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 №2

по дисциплине «Архитектура ЭВМ»

Тема: «Практическое задание № 2. Консоль управления моделью Simple Computer. Текстовая часть.»

Выполнил: Наумов Алексей ИС-142

Проверил: ассистент кафедры ВС Курзин А.С.

Новосибирск 2023

Цель работы:

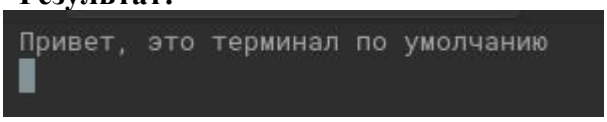
Изучить принципы работы терминалов ЭВМ в текстовом режиме. Понять, каким образом кодируется текстовая информация и как с помощью неё можно управлять работой терминалов. Разработать библиотеку функций myTerm, включающую базовые функции по управлению текстовым терминалом (очистка экрана, позиционирование курсора, управления цветом). Начать разрабатывать консоль управления Simple Computer (вывести на экран текстовую часть).

Задание на лабораторную работу:

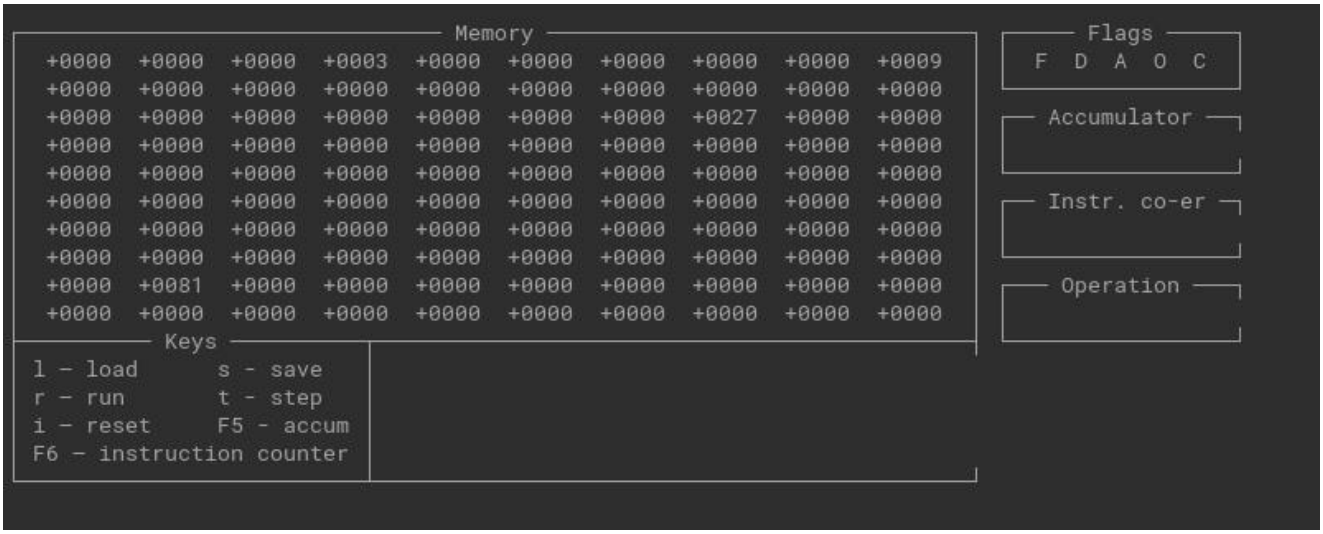
1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграфы 5.4 и 5.5. Изучите страницу man для команды infocmp, базы terminfo, функции ioctl.
2. Откройте текстовый терминал и запустите оболочку bash (оболочка запускается автоматически). Используя команду infocmp, определите (и перепишите их себе) escape-последовательности для терминала, выполняющие следующие действия:
    1. • очистка экрана и перемещение курсора в левый верхний угол (clear_screen); • перемещение курсора в заданную позицию экрана (cursor_address);
    2. • задание цвета последующих выводимых символов (set_a_background);
    3. • определение цвета фона для последующих выводимых символов (set_a_foreground);
    4. • скрытие и восстановление курсора (cursor_invisible, cursor_visible).

3. Используя оболочку bash, команду echo –e и скрипт1 , проверьте работу полученных последовательностей. Символ escape задается как \033 или \E. Например – echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
    1. • очищает экран;
    2. • выводит в пятой строке, начиная с 10 символа Ваше имя красными буквами на черном фоне;
    3. • в шестой строке, начиная с 8 символа Вашу группу зеленым цветом на белом фоне; • перемещает курсор в 10 строку, 1 символ и возвращает настройки цвета в значения «по умолчанию».
    4. Разработать следующие функции:
        1. • int mt_clrscr (void)- производит очистку и перемещение курсора в левый верхний угол экрана;
        2. • int mt_gotoXY (int, int) - перемещает курсор в указанную позицию. Первый параметр номер строки, второй - номер столбца;
        3. • int mt_getscreensize (int * rows, int * cols) - определяет размер экрана терминала (количество строк и столбцов);
        4. • int mt_setfgcolor (enum colors) - устанавливает цвет последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа colors, описывающего цвета терминала;
        5. • int mt_setbgcolor (enum colors) - устанавливает цвет фона последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа colors, описывающего цвета терминала. Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода.

Результат:


Привет, это терминал по умолчанию


Привет, это зеленый фон для вывода текста


Memory
+0000 +0000 +0000 +0003 +0000 +0000 +0000 +0000 +0000 +0009
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0027 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0081 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
Keys
l – load s - save
r – run t - step
i – reset F5 - accum
F6 – instruction counter
Flags
F D A O C
Accumulator
Instr. co-er
Operation

main.c

```c
#include "prototype.h"
#include "term.h"
#include "term_gui.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int
main ()
{
  mt_clrscr ();
  sc_memoryInit ();
  for (int i = 3; i < 100; i *= 3)
    {
      sc_memorySet (i, i);
    }
  sc_regInit ();
  sc_regSet (1, 1);
  sc_regSet (2, 1);
  sc_regSet (4, 1);
  sc_regSet (16, 1);
  out_GUI ();

  mt_clrscr ();
  printf ("Привет, это терминал по умолчанию\n");
  getchar ();

  mt_clrscr ();
  mt_setbgcolor (GREEN);
  printf ("Привет, это зеленый фон для вывода текста\n");
  getchar ();

  mt_setbgcolor (DEFAULT);
  mt_clrscr ();
  mt_setfgcolor (RED);
  printf ("Привет, это красный фон для всего\n");
  getchar ();

  mt_clrscr ();
  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.
```

Prototype_test.c

```
include "prototype.h"

int
prototype_test ()
{
  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;
}
```

Term.c

```
include "term.h"

int
mt_clrscr (void) // очищает экран и перемещает курсор в верхний левый угол
{
  if (write (STDOUT_FILENO, CLEAR, strlen (CLEAR))
      < sizeof (char) * strlen (CLEAR))
    {
      return -1;
    }
  return 0;
}

int
mt_gotoXY (
    int x,
    int y) // перемещает курсор к введенным координатам (x, y) = (row, col)
{
  char go[30];
  sprintf (go, "\E[%d;%dH", x, y);
  if (write (STDOUT_FILENO, go, strlen (go)) < sizeof (char) * strlen (go))
    {
      return -1;
    }
  return 0;
}

int
mt_getscreensize (int *rows, int *cols) // получает размер экрана терминала
                                        // (количество строк и столбцов)
{
  struct winsize ws;
  if (ioctl (1, TIOCGWINSZ, &ws))
    {
      return -1;
    }
  *rows = ws.ws_row;
  *cols = ws.ws_col;
  return 0;
}

int
mt_setfgcolor (enum colors color) // устанавливает цвет фона для всех строк и
                                  // столбцов всего терминала
{
  char foreground[30];
  sprintf (foreground, "\E[38;5;%dm", color);
  if (write (STDOUT_FILENO, foreground, strlen (foreground))
      < sizeof (char) * strlen (foreground))
    {
      return -1;
    }
  return 0;
}

int
mt_setbgcolor (enum colors color) //устанавливает цвет фона только для
                                  //предстоящих символов
{
  char background[30];
  sprintf (background, "\E[48;5;%dm", color);
  if (write (STDOUT_FILENO, background, strlen (background))
```

```
      < sizeof (char) * strlen (background))
    {
      return -1;
    }
  return 0;
}
```

Term.h

```
#pragma once

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define CLEAR "\E[H\E[2J"

enum colors
{
  BLACK,
  RED,
  GREEN,
  YELLOW,
  BLUE,
  MAGENTA,
  CYAN,
  WHITE,
  UNKNOWN,
  DEFAULT
};

int mt_clrscr (void); // очищает экран и перемещает курсор в верхний левый угол
int mt_gotoXY (
    int x,
    int y); // перемещает курсор к введенным координатам (x, y) = (row, col)
int mt_getscreensize (int *rows, int *cols); //получает размер экрана терминала
                                             //(количество строк и столбцов)
int mt_setfgcolor (enum colors color); // устанавливает цвет фона для всех и
                                       // столбцов всего терминала
int mt_setbgcolor (enum colors color); // устанавливает цвет фона только для
                                       // предстоящих символов
```

Term_gui.c

```c
#include "term_gui.h"
#include "prototype.h"

void
out_N_hor (int n)
{
  for (int i = 0; i < n; i++)
    {
      printf ("—");
    }
}

void
out_border_top ()
{
  printf ("┌");
  out_N_hor (32);
  printf (" Memory ");
  out_N_hor (32);
  printf ("┐ ┌");
  out_N_hor (5);
  printf (" Flags ");
  out_N_hor (5);
  printf ("┐\n");
}

void
out_flags ()
{
  int flag = 0;
  sc_regGet (1, &flag);
  char F = flag == 1 ? 'F' : ' ';
  sc_regGet (2, &flag);
  char D = flag == 1 ? 'D' : ' ';
  sc_regGet (4, &flag);
  char A = flag == 1 ? 'A' : ' ';
  sc_regGet (8, &flag);
  char O = flag == 1 ? 'O' : ' ';
  sc_regGet (16, &flag);
  char C = flag == 1 ? 'C' : ' ';
  printf ("%c  %c  %c  %c  %c", F, D, A, O, C);
}

void
out_memcell (int n)
{
  int val;
  char cell[5];
  sc_memoryGet (n, &val);
  if (val < 0)
    {
      printf ("-");
      val *= -1;
    }
  else
    {
      printf ("+");
    }
  sprintf (cell, "%04d", val);
  printf ("%s", cell);
}
```

```
void
out_GUI ()
{
  out_border_top ();
  for (int i = 0; i < 100; i++)
    {
      if (i == 0)
        {
          printf ("│  ");
        }
      out_memcell (i);
      printf ("  ");
      if (i % 10 == 9)
        {
          printf ("│ ");
          switch (i)
            {
            case 9:
              printf ("│  ");
              out_flags ();
              printf ("  │");
              break;
            case 19:
              printf ("└");
              out_N_hor (17);
              printf ("┘");
              break;
            case 29:
              printf ("┌");
              out_N_hor (2);
              printf (" Accumulator ");
              out_N_hor (2);
              printf ("┐");
              break;
            case 39:
              printf ("│ ");
              break;
            case 49:
              printf ("└");
              out_N_hor (17);
              printf ("┘");
              break;
            case 59:
              printf ("┌");
              out_N_hor (2);
              printf (" Instr. co-er ");
              out_N_hor (1);
              printf ("┐");
              break;
            case 69:
              printf ("│ ");
              break;
            case 79:
              printf ("└");
              out_N_hor (17);
              printf ("┘");
              break;
            case 89:
              printf ("┌");
              out_N_hor (3);
              printf (" Operation ");
              out_N_hor (3);
              printf ("┐");
```

```
              break;
            case 99:
              printf ("│ ");
              break;
            }
          printf ("\n");
          if (i < 99)
            {
              printf ("│   ");
            }
        }
    }
  printf ("├");
  out_N_hor (10);
  printf (" Keys ");
  out_N_hor (10);
  printf ("┬");
  out_N_hor (45);
  printf ("┤ └");
  out_N_hor (17);
  printf ("┘\n│ ");
  printf ("l – load\ts - save    ");
  printf ("│\n│ ");
  printf ("r – run\tt - step    ");
  printf ("│\n│ ");
  printf ("i – reset\tF5 - accum ");
  printf ("│\n│ ");
  printf ("F6 – instruction counter ");
  printf ("│\n");
  printf ("└");
  out_N_hor (26);
  printf ("┴");
  out_N_hor (45);
  printf ("┘\n");
}
```

Term_gui.h

```
#pragma once

#include "prototype.h"

void out_N_hor (int n);
void out_border_top ();
void out_flags ();
void out_memcell (int n);
void out_GUI ();
```

term_test.c

```c
#include "term.h"
#include <stdio.h>

int
term_test ()
{
  mt_clrscr ();
  printf ("Привет, это терминал по умолчанию\n");
  getchar ();

  mt_clrscr ();
  mt_setbgcolor (GREEN);
  printf ("Привет, это зеленый фон для вывода текста\n");
  getchar ();

  mt_setbgcolor (DEFAULT);
  mt_clrscr ();
  mt_setfgcolor (RED);
  printf ("Привет, это красный фон для всего\n");
  getchar ();

  mt_clrscr ();
  return 0;
}
```

Makefile

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

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