## 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 Воротников Ярослав Денисович Проверил: Профессор кафедры ВС

Мамойленко С.Н.

Новосибирск – 2023

# СОДЕРЖАНИЕ

[Постановка задачи 3](#_TOC_250003)

[Реализация функций 4](#_TOC_250002)

[Тестирование библиотеки функций 9](#_TOC_250001)

[Список литературы 11](#_TOC_250000)

Приложение 12

Код программы 12

# 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\_init()

{

memory = calloc(RAM, sizeof(int)); sc\_regInit();

if (memory != NULL) {

printf("СуперКомпьютер запущен. Добро пожаловать!\n"); return 0;

} else {

printf("Недостаточно ОЗУ для запуска."); 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) { sc\_regSet(3, 1);

return -1;

}

memory[address] = value; 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 (value == NULL || address < 0 || address > 99) { sc\_regSet(3, 1);

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)

{

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

if (output\_file == NULL) { fclose(output\_file); return -1;

}

if (fwrite(memory, sizeof(int), 100, output\_file) != 100) { return -1;

}

fclose(output\_file); 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\* input\_file = fopen(filename, "rb");

if (input\_file == NULL) { fclose(input\_file); return -1;

}

if (fread(memory, sizeof(int), 100, input\_file) != 100) { return -1;

}

fclose(input\_file); return 0;

}

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

### Инициализирует регистр.

void sc\_regInit()

{

registr = 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) //reg - номер разряда

{

if (reg < 1 || reg > 5) return -1;

if (value != 0 && value != 1) return -1;

if (value == 1)

registr |= (1 << (reg - 1)); // задвигаем единичку на нужную позицию и записываем в регистр

else

registr &= (~(1 << (reg - 1))); // задвигаем единичку на нужную позицию и записываем в регистр

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) //reg - номер разряда

{

if (reg < 1 || reg > 5) { return -1;

}

if (value == NULL) return -1;

\*value = (registr >> (reg - 1)) & BIT;

//printf("%d", registr >> (reg - 1));

return 0;

}

***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)

{

int length = sizeof(commandSet) / sizeof(commandSet[0]);

int\* found = bsearch(&command, commandSet, length, sizeof(int), compare); if (found == NULL)

return -1;

if (value == NULL || operand > MASK || command > MASK) return -1;

\*value = NULLBIT;

\*value = command << 7;

\*value |= 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(5, 1);

return -1;

}

\*operand = \*command = NULLBIT;

\*operand = value & MASK; value >>= 7;

\*command = value; return 0;

}

# 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 "../thirdparty/ctest.h" #include <computer/computer.h> #define SUCCESS 0

#define FAIL -1

CTEST(memory, init)

{

int expCode = SUCCESS; int curCode = sc\_init();

ASSERT\_EQUAL(expCode, curCode);

}

CTEST(memory, get)

{

int val = 24;

int expCode = SUCCESS;

int curCode = sc\_memoryGet(50, &val); ASSERT\_EQUAL(expCode, curCode);

expCode = FAIL;

curCode = sc\_memoryGet(110, &val); // попыткаобращения ASSERT\_EQUAL(expCode, curCode);

}

CTEST(reg, get)

{

int val = 1;

int expCode = SUCCESS;

int curCode = sc\_regGet(1, &val); ASSERT\_EQUAL(expCode, curCode);

expCode = FAIL;

curCode = sc\_regGet(1, NULL); // защита "от дурака". ASSERT\_EQUAL(expCode, curCode);

val = 0; expCode = FAIL;

curCode = sc\_regGet(7, &val); // диапазон разрядов от 0 до 5.

ASSERT\_EQUAL(expCode, curCode);

}

CTEST(command, encode)

{

int command = 41, operand = 51; int value = 0;

int expValue = 5299;

int expCode = SUCCESS;

int curCode = sc\_commandEncode(command, operand, &value); ASSERT\_EQUAL(expCode, curCode);

ASSERT\_EQUAL(expValue, value); // закодированная команда.

command = 27; expCode = FAIL;

curCode = sc\_commandEncode(command, operand, &value); // не существует такой команды.

ASSERT\_EQUAL(expCode, curCode);

command = 20;

operand = 130; expCode = FAIL;

curCode = sc\_commandEncode(command, operand, &value); // операнд превышает 0x7f.

ASSERT\_EQUAL(expCode, curCode);

}

CTEST(command, decode)

{

int expCommand = 41, expOperand = 51; int command = 0, operand = 0;

int value = 5299;

int expCode = SUCCESS;

int curCode = sc\_commandDecode(value, &command, &operand); ASSERT\_EQUAL(expCode, curCode);

ASSERT\_EQUAL(expCommand, command); // раскодированная команда. ASSERT\_EQUAL(expOperand, operand); // полученный операнд.

command = 27; expCode = FAIL;

curCode = sc\_commandDecode(value, NULL, &operand); // защита от "дурака". ASSERT\_EQUAL(expCode, curCode);

}

# 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)

**Main.c**

# ЛИСТИНГ ПРОГРАММЫ

#include <computer/computer.h> #include <stdio.h>

int main(void)

{

int command = 41, operand = 127; int value = 0;

sc\_init();

//sc\_memoryRand(); sc\_memoryLoad("load.bin");

// sc\_memorySave("save.bin");

sc\_outputMemory(); sc\_commandEncode(command, operand, &value); printf("Закодировано: %d\n", value); sc\_commandDecode(value, &command, &operand);

printf("Декодирована команда: %d, операнд: %d\n", command, operand);

}

#### Computer.h

#pragma once int sc\_init();

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

int sc\_regSet(int reg, int value); void sc\_regInit();

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

int sc\_memoryGet(int address, int\* value); int sc\_memorySet(int address, int value); void sc\_memoryRand();

void sc\_outputMemory();

int sc\_commandEncode(int command, int operand, int\* value); int sc\_commandDecode(int value, int\* command, int\* operand); int compare(const void\* n1, const void\* n2);

#### Computer.c

#include "computer.h"

#include <inttypes.h> #include <stdio.h> #include <stdlib.h> #define RAM 100 #define NULLBIT 0x0 #define BIT 0x1 #define MASK 0x7f

static int\* memory = NULL;

static int registr; // переменная, хранящая флаги

static int commandSet[] = { 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,

};

int sc\_init()

{

memory = calloc(RAM, sizeof(int)); sc\_regInit();

if (memory != NULL) {

printf("СуперКомпьютер запущен. Добро пожаловать!\n"); return 0;

} else {

printf("Недостаточно ОЗУ для запуска."); return -1;

}

}

int sc\_memorySave(char\* filename)

{

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

if (output\_file == NULL) { fclose(output\_file); return -1;

}

if (fwrite(memory, sizeof(int), 100, output\_file) != 100) { return -1;

}

fclose(output\_file);

return 0;

}

int sc\_memoryLoad(char\* filename)

{

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

if (input\_file == NULL) { fclose(input\_file); return -1;

}

if (fread(memory, sizeof(int), 100, input\_file) != 100) { return -1;

}

fclose(input\_file);

return 0;

}

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

{

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

sc\_regSet(3, 1);

return -1;

}

\*value = memory[address]; return 0;

}

int sc\_memorySet(int address, int value) //reg - номер разряда

{

if (address < 0 || address > 99) { sc\_regSet(3, 1);

return -1;

}

memory[address] = value; return 0;

}

int sc\_regGet(int reg, int\* value) //reg - номер разряда

{

if (reg < 1 || reg > 5) { return -1;

}

if (value == NULL) return -1;

\*value = (registr >> (reg - 1)) & BIT;

//printf("%d", registr >> (reg - 1)); return 0;

}

int sc\_regSet(int reg, int value) //reg - номер разряда

{

if (reg < 1 || reg > 5) return -1;

if (value != 0 && value != 1) return -1;

if (value == 1)

registr |= (1 << (reg - 1)); // задвигаем единичку на нужную позицию и записываем в регистр

else

registr &= (~(1 << (reg - 1))); // задвигаем единичку на нужную позицию и записываем в регистр

return 0;

}

void sc\_regInit()

{

registr = 0;

}

void sc\_outputMemory()

{

for (u\_int8\_t i = 0; i != 100; ++i) { if (i % 10 == 0) {

printf("\n");

}

printf("%3d ", memory[i]);

}

printf("\n");

}

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

{

int length = sizeof(commandSet) / sizeof(commandSet[0]);

int\* found = bsearch(&command, commandSet, length, sizeof(int), compare); if (found == NULL)

return -1;

if (value == NULL || operand > MASK || command > MASK) return -1;

|  |  |  |
| --- | --- | --- |
|  | \*value | = NULLBIT; |
|  | \*value  \*value return | = command << 7;  |= operand; 0; |

}

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

{

if (command == NULL || operand == NULL) { sc\_regSet(5, 1);

return -1;

}

\*operand = \*command = NULLBIT;

\*operand = value & MASK; value >>= 7;

\*command = value; return 0;

}

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

{

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

}

void sc\_memoryRand()

{

for (uint8\_t i = 0; i < 50; i++)

sc\_memorySet(rand() % 100, rand() % 40);

}