZE) || (address < 0)) {
        sc_regSet(M, 1);
        return 1;
    }
```

```
    sc_regSet(M, 0);
    *value = memory[address];
    return 0;
}

int sc_memorySave(char *filename) {
    FILE *f = fopen(filename, "wb");
    fwrite(&memory, sizeof(int), SIZE, f);
    fclose(f);
    return 0;
}

int sc_memoryLoad(char *filename) {
    FILE *f = fopen(filename, "rb");
    if (!f) {
        return 1;
    } else {
        fread(memory, sizeof(int), SIZE, f);
        fclose(f);
    }
    return 0;
}

int sc_regInit() {
    registr = 0;
    return 0;
}

int sc_regSet(int flag, int value) {
    int k;
    if ((value == 0) || (value == 1)) {
        switch (flag) {
            case M:
                k = 0;
                break;
            case E:
                k = 4;
                break;
            case T:
                k = 8;
                break;
            case O:
                k = 12;
                break;
            case P:
                k = 16;
                break;
            default:
                return 1;
        }
        if (value == 1) registr = registr | (1 << k);
        else registr = registr & (~(1 << k));
        return 0;
    } else {
        return 1;
    }
}

int sc_regGet(int flag, int *value) {
    int k;
    switch (flag) {
        case M:
            k = 0;
            break;
        case E:
            k = 4;
            break;
        case T:
            k = 8;
```

```
            break;
        case O:
            k = 12;
            break;
        case P:
            k = 16;
            break;
        default:
            return 1;
    }
    *value = (registr >> k) & 0x1;
    return 0;
}

int sc_commandEncode(int command, int operand, int *value) {
    int k = 0;
    for (int i = 0; i < 38; i++) {
        if (command == correct_command[i]) {
            k = 1;
            break;
        }
    }
    if (k == 0) {
        sc_regSet(E, 1);
        return 1;
    }
    if ((operand > 0x7F) || (operand < 0)) return 1;
    *value = (command << 7) | operand;
    sc_regSet(E, 0);
    return 0;
}

int sc_commandDecode(int value, int *command, int *operand) {
    int command_1 = 0, operand_1 = 0, k = 0;

    if (value >= 0x4000) return 1;
    command_1 = command_1 | (value >> 7);
    operand_1 = value & 0x7F;

    for (int i = 0; i < 38; i++) {
        if (command_1 == correct_command[i]) {
            k = 1;
            break;
        }
    }
    if (k == 0) {
        sc_regSet(E, 1);
        return 1;
    }
    if ((operand_1 > 0x7F) || (operand_1 < 0)) return 1;
    sc_regSet(E, 0);
    *command = command_1;
    *operand = operand_1;
    return 0;
}
```

3. My_Term.h

```
#ifndef MY_TERM
#define MY_TERM

#include <stdio.h>
#include <termios.h>
#include <sys/ioctl.h>

enum colors {
    black,
```

```
        red,
        green,
        yellow,
        blue,
        violet,
        light_blue,
        white
};

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);

#endif
```

4. My_Term.c

```
#include "My_Term.h"

int mt_clrscr() {
    printf("=\E[H\E[J");
    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_gotoXY(int x, int y) {
    int rows, cols;

    mt_getscreensize(&rows, &cols);
    if ((x >= 0) && (x <= rows) && (y >= 0) && (y <= cols)) {
        printf("\E[%d;%dH", x, y);
        return 0;
    }
    else {
        return 1;
    }
}

int mt_setfgcolor(enum colors color) {
    switch (color) {
              case black:
                     printf("\E[30m");
                     break;
              case red:
                     printf("\E[31m");
                     break;
              case green:
                     printf("\E[32m");
                     break;
              case yellow:
                     printf("\E[33m");
                     break;
              case blue:
                     printf("\E[34m");
```

```
                        break;
                case violet:
                        printf("\E[35m");
                        break;
                case light_blue:
                        printf("\E[36m");
                        break;
                case white:
                        printf("\E[37m");
                        break;
                default:
                        return -1;
        }
        return 0;
}

int mt_setbgcolor(enum colors color) {
    switch (color) {
                case black:
                        printf("\E[40m");
                        break;
                case red:
                        printf("\E[41m");
                        break;
                case green:
                        printf("\E[42m");
                        break;
                case yellow:
                        printf("\E[43m");
                        break;
                case blue:
                        printf("\E[44m");
                        break;
                case violet:
                        printf("\E[45m");
                        break;
                case light_blue:
                        printf("\E[46m");
                        break;
                case white:
                        printf("\E[47m");
                        break;
                default:
                        return -1;
        }
        return 0;
}
```

5. My_Big_Chars.h

```
#ifndef MY_BIG_CHARS
#define MY_BIG_CHARS

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "My_Term.h"

#define VERTICAL_LINE "x"            // │
#define HORIZONTAL_LINE "q"          // ─
#define BOTTOM_RIGHT_CORNER "j"      // ┘
#define TOP_RIGHT_CORNER "k"         // ┐
#define BOTTOM_LEFT_CORNER "m"       // └
#define TOP_LEFT_CORNER "l"          // ┌
#define PAINT "a"                    // ▒
```

```
extern int zero[2];
extern int one[2];
extern int two[2];
extern int three[2];
extern int four[2];
extern int five[2];
extern int six[2];
extern int seven[2];
extern int eight[2];
extern int nine[2];
extern int bigA[2];
extern int bigB[2];
extern int bigC[2];
extern int bigD[2];
extern int bigE[2];
extern int bigF[2];
extern int plus[2];
extern int minus[2];

int bc_printA(char *str);
int bc_box(int x1, int y1, int x2, int y2);
int bc_printbigchar(int *big, int x1, int y1, enum colors fgcolor, enum colors bgcolor);
int bc_setbigcharpos(int *big, int x, int y, int value);
int bc_getbigcharpos(int *big, int x, int y, int *value);
int bc_bigcharwrite(int fd, int *big, int count);
int bc_bigcharread(int fd, int *big, int need_count, int *count);

#endif
```

6. My_Big_Chars.c

```
#include "My_Big_Chars.h"

int zero[2] = {1246118460, 1010983506};
int one[2] = {137893896, 1040713736};
int two[2] = {33702460, 2114979844};
int three[2] = {67634302, 1010958850};
int four[2] = {1210587144, 134744188};
int five[2] = {41697406, 1010958850};
int six[2] = {2084577308, 1010975298};
int seven[2] = {134480510, 269488144};
int eight[2] = {1010975292, 1010975298};
int nine[2] = {1111638588, 939786814};
int bigA[2] = {1111638588, 1111638654};
int bigB[2] = {2084717180, 2084717122};
int bigC[2] = {1077952574, 1044398144};
int bigD[2] = {1111638652, 2084717122};
int bigE[2] = {2118140030, 2118139968};
int bigF[2] = {2118140030, 1077952576};
int plus[2] = {4279769112, 404232447};
int minus[2] = {4278190080, 255};

int bc_printA(char *str) {
    printf("\E(0%s\E(B", str);
    return 0;
}

int bc_box(int x1, int y1, int x2, int y2) {
    int x, y;
    mt_getscreensize(&x, &y);
    mt_setfgcolor(white);

    if ((x1 < 0) || (y1 < 0) || (x2 > x) || (y2 > y) || (x2 - x1 < 1) || (y2 - y1 < 1)) return
-1;

    mt_gotoXY(x1, y1);
    bc_printA(TOP_LEFT_CORNER);
```

```
    mt_gotoXY(x2, y1);
    bc_printA(BOTTOM_LEFT_CORNER);

    mt_gotoXY(x1, y2);
    bc_printA(TOP_RIGHT_CORNER);

    mt_gotoXY(x2, y2);
    bc_printA(BOTTOM_RIGHT_CORNER);

    for (int i = x1 + 1; i < x2; i++) {
        mt_gotoXY(i, y1);
        bc_printA(VERTICAL_LINE);
        mt_gotoXY(i, y2);
        bc_printA(VERTICAL_LINE);
    }

    for (int i = y1 + 1; i < y2; i++) {
        mt_gotoXY(x1, i);
        bc_printA(HORIZONTAL_LINE);
        mt_gotoXY(x2, i);
        bc_printA(HORIZONTAL_LINE);
    }
    return 0;
}

int bc_printbigchar(int *big, int x1, int y1, enum colors fgcolor, enum colors bgcolor) {
    int x, y;
    mt_getscreensize(&x, &y);
    x1++;

    if ((x1 < 0) || (y1 < 0) || (x1 + 8 > x) || (y1 + 8 > y)) return -1;
    int bit = 0;

    mt_setfgcolor(fgcolor);
    mt_setbgcolor(bgcolor);

    for (int i = 0, j = 0, k = 8; i < 32; i++, k--) {
        if (k == 0) {
            k = 8;
            j++;
        }
        bit = (big[0] >> i) & 1;
        mt_gotoXY(x1 + j, y1 + k);
        if (bit == 0) bc_printA(" ");
        else bc_printA(PAINT);
    }
    for (int i = 0, j = 4, k = 8; i < 32; i++, k--) {
        if (k == 0) {
            k = 8;
            j++;
        }
        bit = (big[1] >> i) & 1;
        mt_gotoXY(x1 + j, y1 + k);
        if (bit == 0) bc_printA(" ");
        else bc_printA(PAINT);
    }
    mt_setfgcolor(white);
    mt_setbgcolor(black);
    return 0;
}

int bc_setbigcharpos(int *big, int x, int y, int value) {
    if ((x < 0) || (x > 7) || (y < 0) || (y > 7) || (value < 0) || (value > 1)) return -1;

    int pos;
    if (x >= 4) pos = 1;
    else pos = 0;
```

```
    y = 7 - y;
    if (value == 0) big[pos] &= ~(1 << (8 * x + y));
    else big[pos] |= 1 << (8 * x + y);
    return 0;
}

int bc_getbigcharpos(int *big, int x, int y, int *value) {
    if ((x < 0) || (x > 7) || (y < 0) || (y > 7)) return -1;

    int pos;
    if (x >= 4) pos = 1;
    else pos = 0;

    y = 7 - y;
    *value = (big[pos] >> (8 * x + y) & 1);
    return 0;
}

int bc_bigcharwrite(int fd, int *big, int count) {  //O_WRONLY
    int res;
    if (res = (write(fd, big, count * (sizeof(int) * 2))) == -1) return -1;
    return 0;
}

int bc_bigcharread(int fd, int *big, int need_count, int *count) {  //O_RDONLY
    int res;
    *count = 0;
    if ((res = read(fd, big, need_count * sizeof(int) * 2)) == -1) return -1;
    *count = res / (sizeof(int) * 2);
    return 0;
}
```

7. My_Read_Key.h

```
#ifndef MY_READ_KEY_H
#define MY_READ_KEY_H

#include "My_Big_Chars.h"
#include "My_Term.h"
#include <string.h>

enum keys {
    q,
    l,
    s,
    r,
    t,
    i,
    f5,
    f6,
    up,
    down,
    left,
    right,
    enter
};

int rk_readkey(int *key);
int rk_mytermsave();
int rk_mytermrestore();
int rk_mytermregime(int regime, int vtime, int vmin, int echo, int sigint);

#endif
```

8. My_Read_Key.c

```
#include "My_Read_Key.h"
```

```
int rk_readkey(int *key) {
    struct termios options;
    char term[8];
    int tread;

    if (tcgetattr(0, &options) != 0) return -1;
    if (rk_mytermregime(0, 0, 1, 0, 1) != 0) return -1;

    tread = read(0, term, 8);
    if (tread == -1) return -1;
    term[tread] = '\0';

    if (strcmp(term, "l") == 0) *key = l;
    else if (strcmp(term, "s") == 0) *key = s;
    else if (strcmp(term, "r") == 0) *key = r;
    else if (strcmp(term, "t") == 0) *key = t;
    else if (strcmp(term, "i") == 0) *key = i;
    else if (strcmp(term, "q") == 0) *key = q;
    else if (strcmp(term, "\E[15~") == 0) *key = f5;
    else if (strcmp(term, "\E[17~") == 0) *key = f6;
    else if (strcmp(term, "\E[A") == 0) *key = up;
    else if (strcmp(term, "\E[B") == 0) *key = down;
    else if (strcmp(term, "\E[D") == 0) *key = left;
    else if (strcmp(term, "\E[C") == 0) *key = right;
    else if (strcmp(term, "\n") == 0) *key = enter;
    else return -1;

    if (tcsetattr(0, TCSANOW, &options) != 0) return -1;

    return 0;
}

int rk_mytermsave() {
    struct termios options;
    if (tcgetattr(0, &options) != 0) return -1;

    FILE *file = fopen("termsettings.txt", "wb");
    if (file == NULL) return -1;

    fwrite(&options, sizeof(options), 1, file);
       fclose(file);

       return 0;
}

int rk_mytermrestore() {
    struct termios options;

    FILE *file = fopen("termsettings.txt", "rb");
    if (file == NULL) return -1;

    if (fread(&options, sizeof(options), 1, file) > 0) {
        if (tcsetattr(0, TCSAFLUSH, &options) != 0) return -1;
    } else return -1;
    fclose(file);

    return 0;
}

int rk_mytermregime(int regime, int vtime, int vmin, int echo, int sigint) {
    struct termios options;

    if (tcgetattr(0, &options) != 0) return -1;

    if (regime == 1) options.c_lflag |= ICANON;
    else if (regime == 0) options.c_lflag &= ~ICANON;
    else return -1;
```

```
    if (regime == 0) {
        options.c_cc[VTIME] = vtime;
              options.c_cc[VMIN] = vmin;

        if (echo == 1) options.c_lflag |= ECHO;
        else if (echo == 0) options.c_lflag &= ~ECHO;
        else return -1;

        if (sigint == 1) options.c_lflag |= ISIG;
        else if (sigint == 0) options.c_lflag &= ~ISIG;
        else return 0;
    }
    if (tcsetattr(0, TCSANOW, &options) != 0) return -1;

    return 0;
}
```

9. cpu.h

```
#ifndef CPU_H
#define CPU_H

#include "My_Simple_Computer.h"
#include "My_Term.h"
#include <stdio.h>

extern int inst_counter;
extern int accumulator;
extern int count;

#define READ 0x10
#define WRITE 0x11

#define LOAD 0x20
#define STORE 0x21

#define ADD 0x30
#define SUB 0x31
#define DIVIDE 0x32
#define MUL 0x33

#define JUMP 0x40
#define JNEG 0x41
#define JZ 0x42
#define HALT 0x43

#define XOR 0x54

int CU();
int ALU(int command, int operand);

#endif
```

10. cpu.c

```
#include "cpu.h"
#include "My_Read_Key.h"
#include "main.h"

int inst_counter = 0;
int accumulator = 0;
int count = 0;

int CU() {
    int value = 0;
       int command = 0;
```

```
    int operand = 0;

    sc_memoryGet(inst_counter, &value);

    if (sc_commandDecode(value, &command, &operand)) {
           sc_regSet(E, 1);
           sc_regSet(T, 0);
           return 1;
    }
if (command >= 0x30 && command <= 0x33) {
     next_step();
     ALU(command, operand);
    } else {
     switch (command) {
         case READ:
                         rk_mytermregime(0, 0, 1, 1, 1);
                         printf("\t> ");
                         int value;
           scanf("%d", &value);
                         rk_mytermregime(0, 0, 1, 0, 1);
                         if (value > 0x7FFF) return 1;
            if (value <= 0) value = (-value) + 0x4000;
                         sc_memorySet(operand, value);
                         next_step();
                         count++;
                         break;
                case WRITE:
                         printf("\t< ");
                         sc_memoryGet(operand, &value);
            if (value >= 0x4000) value = value & 0x1FFF;
                         printf("%X\n", value);
                         next_step();
            count++;
                         break;
                case LOAD:
                         sc_memoryGet(operand, &accumulator);
                         next_step();
                         break;
                case STORE:
                         sc_memorySet(operand, accumulator);
                         next_step();
                         break;
                case JUMP:
                         if ((operand < 0) || (operand > 0x63 )) {
                                sc_regSet(M, 1);
                                return 1;
                         }
                         inst_counter = operand;
                         sleep(1);
                         break;
                case JNEG:
                         if ((accumulator >> 14) == 1) {
                                if (((accumulator >> 13 ) & 1) == 1) {
                                       if (operand > 0x63) {
                                              sc_regSet(M, 1);
                                              return 1;
                                       } else inst_counter = operand;
                                } else next_step();
                         } else next_step();
                         sleep(1);
                         break;
                case JZ:
                         if ((accumulator >> 14) == 1) {
                                if (((accumulator >> 13) & 1) == 0) {
                                       if ((accumulator & 0x1FFF) == 0){
                                              if (operand > 0x63) {
                                                     sc_regSet(M, 1);
                                                     return 1;
                                              } else inst_counter = operand;
```

```
                                        } else next_step();
                                    } else next_step();
                                } else next_step();
                                sleep(1);
                                break;
                        case HALT:
                                return 1;
                        case XOR:
                                sc_memoryGet(operand, &value);
                                accumulator ^= value;
                                next_step();
                                break;
                }
        }
        return 0;
}

int ALU(int command, int operand) {
    int value;
    sc_memoryGet(operand, &value);

    switch (command) {
        case ADD:
            if (((value >> 14) == 1) && ((accumulator >> 14) == 1)) {
                int b = (value >> 13) & 1;
                int a = (accumulator >> 13) & 1;
                if (a == b) {
                    if ((accumulator & 0x1FFF) + (value & 0x1FFF) > 0x1FFF) {
                        sc_regSet(P, 1);
                        return -1;
                    } else accumulator = (accumulator & 0x1FFF) + (value & 0x1FFF) + (1 << 14)
+ (a << 13);
                } else {
                    if ((accumulator & 0x1FFF) > (value & 0x1FFF)) {
                        accumulator = (accumulator & 0x1FFF) - (value & 0x1FFF) + (1 << 14) +
(a << 13);
                    } else if ((accumulator & 0x1FFF) < (value & 0x1FFF)) {
                        accumulator = (value & 0x1FFF) - (accumulator & 0x1FFF) + (1 << 14) +
(b << 13);
                    } else if ((accumulator & 0x1FFF) == (value & 0x1FFF)) {
                        accumulator = (value & 0x1FFF) - (accumulator & 0x1FFF) + (1 << 14);
                    }
                }
            } else return -1;
            break;
        case SUB:
            if ((value >> 14) == 1 && (accumulator >> 14 == 1)) {
                int a = accumulator >> 13 & 1;
                int b = value >> 13 & 1;
                if (a == b) {
                    if (b == 0) b = 1;
                    else b = 0;
                    if ((accumulator & 0x1FFF) > (value & 0x1FFF)) {
                        accumulator = (accumulator & 0x1FFF) - (value & 0x1FFF) + (1 << 14) +
(a << 13);
                    } else if ((accumulator & 0x1FFF) < (value & 0x1FFF)) {
                        accumulator = (value & 0x1FFF) - (accumulator & 0x1FFF) + (1 << 14) +
(b << 13);
                    } else if ((accumulator & 0x1FFF) == (value & 0x1FFF)) {
                        accumulator = (value & 0x1FFF) - (accumulator & 0x1FFF) + (1 << 14);
                    }
                } else {
                    if ((accumulator & 0x1FFF) + (value & 0x1FFF) > 0x1FFF) {
                        sc_regSet(P, 1);
                        return -1;
                    } else accumulator = (accumulator & 0x1FFF) + (value & 0x1FFF) + (1 << 14)
+ (a << 13);
                }
            } else return -1;
```

```
            break;
        case DIVIDE:
            if ((value >> 14 == 1) && (accumulator >> 14 == 1)) {
                if ((value & 0x1FFF) == 0) {
                    sc_regSet(O, 1);
                    return -1;
                } else if (((value >> 13) & 1) == ((accumulator >> 13) & 1)) {
                    accumulator = (accumulator & 0x1FFF) / (value & 0x1FFF) + (1 << 14);
                } else accumulator = (accumulator & 0x1FFF) / (value & 0x1FFF) + (1 << 13) +
(1 << 14);
            } else return -1;
            break;
        case MUL:
            if ((value >> 14 == 1) && (accumulator >> 14 == 1)) {
                if ((accumulator & 0x1FFF) * (value & 0x1FFF) > 0x1FFF) {
                    sc_regSet(P, 1);
                    return -1;
                } else if ((accumulator >> 13 & 1) == (value >> 13 & 1)) {
                    accumulator = (accumulator & 0x1FFF) * (value & 0x1FFF) + (1 << 14);
                } else accumulator = (accumulator & 0x1FFF) * (value & 0x1FFF) + (1 << 14) +
(1 << 13);
            } else return -1;
    }
    return 0;
}
```

11. main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "My_Simple_Computer.h"
#include "My_Term.h"
#include "My_Big_Chars.h"
#include "My_Read_Key.h"
#include "cpu.h"
#include <signal.h>

void print_memory();
void print_accumulator();
void select_number(int number, int *big);
void print_instruction_counter();
void print_opertion();
void print_flags();
void print_keys();
void active_window();
void print_boxes();
void print_info(int count);
void next_step();

#endif
```

12. main.c

```
#include "main.h"

void print_memory() {
    int count = 0;

    mt_gotoXY(2, 9);
    for (int i = 0; i < SIZE; i++) {
        if (memory[i] >= 0x4000) printf("-%04X\t", memory[i]);
        else printf("+%04X\t", memory[i]);
        count++;
        if (count == 10) {
            printf("\n\t");
```

```
            count = 0;
        }
    }
}

void print_accumulator() {
    mt_setfgcolor(white);
    mt_gotoXY(2, 101);
    (accumulator >= 0x4000) ? printf("-%04X\t", accumulator) : printf("+%04X\t", accumulator);
}

void select_number(int number, int *big) {
    switch (number) {
        case 0:
            big[0] = zero[0];
            big[1] = zero[1];
            break;
        case 1:
            big[0] = one[0];
            big[1] = one[1];
            break;
        case 2:
            big[0] = two[0];
            big[1] = two[1];
            break;
        case 3:
            big[0] = three[0];
            big[1] = three[1];
            break;
        case 4:
            big[0] = four[0];
            big[1] = four[1];
            break;
        case 5:
            big[0] = five[0];
            big[1] = five[1];
            break;
        case 6:
            big[0] = six[0];
            big[1] = six[1];
            break;
        case 7:
            big[0] = seven[0];
            big[1] = seven[1];
            break;
        case 8:
            big[0] = eight[0];
            big[1] = eight[1];
            break;
        case 9:
            big[0] = nine[0];
            big[1] = nine[1];
            break;
        case 10:
            big[0] = bigA[0];
            big[1] = bigA[1];
            break;
        case 11:
            big[0] = bigB[0];
            big[1] = bigB[1];
            break;
        case 12:
            big[0] = bigC[0];
            big[1] = bigC[1];
            break;
        case 13:
            big[0] = bigD[0];
            big[1] = bigD[1];
            break;
```

```
        case 14:
            big[0] = bigE[0];
            big[1] = bigE[1];
            break;
        case 15:
            big[0] = bigF[0];
            big[1] = bigF[1];
            break;
        default:
            break;
    }
}

void print_instruction_counter() {
    mt_setfgcolor(white);
    mt_gotoXY(5, 101);
    printf("+%04X", inst_counter);

    bc_box(13, 7, 22, 54);
    int i_count = memory[inst_counter];
    if (i_count < 0) bc_printbigchar(minus, 13, 7, white, black);
    else bc_printbigchar(plus, 13, 7, white, black);
    int *big = malloc(sizeof(int) * 2);

    int number = i_count % 16;
    i_count /= 16;
    select_number(number, big);
    bc_printbigchar(big, 13, 45, white, black);

    number = i_count % 16;
    i_count /= 16;
    select_number(number, big);
    bc_printbigchar(big, 13, 35, white, black);

    number = i_count % 16;
    i_count /= 16;
    select_number(number, big);
    bc_printbigchar(big, 13, 25, white, black);

    number = i_count % 16;
    select_number(number, big);
    bc_printbigchar(big, 13, 15, white, black);
}

void print_opertion() {
    int value = memory[inst_counter];
    int command = 0, operand = 0;
    mt_gotoXY(8, 99);
    sc_commandDecode(value, &command, &operand);
    mt_setfgcolor(white);
    printf("+%02X : %02X\n", command, operand);
}

void print_flags() {
    int m, e, t, o, p;

    sc_regGet(M, &m);
    sc_regGet(E, &e);
    sc_regGet(T, &t);
    sc_regGet(O, &o);
    sc_regGet(P, &p);

    mt_gotoXY(11, 93);
    if (m == 1) mt_setfgcolor(red);
    else mt_setfgcolor(green);
    printf("M");

    mt_gotoXY(11, 98);
    if (e == 1) mt_setfgcolor(red);
```

```
    else mt_setfgcolor(green);
    printf("E");

    mt_gotoXY(11, 103);
    if (t == 1) mt_setfgcolor(red);
    else mt_setfgcolor(green);
    printf("T");

    mt_gotoXY(11, 108);
    if (o == 1) mt_setfgcolor(red);
    else mt_setfgcolor(green);
    printf("O");

    mt_gotoXY(11, 113);
    if (p == 1) mt_setfgcolor(red);
    else mt_setfgcolor(green);
    printf("P");
}

void print_keys() {
    mt_setfgcolor(white);
    mt_gotoXY(14, 58);
    printf("l - load\n");
    mt_gotoXY(15, 58);
    printf("s - save\n");
    mt_gotoXY(16, 58);
    printf("r - run\n");
    mt_gotoXY(17, 58);
    printf("t - step\n");
    mt_gotoXY(18, 58);
    printf("i - reset\n");
    mt_gotoXY(19, 58);
    printf("f5 - accumulator\n");
    mt_gotoXY(20, 58);
    printf("f6 - instructionCounter\n");
}

void active_window() {
    int x = 2 + (inst_counter / 10);
    int y = 9 + (inst_counter % 10) * 8;

    mt_gotoXY(x, y);
    mt_setfgcolor(black);
    mt_setbgcolor(white);
    (memory[inst_counter] >= 0x4000) ? printf("-%04X\t", memory[inst_counter]) :
printf("+%04X\t", memory[inst_counter]);
    mt_setfgcolor(white);
    mt_setbgcolor(black);
}

void print_boxes() {
    bc_box(1, 7, 12, 87);
    mt_gotoXY(0, 44);
    mt_setfgcolor(yellow);
    printf(" Memory \n");
    mt_setfgcolor(white);

    bc_box(1, 88, 3, 119);
    mt_gotoXY(0, 97);
    mt_setfgcolor(yellow);
    printf(" accumulator \n");

    bc_box(4, 88, 6, 119);
    mt_gotoXY(4, 94);
    mt_setfgcolor(yellow);
    printf(" instructionCounter \n");

    bc_box(7, 88, 9, 119);
    mt_gotoXY(7, 98);
```

```
    mt_setfgcolor(yellow);
    printf(" Operation \n");

    bc_box(10, 88, 12, 119);
    mt_gotoXY(10, 100);
    mt_setfgcolor(yellow);
    printf(" Flags \n");

    bc_box(13, 55, 22, 119);
    mt_gotoXY(13, 67);
    mt_setfgcolor(yellow);
    printf(" Keys: \n");
}

void print_info(int count) {
    print_memory();
    print_opertion();
    print_flags();
    active_window();
    print_accumulator();
    print_instruction_counter();
    mt_gotoXY(23, 7);
    printf("Input/Output:\n");
    for (int i = 0; i < count; i++) printf("\n");
}

void next_step() {
    sleep(1);
    inst_counter++;
}

int main()
{
    char filename[32];
    int value;
    int command = 0, operand = 0;
    int ch;

    mt_clrscr();
    print_boxes();
    print_keys();

    while (1) {
        value = 0;
        print_info(count);
        rk_readkey(&ch);
        rk_mytermregime(0, 0, 1, 0, 1);
        switch (ch) {
            case up:
                if (inst_counter <= 9) break;
                inst_counter -= 10;
                break;
            case down:
                if (inst_counter >= 90) break;
                inst_counter += 10;
                break;
            case left:
                if (inst_counter % 10 == 0) break;
                inst_counter--;
                break;
            case right:
                if ((inst_counter + 1) % 10 == 0) break;
                inst_counter++;
                break;
            case l:
                rk_mytermregime(0, 0, 1, 1, 1);
                printf("\t> ");
                scanf("%s", filename);
                sc_memoryLoad(filename);
```

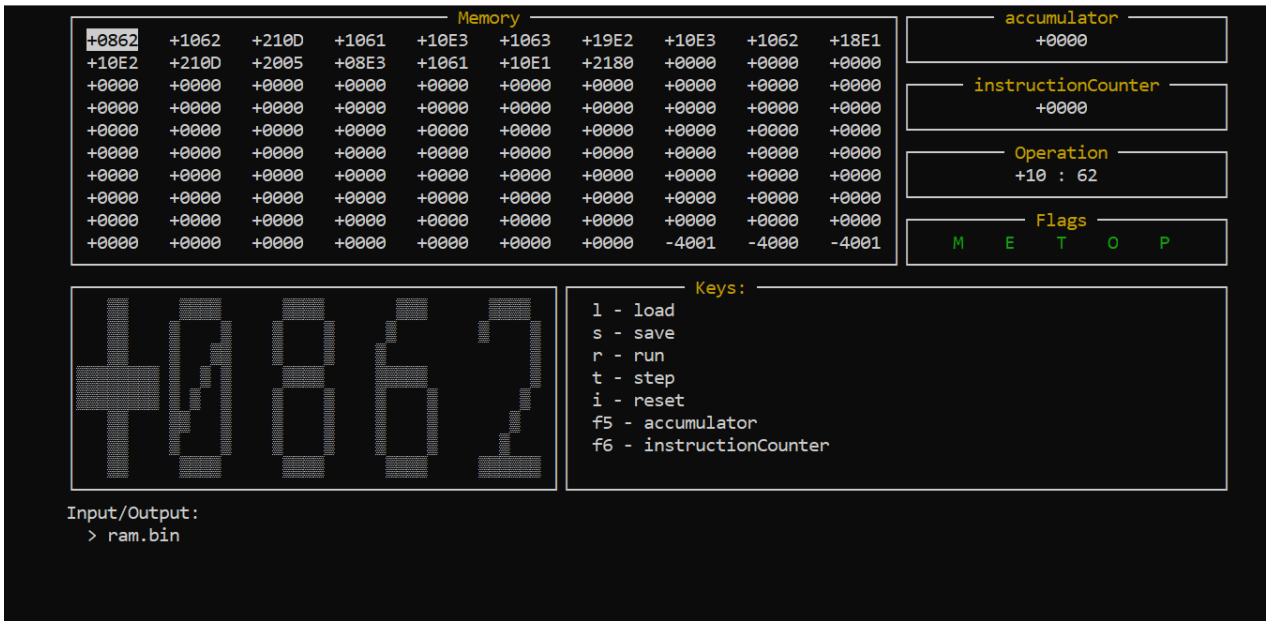
```
                count++;
                rk_mytermregime(0, 0, 1, 0, 1);
                break;
            case s:
                rk_mytermregime(0, 0, 1, 1, 1);
                printf("\t> ");
                scanf("%s", filename);
                sc_memorySave(filename);
                count++;
                rk_mytermregime(0, 0, 1, 0, 1);
                break;
            case enter:
                rk_mytermregime(0, 0, 1, 1, 1);
                printf("\t> ");
                scanf("%x", &value);
                sc_memorySet(inst_counter, value);
                count++;
                rk_mytermregime(0, 0, 1, 0, 1);
                break;
            case r:
                sc_regSet(T, 1);
                while (!CU()) print_info(count);
                sc_regSet(T, 0);
                break;
            case i:
                sc_memoryInit();
                sc_regInit();
                inst_counter = 0;
                accumulator = 0;
                break;
            case f5:
                rk_mytermregime(0, 0, 1, 1, 1);
                printf("\t> ");
                scanf("%x", &value);
                accumulator = value;
                count++;
                rk_mytermregime(0, 0, 1, 0, 1);
                break;
            case f6:
                rk_mytermregime(0, 0, 1, 1, 1);
                printf("\t> ");
                scanf("%x", &value);
                inst_counter = value;
                count++;
                rk_mytermregime(0, 0, 1, 0, 1);
                break;
            case t:
                CU();
                break;
            case q:
                rk_mytermregime(0, 0, 1, 1, 1);
                return 0;
            default:
                break;
        }
        if (count == 10) {
            mt_clrscr();
            print_boxes();
            print_keys();
            count = 0;
        }
    }
}
```

13. Makefile

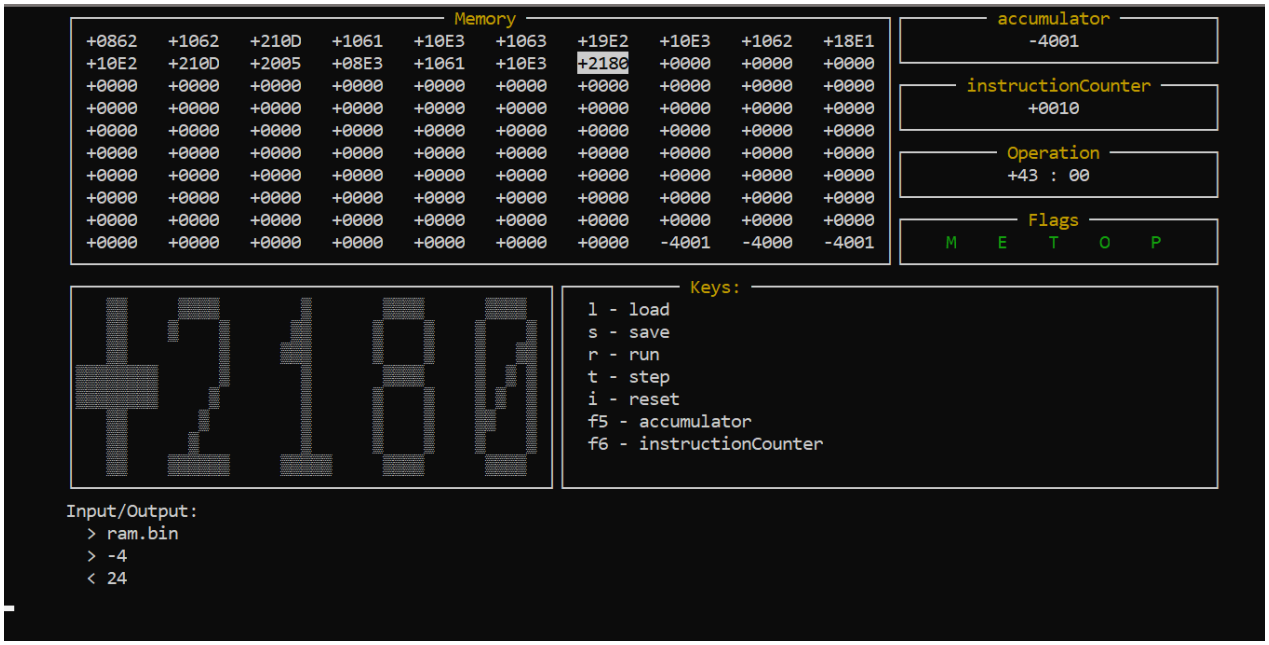
```
computer: main.o libcomputer.a libterm.a libchars.a libkey.a libcpu.a
```

```
	gcc -o computer main.o -L. -lcomputer -lterm -lchars -lkey -lcpu
	rm *.a *.o

main.o: main.c
	gcc -c main.c

libcomputer.a: My_Simple_Computer.o
	ar cr libcomputer.a My_Simple_Computer.o

libterm.a: My_Term.o
	ar cr libterm.a My_Term.o

libchars.a: My_Big_Chars.o
	ar cr libchars.a My_Big_Chars.o

libkey.a: My_Read_Key.o
	ar cr libkey.a My_Read_Key.o

libcpu.a: cpu.o
	ar cr libcpu.a cpu.o

My_Simple_Computer.o: My_Simple_Computer.c
	gcc -c My_Simple_Computer.c

My_Term.o: My_Term.c
	gcc -c My_Term.c

My_Big_Chars.o: My_Big_Chars.c
	gcc -c My_Big_Chars.c

My_Read_Key.o: My_Read_Key.c
	gcc -c My_Read_Key.c

cpu.o: cpu.c
	gcc -c cpu.c

clean:
	rm computer

run:
	./computer
```

5. Результат работы



Программа нахождения факториала числа.



Результат нахождения факториала числа 4.

6. Заключение

В результате выполнения курсового проекта была разработана модель вычислительной машины Simple Computer.

В ходе разработки были изучены устройство и принцип работы функциональных блоков простейшего компьютера, такие как:

— Оперативная память
— Внешние устройства
— Центральный процессор

Благодаря данному курсы был освоен принцип обработки программных прерываний в UNIX-подобных операционных системах, использование ESC-последовательностей, также были изучены особенности ввода данных с клавиатуры в различных режимах терминала. Самое главное было освоено представление о работе электронно-вычислительных машин.