Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «**Архитектура ЭВМ**»

Выполнил:
студент гр. ИС-142
«__» июня 2023 г. _____ /Наумов А.А./

Проверил:
преподаватель
«__» июня 2023 г. _____ /Майданов Ю.С./

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ПОСТАНОВКА ЗАДАЧИ

В рамках курсовой работы необходимо доработать модель *Simple Computer* так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице 1. Из пользовательских функций необходимо реализовать только одну согласно варианту задания (номеру вашей учетной записи). Для разработки программ требуется создать трансляторы с языков *Simple Assembler*.

Обработка команд центральным процессором

Для выполнения программ моделью *Simple Computer* необходимо реализовать две функции:

int *ALU* (*int command, int operand*) – реализует алгоритм работы арифметико-логического устройства. Если при выполнении функции возникла ошибка, которая не позволяет дальше выполнять программу, то функция возвращает -1, иначе 0;

int *CU* (void) – обеспечивает работу устройства управления. Обработку команд осуществляет устройство управления. Функция *CU* вызывается либо обработчиком сигнала от системного таймера, если не установлен флаг «игнорирование тактовых импульсов», либо при нажатии на клавишу *t*.

Алгоритм работы функции следующий:

1. из оперативной памяти считывается ячейка, адрес которой хранится в регистре *instructionCounter*;
2. полученное значение декодируется как команда;
3. если декодирование невозможно, то устанавливаются флаги «указана неверная команда» и «игнорирование тактовых импульсов» (системный таймер можно отключить) и работа функции прекращается.
4. Если получена арифметическая или логическая операция, то вызывается функция *ALU*, иначе команда выполняется самим устройством управления.
5. Определяется, какая команда должна быть выполнена следующей и адрес её ячейки памяти заносится в регистр *instructionCounter*.
6. Работа функции завершается.

Транслятор с языка Simple Assembler

Разработка программ для *Simple Computer* может осуществляться с использованием низкоуровневого языка *Simple Assembler*. Для того чтобы программа могла быть обработана *Simple Computer*необходимо реализовать

транслятор, переводящий текст *Simple Assembler* в бинарный формат, которым может быть считан консолью управления.

Пример программы на **Simple Assembler:**

```
00 READ 09 ; (Ввод A)
01 READ 10 ; (Ввод B)
02 LOAD 09 ; (Загрузка A в аккумулятор)
03 SUB 10 ; (Отнять B)
04 JNEG 07 ; (Переход на 07, если отрицательное)
05 WRITE 09 ; (Вывод A)
06 HALT 00 ; (Останов)
07 WRITE 10 ; (Вывод B)
08 HALT 00 ; (Останов)
09 = +0000 ; (Переменная A)
10 = +9999 ; (Переменная B)
```

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: *sat* файл.*sa* файл.*o*, где файл.*sa* – имя файла, в котором содержится программа на *Simple Assembler*, файл.*o* – результат трансляции.

Оформление отчета по курсовой работе

Отчет о курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается разработанное программное обеспечение. В пояснительную записку должны входить:

- титульный лист;
- полный текст задания к курсовой работе;
- реферат (объем ПЗ, количество таблиц, рисунков, схем, программ, приложений, краткая характеристика и результаты работы);
- содержание:
- постановка задачи исследования;
- блок-схемы используемых алгоритмов;
- программная реализация;
- результаты проведенного исследования;
- выводы;

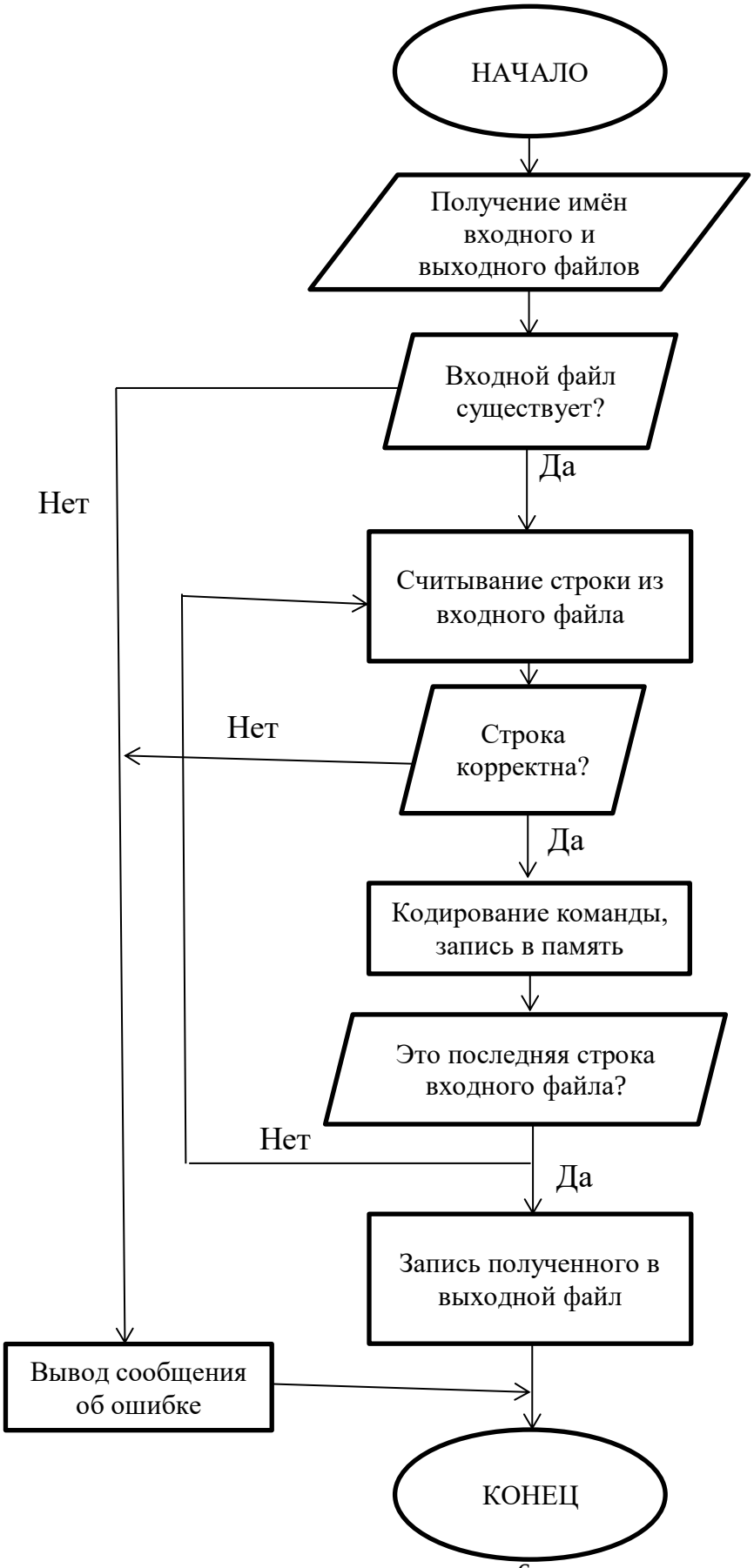
• список использованной литературы;
• подпись, дата.

Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

БЛОК-СХЕМЫ АЛГОРИТМОВ

1. CU()

2. Транслятор с Simple Assembler



ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Main.c

```c
#include "bc.h"
#include "msc.h"
#include "readkey.h"
#include "sig.h"
#include "term.h"
#include "tui.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int
main ()
{
  sc_memoryInit ();
  sc_regInit ();
  sc_regSet (FLAG_IGNORE, 1);
  sc_accumSet (0);
  sc_countSet (0);
  sig_set ();

  rk_mytermsave ();
  rk_mytermregime (0, 0, 1, 0, 1);
  mt_clrscr ();
  g_interface ();
  rk_mytermrestore ();
  return 0;
}
```

Makefile

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

all: bin/simplecomputer
.PHONY: bin/simplecomputer
bin/simplecomputer: src/main.c lib/bc.a lib/msc.a lib/readkey.a lib/tui.a
lib/term.a lib/sig.a lib/alu.a lib/cu.a
	$(DIRGUARD)
	gcc -Wall -Wextra -I src -o $@ $^ -lm

lib/tui.a: obj/tui.o obj/bc.o obj/sig.o
	$(DIRGUARD)
	ar rcs $@ $^

lib/cu.a: obj/cu.o obj/alu.o
	$(DIRGUARD)
	ar rcs $@ $^

lib/%.a: obj/%.o
	$(DIRGUARD)
	ar rcs $@ $<

obj/%.o: src/%.c
```

```
        $(DIRGUARD)
        gcc -Wall -Wextra -c -o $@ $<

test: bin/test
.PHONY: bin/test
bin/test: test/*.c lib/bc.a lib/msc.a lib/readkey.a lib/tui.a lib/term.a
lib/sig.a lib/alu.a lib/cu.a
        $(DIRGUARD)
        gcc -Wall -Wextra -I src -MMD -I thirdparty -o $@ $^ -lm

.PHONY: clean
clean:
        rm -rf bin/ lib/ obj/

sat: bin/sat
.PHONY: bin/sat
bin/sat: src/sat.c src/msc.c
        $(DIRGUARD)
        gcc -Wall -Wextra -o $@ $^
```

testprog.sa

```
00 READ 88 ; (Ввод A)
01 READ 89 ; (Ввод B)
02 LOAD 88 ; (Загрузка A в аккумулятор)
03 SUB 89 ; (Отнять B)
04 JNEG 07 ; (Переход на 07, если отрицательное)
05 WRITE 88 ; (Вывод A)
06 HALT 00 ; (Останов)
07 WRITE 89 ; (Вывод B)
08 HALT 00 ; (Останов)
09 = +0000 ; (Переменная A)
10 = +9999 ; (Переменная B)
```

Alu.c

```c
#include "alu.h"

int
ALU (int command, int operand)
{
  int tmp, accum, a, b;
  sc_memoryGet (operand, &tmp);
  sc_accumGet (&accum);
  switch (command)
    {
    case 0x30: // summation
      sc_accumSet (accum + tmp);
      break;
    case 0x31: // substraction
      sc_accumSet (accum - tmp);
      break;
    case 0x32: // division
      if (tmp == 0)
        {
```

```
          sc_regSet (FLAG_DIV_BY_ZERO, 1);
          return -1;
        }
      sc_regSet (FLAG_DIV_BY_ZERO, 0);
      sc_accumSet (accum / tmp);
      break;
    case 0x33: // multiplication
      sc_accumSet (accum * tmp);
      break;
    case 0x52: // conjunction
      a = accum;
      b = tmp;
      for (int i = 0; i < 8; i++)
        {
          if ((a & (1 << i)) & (b & (1 << i)))
            {
              sc_accumSet (accum | (1 << i));
            }
        }
      break;
    }
  return 0;
}
```

Alu.h

```
#pragma once

#include "msc.h"

int ALU (int command, int operand);
```

bc.c

```
#include "bc.h"

int
bc_printA (char *str)
{
  ssize_t len = strlen (str) * sizeof (char);
  write (STDOUT_FILENO, "\E(0", 3);
  if (write (STDOUT_FILENO, str, len) != len)
    {
      return -1;
    }
  write (STDOUT_FILENO, "\E(B", 3);
  return 0;
}

int
bc_box (int x1, int y1, int x2, int y2)
{
  for (int i = 0; i < x2; i++)
    {
      for (int j = 0; j < y2; j++)
```

```
        {
          mt_gotoXY (x1 + i, y1 + j);
          if (i == 0 && j == 0)
            {
              bc_printA ("l");
            }
          else if (i == 0 && j == y2 - 1)
            {
              bc_printA ("k\n");
            }
          else if (i == x2 - 1 && j == 0)
            {
              bc_printA ("m");
            }
          else if (i == x2 - 1 && j == y2 - 1)
            {
              bc_printA ("j\n");
            }
          else if ((i == 0 || i == x2 - 1)
                   && (j > 0 && j < y2)) // horizontal line
            {
              bc_printA ("q");
            }
          else if ((i > 0 && i < x2)
                   && (j == 0 || j == y2 - 1)) // vertical line
            {
              bc_printA ("x");
            }
          else
            {
              write (STDOUT_FILENO, " ", sizeof (char));
            }
        }
    }
  return 0;
}

int
bc_printbigchar (int *big, int x, int y, enum colors fgcolor,
                 enum colors bgcolor)
{
  mt_setbgcolor (bgcolor);
  mt_setfgcolor (fgcolor);
  for (int i = 0; i < 8; i++)
    {
      for (int j = 0; j < 8; j++)
        {
          mt_gotoXY (x + i, y + j);
          short value;
          bc_getbigcharpos (big, i, j, &value);
          if (!value)
            {
              write (STDOUT_FILENO, " ", sizeof (char));
            }
          else
            {
              bc_printA ("a");
            }
```

```
        }
      write (STDOUT_FILENO, "\n", sizeof (char));
    }
  mt_setbgcolor (GREY);
  mt_setfgcolor (LIGHT_GREY);
  return 0;
}

int
bc_setbigcharpos (int *big, int x, int y, short int value)
{
  if (x < 0 || x > 7 || y < 0 || y > 7 || value > 1 || value < 0)
    {
      return 1;
    }
  int part = x / 4;
  x %= 4;
  if (value)
    {
      big[part] |= (1 << (8 * x + y));
    }
  else
    {
      big[part] &= ~(1 << (8 * x + y));
    }
  return 0;
}

int
bc_getbigcharpos (int *big, int x, int y, short int *value)
{
  if (x < 0 || x > 7 || y < 0 || y > 7)
    {
      return -1;
    }
  short int part = x / 4;
  x %= 4;
  if (big[part] & (1 << (8 * x + y)))
    {
      *value = 1;
    }
  else
    {
      *value = 0;
    }
  return 0;
}

int
bc_bigcharwrite (int fd, int *big, int count)
{
  for (int i = 0; i < count * 2; i++)
    {
      if (write (fd, &big[i], sizeof (int)) == -1)
        {
          return 1;
        }
    }
```

```
  return 0;
}

int
bc_bigcharread (int fd, int *big, int need_count, int *count)
{
  for (*count = 0; (*count < need_count * 2); *count += 1)
    {
      if (read (fd, &big[*count], sizeof (int)) == -1)
        {
          return 1;
        }
    }
  return 0;
}
```

Bc.h

```
#pragma once

#include "term.h"

int bc_printA (char *str);
int bc_box (int x1, int y1, int x2, int y2);
int bc_printbigchar (int *big, int x, int y, enum colors fgcolor,
                     enum colors bgcolor);
int bc_setbigcharpos (int *big, int x, int y, short int value);
int bc_getbigcharpos (int *big, int x, int y, short int *value);
int bc_bigcharwrite (int fd, int *big, int count);
int bc_bigcharread (int fd, int *big, int need_count, int *count);
```

cu.c

```
#include "cu.h"
#include "alu.h"
#include "term.h"
#include "tui.h"

int
READ (int operand) // read new content for memory unit from console
{
  g_clearfields ();
  mt_gotoXY (INPUTFIELD_X, 1);
  mt_printtext (" Input (hex):> ");
  rk_mytermsave ();
  rk_mytermregime (0, 0, 4, 1, 1);
  char buff[5];
  mt_readtext (buff, sizeof (buff));
  int value;
  sscanf (buff, "%x", &value);
  sc_regSet (FLAG_WRONG_COMMAND, 0);
  sc_regSet (FLAG_WRONG_OPERAND, 0);
  int cmd, oper;
  int err = sc_commandDecode (value, &cmd, &oper);
  mt_gotoXY (RESULTFIELD_X, 1);
```

```
  mt_printtext (" ");
  if (err == 0)
    {
      sc_memorySet (operand, value);
      mt_setbgcolor (GREEN);
      mt_printtext (" SUCCESS ");
    }
  else
    {
      sc_memorySet (operand, value);
      mt_setbgcolor (RED);
      mt_printtext (" FAIL : WRONG ");
      if (err == ERR_WRONG_COMMAND)
        {
          sc_regSet (FLAG_WRONG_COMMAND, 1);
          mt_printtext ("COMMAND");
        }
      else
        {
          sc_regSet (FLAG_WRONG_OPERAND, 1);
          mt_printtext ("OPERAND");
        }
      mt_printtext (" : WROTE ANYWAY ");
    }
  mt_setbgcolor (GREY);
  rk_mytermrestore ();
  g_drawmemorybox ();
  return 0;
}

int
WRITE (int operand) // write memory unit contents to console
{
  g_clearfields ();
  char buff[6];
  g_getunit (operand, &buff);
  char tmp[32];
  sprintf (tmp, " Output (hex):> %s", buff);
  mt_gotoXY (RESULTFIELD_X, 1);
  mt_printtext (tmp);
  return 0;
}

int
LOAD (int operand) // put value from accumulator to operand# memory cell
{
  int value = 0;
  sc_memoryGet (operand, &value);
  sc_accumSet (value);
  return 0;
}

int
STORE (int operand) // put operand from accumulator to memory
{
  int accum;
  sc_accumGet (&accum);
  sc_memorySet (operand, accum);
```

```
  return 0;
}

int
JUMP (int operand) // jump to instruction
{
  sc_countSet (operand);
  CU ();
  return 0;
}

int
JNEG (int operand) // jump to instruction if accumulator is negative
{
  int accum;
  sc_accumGet (&accum);
  if (accum < 0)
    {
      sc_countSet (operand);
      CU ();
    }
  return 0;
}

int
JZ (int operand) // jump if accumulator equals to zero
{
  int accum;
  sc_accumGet (&accum);
  if (accum == 0)
    {
      sc_countSet (operand);
      CU ();
    }
  return 0;
}

int
HALT () // set ignore flag to 1
{
  sc_regSet (FLAG_IGNORE, 1);
  alarm (0);
  sc_countSet (0);
  return 0;
}

int
JNS (int operand)
{
  int accum;
  sc_accumGet (&accum);
  if (accum > 0)
    {
      sc_countSet (operand);
    }
  return 0;
}
```

```
int
CU ()
{
  int value = 0, count;
  sc_countGet (&count);
  sc_memoryGet (count, &value);
  int command, operand;
  if (sc_commandDecode (value, &command, &operand) < 0)
    {
      sc_regSet (FLAG_IGNORE, 1);
      return 0;
    }
  if (((command >= 0x30) && (command <= 0x33)) || (command == 0x52))
    {
      ALU (command, operand);
    }
  else
    {
      switch (command)
        {
        case 0x10:
          READ (operand);
          break;
        case 0x11:
          WRITE (operand);
          break;
        case 0x20:
          LOAD (operand);
          break;
        case 0x21:
          STORE (operand);
          break;
        case 0x40:
          JUMP (operand);
          break;
        case 0x41:
          JNEG (operand);
          break;
        case 0x42:
          JZ (operand);
          break;
        case 0x43:
          HALT ();
          break;
        case 0x55:
          JNS (operand);
          break;
        }
    }
  return 0;
}
```

cu.h

```
#pragma once

#include "alu.h"
```

```c
#include "msc.h"
#include "term.h"
#include "tui.h"
#include <string.h>

int READ (int operand);
int WRITE (int operand);
int LOAD (int operand);
int STORE (int operand);
int JUMP (int operand);
int JNEG (int operand);
int JZ (int operand);
int HALT (void);
int JNS (int operand);
int CU (void);
```

msc.c

```c
#include "msc.h"

int sc_memory[MEMSIZE];
int sc_register;
int sc_accum;
int sc_count;

int
sc_memoryInit () // initializes the array of 100 elements
{
  memset (sc_memory, 0, MEMSIZE * sizeof (sc_memory[0]));
  return 0;
}

int
sc_memorySet (int address,
              int value) // sets the value of [address] memory unit
{
  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)  // returns it into value var
{
  if (address < 0 || address >= MEMSIZE)
    {
      BIT_SET (sc_register, FLAG_WRONG_ADDRESS);
      return ERR_WRONG_ADDRESS;
    }
  *value = sc_memory[address];
  return 0;
}
```

```c
int
sc_memorySave (
    char *filename) // saves memory into a binary file (write/fwrite)
{
  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) // loads RAM from a file (read/fread)
{
  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) // inits the register of flags with 0
{
  sc_register = 0;
  return 0;
}

int
sc_regSet (int reg,   // sets the flag register value, #define-s are used for
           int value) // register numbers, if wrong register number - error
{
  if (reg < 1 || reg > 64)
    {
      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) // gets the flag value, if wrong register - error
{
  if (reg < 1 || reg > 64)
    {
      return ERR_WRONG_FLAG;
    }
  *value = BIT_GET (sc_register, reg);
  return 0;
}

int
sc_commandEncode (int command, // encodes command with a specific number and
                  int operand, // operand, puts the result in value, if wrong
                  int *value)  // command or operand - error, value not
changes
{
  if ((command > 0x0 && command < 0x10) || (command > 0x11 && command < 0x20)
      || (command > 0x21 && command < 0x30)
      || (command > 0x33 && command < 0x40)
      || (command > 0x44 && command < 0x51) || command > 0x79)
    {
      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;
    }
  *value = *value | (command << 7);
  *value = *value | operand;
  return 0;
}

int
sc_commandDecode (
    int value,
    int *command, // decodes value as a sc command, if decoding is impossible
-
    int *operand) // sets error command and returns an error
{
  if ((value >> 14) != 0)
    {
      sc_regSet (FLAG_WRONG_COMMAND, 1);
      return ERR_WRONG_COMMAND;
    }
  *command = (value >> 7);
  if ((*command > 0x0 && *command < 0x10)
      || (*command > 0x11 && *command < 0x20)
      || (*command > 0x21 && *command < 0x30)
      || (*command > 0x33 && *command < 0x40)
      || (*command > 0x44 && *command < 0x51) || *command > 0x79)
    {
      sc_regSet (FLAG_WRONG_COMMAND, 1);
      return ERR_WRONG_COMMAND;
    }
  *operand = value & 0b1111111;
```

```
  return 0;
}

int
sc_accumSet (int value)
{
  sc_accum = value;
  return 0;
}

int
sc_accumGet (int *value)
{
  *value = sc_accum;
  return 0;
}

int
sc_countSet (int value)
{
  if (value < 0 || value > 99)
    {
      value = 0;
    }
  sc_count = value;
  return 0;
}

int
sc_countGet (int *value)
{
  *value = sc_count;
  return 0;
}
```

msc.h

```
#pragma once

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

#define MEMSIZE 100

// flags
#define FLAG_IGNORE 32
#define FLAG_WRONG_COMMAND 16
#define FLAG_WRONG_OPERAND 8
#define FLAG_WRONG_ADDRESS 4
#define FLAG_DIV_BY_ZERO 2
#define FLAG_OVERFLOW 1

// errors
#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

// bit operations
#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 (); // initializes the array of 100 elements

int sc_memorySet (int address,
                  int value); // sets the value of [address] memory unit

int sc_memoryGet (int address, // gets the value of [address] memory unit and
                  int *value); // returns it into value var

int sc_memorySave (
    char *filename); // saves memory into a binary file (write/fwrite)

int sc_memoryLoad (char *filename); // loads RAM from a file (read/fread)

int sc_regInit (void); // inits the register of flags with 0

int sc_regSet (
    int register, // sets the flag register value, #define-s are used for
    int value);   // register numbers, if wrong register number - error

int sc_regGet (int register,
               int *value); // gets the flag value, if wrong register - error

int
sc_commandEncode (int command, // encodes command with a specific number and
                  int operand, // operand, puts the result in value, if wrong
                  int *value); // command or operand - error, value not
changes

int sc_commandDecode (
    int value,    // decodes value as a sc command, if decoding is impossible
    int *command, // - sets error command and returns an error
    int *operand);

int sc_accumGet (int *value);

int sc_accumSet (int value);

int sc_countSet (int value);

int sc_countGet (int *value);
```

readkey.c

```
#include "readkey.h"

struct termios current, backup;

int
```

```
rk_readkey (enum keys *key)
{
  char buff[16];
  int numRead;
  if (tcgetattr (STDOUT_FILENO, &backup) != 0)
    {
      return -1;
    }
  if (rk_mytermregime (0, 0, 1, 0, 1) != 0)
    {
      return -1;
    }
  numRead = read (STDOUT_FILENO, buff, 15);
  buff[numRead] = '\0';
  if (strcmp (buff, "l") == 0)
    {
      *key = KEY_L;
    }
  else if (strcmp (buff, "s") == 0)
    {
      *key = KEY_S;
    }
  else if (strcmp (buff, "r") == 0)
    {
      *key = KEY_R;
    }
  else if (strcmp (buff, "q") == 0)
    {
      *key = KEY_Q;
    }
  else if (strcmp (buff, "t") == 0)
    {
      *key = KEY_T;
    }
  else if (strcmp (buff, "i") == 0)
    {
      *key = KEY_I;
    }
  else if ((strcmp (buff, "\n") == 0) || (strcmp (buff, "\r") == 0))
    {
      *key = KEY_ENTER;
    }
  else if ((strcmp (buff, "\E[15~") == 0) || (strcmp (buff, "\E[[E") == 0))
    {
      *key = KEY_F5;
    }
  else if (strcmp (buff, "\E[17~") == 0)
    {
      *key = KEY_F6;
    }
  else if (strcmp (buff, "\E[A") == 0)
    {
      *key = KEY_UP;
    }
  else if (strcmp (buff, "\E[B") == 0)
    {
      *key = KEY_DOWN;
    }
```

```
  else if (strcmp (buff, "\E[C") == 0)
    {
      *key = KEY_RIGHT;
    }
  else if (strcmp (buff, "\E[D") == 0)
    {
      *key = KEY_LEFT;
    }
  if (tcsetattr (STDOUT_FILENO, TCSANOW, &backup) != 0)
    {
      return -1;
    }
  return 0;
}

int
rk_mytermsave ()
{
  tcsetattr (STDOUT_FILENO, TCSANOW, &current);
  memcpy (&backup, &current, sizeof (backup));
  return 0;
}

int
rk_mytermrestore ()
{
  return tcsetattr (STDOUT_FILENO, TCSANOW, &backup);
}

int
rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)
{
  if (tcgetattr (STDOUT_FILENO, &current) != 0)
    {
      return -1;
    }
  // канонический режим = 1
  if (regime)
    {
      current.c_lflag |= ICANON;
    }
  else if (!regime)
    {
      current.c_lflag &= ~ICANON;
    }
  else
    {
      return -1;
    }
  // неканонический режим работы
  if (!regime)
    {
      // количество символов в очереди, чтобы read завершился
      current.c_cc[VTIME] = vtime;
      // сколько времени ждать появления символа
      current.c_cc[VMIN] = vmin;
    }
  // символы будут отражаться по мере набора
```

```
  if (echo == 1)
    {
      current.c_lflag |= ECHO;
    }
  else if (!echo)
    {
      current.c_lflag &= ~ECHO;
    }
  else
    {
      return -1;
    }
  // обработка клавиш прерывания
  if (sigint)
    {
      current.c_lflag |= ISIG;
    }
  else if (!sigint)
    {
      current.c_lflag &= ~ISIG;
    }
  else
    {
      return -1;
    }
  return tcsetattr (STDOUT_FILENO, TCSANOW, &current);
}
```

readkey.h

```
#pragma once

#include "term.h"
#include <string.h>
#include <termios.h>
#include <unistd.h>

enum keys
{
  KEY_S,
  KEY_L,
  KEY_R,
  KEY_T,
  KEY_I,
  KEY_F5,
  KEY_F6,
  KEY_UP,
  KEY_DOWN,
  KEY_RIGHT,
  KEY_LEFT,
  KEY_ENTER,
  KEY_Q,
  KEY_DEFAULT,
};

int rk_readkey (enum keys *key);
int rk_mytermsave (void);
```

```
int rk_mytermrestore (void);
int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint);
```

sat.c

```
#include "msc.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *input = NULL;

void
errOutput (char *message)
{
  fprintf (stderr, "%s", message);
  exit (EXIT_FAILURE);
}

void
loadFile (const char *filename)
{
  if ((input = fopen (filename, "r")) == NULL)
    {
      errOutput ("Input file does not exist\n");
    }
}

int
getCommandNum (char *command)
{
  int result = 0;
  if (strcmp (command, "READ") == 0)
    {
      result = 0x10;
    }
  else if (strcmp (command, "WRITE") == 0)
    {
      result = 0x11;
    }
  else if (strcmp (command, "LOAD") == 0)
    {
      result = 0x20;
    }
  else if (strcmp (command, "STORE") == 0)
    {
      result = 0x21;
    }
  else if (strcmp (command, "ADD") == 0)
    {
      result = 0x30;
    }
  else if (strcmp (command, "SUB") == 0)
    {
      result = 0x31;
    }
  else if (strcmp (command, "DIVIDE") == 0)
```

```
    {
      result = 0x32;
    }
  else if (strcmp (command, "MUL") == 0)
    {
      result = 0x33;
    }
  else if (strcmp (command, "JUMP") == 0)
    {
      result = 0x40;
    }
  else if (strcmp (command, "JNEG") == 0)
    {
      result = 0x41;
    }
  else if (strcmp (command, "JZ") == 0)
    {
      result = 0x42;
    }
  else if (strcmp (command, "HALT") == 0)
    {
      result = 0x43;
    }
  else if (strcmp (command, "AND") == 0)
    {
      result = 0x52;
    }
  else if (strcmp (command, "JNS") == 0)
    {
      result = 0x55;
    }
  else
    {
      char errMsg[64];
      sprintf (errMsg, "Not recognized command %s\n", command);
      errOutput (errMsg);
    }
  return result;
}

void
writeTranslationTo (const char *filename)
{
  for (int i = 0; !feof (input); i++)
    {
      char line[255];
      if (!fgets (line, 255, input))
        {
          if (feof (input))
            {
              break;
            }
          else
            {
              char errMsg[64];
              sprintf (errMsg, "Can't read line %d from input file\n", i);
              errOutput (errMsg);
            }
```

```
        }

      char *ptr = strtok (line, " ");
      char *str_address = ptr;
      int address = atoi (str_address);
      if (address < 0 || address > 99)
        {
          char errMsg[64];
          sprintf (errMsg, "Expected address of memory cell on line %d\n",
i);
          errOutput (errMsg);
        }

      ptr = strtok (NULL, " ");
      char *command = ptr;

      ptr = strtok (NULL, " +");
      char *str_operand = ptr;

      ptr = strtok (NULL, " ");

      if (ptr != NULL && ptr[0] != ';')
        {
          char errMsg[64];
          sprintf (errMsg, "Unexpected symbols on line %d\n", i);
          errOutput (errMsg);
        }

      int operand = atoi (str_operand);
      char buffer[255];
      sprintf (buffer, "%02x", operand);
      sscanf (buffer, "%02x", &operand);

      if (command[0] == '=')
        {
          sscanf (str_operand, "%x", &operand);
          sc_memorySet (address, operand);
          continue;
        }
      int num_of_cmd = getCommandNum (command);

      // command encoding, correctness checking, writing to mem
      int value = 0;
      if (sc_commandEncode (num_of_cmd, operand, &value) < 0)
        {
          char errMsg[64];
          sprintf (errMsg, "Encoding error on line %d\n", i);
          errOutput (errMsg);
        }
      sc_memorySet (address, value);
    }

  // write the result memory to output file
  sc_memorySave (strdup (filename));
}

int
main (int argc, const char **argv)
```

```
{
  if (argc < 3)
    {
      errOutput ("Usage: ./sat <input_file.sa> <output_object_file.o>\n");
    }
  sc_memoryInit ();
  loadFile (argv[1]);
  writeTranslationTo (argv[2]);
  return 0;
}
```

sig.c

```
#include "sig.h"
#include "cu.h"
#include "tui.h"

int
sig_handle_reset ()
{
  sc_memoryInit ();
  sc_regInit ();
  sc_regSet (FLAG_IGNORE, 1);
  alarm (0);
  return 0;
}

int
sig_handle_alarm ()
{
  CU ();
  int value;
  sc_countGet (&value);
  sc_countSet (value + 1);
  sc_countGet (&value);
  g_drawboxes ();
  int flag;
  int x = value / 10;
  int y = value % 10;
  g_highlightmemory (x, y);
  g_drawbcbox ();
  sc_regGet (FLAG_IGNORE, &flag);
  if (flag == 0)
    {
      ualarm (100, 0);
    }
  else
    {
      alarm (0);
    }
  return 0;
}

int
sig_set ()
{
```

```
  signal (SIGUSR1, sig_handle_reset);
  signal (SIGALRM, sig_handle_alarm);
  return 0;
}
```

sig.h

```
#pragma once

#include "msc.h"
#include <signal.h>
#include <unistd.h>

int sig_handle_reset (void);
int sig_handle_alarm (void);
int sig_set (void);
```

term.c

```
#include "term.h"

int
mt_clrscr (void)
{
  if ((unsigned long)write (STDOUT_FILENO, CLEAR, strlen (CLEAR))
      < sizeof (char) * strlen (CLEAR))
    {
      return -1;
    }
  return 0;
}

int
mt_gotoXY (int x, int y)
{
  char go[30];
  sprintf (go, "\E[%d;%dH", x, y);
  if ((unsigned long)write (STDOUT_FILENO, go, strlen (go))
      < sizeof (char) * strlen (go))
    {
      return -1;
    }
  return 0;
}

int
mt_getscreensize (int *rows, int *cols)
{
  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)
{
  char foreground[30];
  sprintf (foreground, "\E[38;5;%dm", color);
  if ((unsigned long)write (STDOUT_FILENO, foreground, strlen (foreground))
      < sizeof (char) * strlen (foreground))
    {
      return -1;
    }
  return 0;
}

int
mt_setbgcolor (enum colors color)
{
  char background[30];
  sprintf (background, "\E[48;5;%dm", color);
  if ((unsigned long)write (STDOUT_FILENO, background, strlen (background))
      < sizeof (char) * strlen (background))
    {
      return -1;
    }
  return 0;
}

int
mt_readtext (char *text, int size)
{
  int numRead = read (STDOUT_FILENO, text, size);
  text[numRead] = '\0';
  return 0;
}

int
mt_printtext (char *text)
{
  write (STDOUT_FILENO, text, strlen (text));
  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
{
  LIGHT_GREY = 247,
  GREY = 237,
  BLACK = 16,
  RED = 124,
  GREEN = 41,
  BLUE = 20,
  YELLOW = 184,
};

int mt_clrscr (void); // clears screen and moves cursor to upper left corner
int mt_gotoXY (
    int x,
    int y); // moves cursor to the entered coordinates (x, y) = (row, col)
int mt_getscreensize (
    int *rows, int *cols); // gets terminal screen size (num of rows and
cols)
int mt_setfgcolor (enum colors color); // sets a background color for all
rows
                                       // and cols, entire terminal
int mt_setbgcolor (
    enum colors color); // sets a background color only for upcoming
characters
int mt_readtext (char *text, int size);
int mt_printtext (char *text);
```

tui.c

```
#include "tui.h"
#include "cu.h"

int
g_writeflags (char **val)
{
  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' : ' ';
  sc_regGet (32, &flag);
  char I = flag == 1 ? 'I' : ' ';
  char buff[24] = { 0 };
  sprintf (buff, "%c  %c  %c  %c  %c  %c\0", F, D, A, O, C, I);
  *val = buff;
  return 0;
}

int
g_drawborders (void)
{
```

```
  bc_box (1, 1, 12, 63);
  mt_gotoXY (0, 28);
  write (STDOUT_FILENO, " Memory ", 8 * sizeof (char));
  bc_box (1, 64, 3, 39);
  mt_gotoXY (1, 77);
  write (STDOUT_FILENO, " accumulator ", 13 * sizeof (char));
  bc_box (4, 64, 3, 39);
  mt_gotoXY (4, 73);
  write (STDOUT_FILENO, " instructionCounter ", 20 * sizeof (char));
  bc_box (7, 64, 3, 39);
  mt_gotoXY (7, 78);
  write (STDOUT_FILENO, " Operation ", 11 * sizeof (char));
  bc_box (10, 64, 3, 39);
  mt_gotoXY (10, 79);
  write (STDOUT_FILENO, " Flags ", 7 * sizeof (char));
  bc_box (13, 1, 12, 63);
  bc_box (13, 64, 12, 39);
  mt_gotoXY (13, 67);
  write (STDOUT_FILENO, " Keys ", 7 * sizeof (char));
  char *str[7] = { "l - load",
                   "s - save",
                   "r - run",
                   "t - step",
                   "i - reset",
                   "F5 - accumulator",
                   "F6 - instructionCounter" };
  for (int i = 0; i < 7; i++)
    {
      mt_gotoXY (15 + i, 66);
      write (STDOUT_FILENO, str[i], strlen (str[i]));
    }
  mt_gotoXY (33, 0);
  return 0;
}

int
g_drawaccumbox (void)
{
  mt_gotoXY (2, 80);
  char buff[5];
  int val;
  sc_accumGet (&val);
  sprintf (buff, "%d", val);
  write (STDOUT_FILENO, buff, 5 * sizeof (char));
  mt_gotoXY (33, 0);
  return 0;
}

int
g_drawflagbox (void)
{
  mt_gotoXY (11, 78);
  char *val;
  g_writeflags (&val);
  write (STDOUT_FILENO, val, 24 * sizeof (char));
  mt_gotoXY (33, 0);
  return 0;
}
```

```
int
g_drawbcbox ()
{
  int big[18][2] = {
    { 0x4242423C, 0x3C424242 }, // 0
    { 0x48506040, 0x40404040 }, // 1
    { 0x2042423C, 0x7E020418 }, // 2
    { 0x3840423C, 0x3C424040 }, // 3
    { 0x7E424478, 0x40404040 }, // 4
    { 0x3E02027E, 0x3C424040 }, // 5
    { 0x3E02423C, 0x3C424242 }, // 6
    { 0x2040407E, 0x10101010 }, // 7
    { 0x3C42423C, 0x3C424242 }, // 8
    { 0x7C42423C, 0x3C424040 }, // 9
    { 0x66663C18, 0x66667E7E }, // A
    { 0x3E66663E, 0x3E666666 }, // B
    { 0x0202423C, 0x3C420202 }, // C
    { 0x4444443E, 0x3E444444 }, // D
    { 0x3E02027E, 0x7E020202 }, // E
    { 0x1E02027E, 0x02020202 }, // F
    { 0x7E181800, 0x0018187E }, // +
    { 0x7E000000, 0x7E }        // -
  };
  int address, val;
  sc_countGet (&address);
  char buff[6];
  g_getunit (address, &buff);
  int *digit = malloc (2 * sizeof (int));
  for (int i = 0; i < 5; i++)
    {
      int k = 0;
      switch (buff[i])
        {
        case '0':
          k = 0;
          break;
        case '1':
          k = 1;
          break;
        case '2':
          k = 2;
          break;
        case '3':
          k = 3;
          break;
        case '4':
          k = 4;
          break;
        case '5':
          k = 5;
          break;
        case '6':
          k = 6;
          break;
        case '7':
          k = 7;
          break;
```

```
        case '8':
          k = 8;
          break;
        case '9':
          k = 9;
          break;
        case 'a':
          k = 10;
          break;
        case 'b':
          k = 11;
          break;
        case 'c':
          k = 12;
          break;
        case 'd':
          k = 13;
          break;
        case 'e':
          k = 14;
          break;
        case 'f':
          k = 15;
          break;
        case '+':
          k = 16;
          break;
        default:
          k = 17;
          break;
        }
      digit[0] = big[k][0];
      digit[1] = big[k][1];
      bc_printbigchar (digit, BC_X, BC_START + i * BC_STEP, GREEN, GREY);
    }
  mt_gotoXY (33, 0);
  return 0;
}

int
g_clearfields ()
{
  for (int j = 25; j < 28; j++)
    {
      mt_gotoXY (j, 1);
      for (int i = 0; i < 102; i++)
        {
          mt_printtext (" ");
        }
    }
  return 0;
}

int
g_loadmemory (void)
{
  g_clearfields ();
  mt_gotoXY (INPUTFIELD_X, 1);
```

```
  mt_printtext (" Load from file:> ");
  rk_mytermregime (0, 0, 1, 1, 1);
  char *buff = calloc (0, sizeof (char) * 100);
  scanf ("%s", buff);
  mt_gotoXY (RESULTFIELD_X, 1);
  mt_printtext (" ");
  if (sc_memoryLoad (buff) == 0)
    {
      mt_setbgcolor (GREEN);
      mt_printtext (" SUCCESS ");
    }
  else
    {
      mt_setbgcolor (RED);
      mt_printtext (" FAIL ");
    }
  mt_setbgcolor (GREY);
  g_drawboxes ();
  return 0;
}

int
g_savememory (void)
{
  g_clearfields ();
  mt_gotoXY (INPUTFIELD_X, 1);
  mt_printtext (" Save to file:> ");
  rk_mytermregime (0, 0, 1, 1, 1);
  char *buff = calloc (0, sizeof (char) * 100);
  scanf ("%s", buff);
  sc_memorySave (buff);
  mt_gotoXY (RESULTFIELD_X, 1);
  mt_printtext (" ");
  mt_setbgcolor (GREEN);
  mt_printtext (" SUCCESS ");
  mt_setbgcolor (GREY);
  g_drawboxes ();
  return 0;
}

int
g_getunit (int address, char *buff)
{
  int val, command, operand;
  sc_memoryGet (address, &val);
  if (val < 0)
    {
      sprintf (buff, "-%04x", abs (val));
      return 0;
    }
  char temp[6];
  int err = sc_commandDecode (val, &command, &operand);
  sprintf (temp, " %04x", val);
  if (err == 0)
    {
      temp[0] = '+';
    }
  else
```

```
    {
      temp[0] = '-';
    }
  strcpy (buff, temp);
  return 0;
}

int
g_highlightmemory (int x, int y)
{
  g_drawmemorybox ();
  int address = x * 10 + y;
  char buff[6];
  g_getunit (address, &buff);
  mt_gotoXY (X_START + x, Y_START + y * Y_STEP);
  mt_setbgcolor (BLUE);
  mt_printtext (buff);
  mt_setbgcolor (GREY);
  return 0;
}

int
g_drawoperationbox (void)
{
  mt_gotoXY (8, 79);
  int address, val, cmd = 0, oper = 0;
  sc_countGet (&address);
  sc_memoryGet (address, &val);
  char buff[11];
  int err = sc_commandDecode (val, &cmd, &oper);
  sprintf (buff, " %02x:%d", cmd, oper);
  if (err == 0)
    {
      buff[0] = '+';
    }
  else
    {
      buff[0] = '-';
    }
  mt_printtext (buff);
  mt_gotoXY (33, 0);
  return 0;
}

int
g_drawcounterbox (void)
{
  mt_gotoXY (5, 80);
  char buff[5];
  int val;
  sc_countGet (&val);
  sprintf (buff, "%04d", val);
  write (STDOUT_FILENO, buff, 5 * sizeof (char));
  mt_gotoXY (33, 0);
  return 0;
}

int
```

```
g_drawmemorybox (void)
{
  int k = 0;
  for (int i = X_START; i < 12; i++)
    {
      for (int j = Y_START; j < 63; j += Y_STEP)
        {
          mt_gotoXY (i, j);
          char buff[6];
          g_getunit (k++, &buff);
          write (STDERR_FILENO, buff, 6 * sizeof (char));
        }
    }
  return 0;
}

int
g_drawboxes ()
{
  g_drawmemorybox ();
  g_drawaccumbox ();
  g_drawcounterbox ();
  g_drawoperationbox ();
  g_drawflagbox ();
  g_drawbcbox ();
  return 0;
}

int
g_interface ()
{
  mt_setbgcolor (GREY);
  g_drawborders ();
  g_clearfields ();
  int exit = 0, address = 0, flag = 0;
  while (!exit)
    {
      sc_countGet (&address);
      g_drawboxes ();
      int x = address / 10, y = address % 10;
      g_highlightmemory (x, y);
      enum keys key = KEY_DEFAULT;
      rk_readkey (&key);
      switch (key)
        {
        case KEY_Q: // exit
          exit++;
          break;
        case KEY_UP:
          address -= 10;
          if (address < 0)
            {
              address += 100;
            }
          sc_countSet (address);
          break;
        case KEY_DOWN:
          address += 10;
```

```
          if (address > 99)
            {
              address -= 100;
            }
          sc_countSet (address);
          break;
        case KEY_LEFT:
          address -= 1;
          if (address % 10 == 9 || address == -1)
            {
              address += 10;
            }
          sc_countSet (address);
          break;
        case KEY_RIGHT:
          address += 1;
          if (address % 10 == 0)
            {
              address -= 10;
            }
          sc_countSet (address);
          break;
        case KEY_L: // load memory
          g_loadmemory ();
          break;
        case KEY_S: // save memory
          g_savememory ();
          break;
        case KEY_I: // reset
          raise (SIGUSR1);
          x = 0, y = 0;
          g_drawboxes ();
          break;
        case KEY_ENTER: // setting values
          READ (address);
          break;
        case KEY_R: // ignore flag
          sc_regGet (FLAG_IGNORE, &flag);
          sc_regSet (FLAG_IGNORE, !flag);
          alarm (flag);
          break;
        case KEY_T: // step key - calling control unit
          CU ();
          break;
        case KEY_F5:
          break;
        case KEY_F6:
          break;
        }
    }
  return 0;
}
```

tui.h

```
#pragma once
```

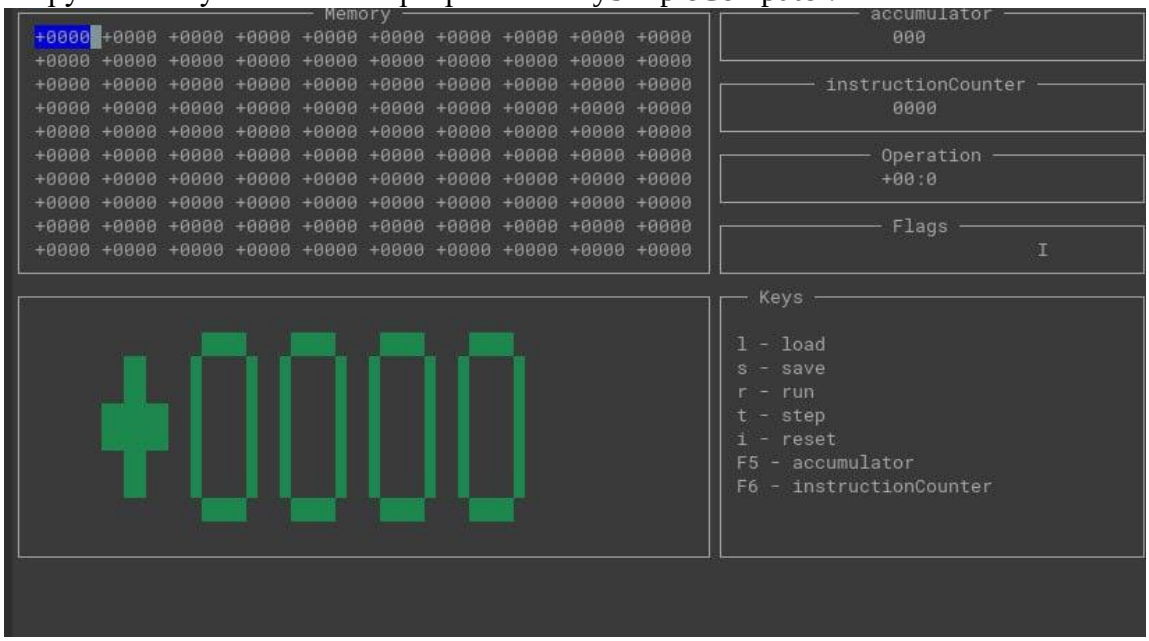
```
#include "bc.h"
#include "msc.h"
#include "readkey.h"
#include "sig.h"
#include "term.h"
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <sys/param.h>
#include <unistd.h>

#define BC_X 15
#define BC_START 8
#define BC_STEP 8
#define X_START 2
#define Y_START 3
#define Y_STEP 6
#define INPUTFIELD_X 25
#define RESULTFIELD_X 26

int g_writeflags (char **val);
int g_drawborders (void);
int g_drawaccumbox (void);
int g_drawflagbox (void);
int g_drawbcbox (void);
int g_clearfields (void);
int g_loadmemory (void);
int g_savememory (void);
int g_getunit (int address, char *buff);
int g_highlightmemory (int x, int y);
int g_drawoperationbox (void);
int g_drawcounterbox (void);
int g_drawmemorybox (void);
int g_drawboxes (void);
int g_interface (void);
```

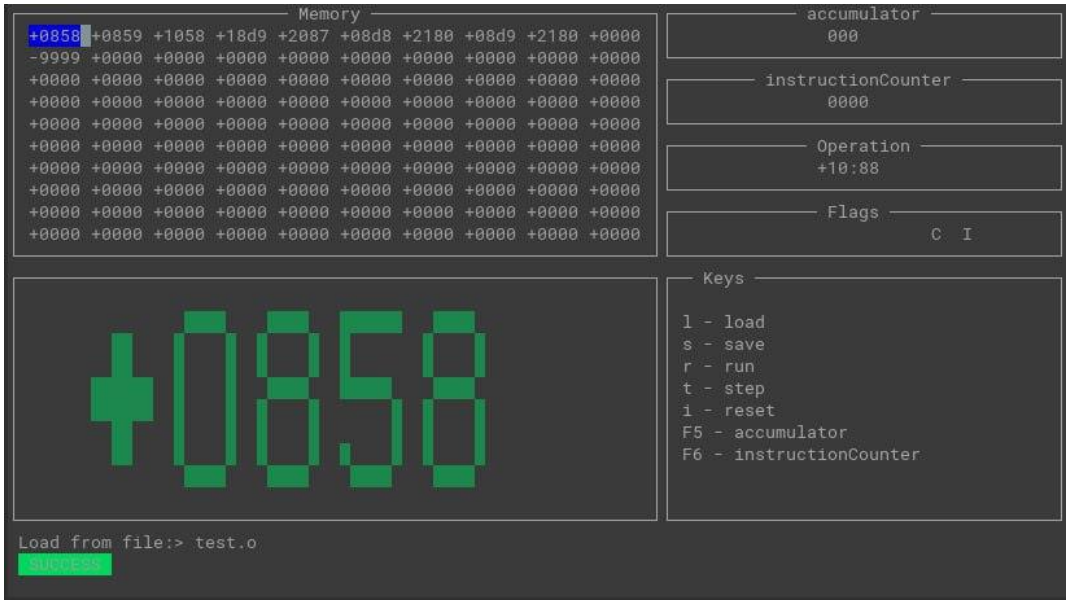
РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Трансляция программы на Simple Assembler в объектный файл:

```
alexeynaumov@Lenovo-Legion-5-15ARH05H-267a6435:~/ArcPC/simplecomputer$ cat testprog.sa
00 READ 88 ; (Ввод A)
01 READ 89 ; (Ввод B)
02 LOAD 88 ; (Загрузка A в аккумулятор)
03 SUB 89 ; (Отнять B)
04 JNEG 07 ; (Переход на 07, если отрицательное)
05 WRITE 88 ; (Вывод A)
06 HALT 00 ; (Останов)
07 WRITE 89 ; (Вывод B)
08 HALT 00 ; (Останов)
09 = +0000 ; (Переменная A)
10 = +9999 ; (Переменная B)
alexeynaumov@Lenovo-Legion-5-15ARH05H-267a6435:~/ArcPC$ cd simplecomputer/
alexeynaumov@Lenovo-Legion-5-15ARH05H-267a6435:~/ArcPC/simplecomputer$ bin/sat testprog.sa test.o
alexeynaumov@Lenovo-Legion-5-15ARH05H-267a6435:~/ArcPC/simplecomputer$ ls
bin  factorial.sb  lib  Makefile  memory.mem  obj  src  test  test.o  testprog.sa  thirdparty
```

Загрузка и запуск тестовой программы в mySimpleComputer:





ЗАКЛЮЧЕНИЕ

В рамках выполнения курсовой работы мной была доработана модель *Simple Computer* так, чтобы она обрабатывала команды, записанные в оперативной памяти с помощью функций *ALU()* и *CU()*. Также были разработаны транслятор с языка программирования *Simple Assembler*, переводящий код с него в бинарный формат.

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. Организация ЭВМ и систем. Практикум // С.Н. Мамойленко, Новосибирск: ГОУ ВПО «СибГУТИ», 2005 г.

2. Архитектура компьютера. 4-е изд. // Э. Танненбаум. – СПб.: Издательство «Питер», 2003.

3. Организация ЭВМ. 5-е изд. / К. Хамахер, З. Вранешич, С. Заки. – СПб.: Издательство «Питер», 2003.