Федеральное агентство связи
Государственное образовательное учреждение
высшего профессионального образования
«Сибирский государственный университет
телекоммуникаций и информатики»

Кафедра вычислительных систем

Курсовая работа по дисциплине «Архитектура ЭВМ»

Выполнил:
Студент группы ИП-812
Дорогин Н.С.

Проверил:
Преподаватель кафедры
Челканова Т.В.

20.05.2025 _____

1. Постановка задачи.

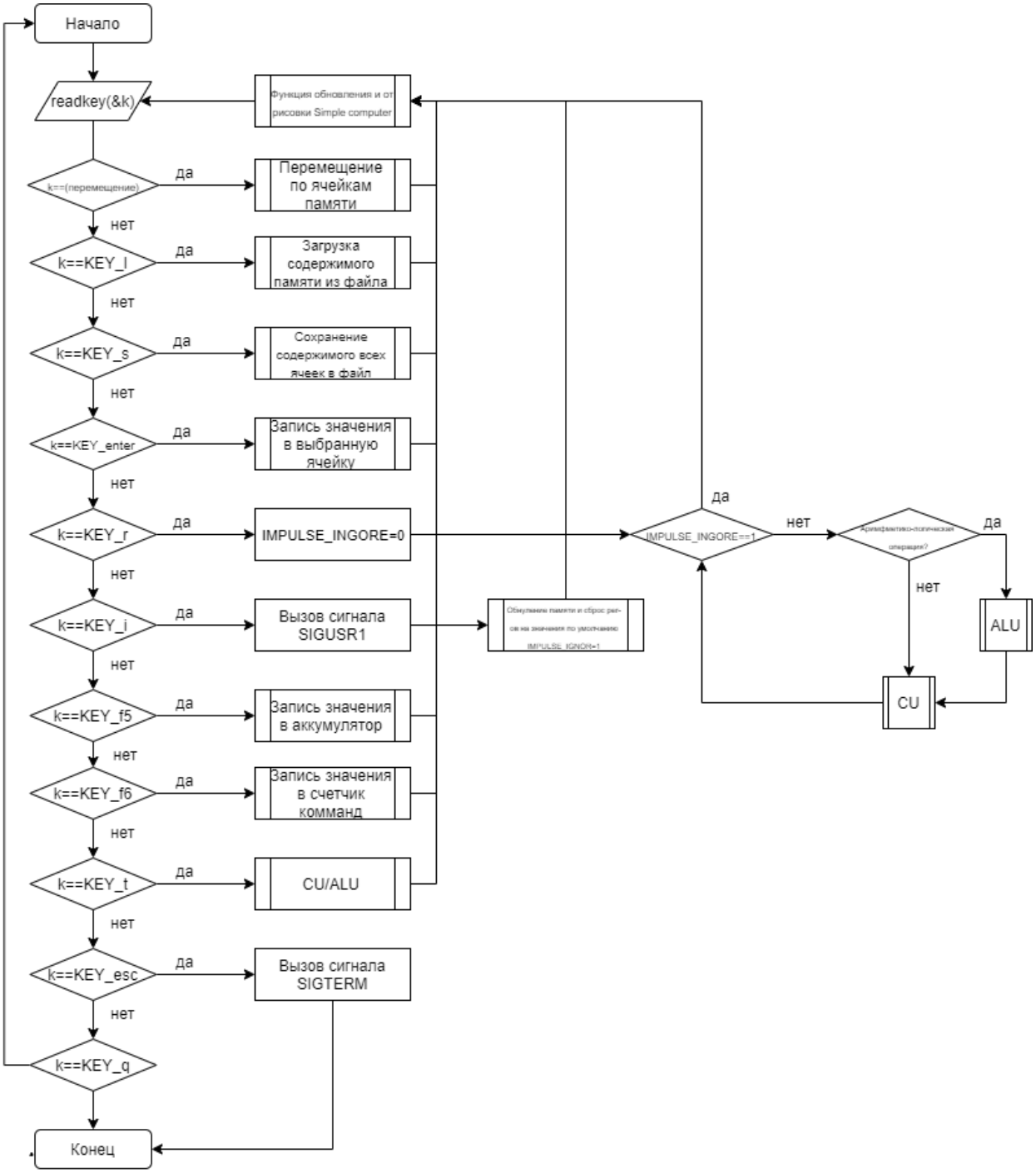
В рамках курсовой работы необходимо:

➢ Разработать транслятор с языка Simple Basic. Итог работы транслятора – бинарный файл

с образом оперативной памяти Simple Computer, который можно загрузить в модель и выполнить.

2. Выполнение работы.

Блок-схемы алгоритмов программы. (SimpleComputer, Assembler translator)




Начало
Открытие файлов
Это конец файла?
да
Запись в файл данных из временной памяти транслятора
Конец
нет
Считывание строки из файла
Проверка правильности формата команды и операндов в считанной строке
Кодирование команды и операнда в формат значения ячейки памяти Simple Computer
Запись закодированной команды во временную память транслятора аналогичную Simple Computer

2.1. Модель центрального процессора.

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. Является функцией обратного вызова, вызываемая системным таймером или по нажатию клавиши «t» в случае установленного флага пошаговой работы. Производит чтение из массива – модели оперативной памяти со смещением, хранящемся в регистре - указателе команд icounter и декодирует полученное значение. В результате декодирования возможны следующие исходы: - устройство управления самостоятельно обрабатывает инструкцию, если команда не требует арифметических вычислений. Возвращает «0». - устройство управления передает обработку команды в АЛУ, если команда требует арифметических вычислений над операндами. К этой группе команд относятся команды, выполняющие арифметические и логические действия. Возвращает «0». - в случае поступления некорректной команды (с недопустимым операционным кодом или операндом) устройство управления останавливает работу SimpleComputer и возвращает «-1».

2.2. Транслятор с языка Simple Assembler.

Задачей транслятора является преобразование текстового файла, содержащего программный код на языке SimpleAssembler, в бинарный файл, содержащий дамп памяти SimpleComputer (дамп содержит программу в машинном коде).

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)
```

2.4 Пользовательская функция

JC с кодом 56. Переход к указанному адресу памяти, если при сложении произошло переполнение.

Код функции:

```
case 0x38: // JC (56)
{
   int flag;
   sc_regGet(REG_OPERATION_OVERFLOW, &flag);
   if (flag)
   {
      icounter = operand;
   }
   else
   {
      icounter++;
   }
   break;
}
```

JNC с кодом 57. Переход к указанному адресу памяти, если при сложении не произошло переполнение.

Код функции:

```
case 0x39: // JNC (57)
{
   int flagt;
   sc_regGet(REG_OPERATION_OVERFLOW, &flagt);
   if (!flagt)
   {
      icounter = operand;
   }
   else
   {
      icounter++;
   }
   break;
}
```

JP с кодом 58. Переход к указанному адресу памяти, если при сложении произошло переполнение.

```
case 0x3A: // JP (58)
{
   if ((accumulator & 0x0001) == 0)
   {
      icounter = operand;
   }
   else
   {
      icounter++;
   }
   break;
}
```

JNP с кодом 59. Переход к указанному адресу памяти, если при сложении не произошло переполнение.

```
case 0x3B: // JNP (59)
{
   if (accumulator & 0x0001)
   {
      icounter = operand;
   }
   else
   {
      icounter++;
   }
   break;
}
```

3.Пример работы Simple Computer.

Код на Simple Assembler:

00 CPUINFO 00 ;

01 READ 10 ;

02 READ 11 ;

03 LOAD 10 ;

04 SUB   11 ;

05 JNEG 14 ;

06 STORE 12 ;

07 WRITE 12 ;

08 HALT 00 ;

14 STORE 12 ;

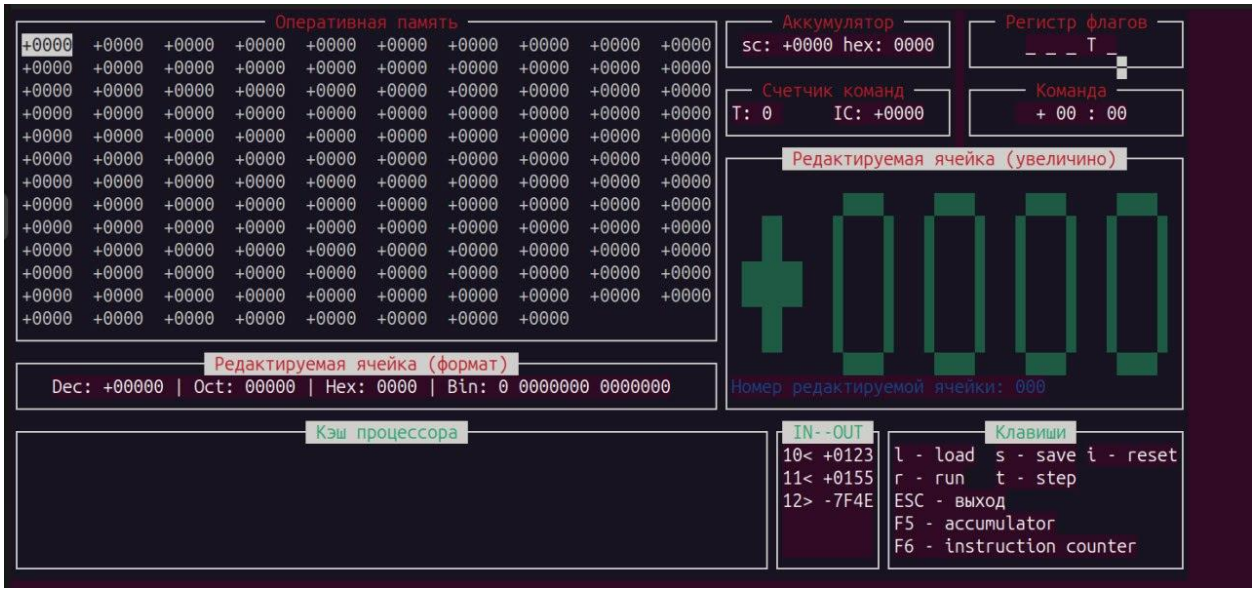
15 WRITE 12 ;

16 JUMP 127 ;

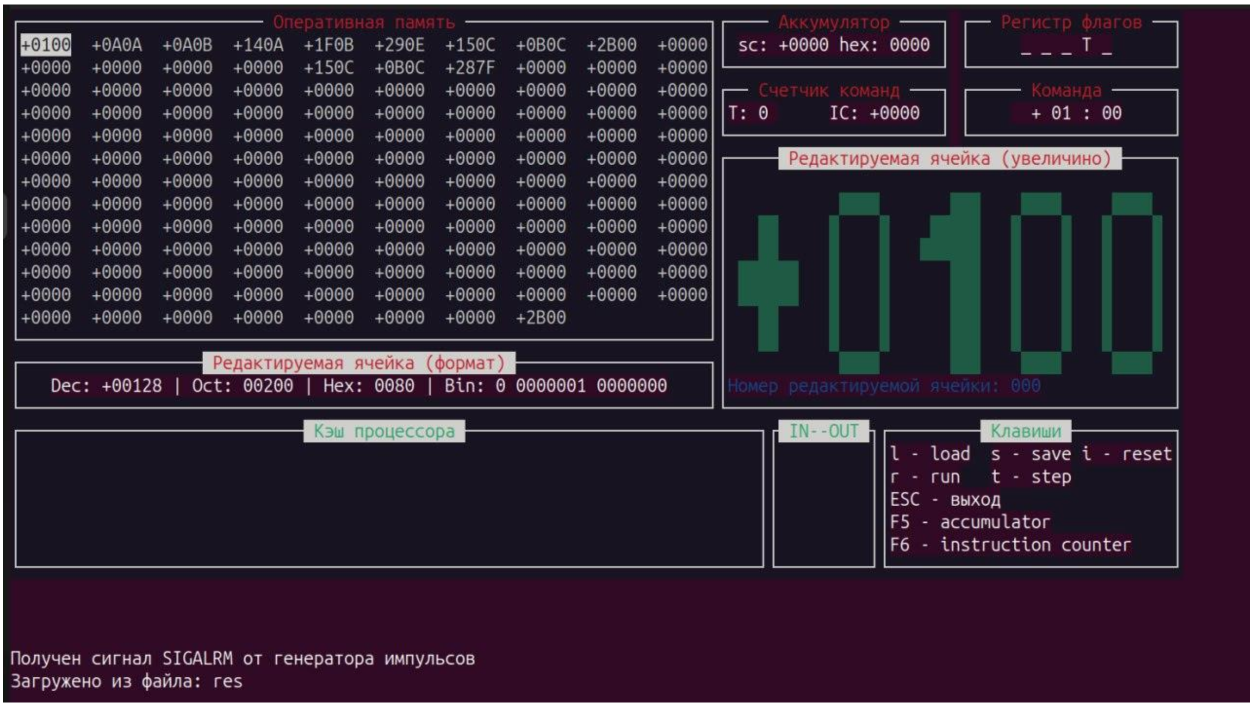
127 HALT 00 ;

```
user@user-MaiBook-X-series:~/Desktop/АрхитектураЭВМ/Репозиторий/simplecomputer$ make sat pr.sa res.bin
make -C ./console sat pr.sa res.bin
make[1]: вход в каталог «/home/user/Desktop/АрхитектураЭВМ/Репозиторий/simplecomputer/console»
Successfully compiled to ./saves/res.bin
make[1]: выход из каталога «/home/user/Desktop/АрхитектураЭВМ/Репозиторий/simplecomputer/console»
```

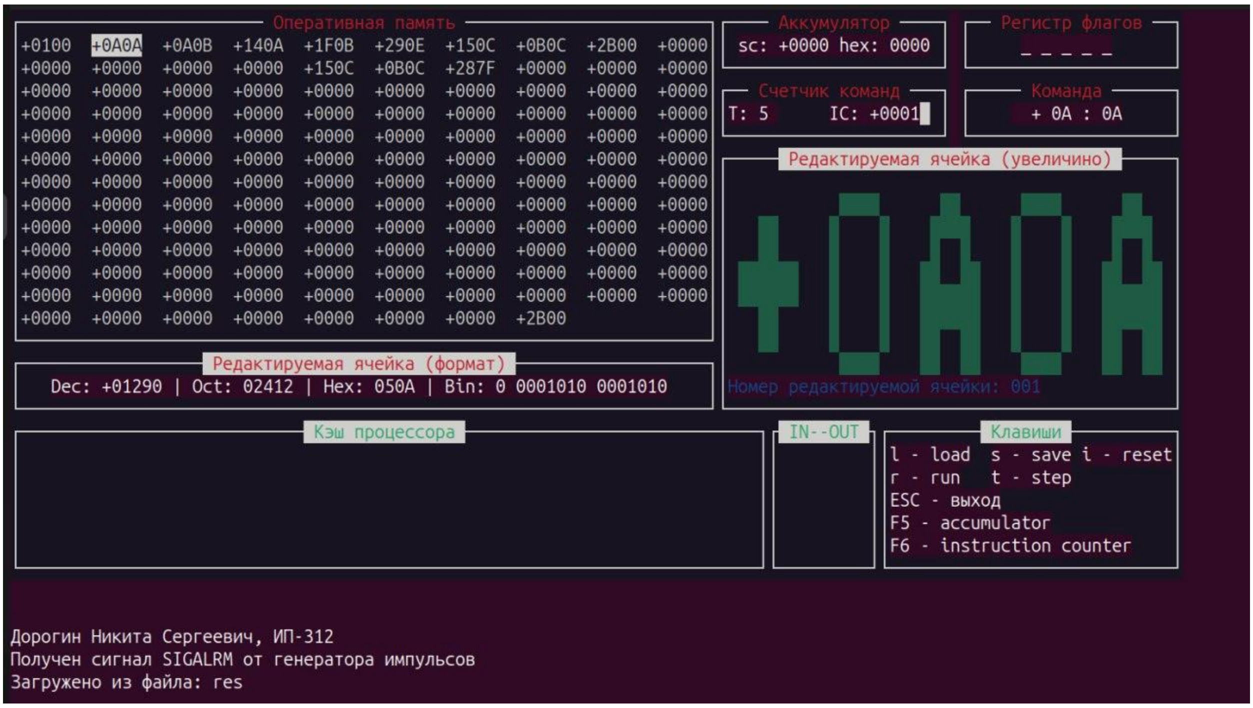
Перевод программы pr.sa на Simple Assembler в бинарный файл res.bin, для дальнейшего считывания в память нашего Simple Computer.



Simple Computer при запуске.

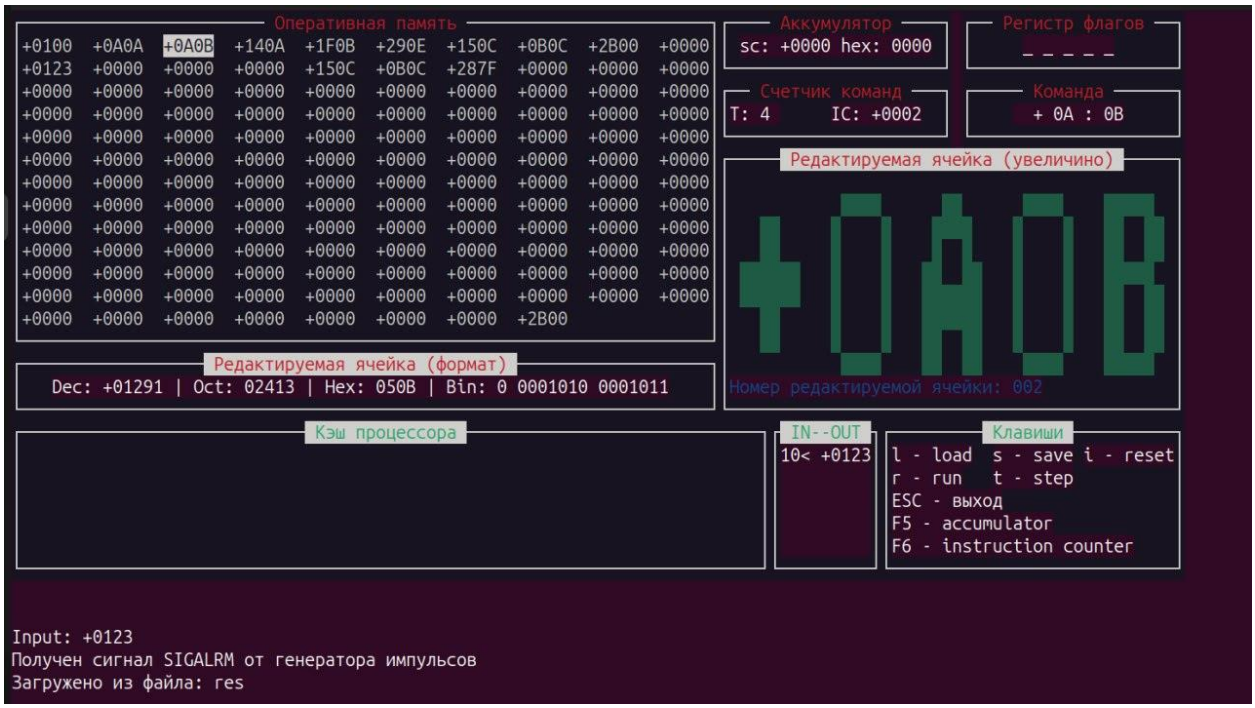


Жмём клавишу l. Нам предложат ввести название файла, из которого мы считываем. Вводим res (без формата) и получаем изображение выше.

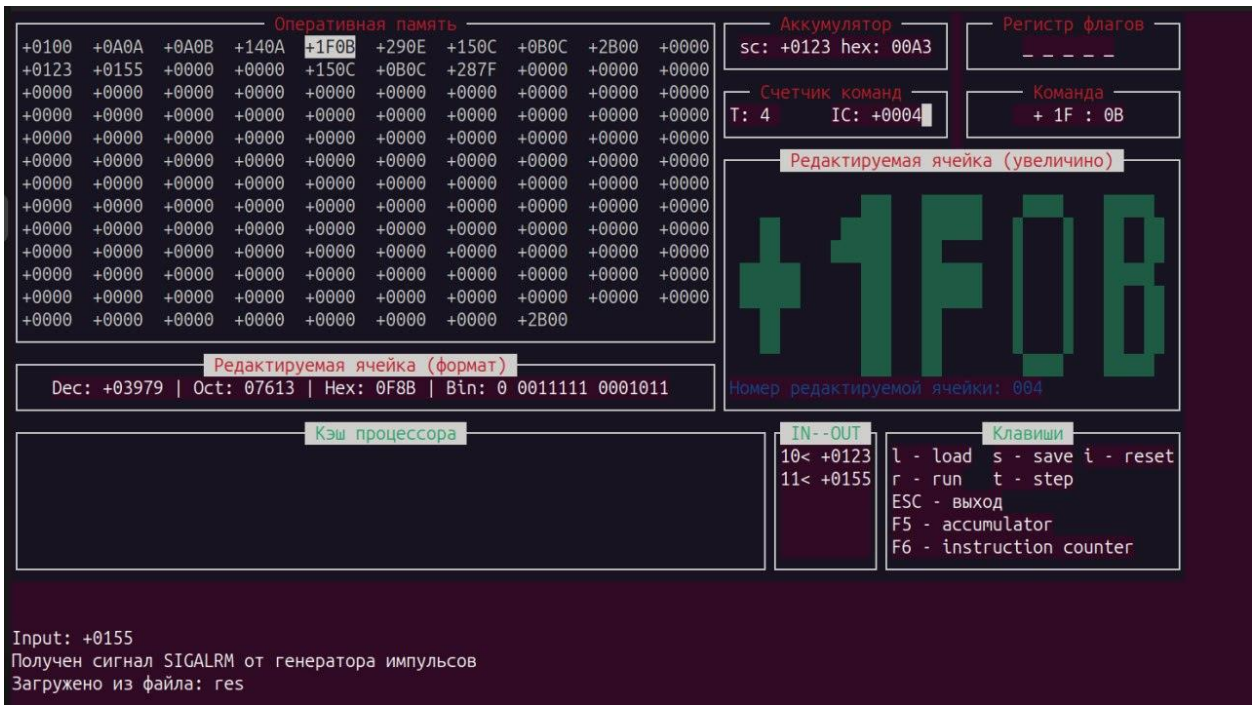


Запускаем программу клавишей r или идём по тактам клавишей t.

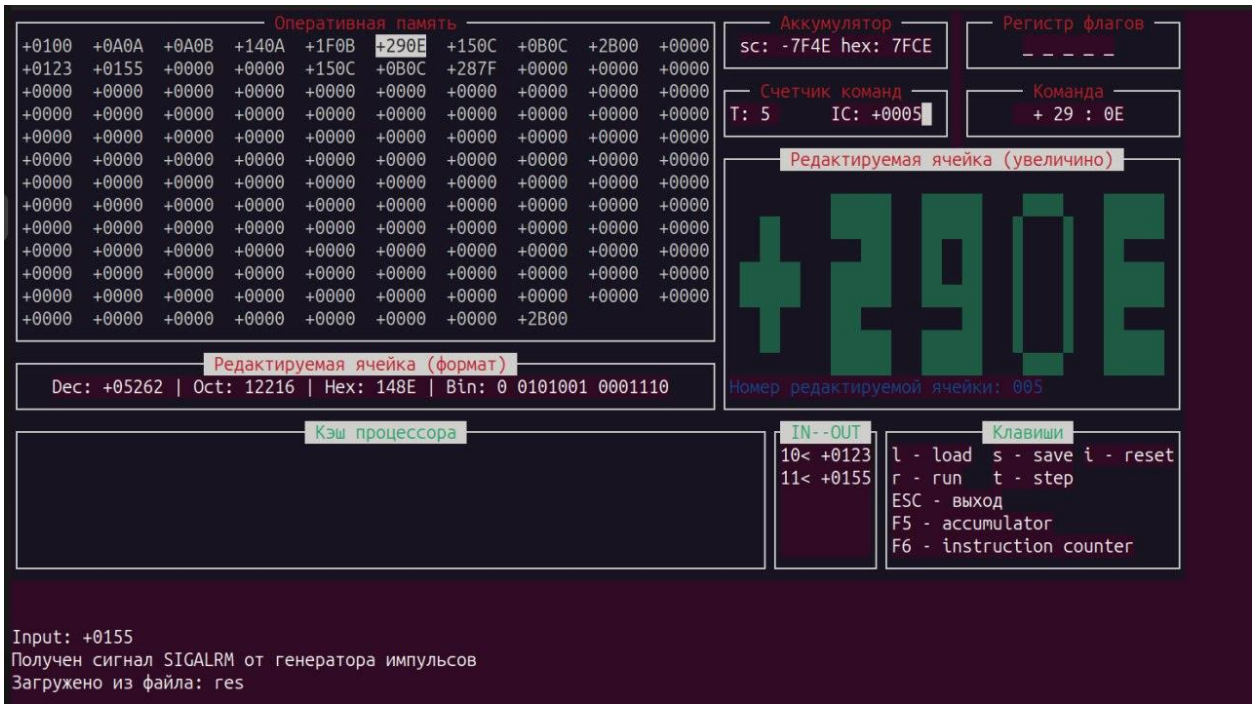
В первой ячейке мы выводим информацию о студенте.

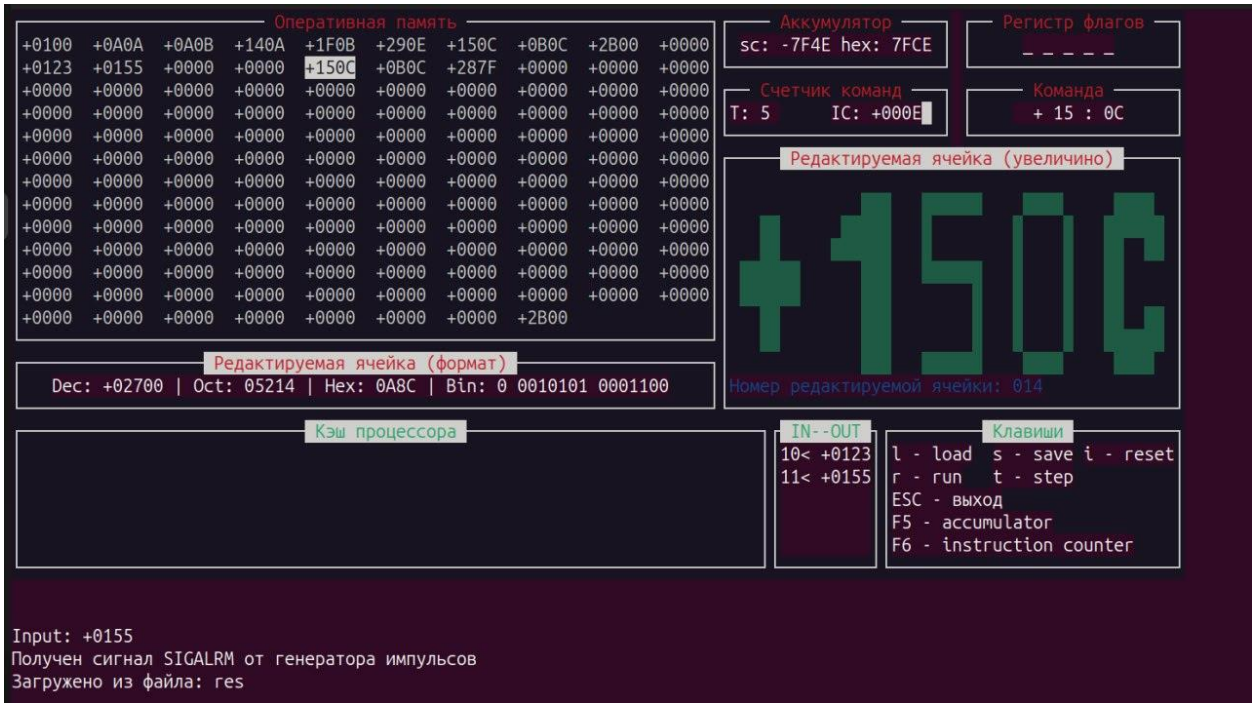


Во второй записываем в 11 ячейку новое значение.

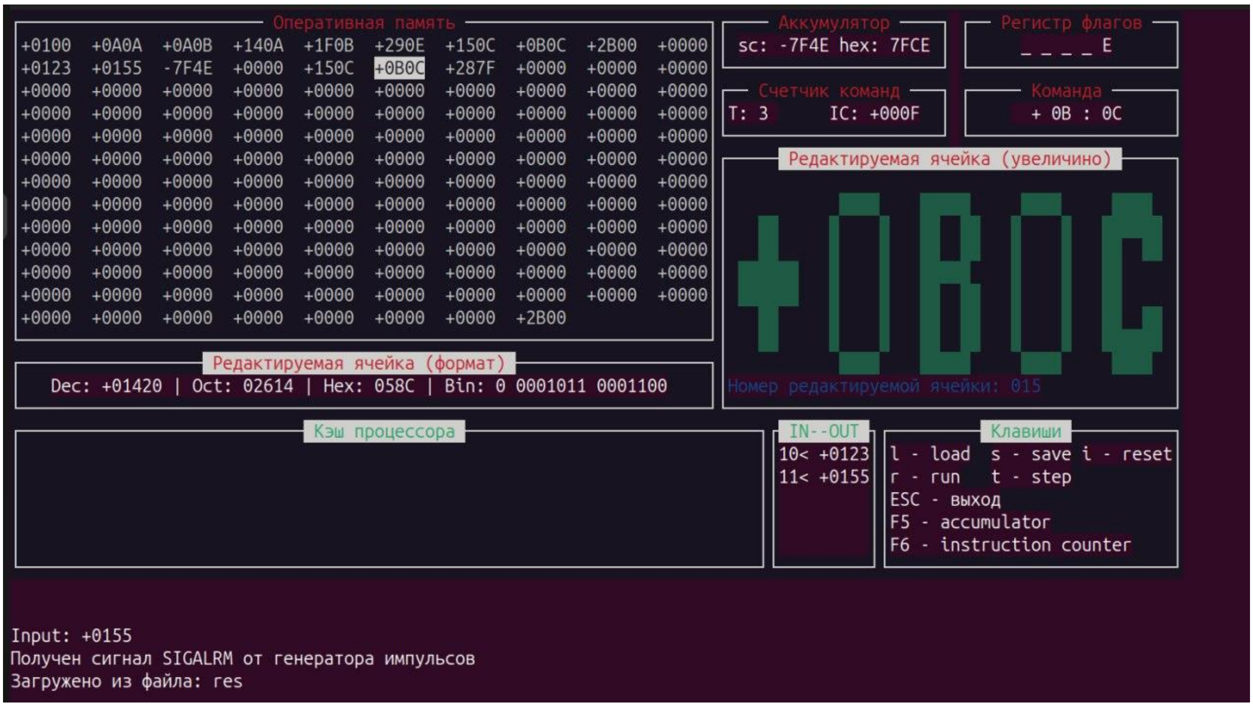


После ввода значения для ещё одной ячейки, ввели значение 11 ячейки в аккумулятор и перешли на ячейку с командой вычитания из аккумулятора значения в 11 ячейке.



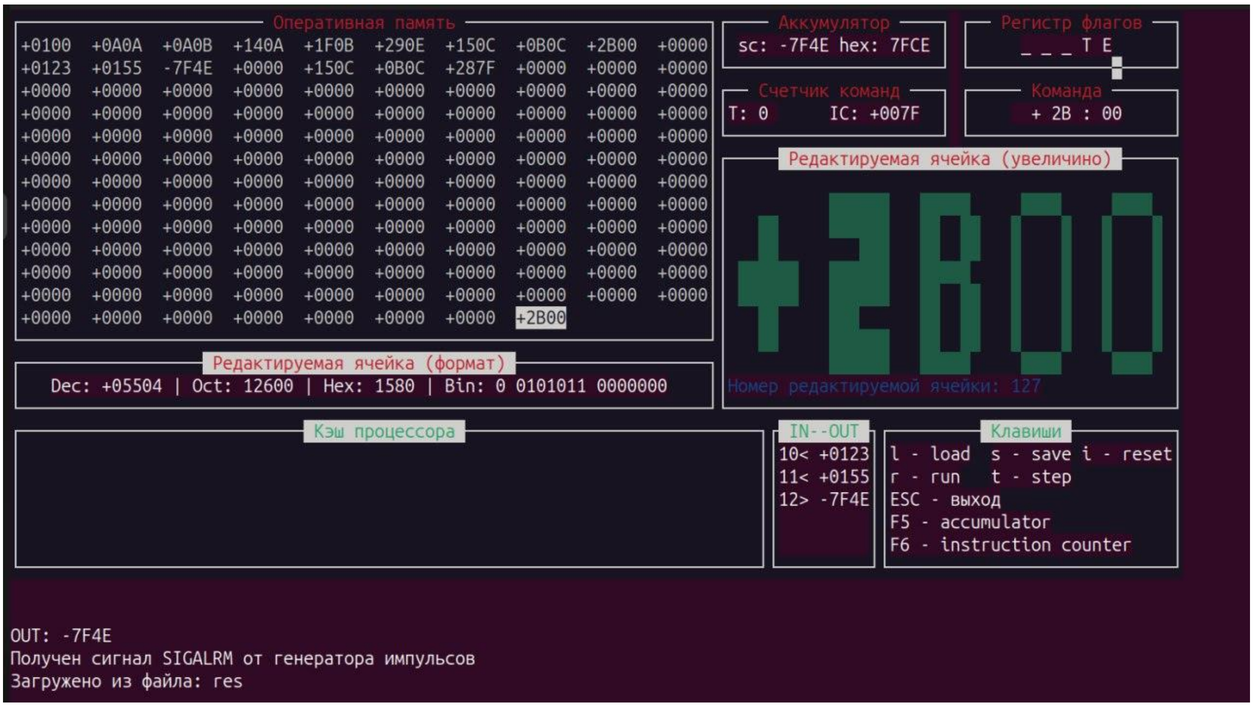


Так поскольку получилось отрицательное значение, на следующей ячейке выполняется прыжок на 14 ячейку



Значение аккумулятора присвоилось 12 ячейке.

Так как команды 7F не существует поднимается флаг инвалидной команды.



На следующей ячейке в OUT выводится значение 12 ячейки.

И ячейка +287F перекидывает нас на последнюю ячейку памяти, где программа останавливается.

4.Вывод.

В рамках данной курсовой работы была реализована модель Simple Computer, выполнение программ которой было осуществлено реализованными функциональными блоками центрального процессора. Для разработки программы был создан транслятор с языка Simple Assembler.

Была разработана простейшая модель вычислительной машины, включающая в себя:
- память
- устройства ввода-вывода
- устройство управления
- арифметико-логическое устройство

5.Листинг программы

MyTermTest.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/ioctl.h>
#include "myBigChars.h"
#include "MyTerm.h"
#include "Enum.h"
#include "mySimpleComputer.h"
#include "myReadkey.h"
#include "CU.h"

int main()
{
   int rows, cols;
   mt_getscreensize(&rows, &cols);
   if (rows < 30 || cols < 120)
   {
      printf("Invalid terminale size!");
      return -1;
   }

   mt_clrscr();
```

```
char *headerRAM = " Оперативная память ";
bc_box(1, 1, 15, 70, WHITE, BLACK, headerRAM, RED, BLACK);

int b;
for (b = 0; b < 128; b++)
{
   printCell(b, 0, WHITE, BLACK);
}

int value;
sc_memoryGet(0, &value);

char *headerDecode = " Редактируемая ячейка (формат) ";
bc_box(16, 1, 3, 70, WHITE, BLACK, headerDecode, RED, WHITE);
printDecodedCommand(value, 17, 5);

sc_regSet(REG_OPERATION_OVERFLOW, 0);
sc_regSet(REG_DIVISION_ZERO, 0);
sc_regSet(REG_MEMORY_OVERFLOW, 0);
sc_regSet(REG_IMPULSE_IGNORE, 1);
sc_regSet(REG_INVALID_COMMAND, 0);
char *headerAccum = " Аккумулятор ";
bc_box(1, 71, 3, 23, WHITE, BLACK, headerAccum, RED, BLACK);
printAccumulator(2, 73);
char *headerFlags = " Регистр флагов ";
bc_box(1, 95, 3, 22, WHITE, BLACK, headerFlags, RED, BLACK);
printFlags(2, 101);
char *headerCount = " Счетчик команд ";
bc_box(4, 71, 3, 23, WHITE, BLACK, headerCount, RED, BLACK);
printCounters(5, 82);
char *headerCommand = " Команда ";
bc_box(4, 95, 3, 22, WHITE, BLACK, headerCommand, RED, BLACK);
printCommand(5, 102);

char *headerBig = " Редактируемая ячейка (увеличино) ";
bc_box(7, 71, 12, 46, WHITE, BLACK, headerBig, RED, WHITE);
printBigCell(0, 9, 72, BLUE, GREEN, BLACK);
```

```
char *headerKeys = " Клавиши ";
bc_box(19, 87, 7, 30, WHITE, BLACK, headerKeys, GREEN, WHITE);
mt_gotoXY(20, 88);
printf("l - load");
mt_gotoXY(21, 88);
printf("r - run");
mt_gotoXY(20, 98);
printf("s - save");
mt_gotoXY(21, 98);
printf("t - step");
mt_gotoXY(20, 107);
printf("i - reset");
mt_gotoXY(22, 88);
printf("ESC - выход");
mt_gotoXY(23, 88);
printf("F5 - accumulator");
mt_gotoXY(24, 88);
printf("F6 - instruction counter");

char *headerCash = " Кэш процессора ";
bc_box(19, 1, 7, 75, WHITE, BLACK, headerCash, GREEN, WHITE);

char *headerInOut = " IN--OUT ";
bc_box(19, 76, 7, 11, WHITE, BLACK, headerInOut, GREEN, WHITE);

mt_gotoXY(80, 1);
init_editor();

enum keys key;

signal(SIGALRM, IRC);
signal(SIGUSR1, IRC);

while (1)
{

   if (interactive)
   {
```

```
if (rk_readkey(&key) == 0)
{
   if (key == ESC)
   {
      mt_setcursorvisible(1);
      break;
   }
   else if (key == ENTER && interactive)
   {
      int new_value;
      int x = (cursor_pos % 10) * 7 + 2;
      int y = (cursor_pos / 10) + 2;
      mt_gotoXY(y, x);
      printf("\033[5X");
      fflush(stdout);
      mt_setcursorvisible(1);

      if (rk_readvalue(&new_value, 10) == 0)
      {
         sc_memorySet(cursor_pos, new_value);
         printCell(cursor_pos, cursor_pos, WHITE, BLACK);
         printDecodedCommand(new_value, 17, 5);
         printBigCell(cursor_pos, 9, 72, BLUE, GREEN, BLACK);
         printCounters(5, 82);
         printCommand(5, 102);
         fflush(stdout);
      }
      else
      {
         printCell(cursor_pos, cursor_pos, WHITE, BLACK);
         fflush(stdout);
      }
      mt_setcursorvisible(0);
      fflush(stdout);
   }
   else if (key == i)
   {
      impulse_counter = 0;
```

```
    IRC(SIGUSR1);
    cursor_pos = 0;
    program_pos = 0;
    printFlags(2, 101);
    fflush(stdout);
    for (b = 0; b < MEMORY_SIZE; b++)
    {
      printCell(b, 0, WHITE, BLACK);
      fflush(stdout);
    }
    printCounters(5, 82);
    printAccumulator(2, 73);
    printCommand(5, 102);
    printDecodedCommand(0, 17, 5);
    printBigCell(0, 9, 72, BLUE, GREEN, BLACK);
    fflush(stdout);
}
else if (key == r)
{
    impulse_counter = 0;
    update_cell_display(cursor_pos, program_pos);
    sc_regSet(REG_IMPULSE_IGNORE, 0);
    interactive = 0;
    alarm_received = false;
    alarm(2);
    printCounters(5, 82);
    printFlags(2, 101);
    fflush(stdout);
}
else if (key == s && interactive)
{
    rk_mytermsave();
}
else if (key == l && interactive)
{
    rk_mytermrestore();
    cursor_pos = 0;
    program_pos = 0;
```

```
    sc_regSet(REG_IMPULSE_IGNORE, 1);
    for (b = 0; b < 128; b++)
    {
        sc_memorySet(b, MEMORY[b]);
        printCell(b, 0, WHITE, BLACK);
        fflush(stdout);
    }
    printCommand(5, 102);
    printDecodedCommand(MEMORY[0], 17, 5);
    printBigCell(0, 9, 72, BLUE, GREEN, BLACK);
}
else if (key == t)
{

    if (!sc_regGet(REG_IMPULSE_IGNORE, &value))
    {
        CU();
        fflush(stdout);

        for (b = 0; b < 128; b++)
        {
            printCell(b, icounter, WHITE, BLACK);
        }
        printAccumulator(2, 73);
        printFlags(2, 101);
        printCounters(5, 82);
        printCommand(5, 102);
        program_pos = icounter;
        printDecodedCommand(MEMORY[icounter], 17, 5);
        printBigCell(icounter, 9, 72, BLUE, GREEN, BLACK);
        fflush(stdout);
    }
}

else if (key == F5 && interactive)
{
    int new_ac;
    mt_gotoXY(2, 77);
```

```
    printf("\033[5X");
    fflush(stdout);
    mt_setcursorvisible(1);

    if (rk_readvalue(&new_ac, 10) == 0)
    {
        sc_accumulatorSet(new_ac);
        fflush(stdout);
    }
    mt_setcursorvisible(0);
    printAccumulator(2, 73);
    fflush(stdout);
}
else if (key == F6 && interactive)
{
    int new_count;
    mt_gotoXY(5, 87);
    printf("\033[4X");
    fflush(stdout);
    char input[5] = {0};
    int pos = 0;

    while (1)
    {
        char c;
        if (read(STDIN_FILENO, &c, 1) != 1)
        {
            break;
        }

        c = toupper(c);
        mt_gotoXY(5, 87 + pos);
        printf("%c", c);
        fflush(stdout);

        if (c == '\n' || c == '\r')
        {
            if (pos > 0)
```

```
    {
        input[pos] = '\0';

        char *endptr;
        unsigned long hex_value = strtoul(input, &endptr, 16);

        if (*endptr != '\0')
        {
            break;
        }
        new_count = (int)hex_value;
        if (new_count > 128)
        {
            sc_regSet(REG_MEMORY_OVERFLOW, 1);
        }
        else
        {
            sc_regSet(REG_MEMORY_OVERFLOW, 0);
        }
        sc_icounterSet(new_count);
        impulse_counter = 0;
        printFlags(2, 101);
        printCounters(5, 82);
        printCommand(5, 102);
        fflush(stdout);
        break;
    }
    break;
}
else if (c == 127 || c == 8)
{
    if (pos > 0)
    {
        pos--;
        mt_gotoXY(5, 87 + pos);
        printf(" ");
        fflush(stdout);
    }
```

```
            }
            else if (pos < 4)
            {
               if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F'))
               {
                  input[pos++] = c;
               }
            }
         }
      }
      else if (interactive)
      {
         move_cursor(key);
      }
   }
}

if (!interactive)
{
   if (alarm_received)
   {
      alarm_received = false;

      if (!sc_regGet(REG_IMPULSE_IGNORE, &value))
      {
         CU();
         fflush(stdout);
         for (b = 0; b < 128; b++)
         {
            printCell(b, icounter, WHITE, BLACK);
         }
         printAccumulator(2, 73);
         printFlags(2, 101);
         printCounters(5, 82);
         printCommand(5, 102);
         printDecodedCommand(MEMORY[icounter], 17, 5);
         program_pos = icounter;
         printBigCell(icounter, 9, 72, BLUE, GREEN, BLACK);
```

```
            fflush(stdout);
        }

        if (!sc_regGet(REG_IMPULSE_IGNORE, &value))
        {
            alarm(2);
        }
        else
        {
            interactive = 1;
            rk_mytermregime(1, 0, 1, 0, 1);
            update_cell_display(program_pos, cursor_pos);
        }
      }

      // if (reset_received)
      // {
      //     reset_received = false;
      //     interactive = 1;
      //     rk_mytermregime(1, 0, 1, 0, 1);
      //     update_cell_display(program_pos, cursor_pos);
      // }
    }
  }

  if (rk_mytermregime(0, 0, 1, 1, 0) != 0)
    return -1;
  mt_gotoXY(30, 1);

  return 0;
}
```

CU.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.h>
```

```
#include <sys/ioctl.h>
#include "Enum.h"
#include "mySimpleComputer.h"
#include "myReadkey.h"
#include "MyTerm.h"
#include "CU.h"

#define VALUE_MASK 0x3FFF

int interactive = 1;
int impulse_counter = 0;
volatile sig_atomic_t alarm_received = false;
// volatile sig_atomic_t reset_received = false;

void CU()
{
   impulse_counter++;
   int value;
   int command;
   int operand;

   if (icounter > 127 || icounter < 0)
   {
      sc_regSet(REG_MEMORY_OVERFLOW, 1);
      sc_regSet(REG_IMPULSE_IGNORE, 1);
      icounter = 0;
      return;
   }

   if (sc_memoryGet(icounter, &value) != 0)
   {
      sc_regSet(REG_IMPULSE_IGNORE, 1);
      return;
   }

   // int sign = (value >> 14) & 1;
   int magnitude = value & 0x3FFF;
   command = (magnitude >> 7) & 0x7F;
```

```
operand = magnitude & 0x7F;

if (sc_commandValidate(command) != 0)
{
   sc_regSet(REG_INVALID_COMMAND, 1);
   sc_regSet(REG_IMPULSE_IGNORE, 1);
   interactive = 1;
   return;
}

if (operand >= MEMORY_SIZE)
{
   sc_regSet(REG_INVALID_COMMAND, 1);
   sc_regSet(REG_IMPULSE_IGNORE, 1);
   return;
}

if (command >= 0x1E && command <= 0x21)
{
   if (ALU(command, operand) != 0)
   {
      sc_regSet(REG_IMPULSE_IGNORE, 1);
      return;
   }
}

switch (command)
{
case 0x00: // NOP
   break;

case 0x01: // CPUINFO
{
   mt_gotoXY(28, 0);
   printf("Грязин Алексей Владимирович, Дорогин Никита Сергеевич, ИП-312");
   fflush(stdout);
   break;
}
```

```
case 0x0A: // READ
{
   mt_gotoXY(28, 0);
   printf("                                                                  ");
   mt_gotoXY(28, 0);
   printf("Input: ");
   fflush(stdout);
   mt_gotoXY(28, 8);
   int input_value;
   rk_readvalue(&input_value, 10);
   // sc_commandDecode(input_value, &input_sign, &input_command, &input_operand);
   fflush(stdout);
   sc_memorySet(operand, input_value);
   printTerm(operand, 1, 20, 77);
   mt_gotoXY(28, 0);
   printf("                                                                  ");
   fflush(stdout);
   break;
}

case 0x0B: // WRITE
{
   sc_memoryGet(operand, &value);
   mt_gotoXY(28, 0);
   printf("                                                                   ");

   int sign = (value >> 14) & 1;
   int magnitude = value & 0x3FFF;
   int command = (magnitude >> 7) & 0x7F;
   int operand_out = magnitude & 0x7F;
   mt_gotoXY(28, 0);
   printf("OUT: %c%02X%02X", sign ? '-' : '+', command, operand_out);
   printTerm(operand, 0, 20, 77);
   fflush(stdout);
   break;
}
```

```
case 0x14: // LOAD
{
   sc_memoryGet(operand, &value);
   accumulator = value;
   break;
}
case 0x15: // STORE
{
   sc_memorySet(operand, accumulator);
   break;
}
case 0x28: // JUMP
{
   if (operand == icounter)
   { // Защита от самоперехода
      sc_regSet(REG_INVALID_COMMAND, 1);
      sc_regSet(REG_IMPULSE_IGNORE, 1);
      return;
   }
   icounter = operand;
   break;
}
case 0x29: // JNEG
{
   if (accumulator & 0x4000)
   {
      icounter = operand;
   }
   else
   {
      icounter++;
   }
   break;
}
case 0x2A: // JZ
{
   if ((accumulator & 0x3FFF) == 0)
   {
```

```
        icounter = operand;
    }
    break;
}
case 0x2B: // HALT
{
    sc_regSet(REG_IMPULSE_IGNORE, 1);
    interactive = 1;
    icounter--;
    impulse_counter--;
    break;
}
// Пользовательские команды
case 0x38: // JC (56)
{
    int flag;
    sc_regGet(REG_OPERATION_OVERFLOW, &flag);
    if (flag)
    {
        icounter = operand;
    }
    else
    {
        icounter++;
    }
    break;
}
case 0x39: // JNC (57)
{
    int flagt;
    sc_regGet(REG_OPERATION_OVERFLOW, &flagt);
    if (!flagt)
    {
        icounter = operand;
    }
    else
    {
        icounter++;
```

```
    }
    break;
  }
  case 0x3A: // JP (58)
  {
    if ((accumulator & 0x0001) == 0)
    {
      icounter = operand;
    }
    else
    {
      icounter++;
    }
    break;
  }
  case 0x3B: // JNP (59)
  {
    if (accumulator & 0x0001)
    {
      icounter = operand;
    }
    else
    {
      icounter++;
    }
    break;
  }
  default:
    break;
  }

  if (!(command == 0x28 || command == 0x29 || command == 0x2A || command == 0x38 || command
== 0x39 || command == 0x3A || command == 0x3B))
  {
    icounter++;
  }

  return;
```

```
}

int ALU(int command, int operand)
{
   int mem_value;
   if (sc_memoryGet(operand, &mem_value) != 0)
   {
      return -1;
   }

   int result;

   switch (command)
   {
   case 0x1E: // ADD
      result = accumulator + mem_value;
      break;
   case 0x1F: // SUB
      result = accumulator - mem_value;
      break;
   case 0x20: // DIVIDE
      if (mem_value == 0)
      {
         sc_regSet(REG_DIVISION_ZERO, 1);
         return -1;
      }
      result = accumulator / mem_value;
      break;
   case 0x21: // MUL
      result = accumulator * mem_value;
      break;
   default:
      return -1;
   }

   if (result > MAX_VALUE || result < MIN_VALUE)
   {
      sc_regSet(REG_OPERATION_OVERFLOW, 1);
```

```
        int sign = (result < 0) ? -1 : 1;

        unsigned int abs_value = (unsigned int)(sign * result);
        abs_value &= VALUE_MASK;

        result = sign * (int)abs_value;
    }

    accumulator = result;
    return 0;
}

void IRC(int signum)
{
    mt_gotoXY(29, 0);
    mt_delline();
    switch (signum)
    {
    case SIGALRM:
        alarm_received = true;
        printf("Получен сигнал SIGALRM от генератора импульсов\n");
        fflush(stdout);
        break;

    case SIGUSR1:
        // reset_received = true;
        sc_regSet(REG_OPERATION_OVERFLOW, 0);
        sc_regSet(REG_DIVISION_ZERO, 0);
        sc_regSet(REG_MEMORY_OVERFLOW, 0);
        sc_regSet(REG_IMPULSE_IGNORE, 1);
        sc_regSet(REG_INVALID_COMMAND, 0);
        sc_accumulatorSet(0);
        sc_icounterSet(0);
        interactive = 1;
        int b;
        for (b = 0; b < MEMORY_SIZE; b++)
        {
```

```
            sc_memorySet(b, 0);
        }
        printf("Получен сигнал SIGUSR1 от Reset\n");
        fflush(stdout);
        break;

    default:
        printf("Получен неизвестный сигнал: %d\n", signum);
        fflush(stdout);
        break;
    }
}
```

SimpleAssembler.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <libgen.h>
#include <limits.h>

#define MEM_SIZE 128
#define MAX_PATH 256
#define MIN_VALUE -16383
#define MAX_VALUE 16383

typedef struct
{
    char *name;
    int code;
} Command;

Command commands[] = {
    {"NOP", 0x00}, {"CPUINFO", 0x01}, {"READ", 0x0A}, {"WRITE", 0x0B}, {"LOAD", 0x14}, {"STORE", 0x15},
{"ADD", 0x1E}, {"SUB", 0x1F}, {"DIVIDE", 0x20}, {"MUL", 0x21}, {"JUMP", 0x28}, {"JNEG", 0x29}, {"JZ",
0x2A}, {"HALT", 0x2B}, {"NOT", 0x33}, {"AND", 0x34}, {"OR", 0x35}, {"XOR", 0x36}, {"JNS", 0x37}, {"JC",
0x38}, {"JNC", 0x39}, {"JP", 0x3A}, {"JNP", 0x3B}, {"CHL", 0x3C}, {"SHR", 0x3D}, {"RCL", 0x3E}, {"RCR",
0x3F}, {"NEG", 0x40}, {"ADDC", 0x41}, {"SUBC", 0x42}, {"LOGLC", 0x43}, {"LOGRC", 0x44}, {"RCCL", 0x45},
{"RCCR", 0x46}, {"MOVA", 0x47}, {"MOVR", 0x48}, {"MOVCA", 0x49}, {"MOVCR", 0x4A}, {"ADDC", 0x4B},
{"SUBC", 0x4C}};
```

```
int memory[MEM_SIZE] = {0};

int find_command_code(const char *name)
{
    for (size_t i = 0; i < sizeof(commands) / sizeof(commands[0]); i++)
    {
        if (strcmp(commands[i].name, name) == 0)
        {
            return commands[i].code;
        }
    }
    return -1;
}

int parse_number(const char *str, int *success)
{
    int sign = 1;
    int value = 0;
    *success = 1;

    if (*str == '+')
    {
        str++;
    }
    else if (*str == '-')
    {
        sign = -1;
        str++;
    }

    if (!isdigit(*str))
    {
        *success = 0;
        return 0;
    }

    while (*str && isdigit(*str))
```

```
    {
        value = value * 10 + (*str - '0');
        str++;
    }

    if (*str != '\0' && !isspace(*str) && *str != ';')
    {
        *success = 0;
    }

    int result = sign * value;

    if (result < MIN_VALUE || result > MAX_VALUE)
    {
        *success = 0;
    }

    return result;
}

void process_line(char *line, int line_num)
{
    char *comment = strchr(line, ';');
    if (comment)
        *comment = '\0';

    char *end = line + strlen(line) - 1;
    while (end >= line && isspace(*end))
        end--;
    *(end + 1) = '\0';

    if (strlen(line) == 0)
        return;

    char *token = strtok(line, " \t");
    if (!token)
        return;
```

```
    int address;
    if (sscanf(token, "%d", &address) != 1)
    {
        fprintf(stderr, "Line %d: Invalid address format\n", line_num);
        return;
    }

    if (address < 0 || address >= MEM_SIZE)
    {
        fprintf(stderr, "Line %d: Address %d out of range (0-%d)\n", line_num, address, MEM_SIZE - 1);
        return;
    }

    token = strtok(NULL, " \t");
    if (!token)
    {
        fprintf(stderr, "Line %d: Missing command or '='\n", line_num);
        memory[address] = 0;
        return;
    }

    if (strcmp(token, "=") == 0)
    {
        token = strtok(NULL, " \t");
        if (!token)
        {
            fprintf(stderr, "Line %d: Missing value after '='\n", line_num);
            memory[address] = 0;
            return;
        }

        int success;
        int value = parse_number(token, &success);
        if (!success)
        {
            fprintf(stderr, "Line %d: Invalid number format or out of range '%s' (must be between %d and
%d)\n", line_num, token, MIN_VALUE, MAX_VALUE);
            memory[address] = 0;
```

```
            return;
        }

        memory[address] = value;
    }
    else
    {
        int command_code = find_command_code(token);
        if (command_code == -1)
        {
            fprintf(stderr, "Line %d: Unknown command '%s'\n", line_num, token);
            memory[address] = 0;
            return;
        }

        token = strtok(NULL, " \t");
        if (!token)
        {
            fprintf(stderr, "Line %d: Missing operand for command\n", line_num);
            memory[address] = 0;
            return;
        }

        int success;
        int operand = parse_number(token, &success);
        if (!success)
        {
            fprintf(stderr, "Line %d: Invalid operand format or out of range '%s' (must be between %d and
%d)\n", line_num, token, MIN_VALUE, MAX_VALUE);
            memory[address] = 0;
            return;
        }

        if (operand < 0 || operand >= MEM_SIZE)
        {
            fprintf(stderr, "Line %d: Operand %d out of range (0-%d)\n", line_num, operand, MEM_SIZE - 1);
            memory[address] = 0;
            return;
```

```
        }

        memory[address] = (command_code << 7) | (operand & 0x7F);
    }
}

void write_binary(const char *filename)
{
    FILE *f = fopen(filename, "wb");
    if (!f)
    {
        perror("Error opening output file");
        exit(1);
    }

    for (int i = 0; i < MEM_SIZE; i++)
    {
        int word = memory[i];
        fwrite(&word, sizeof(int), 1, f);
    }

    fclose(f);
}

int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        printf("Usage: %s input.sa output.bin\n", argv[0]);
        return 1;
    }

    memset(memory, 0, sizeof(memory));

    FILE *f = fopen(argv[1], "r");
    if (!f)
    {
        perror("Error opening input file");
```

```
        return 1;
    }

    char line[256];
    int line_num = 0;
    while (fgets(line, sizeof(line), f))
    {
        line_num++;
        process_line(line, line_num);
    }

    fclose(f);

    char input_path[MAX_PATH];
    char output_path[MAX_PATH];
    char *base;

    realpath(argv[1], input_path);
    base = basename(input_path);

    char *dot = strrchr(base, '.');
    if (dot && strcmp(dot, ".sa") == 0)
    {
        *dot = '\0';
    }

    snprintf(output_path, MAX_PATH, "../console/saves/%s.bin", base);

    char cmd[MAX_PATH + 10];
    snprintf(cmd, sizeof(cmd), "mkdir -p ../console/saves");
    system(cmd);

    write_binary(argv[2]);
    printf("Successfully compiled to %s\n", argv[2]);

    return 0;
}
```

myReadKey.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <string.h>
#include <ctype.h>
#include <poll.h>
#include "myReadkey.h"
#include "mySimpleComputer.h"
#include "Enum.h"
#include "MyTerm.h"
#include "CU.h"
#include <signal.h>
#include <stdbool.h>

static struct termios original_termios;

int rk_readkey(enum keys *address)
{
  if (rk_mytermregime(1, 0, 1, 0, 0) != 0)
    return -1;
  tcflush(STDIN_FILENO, TCIFLUSH);

  char buf[8] = {0};
  int bytes_read = read(STDIN_FILENO, buf, sizeof(buf));

  if (bytes_read <= 0)
  {
    return -1;
  }

  if (bytes_read == 1)
  {
    switch (buf[0])
    {
    case 'l':
      *address = l;
```

```
            break;
        case 's':
            *address = s;
            break;
        case 'i':
            *address = i;
            break;
        case 'r':
            *address = r;
            break;
        case 't':
            *address = t;
            break;
        case '\x1B':
            *address = ESC;
            break;
        case '\n':
        case '\r':
            *address = ENTER;
            break;
        default:
            return -1;
        }
    }
    else if (bytes_read >= 3)
    {

        if (strncmp(buf, "\x1B[15~", 5) == 0)
        {
            *address = F5;
        }
        else if (strncmp(buf, "\x1B[17~", 5) == 0)
        {
            *address = F6;
        }
        else if (strncmp(buf, "\x1B[A", 3) == 0)
        {
            *address = UP;
```

```
        }
        else if (strncmp(buf, "\x1B[B", 3) == 0)
        {
            *address = DOWN;
        }
        else if (strncmp(buf, "\x1B[C", 3) == 0)
        {
            *address = RIGHT;
        }
        else if (strncmp(buf, "\x1B[D", 3) == 0)
        {
            *address = LEFT;
        }
        else
        {
            return -1;
        }
    }
    else
    {
        return -1;
    }
    return 0;
}

int rk_mytermsave(void)
{
    char filename[256] = {0};
    char full_path[512] = {0};
    struct termios old_term, new_term;

    if (tcgetattr(STDIN_FILENO, &original_termios) != 0)
    {
        return -1;
    }

    old_term = original_termios;
    new_term = old_term;
```

```
new_term.c_lflag |= (ICANON | ECHO);
if (tcsetattr(STDIN_FILENO, TCSANOW, &new_term) != 0)
{
   return -1;
}

mt_gotoXY(30, 0);
mt_delline();
printf("Введите имя файла для сохранения: ");
fflush(stdout);

if (fgets(filename, sizeof(filename), stdin) == NULL)
{
   tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
   rk_mytermregime(1, 0, 1, 0, 1);
   return -1;
}
// alarm(5);
// pause();
IRC(SIGALRM);
filename[strcspn(filename, "\n")] = '\0';
snprintf(full_path, sizeof(full_path), "saves/%s.bin", filename);

FILE *file = fopen(full_path, "wb");
if (file == NULL)
{
   mt_gotoXY(30, 0);
   mt_delline();
   printf("Ошибка открытия файла для записи!");
   fflush(stdout);
   tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
   rk_mytermregime(1, 0, 1, 0, 1);
   return -1;
}

fwrite(MEMORY, sizeof(int), MEMORY_SIZE, file);
fclose(file);
tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
```

```
    mt_gotoXY(30, 0);
    mt_delline();
    printf("Сохранено в файл: %s", filename);
    fflush(stdout);
    rk_mytermregime(1, 0, 1, 0, 1);
    return 0;
}

int rk_mytermrestore(void)
{
    char filename[256] = {0};
    char full_path[512] = {0};
    struct termios old_term, new_term;

    if (tcgetattr(STDIN_FILENO, &old_term) != 0)
    {
        return -1;
    }
    new_term = old_term;
    new_term.c_lflag |= (ICANON | ECHO);

    if (tcsetattr(STDIN_FILENO, TCSANOW, &new_term) != 0)
    {
        return -1;
    }

    mt_gotoXY(30, 0);
    mt_delline();
    printf("Введите имя файла для загрузки: ");
    fflush(stdout);

    if (fgets(filename, sizeof(filename), stdin) == NULL)
    {
        tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
        return -1;
    }
    // alarm(5);
    // pause();
```

```
    filename[strcspn(filename, "\n")] = '\0';
    snprintf(full_path, sizeof(full_path), "saves/%s.bin", filename);

    FILE *file = fopen(full_path, "rb");
    if (file == NULL)
    {
        mt_gotoXY(30, 0);
        mt_delline();
        printf("Файл %s не найден!", filename);
        fflush(stdout);
        tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
        return -1;
    }

    if (fread(MEMORY, sizeof(int), MEMORY_SIZE, file) != MEMORY_SIZE)
    {
        mt_gotoXY(30, 0);
        mt_delline();
        printf("Ошибка чтения данных!");
        fclose(file);
        tcsetattr(STDIN_FILENO, TCSANOW, &old_term);
        return -1;
    }
    fclose(file);

    if (tcsetattr(STDIN_FILENO, TCSANOW, &original_termios) != 0)
    {
        return -1;
    }
    mt_gotoXY(30, 0);
    mt_delline();
    printf("Загружено из файла: %s", filename);
    fflush(stdout);
    rk_mytermregime(1, 0, 1, 0, 1);
    return 0;
}

int rk_mytermregime(int regime, int vtime, int vmin, int echo, int sigint)
```

```
{
   struct termios new_termios;

   if (tcgetattr(STDIN_FILENO, &new_termios) != 0)
   {
      return -1;
   }
   if (regime)
   {
      new_termios.c_lflag &= ~(ICANON | ECHO | (sigint ? 0 : ISIG));
      new_termios.c_cc[VTIME] = vtime;
      new_termios.c_cc[VMIN] = vmin;
   }
   else
   {
      new_termios.c_lflag |= (ICANON | (echo ? ECHO : 0) | (sigint ? ISIG : 0));
   }
   return tcsetattr(STDIN_FILENO, TCSANOW, &new_termios);
}

int rk_readvalue(int *value, int timeout)
{
   char input[6] = {0};
   int pos = 0;
   struct pollfd fds = {STDIN_FILENO, POLLIN, 0};

   if (rk_mytermregime(1, 0, 1, 1, 0) != 0)
   {
      return -1;
   }

   while (1)
   {
      int poll_result = poll(&fds, 1, timeout * 1000);
      if (poll_result <= 0)
         break;

      char c;
```

```
if (read(STDIN_FILENO, &c, 1) != 1)
   break;

c = toupper(c);

if (c == '\n' || c == '\r')
{
   if (pos > 0)
   {
      input[pos] = '\0';
      int is_negative = (input[0] == '-');
      char *endptr;
      unsigned long hex_value = strtoul(is_negative ? input + 1 : input, &endptr, 16);

      if (*endptr != '\0' || hex_value > 0x7FFF)
         break;

      unsigned int command = (hex_value >> 8) & 0x7F;
      unsigned int operand = hex_value & 0x7F;
      unsigned int full_value = (is_negative << 14) | (command << 7) | operand;

      if (is_negative)
      {
         full_value = (full_value - 1) & 0x7FFF;
         full_value = ~full_value & 0x7FFF;
         *value = -(full_value);
      }
      else
      {
         *value = full_value;
      }
      return 0;
   }
   break;
}
else if (c == 127 || c == 8)
{
   if (pos > 0)
```

```
            {
                pos--;
                printf("\b \b");
                fflush(stdout);
            }
        }
        else if (pos < 5)
        {
            if ((pos == 0 && (c == '-' || c == '+')) || (pos > 0 && ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F'))))
            {
                input[pos++] = c;
                printf("%c", c);
                fflush(stdout);
            }
        }
    }

    tcflush(STDIN_FILENO, TCIFLUSH);
    rk_mytermregime(1, 0, 1, 0, 0);
    return -1;
}
```

6.Список литературы

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

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

3. Организация ЭВМ. 5-е изд. / К. Хамахер, З. Вранешич, С. Заки. – СПб.: Питер; Киев: Издательская группа BHV, 2003.

4. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: учебник для ВУЗов. – СПб.: Питер, 2004.