Федеральное агентство связи Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Факультет: Информатики и вычислительной техники

Кафедра телекоммуникационных сетей и вычислительных средств

Дисциплина: Архитектура ЭВМ

Отчёт по лабораторной работе №2

«Разработка библиотеки mySimpleComputer. Оперативная память, регистр флагов, декодирование операций.»

Выполнили студенты группы ИА-831:

Зарубин Максим Евгеньевич

Дорощук Никита Андреевич

Проверил преподаватель:

Токмашева Елизавета Ивановна

Новосибирск

2020

Задание к лабораторной работе.

1. Прочитайте главу 4 практикума по курсу «Организация ЭВМ и систем». Изучите принципы работы разрядных операций в языке Си: как можно изменить значение указанного разряда целой переменной или получить его значение. Вспомните, как сохранять информацию в файл и считывать еѐ оттуда в бинарном виде.

2. Разработайте функции по взаимодействию с оперативной памятью, управлению регистром флагов и кодированию/декодированию команд:

a. int sc_memoryInit () – инициализирует оперативную память Simple Computer, задавая всем еѐ ячейкам нулевые значения. В качестве «оперативной памяти» используется массив целых чисел, определенный статически в рамках библиотеки. Размер массива равен 100 элементам.

b. int sc_memorySet (int address, int value) – 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;

c. int sc_memoryGet (int address, int * value) – 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.

d. int sc_memorySave (char * filename) – сохраняет содержимое памяти в файл в бинарном виде (используя функцию write или fwrite);

e. int sc_memoryLoad (char * filename) – загружает из указанного файла содержимое оперативной памяти (используя функцию read или fread);

f. int sc_regInit (void) – инициализирует регистр флагов нулевым значением;

g. int sc_regSet (int register, int value) – 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.

h. int sc_regGet (int register, int * value) – возвращает значение указанного флага. Если указан недопустимый номер регистра, то функция завершается с ошибкой.

i. int sc_commandEncode (int command, int operand, int * value) – 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.

j. int sc_commandDecode (int value, int * command, int * operand) – декодирует значение как команду Simple Computer. Если декодирование невозможно, то устанавливается флаг «ошибочная команда» и функция завершается с ошибкой.

3. Оформите разработанные функции как статическую библиотеку. Подготовьте заголовочный файл для неѐ.

Описание реализованных функций.

1. int sc_memoryInit () – Инициализируем 100 элементов массива нулями.

2. int sc_memorySet (int address, int value) - Проверяем выход за границы, если адрес не выходит за границы, то присваиваем ячейке массива с индексом адреса значение, переданное в функцию.

3. int sc_memoryGet (int address, int * value) - Тоже сначала проверяем выход за границы, если адрес не выходит за границы, то возвращаем значение из ячейки массива с индексом адреса в value.

4. int sc_memorySave (char * filename) - Открываем файл и через функцию fwrite записываем значения массива в него.

5. int sc_memoryLoad (char * filename) - Открываем файл и, если он не пустой, через функцию fread записываем из него данные в массив.

6. int sc_regInit (void) - Инициализируем флаги нулями.

7. int sc_regSet (int register, int value) - Проверяем номер регистра и переданное значение на допустимость, если всё нормально, то устанавливаем значение указанного регистра флагов.

8. int sc_regGet (int register, int * value) - Проверяем номер регистра на допустимость, если всё нормально, то возвращаем значение указанного указанного флага в value.

9. int sc_commandEncode (int command, int operand, int * value) - Проходим циклом через все элементы массива, если команда совпадает со значением, то кодируем команду с указанным номером и операндом и помещаем результат в value, завершая функцию, иначе, если не было совпадений, то завершаем функцию с ошибкой.

10. int sc_commandDecode (int value, int * command, int * operand) - проверяем value, command и operand на возможность

декодирования, если оно невозможно, то устанавливаем флаг и завершаем функцию с ошибкой, иначе декодируем значение.

скриншоты проверки работы функций.

Результат с выходом за пределы массива во 2 и 3 функции

```
sc_memoryInit:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_memorySet(101,1):
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_memoryGet(101):
0
sc_memoryLoad:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_regInit:
0
sc_regSet(7,2):
0
sc_regGet:
-858993460
sc_commandEncode(0x10, 0x90, &value_3):
-858993460
100110011001100
sc_commandDecode:
-1
-858993460
-858993460
```

Результат работы с корректными входными данными

```
sc_memoryInit:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_memorySet(101,1):
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_memoryGet(101):
0
sc_memoryLoad:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sc_regInit:
0
sc_regSet(7,2):
0
sc_regGet:
-858993460
sc_commandEncode(0x10, 0x90, &value_3):
-858993460
100110011001100
sc_commandDecode:
2
-858993460
-858993460
```

Листинг программы.

Header.h

```
#pragma once
#ifndef HEADER_H
#define HEADER_H
#define _CRT_SECURE_NO_WARNINGS

#define N_MEM 100
#define N_COM 38

#define OVERFLOW 1
#define WRONGCOMMAND 2
#define ZEROFLAG 3
#define IGNOREFLAG 4
#define ISRUN 5
#define LIMIT 6

#define ENCODE 127
#include <stdio.h>
#include <stdlib.h>



int sc_memoryInit();
int sc_memorySet(int, int);
int sc_memoryGet(int, int*);

int sc_memorySave(char*);
int sc_memoryLoad(char*);

int sc_regInit();
int sc_regSet(int, int);
int sc_regGet(int, int*);

int sc_commandEncode(int, int, int*);
int sc_commandDecode(int, int*, int*);

#endif
```

Header.c

```
#include "Header.h"
#include <stdio.h>
#include <stdlib.h>
int arrMemory[N_MEM];
int reg;
int arrCommand[N_COM];
int sc_memoryInit() {
	int i;
	for (i = 0; i < N_MEM; i++)
		arrMemory[i] = 0;
```

```
	int tempArr[] = { 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 };
	memcpy(arrCommand, tempArr, (sizeof(tempArr)));
	return 0;
}
int sc_memorySet(int address, int value) {
	if (address < 0 || address > N_MEM) {
		sc_regSet(OVERFLOW, 1);
		return -1;
	}
	arrMemory[address] = value;
	return 0;
}
int sc_memoryGet(int address, int* value) {
	if (address < 0 || address > N_MEM) {

		sc_regSet(OVERFLOW, 1);

		return -1;
	}
	*value = arrMemory[address];
	return 0;
}
int sc_memorySave(char* filename) {
	FILE* output;
	output = fopen(filename, "wb");
	fwrite(arrMemory, N_MEM, sizeof(int), output);
	fclose(output);
	return 0;
}
int sc_memoryLoad(char* filename) {
	FILE* input;
	input = fopen(filename, "rb");
	if (input != NULL) {
		fread(arrMemory, N_MEM, sizeof(int), input);
	}
	else {
		return -1;
	}
	fclose(input);
	return 0;
}
int sc_regInit() {
	reg = 0;
	return 0;
}
int sc_regSet(int digNum, int value) {
	if ((value > 1) || (value < 0) || (digNum < 1) || (digNum > 6))
		return -1;
	if (value == 1) {
		reg = reg | (1 << (digNum - 1));
	}
	else {
		reg = reg & (~(1 << (digNum - 1)));
	}
	return 0;
}
int sc_regGet(int digNum, int* value) {
	if ((digNum < 1) || (digNum > 6)) return -1;
	else {
		*value = (reg >> (digNum - 1)) & 0x1;
		return 0;
	}
}
```

```
int sc_commandEncode(int command, int operand, int* value) {
	int i;
	for (i = 0; i < N_COM; i++) {
		if (command == arrCommand[i]) {
			*value = (command << 8) | (operand & ENCODE);
			sc_regSet(WRONGCOMMAND, 0);
			return 0;
		}
	}
	sc_regSet(WRONGCOMMAND, 1);
	return -1;
}
const int correct_ops[] = { 0x10, 0x11, 0x20, 0x21, 0x30, 0x31, 0x32, 0x33, 0x40,
                                          0x41, 0x42, 0x43, 0x59 };
const int ops_num = 13;
int int_cmp(const void* a, const void* b)
{
	if (*(int*)a < *(int*)b)
		return -1;
	else
		if (*(int*)a > * (int*)b)
			return 1;
		else
			return 0;
}

int sc_commandDecode(int value, int* command, int* operand)
{
	void* correct_command;
	int attribute;
	int tmp_command, tmp_operand;

	attribute = (value >> 14) & 1;
	if (attribute == 0)
	{
		tmp_command = (value >> 7) & 0x7F;
		tmp_operand = value & 0x7F;
		correct_command = bsearch(&tmp_command, correct_ops, ops_num, sizeof(int),
int_cmp);
		if (correct_command != NULL)
		{
			*command = tmp_command;
			*operand = tmp_operand;
		}
		else
			return 1;
	}
	else
		return 2;
	return 0;
}
```

Main.c

```
include "Header.h"
#include <stdio.h>
#include <stdlib.h>
extern int arrMemory[N_MEM];
extern int reg;
int main() {
	int value =0;
```

```
    sc_memoryInit();
    for (int i = 0; i < N_MEM;i++) printf("%d", arrMemory[i]);
    printf("\n");

    sc_memorySet(101, 5);
    for (int i = 0; i < N_MEM;i++) printf("%d", arrMemory[i]);
    printf("\n");

    sc_memoryGet(101, &value);
    printf("%d\n", value);

    sc_memorySave("output.bin");

    sc_memoryLoad("input.bin");
    for (int i = 0; i < N_MEM;i++) printf("%d", arrMemory[i]);
    printf("\n");

    sc_regInit();
    printf("%d\n", reg);

    sc_regSet(4, 2);
    printf("%d\n", reg);

    int           value_2;
    sc_regGet(4, &value_2);
    printf("%d\n", value_2);

    int value_3;
    sc_commandEncode(0x20, 0x40, &value_3);
    printf("%d\n", value_3);
    for (int i = 14;i >= 0;--i) printf("%d", (value_3 >> i) & 1);
    printf("\n");

    int operand;
    int command;
    int f;
    f = sc_commandDecode(value_3, &command, &operand);
    printf("%d\n", f);
    printf("%d\n", operand);
    printf("%d\n", command);
    printf("\n");
    return 0;
}
```