МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ПРАКТИЧЕСКАЯ РАБОТА №1

по дисциплине «Архитектура ЭВМ»

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

Выполнил: Наумов Алексей ИС-142

Проверил: ассистент кафедры ВС Курзин А.С.

Новосибирск 2023

Цель работы:

Изучить принципы работы оперативной памяти. Познакомиться с разрядными операциями языка Си. Разработать библиотеку mySimpleComputer, включающую функции по декодированию команд, управлению регистрами и взаимодействию с оперативной памятью.

Задание на лабораторную работу:

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 её оттуда в бинарном виде.
2. Разработайте функции по взаимодействию с оперативной памятью, управлению регистром флагов и кодированию/декодированию команд:
    1. ***int sc_memoryInit ()*** – 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.
    2. ***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;
    3. ***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.
    4. ***int sc_memorySave (char * filename)*** – сохраняет содержимое памяти в файл в бинарном виде (используя функцию write или fwrite);
    5. ***int sc_memoryLoad (char * filename)*** – загружает из указанного файла содержимое оперативной памяти (используя функцию read или fread);
    6. ***int sc_regInit (void)*** – инициализирует регистр флагов нулевым значением;
    7. ***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.
    8. ***int sc_regGet (int register, int * value)*** – возвращает значение указанного флага. Если указан недопустимый номер регистра, то функция завершается с ошибкой.
    9. ***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.
    10. ***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.

Результат:

```
alexeynaumov@Lenovo-Legion-5-15ARH05H-267a6435:~/otchet/11$ bin/main
--------------------------------------------------------------------------------
Инициализация памяти
Запись в массив sc_memory[1] to 3
Запись в массив sc_memory[2] to 6
Запись в массив sc_memory[3] to 9
получение из массива sc_memory[1] = 3
получение из массива sc_memory[2] = 6
получение из массива sc_memory[3] = 9
--------------------------------------------------------------------------------
запись в файл memory -> memory.mem
Запись sc_memory[1] to 3
Запись sc_memory[2] to 6
Запись sc_memory[3] to 9
--------------------------------------------------------------------------------
Загружаем файл memory <- memory.mem
Получаем sc_memory[1] = 3
Получаем sc_memory[2] = 6
Получаем sc_memory[3] = 9
--------------------------------------------------------------------------------
инициализация register = 0
Получение FLAG_DIV_BY_ZERO = 1
Получение FLAG_OVERFLOW = 0
Получение FLAG_OVERFLOW = 1
Кодирование команды '33' операнда '11' = '4235'
Декодирование '4235' = команды '33' операнда'11'
Кодирование value = 4235
FLAG_WRONG_COMMAND = 1
--------------------------------------------------------------------------------
```

Листинг кода:

main.c

```
#include "prototype.h"
#include <stdio.h>

int
main ()
{
  printf ("_____"
          "____\n");
  sc_memoryInit ();
  printf ("Инициализация памяти\n");
  for (int i = 1; i < 4; i++)
    {
      int val = i * 3;
      sc_memorySet (i, val);
      printf ("Запись в массив sc_memory[%d] to %d\n", i, val);
    }
  for (int i = 1; i < 4; i++)
    {
      int val = 0;
      sc_memoryGet (i, &val);
      printf ("получение из массива sc_memory[%d] = %d\n", i, val);
    }
  printf ("_____"
          "____\n");
  char *memfile = "memory.mem";
  sc_memorySave (memfile);
  printf ("запись в файл memory -> %s\n", memfile);
  for (int i = 1; i < 4; i++)
    {
      int val = i * 3;
      sc_memorySet (i, val);
      printf ("Запись sc_memory[%d] to %d\n", i, val);
    }
  printf ("_____"
          "____\n");
  sc_memoryLoad (memfile);
  printf ("Загружаем файл memory <- %s\n", memfile);
  for (int i = 1; i < 4; i++)
    {
      int val = 0;
      sc_memoryGet (i, &val);
      printf ("Получаем sc_memory[%d] = %d\n", i, val);
    }
  printf ("_____"
          "____\n");
  sc_regInit ();
  printf ("инициализация register = 0\n");
  sc_regSet (FLAG_DIV_BY_ZERO, 1);
  int val = 0;
  sc_regGet (FLAG_DIV_BY_ZERO, &val);
  printf ("Получение FLAG_DIV_BY_ZERO = %d\n", val);
  sc_regGet (FLAG_OVERFLOW, &val);
  printf ("Получение FLAG_OVERFLOW = %d\n", val);
  sc_regSet (FLAG_DIV_BY_ZERO, 2); // пытаемся перегрузить бит)
  sc_regGet (FLAG_OVERFLOW, &val);
  printf ("Получение FLAG_OVERFLOW = %d\n", val);
  int command = 0x21; //команда
  int operand = 11;   //что записываешь в команду
```

```
  sc_commandEncode (command, operand, &val);
  printf ("Кодирование команды '%d' операнда '%d' = '%d' \n", command, operand,
          val);
  sc_commandDecode (val, &command, &operand);
  printf ("Декодирование '%d' = команды '%d' операнда'%d'\n", val, command,
          operand);
  sc_commandEncode (0x0, 129,
                    &val); // пытаемся закодировать неправильную команду
  printf ("Кодирование value = %d\n", val); // ничего не меняется
  sc_regGet (FLAG_WRONG_COMMAND, &val);
  printf ("FLAG_WRONG_COMMAND = %d\n", val);
  printf ("_____________________________________________________________________"
          "____\n");
  return 0;
}
```

Prototype.c

```c
#include "prototype.h"

int sc_memory[MEMSIZE];
int sc_register;

int
sc_memoryInit () //инициализирует массив из 100 элементов
{
  memset (sc_memory, 0, MEMSIZE * sizeof (sc_memory[0]));
  return 0;
}

int
sc_memorySet (int address,
              int value) // устанавливает значение блока памяти
{
  if (address < 0 || address >= MEMSIZE)
    {
      BIT_SET (sc_register, FLAG_WRONG_ADDRESS);
      return ERR_WRONG_ADDRESS;
    }
  sc_memory[address] = value;
  return 0;
}

int
sc_memoryGet (
    int address, // gets the value of [address] memory unit and
    int *value) //получает значение блока памяти и возвращет его в переменную
{
  if (address < 0 || address >= MEMSIZE)
    {
      BIT_SET (sc_register, FLAG_WRONG_ADDRESS);
      return ERR_WRONG_ADDRESS;
    }
  *value = sc_memory[address];
  return 0;
}

int
sc_memorySave (char *filename) //сохраняет память в бинарный файл
{
  FILE *f = fopen (filename, "wb");
  if (!f)
    {
      return 1;
    }
  fwrite (sc_memory, sizeof (int), sizeof (sc_memory) / sizeof (int), f);
  fclose (f);
  return 0;
}

int
sc_memoryLoad (char *filename) //загружает оперативную память из файла
{
  FILE *f = fopen (filename, "rb");
  if (!f)
    {
      return 1;
    }
```

```
  fread (sc_memory, sizeof (int), sizeof (sc_memory) / sizeof (int), f);
  fclose (f);
  return 0;
}

int
sc_regInit (void) //инициализирует регистр флагов с 0
{
  sc_register = 0;
  return 0;
}

int
sc_regSet (
    int reg,
    int value) //устанавливает значение регистра флага #define-s используются
               //для номера регистров, если неверный номер регистра то ошибка
{
  if (reg < 0 || reg > 5)
    {
      return ERR_WRONG_FLAG;
    }
  if (!value)
    {
      BIT_DEL (sc_register, reg);
      return 0;
    }
  if (value != 1)
    {
      BIT_SET (sc_register, FLAG_OVERFLOW);
      return ERR_WRONG_VALUE;
    }
  BIT_SET (sc_register, reg);
  return 0;
}

int
sc_regGet (
    int reg,
    int *value) //получает значение флага, если неверный регистр то ошибка
{
  if (reg < 0 || reg > 5)
    {
      return ERR_WRONG_FLAG;
    }
  *value = BIT_GET (sc_register, reg);
  return 0;
}

int
sc_commandEncode (
    int command, int operand,
    int *value) //кодирует команду с определенным номером и операндом
                // помещает результат в значение, если он неправильный
                // или операнд - ошибка, значение не меняется.
{
  // commands list: 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
  if (((command > 0x76) || (command < 0x10))
```

```
      || ((command > 0x11) & (command < 0x20))
      || ((command > 0x21) & (command > 0x30))
      || ((command > 0x33) & (command < 0x40))
      || ((command > 0x43) & (command < 0x51)))
    {
      sc_regSet (FLAG_WRONG_COMMAND, 1);
      return ERR_WRONG_COMMAND;
    }
  if (operand < 0 || operand > 127)
    {
      sc_regSet (FLAG_WRONG_OPERAND, 1);
      return ERR_WRONG_OPERAND;
    }
  int encoded = 0b0000000000000000 | command;
  encoded <<= 7;
  encoded |= operand;
  *value = encoded;
  return 0;
}

int
sc_commandDecode (int value, int *command,
                  int *operand) // декодирует значение как комнаду sc, если
                                // декодировавние невозможно устанавливает
                                // команду error и возвразает ошибку.
{
  if ((value & (1 << 14)) != 0)
    {
      sc_regSet (FLAG_WRONG_COMMAND, 1);
      return ERR_WRONG_COMMAND;
    }
  *command = (value >> 7);
  value -= (*command << 7);
  *operand = value;
  return 0;
}
```

Prototype.h

```
#pragma once

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

#define MEMSIZE 100

// флаги
#define FLAG_WRONG_COMMAND 5
#define FLAG_WRONG_OPERAND 4
#define FLAG_WRONG_ADDRESS 3
#define FLAG_DIV_BY_ZERO 2
#define FLAG_OVERFLOW 1

// ошибки
#define ERR_WRONG_ADDRESS -1
#define ERR_WRONG_FLAG -2
#define ERR_WRONG_VALUE -3
#define ERR_WRONG_COMMAND -4
#define ERR_WRONG_OPERAND -5

// битовые операции
#define BIT_SET(X, Y) X = X | (1 << (Y - 1))
#define BIT_DEL(X, Y) X = X & (~(1 << (Y - 1)))
#define BIT_GET(X, Y) X >> (Y - 1) & 0x1

int sc_memoryInit (); // инициализация массива из 100 элементов

int sc_memorySet (int address,
                  int value); // устанавливает значение блока памяти

int sc_memoryGet (int address, int *value); // получает значение блока памяти и
                                            // возвращает его в значение var

int sc_memorySave (char *filename); // сохраняет память в бинарный файл

int sc_memoryLoad (char *filename); // загружает оперативную память из файла

int sc_regInit (void); // инициализирует регистр флагов с нуля

int sc_regSet (
    int reg,
    int value); //устанаввливает флаг значение регистра #define-s используется
                // для номеров регистров если неверный номер - то ошибка.

int sc_regGet (
    int reg,
    int *value); //получает значение флага, если неправильный регистр то ошибка

int sc_commandEncode (
    int command, int operand,
    int *value); // кодирует команду с определенным номером и  операндом
                 // помещает результат в значение если он неправильная  команда
                 // или операнд - ошибка, значение не меняется

int sc_commandDecode (int value, int *command,
                      int *operand); // декодирует значение как команду sc если
                                     // декодировние невозможно  устанавливает
```

```
                                        // команду ошибки и возвращает ошибку.
```

Makefile

```
DIRGUARD = @mkdir -p $(@D)

all: bin/main
.PHONY: bin/main
bin/main: *.c
    $(DIRGUARD)
gcc -Wall -Wextra -o $@ $^
```