Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет

телекоммуникаций и информатики»

кафедра ВС

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине «Архитектура ЭВМ»

Тема: «Разработка библиотеки *mySimpleComputer*»

Выполнил: студент группы ИВ-121

Бессонов Алексей Евгеньевич

Проверил: Профессор кафедры ВС

Мамойленко С.Н.

Новосибирск – 2023

**СОДЕРЖАНИЕ**

[**Постановка задачи** 3](#_Toc39155)

[**Реализация функций** 4](#_Toc39156)

[**Тестирование библиотеки функций** 9](#_Toc39157)

[**Список литературы** 11](#_Toc39158)

[**Приложение** 12](#_Toc39159)

[**Код программы** 12](#_Toc39160)

# ПОСТАНОВКА ЗАДАЧИ

Необходимо выполнить программную реализацию основных операций работы простейшего вычислительного устройства на языке программирования Си. Операции включают в себя: кодирование, декодирование команд, управление регистрами, взаимодействие с оперативной памятью.

Список команд и их описание представлены ниже:

* int sc\_memoryInit ()

Инициализирует оперативную память, задавая всем её ячейкам нулевые значения.

* int sc\_memorySet (int address, int value)

Задаёт значение указанной ячейки памяти как *value.*

* int sc\_memoryGet (int address, int \* value)

Возвращает значение из указанной ячейки памяти.

* int sc\_memorySave (char \* filename)

Сохраняет содержимое памяти в файл в бинарном виде.

* int sc\_memoryLoad (char \* filename)

Загружает из указанного файла содержимое оперативной памяти.

* int sc\_regInit (void)

Инициализирует регистр флагов нулевым значением.

* int sc\_regSet (int register, int value)

Устанавливает значение указанного регистра флагов.

* int sc\_regGet (int register, int \* value)

Возвращает значение указанного регистра флагов.

* int sc\_commandEncode (int command, int operand, int \* value) Кодирует команду с указанным номером и операндом в *value.*
* int sc\_commandDecode (int value, int \* command, int \* operand) Декодирует команду и помещает результат в *command* и *operand.*

# РЕАЛИЗАЦИЯ ФУНКЦИЙ

***int sc\_init()***

*Функция инициализирует оперативную память значениями 0. Если инициализация не прошла успешно – функция возвращает -1.*

int sc\_memoryInit()

{

    memory = calloc(MEMORY\_SIZE, sizeof(int));

    if (memory != NULL) {

        printf("SimpleComputer start!\n");

        return 0;

    } else {

        printf("Error\n");

        return -1;

    }

}

***int sc\_memorySet(int address, int value)***

*Функция устанавливает значение value по индексу address. Если установочный индекс выходит на пределы оперативной памяти, устанавливается флаг выхода за границу памяти (3) и возвращается -1.*

int sc\_memorySet(int address, int value)

{

    if (address <= 0 || address < 99) {

        memory[address] = value;

        // printf("%d\n", memory[address]);

    }

    else {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        printf("Error: Out of memory bounds\n");

        return -1;

    }

    return 0;

}

***int sc\_memoryGet(int address, int\* value)***

*Функция присваивает указателю value значение, хранимое в оперативной памяти по адресу address. Если адрес выходит за области оперативной памяти или таковая не была инициализированая, то функция возвращает -1.*

int sc\_memoryGet(int address, int\* value)

{

    if (address < 0 || address > 99 || value == NULL) {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        printf("Error: Out of memory bounds\n");

        return -1;

    }

    \*value = memory[address];

    return 0;

}

***int sc\_memorySave(char\* filename)***

*Функция записывает содержимое оперативной памяти в файл с именем filename в бинарном виде. Если файл с указанным именем не существует, либо, по какой-либо причине не удалось записать данное количество элементов, возвращается -1.*

int sc\_memorySave(char\* filename)

{

    int memoryMEMORY\_SIZE = sizeof(memory); // размер данных в байтах

    FILE\* fp = fopen(filename, "wb");

    if (fp == NULL) {

        printf("Не удалось открыть файл для записи\n");

        return -1;

    }

    if (fwrite(memory, memoryMEMORY\_SIZE, 1, fp) != 1) {

        printf("Не удалось записать данные в файл\n");

        return -1;

    }

    fclose(fp);

    printf("Данные успешно записаны в файл\n");

    return 0;

}

***int sc\_memoryLoad(char\* filename)***

*Функция загружает содержимое файла с именем filename в оперативную память. Если такой файл не существует, либо не удалось записать указанное количество элементов, функция возвращает -1.*

int sc\_memoryLoad(char\* filename)

{

    FILE\* fp = fopen(filename, "rb");

    if (fp == NULL) {

        printf("Не удалось открыть файл для чтения\n");

        return -1;

    }

    size\_t n = fread(memory, sizeof(int), MEMORY\_SIZE, fp);

    if (n == 0) {

        printf("Не удалось прочитать данные из файла\n");

        return -1;

    }

    fclose(fp);

    printf("Данные успешно загружены из файла\n");

    return 0;

}

***void sc\_regInit()***

*Инициализирует регистр.*

int sc\_regInit()

{

    flags = 0;

    return 0;

}

***int sc\_regSet(int reg, int value)***

*Функция устанавливает значение флага register в значение value, при этом, значение value может быть только 0 или 1. В противном случае, функция возвращает -1.*

int sc\_regSet(int reg, int value)

{

    if (reg < 0 || reg > 4) {

        return -1;

    }

    if (value != 0 && value != 1) {

        return -1;

    }

    if (value == 1) {

        switch (reg) {

        case 0:

            printf("Cannot set flag\n");

            break;

        case 1:

            printf("Overflow flag is set\n");

            break;

        case 2:

            printf("Division by zero flag is set\n");

            break;

        case 3:

            printf("Memory access error flag is set\n");

            break;

        case 4:

            printf("Wrong command flag is set\n");

            break;

        }

    }

    (value == 1) ? (flags |= (1 << (reg - 1))) : (flags &= (~(1 << (reg - 1))));

    return 0;

}

***int sc\_regGet(int reg, int\* value)***

*Функция присваивает указателю value значение, хранимое в регистре register. Если регистр превышает максимальное их количество (5) – функция возвращает -1.*

int sc\_regGet(int reg, int\* value)

{

    if (reg < 0 || reg > 4

        || value == NULL) { // проверка допустимости номера регистра

        return -1; // возврат кода ошибки

    }

    \*value = (flags >> (reg - 1)) & 0x1; // получение значения флага

    return 0; // успешное завершение функции

}

***int sc\_commandEncode(int command, int operand, int\* value)***

*Функция кодирует команду по шаблону. Если значение value неинициализировано, либо команда command является некорректной, или операнд operand превышает максимально допустимое значение, функция возвращает -1.*

int sc\_commandEncode(int command, int operand, int\* value)

{

    if (bsearch(&command,

                commands,

                sizeof(commands) / sizeof(commands[0]),

                sizeof(commands[0]),

                comp)

                == NULL

        || command < 0 || command > 127 || operand < 0 || operand > 127) {

        return -1; // неправильное значение для команды

    }

    \*value = 0;

    \*value = (command << COMMAND\_SHIFT) | operand;

    return 0;

}

***int sc\_commandDecode(int value, int\* command, int\* operand)***

*Функция декодирует значение value по шаблону. Декодированные значения помещаются в command и operand соответственно.*

int sc\_commandDecode(int value, int\* command, int\* operand)

{

    if (command == NULL || operand == NULL) {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        return -1;

    }

    \*operand = \*command = 0;

    \*operand = value & MASK\_COMMAND;

    value >>= COMMAND\_SHIFT;

    \*command = value;

    return 0;

}

Функция comp сравнивает два симовола, если первый равен второму возвращается 0 если меньше или больше -1.

int comp(const void\* n1, const void\* n2)

{

    return (\*(int\*)n1 - \*(int\*)n2);

}

# ТЕСТИРОВАНИЕ БИБЛИОТЕКИ ФУНКЦИЙ

Для тестирования корректности работы библиотеки разработанных функций, используется библиотека *ctest.h.* При реализации тестирования проверялась корректность работы библиотеки при стандартных, краевых и ошибочных случаев. Исходный код тестов представлен ниже.

#include <libcomputer/comp.h>

#include <ctest.h>

#include <stdlib.h>

#include <string.h>

CTEST(init\_memory, successfully\_init\_memory)

{

    ASSERT\_EQUAL(sc\_memoryInit(), 0);

}

CTEST(memory\_seting\_getting\_tests, set\_get)

{

    for (int x = 25; x != 75; ++x) {

        sc\_memorySet(x, x);

    }

    int\* accum = malloc(sizeof(int));

    for (int x = 0; x != 100; ++x) {

        int actual = sc\_memoryGet(x, accum);

        if (x > 24 && x < 75) {

            ASSERT\_EQUAL(\*accum, x);

        } else {

            ASSERT\_EQUAL(\*accum, 0);

        }

        ASSERT\_EQUAL(actual, 0);

    }

    free(accum);

}

CTEST(memory\_save\_load\_tests, binary\_saving\_loading)

{

    char\* mem\_path1 = "test/test\_memory\_snapshots/mem1";

    char\* mem\_path2 = "test/test\_memory\_snapshots/mem2";

    for (int x = 0; x != 100; ++x) {

        sc\_memorySet(x, 0);

    }

    sc\_memorySave(mem\_path2);

    sc\_memorySet(10, 1);

    sc\_memorySave(mem\_path1);

    sc\_memoryInit();

    sc\_memoryLoad(mem\_path1);

    int\* accum = malloc(sizeof(int));

    for (int x = 0; x != 100; ++x) {

        sc\_memoryGet(x, accum);

        if (x == 10) {

            ASSERT\_EQUAL(\*accum, 1);

        } else {

            ASSERT\_EQUAL(\*accum, 0);

        }

    }

    sc\_memoryLoad(mem\_path2);

    for (int x = 0; x != 100; ++x) {

        sc\_memoryGet(x, accum);

        ASSERT\_EQUAL(\*accum, 0);

    }

    free(accum);

}

CTEST(init\_reg, successfully\_init\_reg)

{

    ASSERT\_EQUAL(sc\_regInit(), 0);

}

CTEST(memory\_register\_tests, register\_seting\_getting)

{

    for (int x = 0; x != 5; x++) {//для выявления ошибок

        sc\_regSet(x, 1);

    }

    int\* actual = malloc(sizeof(int));

    for (int x = 0; x != 5; x++) {

        sc\_regGet(x + 1, actual);

        ASSERT\_EQUAL(1, \*actual);

    }

    for (int x = -1; x > -5; --x) {

        ASSERT\_EQUAL(-1, sc\_regGet(x, actual));

    }

    sc\_regInit();

    for (int x = 0; x > -5; --x) {

        sc\_regGet(x + 1, actual);

        ASSERT\_EQUAL(\*actual, 0);

    }

}

CTEST(memory\_commands, commands\_encoding\_decoding)

{

    int\* encode\_value = malloc(sizeof(int));

    sc\_commandEncode(0x33, 0x59, encode\_value);

    ASSERT\_EQUAL(6617, \*encode\_value);

    sc\_commandEncode(0xa, 0xa, encode\_value);

    ASSERT\_EQUAL(1290, \*encode\_value);

    sc\_commandEncode(0x3c, 0x55, encode\_value);

    ASSERT\_EQUAL(7765, \*encode\_value);

    free(encode\_value);

}

# СПИСОК ЛИТЕРАТУРЫ

Мамойленко С.Н. Молдованова О.В, ЭВМ и ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА Учебное пособие. - Новосибирск: СибГУТИ, 2012. - 106 с.

Синтаксис языка С // Электр. ресурс Режим доступа: https://devdocs.io/c/ (дата обращения 02.02.2023)

**ЛИСТИНГ ПРОГРАММЫ**

**Main.c**

#include <computer/computer.h>

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <libcomputer/comp.h>

int main(void)

{

    // int \*value = malloc(sizeof(int));

    // sc\_commandEncode(0x33, 0x59, value);

    // int \*command = malloc(sizeof(int));

    // int \*operand = malloc(sizeof(int));

    // sc\_commandDecode(\*value, command, operand);

    sc\_memoryInit();

    // sc\_memorySet(1, 333);

    // sc\_memorySave("2.txt");

    // sc\_memoryLoad("2.txt");

    // sc\_regInit();

    // sc\_regSet(1, 1);  // Set overflow flag

    // sc\_regSet(5, 1);  // Invalid register number

    // sc\_regSet(0, 1);  // Cannot set flag

    // sc\_regSet(1, 2);  // Invalid value

    return 0;

}

#ifndef SIMPLECOMPUTER\_H

#define SIMPLECOMPUTER\_H

#define MEMORY\_SIZE 100

#define REG\_SIZE 32 // размер регистра флагов

#define FLAG\_OVERFLOW 0x1 // маска флага переполнения

#define FLAG\_DIVISION\_BY\_ZERO 0x2 // маска флага нуля

#define FLAG\_MEMORY\_ACCESS\_ERROR 0x3 // маска флага выхода за границы памяти

#define FLAG\_WRONG\_COMMAND 0x8 //Wrong command flag is set

#define MASK\_COMMAND 0x7F00

#define MASK\_OPERAND 0x00FF

#define COMMAND\_SHIFT 7

// #define ERROR\_OUT\_OF\_MEMORY 1

// #define ERROR\_OUT\_OF\_BOUNDS 2

// #define ERROR\_INVALID\_REGISTER 3

// #define ERROR\_INVALID\_VALUE 4

// #define ERROR\_INVALID\_COMMAND 5

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);

int comp(const void \*n1, const void \*n2);

#endif

**Computer.c**

#include "comp.h"

#include <search.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int reg\_flags[REG\_SIZE]; // массив для хранения значений флагов

int col = 0;

static int\* memory = NULL;

int reg\_flags[REG\_SIZE];

int ar[MEMORY\_SIZE];

int flags = 0;

int commands[] = {

        10, 11, 20, 21, 30, 31, 32, 33, 40, 41, 42, 43, 51,

        52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,

        65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,

};

// инициализирует оперативную память Simple

// Computer, задавая всем её ячейкам нулевые значения. В качестве

// «оперативной памяти» используется массив целых чисел, определенный

// статически в рамках библиотеки. Размер массива равен 100 элементам.

int sc\_memoryInit()

{

    memory = calloc(MEMORY\_SIZE, sizeof(int));

    if (memory != NULL) {

        printf("SimpleComputer start!\n");

        return 0;

    } else {

        printf("Error\n");

        return -1;

    }

}

// задает значение

// указанной ячейки памяти как value. Если адрес выходит за допустимые

// границы, то устанавливается флаг «выход за границы памяти» и работа

// функции прекращается с ошибкой;

int sc\_memorySet(int address, int value)

{

    if (address <= 0 || address < 99) {

        memory[address] = value;

        // printf("%d\n", memory[address]);

    }

    else {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        printf("Error: Out of memory bounds\n");

        return -1;

    }

    return 0;

}

// возвращает значение

// указанной ячейки памяти в value. Если адрес выходит за допустимые

// границы, то устанавливается флаг «выход за границы памяти» и работа

// функции прекращается с ошибкой. Значение value в этом случае не

// изменяется.

int sc\_memoryGet(int address, int\* value)

{

    if (address < 0 || address > 99 || value == NULL) {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        printf("Error: Out of memory bounds\n");

        return -1;

    }

    \*value = memory[address];

    return 0;

}

// – сохраняет содержимое памяти

// в файл в бинарном виде (используя функцию write или fwrite);

int sc\_memorySave(char\* filename)

{

    int memoryMEMORY\_SIZE = sizeof(memory); // размер данных в байтах

    FILE\* fp = fopen(filename, "wb");

    if (fp == NULL) {

        printf("Не удалось открыть файл для записи\n");

        return -1;

    }

    if (fwrite(memory, memoryMEMORY\_SIZE, 1, fp) != 1) {

        printf("Не удалось записать данные в файл\n");

        return -1;

    }

    fclose(fp);

    printf("Данные успешно записаны в файл\n");

    return 0;

}

// загружает из указанного файла

// содержимое оперативной памяти (используя функцию read или fread);

int sc\_memoryLoad(char\* filename)

{

    FILE\* fp = fopen(filename, "rb");

    if (fp == NULL) {

        printf("Не удалось открыть файл для чтения\n");

        return -1;

    }

    size\_t n = fread(memory, sizeof(int), MEMORY\_SIZE, fp);

    if (n == 0) {

        printf("Не удалось прочитать данные из файла\n");

        return -1;

    }

    fclose(fp);

    printf("Данные успешно загружены из файла\n");

    return 0;

}

// инициализирует регистр флагов нулевым

// значением;

int sc\_regInit()

{

    flags = 0;

    return 0;

}

// устанавливает значение

// указанного регистра флагов. Для номеров регистров флагов должны

// использоваться маски, задаваемые макросами (#define). Если указан

// недопустимый номер регистра или некорректное значение, то функция

// завершается с ошибкой.

int sc\_regSet(int reg, int value)

{

    if (reg < 0 || reg > 4) {

        return -1;

    }

    if (value != 0 && value != 1) {

        return -1;

    }

    if (value == 1) {

        switch (reg) {

        case 0:

            printf("Cannot set flag\n");

            break;

        case 1:

            printf("Overflow flag is set\n");

            break;

        case 2:

            printf("Division by zero flag is set\n");

            break;

        case 3:

            printf("Memory access error flag is set\n");

            break;

        case 4:

            printf("Wrong command flag is set\n");

            break;

        }

    }

    (value == 1) ? (flags |= (1 << (reg - 1))) : (flags &= (~(1 << (reg - 1))));

    return 0;

}

// – возвращает значение

// указанного флага. Если указан недопустимый номер регистра, то функция

// завершается с ошибкой.

// int reg\_flags = 0;  // регистр флагов

int sc\_regGet(int reg, int\* value)

{

    if (reg < 0 || reg > 4

        || value == NULL) { // проверка допустимости номера регистра

        return -1; // возврат кода ошибки

    }

    \*value = (flags >> (reg - 1)) & 0x1; // получение значения флага

    return 0; // успешное завершение функции

}

// кодирует команду с указанным номером и операндом и помещает результат

// в value. Если указаны неправильные значения для команды или операнда, то

// функция завершается с ошибкой. В этом случае значение value не изменяется.

int comp(const void\* n1, const void\* n2)

{

    return (\*(int\*)n1 - \*(int\*)n2);

}

int sc\_commandEncode(int command, int operand, int\* value)

{

    if (bsearch(&command,

                commands,

                sizeof(commands) / sizeof(commands[0]),

                sizeof(commands[0]),

                comp)

                == NULL

        || command < 0 || command > 127 || operand < 0 || operand > 127) {

        return -1; // неправильное значение для команды

    }

    \*value = 0;

    \*value = (command << COMMAND\_SHIFT) | operand;

    return 0;

}

// декодирует значение как команду Simple Computer. Если декодирование

// невозможно, то устанавливается флаг «ошибочная команда» и функция

// завершается с ошибкой.

int sc\_commandDecode(int value, int\* command, int\* operand)

{

    if (command == NULL || operand == NULL) {

        sc\_regSet(FLAG\_MEMORY\_ACCESS\_ERROR, 1);

        return -1;

    }

    \*operand = \*command = 0;

    \*operand = value & MASK\_COMMAND;

    value >>= COMMAND\_SHIFT;

    \*command = value;

    return 0;

}

// int main(void) {

//     sc\_memoryInit();

//     sc\_memorySet(1, 333);

//     sc\_memoryGet(10, &memory[3]);

//     sc\_memorySave("2.txt");

//     sc\_memoryLoad("2.txt");

//     sc\_regInit();

//     sc\_regSet(1, 1);  // Set overflow flag

//     //sc\_regSet(5, 1);  // Invalid register number

//     sc\_regSet(0, 1);  // Cannot set flag

//     //sc\_regSet(1, 2);  // Invalid value

//     return 0;

// }