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-241
«__» мая 2024 г. ____________ / Кулик П.Е. /

Проверил:
преподаватель
«__» мая 2024 г. ____________ / Майданов Ю.С. /

Оценка «_____________»

Новосибирск 2024

Оглавление

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;

➢ Доработать модель Simple Computer – реализовать алгоритм работы блока «L1-кэш команд и данных» и модифицировать работу контроллера оперативной памяти и обработчика прерываний таким образом, чтобы учитывался простой процессора при прямом доступе к оперативной памяти;

➢ Разработать транслятор с языка Simple Basic. Итог работы транслятора – текстовый файл с программой на языке Simple Basic.

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

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.

Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Пример программы на Simple Basic:

```
10 REM Это комментарий
20 INPUT A
30 INPUT B
40 LET C = A – B
50 IF C < 0 GOTO 20
60 PRINT C
70 END
```

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, *, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

Оформление отчета по курсовой работе

Отчет о курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается диск с разработанным программным обеспечением. 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.

ВЫПОЛНЕНИЕ РАБОТЫ

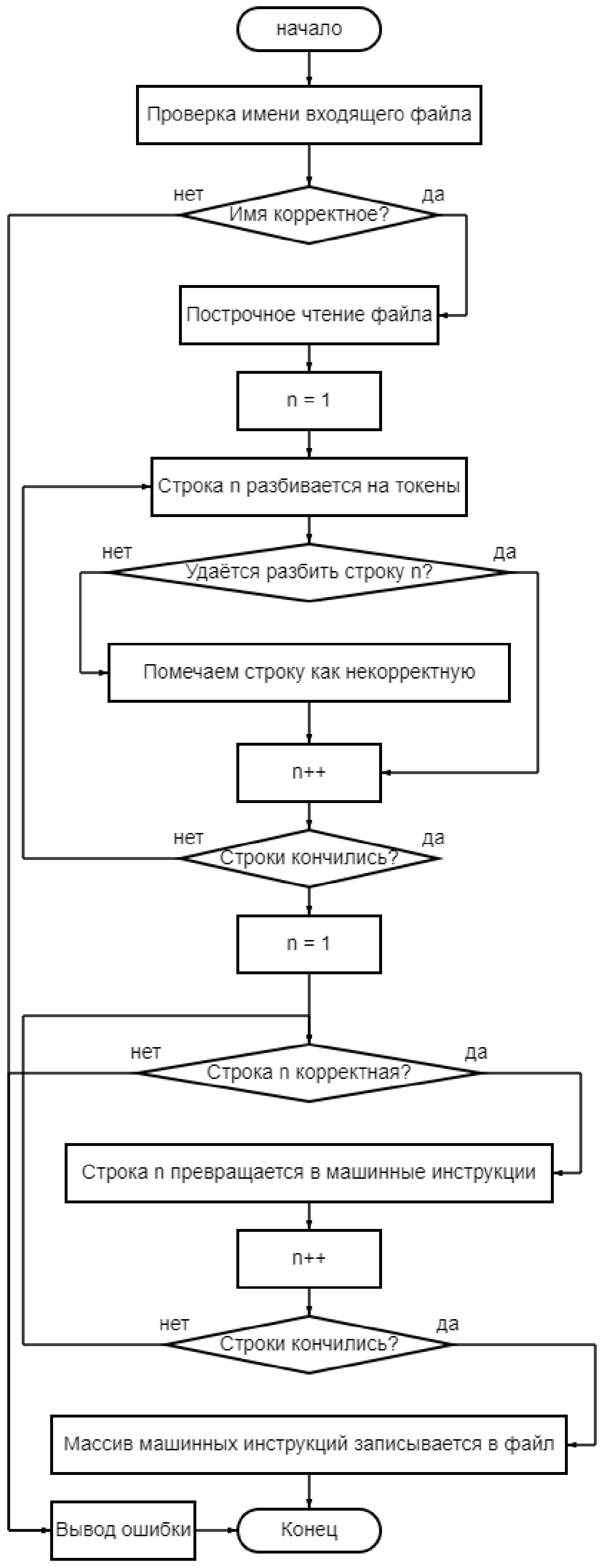
В первую очередь был реализован транслятор с языка simple assembler. Это было самой простой задачей, которая не потребовала поиска дополнительной информации, так как после реализации большей части проекта simple computer уже было понятно, как инструкции simple assembler преобразуются в машинные инструкции.

За этим последовала реализация транслятора с simple basic. Данная задача оказалась менее тривиальной, так как язык simple basic поддерживает арифметические выражения, составленные из множества операций, которые обладают определённым порядком действий. Для решения задачи правильного порядка выполнения пришлось узнать о том, что такое обратная польская нотация и реализовать её. Помимо этого, в языке simple basic могут быть как константы, так и переменные, для поддержки которых требуется построить таблицу символов, чтобы правильно расположить их в памяти и они не пересекались с инструкциями. Проблема размещения данных в памяти была решена очень просто: так как команды располагаются по порядку, начиная с нулевого адреса, то переменные и константы располагаются в обратном порядке, начиная с последнего адреса. Транслятор был написан не с первой попытки, но задача в итоге была решена.

Последним был реализован блок кэша процессора. Предварительно были изучены различные алгоритмы замещения кэша, из которых был выбран LRU кэш как наиболее оптимальный, так как он хорошо работает, учитывая важность часто используемых строк и при этом лёгок в реализации. В моей реализации за каждой строкой кэша закреплена переменная downtime. При каждом обращении к памяти эта переменная обнуляется для той строки, которая была запрошена и инкрементируется для всех остальных строк.

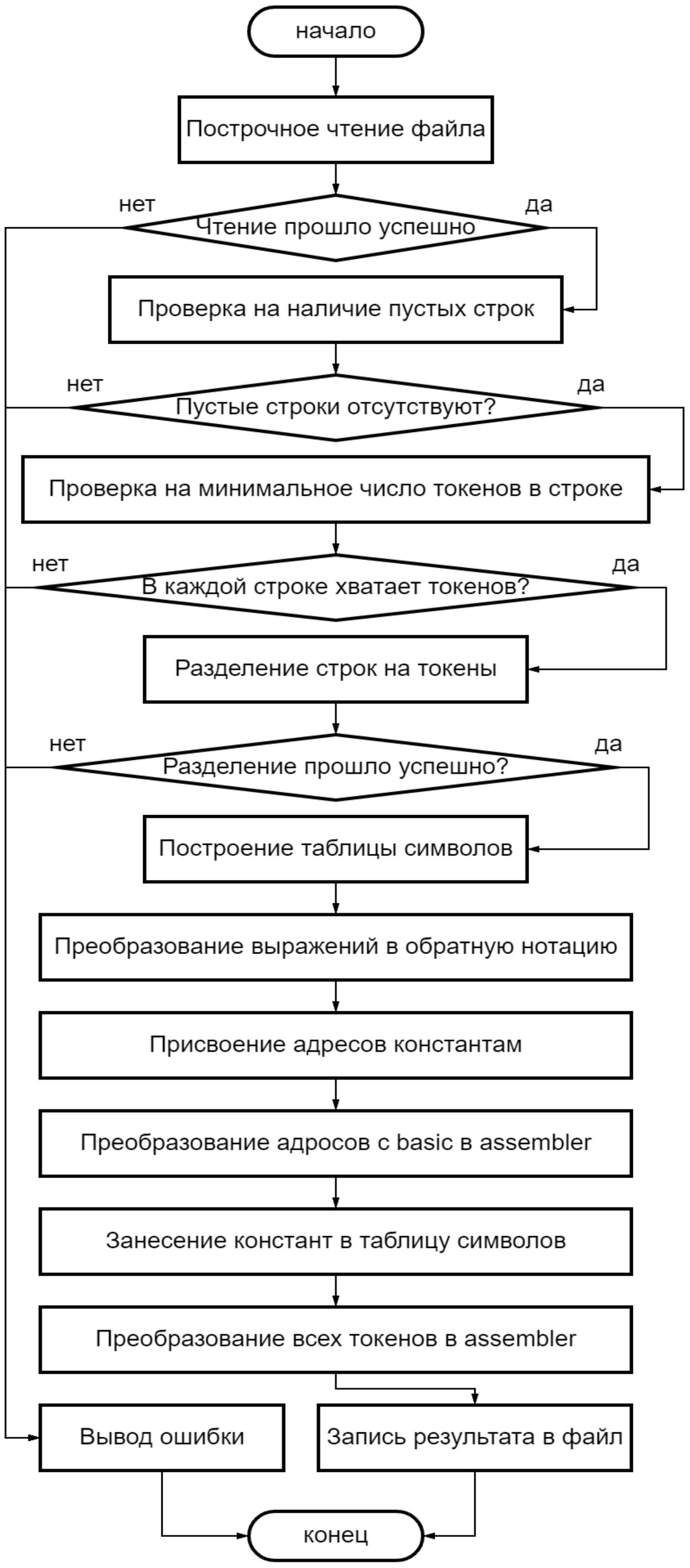
БЛОК-СХЕМЫ АЛГОРИТМОВ

1. Транслятор с Simple Assembler



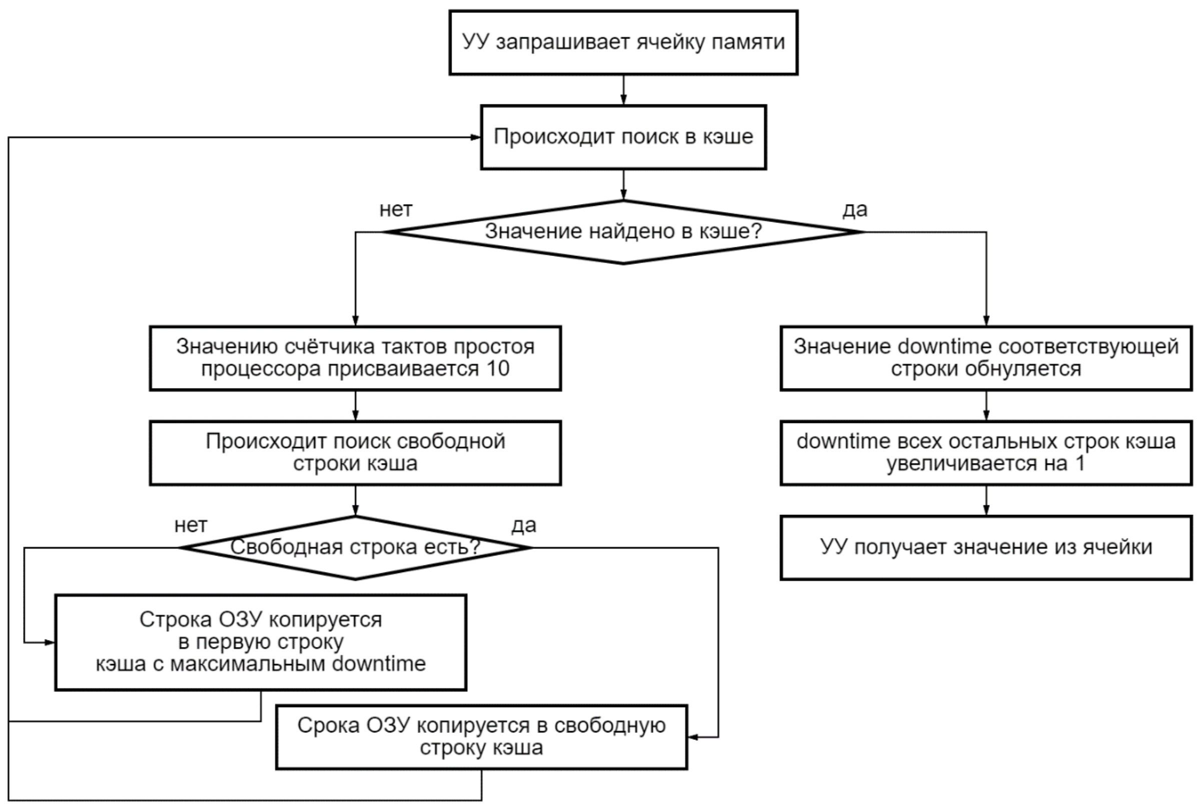
начало
Проверка имени входящего файла
нет
Имя корректное?
да
Построчное чтение файла
n = 1
Строка n разбивается на токены
нет
Удаётся разбить строку n?
да
Помечаем строку как некорректную
n++
нет
Строки кончились?
да
n = 1
нет
Строка n корректная?
да
Строка n превращается в машинные инструкции
n++
нет
Строки кончились?
да
Массив машинных инструкций записывается в файл
Вывод ошибки
Конец

2. Транслятор с Simple Basic



начало
Построчное чтение файла
нет
да
Чтение прошло успешно
Проверка на наличие пустых строк
нет
да
Пустые строки отсутствуют?
Проверка на минимальное число токенов в строке
нет
да
В каждой строке хватает токенов?
Разделение строк на токены
нет
да
Разделение прошло успешно?
Построение таблицы символов
Преобразование выражений в обратную нотацию
Присвоение адресов константам
Преобразование адросов с basic в assembler
Занесение констант в таблицу символов
Преобразование всех токенов в assembler
Вывод ошибки
Запись результата в файл
конец

3. Кэш


УУ запрашивает ячейку памяти
Происходит поиск в кэше
Значение найдено в кэше?
нет
да
Значению счётчика тактов простоя процессора присваивается 10
Происходит поиск свободной строки кэша
Свободная строка есть?
нет
да
Строка ОЗУ копируется в первую строку кэша с максимальным downtime
Срока ОЗУ копируется в свободную строку кэша
Значение downtime соответствующей строки обнуляется
downtime всех остальных строк кэша увеличивается на 1
УУ получает значение из ячейки

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

console

```
alu.c
#include "console.h"
#include <mySimpleComputer.h>

int
ADD_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return (((accumulator_value << 17) + (memory_value << 17)) >> 17) & 0x7FFF;
}

int
SUB_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return (((accumulator_value << 17) + ((~memory_value + 1) << 17)) >> 17)
         & 0x7FFF;
}

int
DIVIDE_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  if (memory_value == 0)
    {
      sc_regSet (SC_DIVIDING_BY_ZERO, 1);
      sc_regSet (SC_THROTTLE, 1);
      return accumulator_value;
    }
  int accumulator_sign = accumulator_value >> 14;
  int memory_sign = memory_value >> 14;
  accumulator_value = accumulator_value << 17;
  memory_value = memory_value << 17;
  int value = (accumulator_value / memory_value) & 0x3FFF;
  if (accumulator_sign ^ memory_sign)
    value = value | 0x4000;
  return value;
}

int
MUL_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  int accumulator_sign = accumulator_value >> 14;
  int memory_sign = memory_value >> 14;
  if (accumulator_sign)
    accumulator_value = ~(accumulator_value - 1);
  if (memory_sign)
    memory_value = ~(memory_value - 1);
  int value = (accumulator_value * memory_value) & 0x3FFF;
  if (accumulator_sign ^ memory_sign)
    value = ((~value & 0x3FFF) + 1) | 0x4000;
  return value;
}

int
NOT_C (void)
{
  int memory_value;
  sc_accumulatorGet (&memory_value);
  return ((~(memory_value << 17)) >> 17) & 0x7FFF;
}

int
AND_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return accumulator_value & memory_value;
}

int
OR_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return accumulator_value | memory_value;
}

int
XOR_C (int cell_number)
{
  int accumulator_value, memory_value;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return accumulator_value ^ memory_value;
}

int
CHL_C (int cell_number)
{
  int memory_value;
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return ((memory_value << 18) >> 17) & 0x7FFF;
}

int
SHR_C (int cell_number)
```

```
{
  int memory_value;
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return memory_value >> 1;
}

int
RCL_C (int cell_number)
{
  int memory_value;
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return ((memory_value << 1) | (memory_value >> 14)) & 0x3FFF;
}

int
RCR_C (int cell_number)
{
  int memory_value;
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return ((memory_value >> 1) | (memory_value << 14)) & 0x3FFF;
}

int
NEG_C (int cell_number)
{
  int memory_value;
  if (sc_memoryGet (cell_number, &memory_value) == -2)
    return 1000000;
  return (-(memory_value << 17) >> 17) & 0x7FFF;
}

int
ADDC_C (int cell_number)
{
  int accumulator_value, memory_value1, memory_value2;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value1) == -2)
    return 1000000;
  if (sc_memoryGet (accumulator_value & 0x3F, &memory_value2) == -2)
    return 1000000;
  return (((memory_value1 << 17) + (memory_value2 << 17)) >> 17);
}

int
SUBC_C (int cell_number)
{
  int accumulator_value, memory_value1, memory_value2;
  sc_accumulatorGet (&accumulator_value);
  if (sc_memoryGet (cell_number, &memory_value1) == -2)
    return 1000000;
  if (sc_memoryGet (accumulator_value & 0x3F, &memory_value2) == -2)
    return 1000000;
  return (((memory_value1 << 17) - (memory_value2 << 17)) >> 17);
}

int
alu (int command, int operand)
{
  switch (command)
    {
    case ADD:
      return ADD_C (operand);
    case SUB:
      return SUB_C (operand);
    case DIVIDE:
      return DIVIDE_C (operand);
    case MUL:
      return MUL_C (operand);
    case NOT:
      return NOT_C ();
    case AND:
      return AND_C (operand);
    case OR:
      return OR_C (operand);
    case XOR:
      return XOR_C (operand);
    case CHL:
      return CHL_C (operand);
    case SHR:
      return SHR_C (operand);
    case RCL:
      return RCL_C (operand);
    case RCR:
      return RCR_C (operand);
    case NEG:
      return NEG_C (operand);
    case ADDC:
      return ADDC_C (operand);
    case SUBC:
      return SUBC_C (operand);
    }
  return 0;
}

check_terminal_size.c
#include "console.h"
#include <myTerm.h>

#include <stdio.h>

int
check_terminal_size (void)
{
  int rows = 0, cols = 0;
  if (mt_getscreensize (&rows, &cols))
    return -1;
  if (rows < 27 || cols < 108)
    {
      printf ("Terminal is too small\n");
      printf ("Needs 27x108, but it's %dx%d\n", rows, cols);
      return -1;
    }
  return 0;
}

console.h
#pragma once

#include <myTerm.h>
```

```
enum 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,
  ADDC2 = 0x4B,
  SUBC2 = 0x4C
};

extern int cell;
extern int big[36];

void printAccumulator (void);
void printCell (int address, enum colors fg, enum colors bg);
void printCounters (void);
void printDecodedCommand (int value);
void printFlags (void);
int printTerm (int address, int input);
void printInfo (void);
void printBigCell (void);
void printCommand (void);
void CU (void);
int alu (int command, int operand);
void IRC (int signum);
void print_all_mem_cells_def (void);
void printCache (void);
int get_font (char *filename);
int check_terminal_size (void);
void draw_boxes (void);
void print_all_mem_cells_def (void);
void default_state (void);
void draw_interface (void);
void running_application ();

cu.c
#include "console.h"
#include <myReadKey.h>
#include <mySimpleComputer.h>
#include <myTerm.h>

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

void
CPUINFO_C (void) // 0x01
{
  mt_gotoXY (20, 79);
  mt_print ("Кулик Павел Евгеньевич, ИС241");
  for (int i = 0; i < 4; i++)
    {
      mt_gotoXY (21 + i, 79);
      mt_print ("                             ");
    }
  sleep (2);
  printInfo ();
}

void
READ_C (int cell_number) // 0x0A
{
  sc_regSet (SC_THROTTLE, 1);
  sc_setIgnoreCache (1);
  printTerm (cell_number, 1);
  sc_setIgnoreCache (0);
  rk_mytermregime (0, 1, 0, 0, 0);
  sc_regSet (SC_THROTTLE, 0);
}

int
WRITE_C (int cell_number) // 0x0B
{
  if (printTerm (cell_number, 0) == -2)
    return -2;
  return 0;
}

int
LOAD_C (int cell_number) // 0x14
{
  int value;
  if (sc_memoryGet (cell_number, &value) == -2)
    return -2;
  sc_accumulatorSet (value);
  return 0;
}
```

```
int
STORE_C (int cell_number) // 0x15
{
  int value;
  sc_accumulatorGet (&value);
  if (sc_memorySet (cell_number, value) == -2)
    return -2;
  return 0;
}

void
JUMP_C (int cell_number) // 0x28
{
  sc_icounterSet (cell_number);
}

void
JNEG_C (int cell_number) // 0x29
{
  int value;
  sc_accumulatorGet (&value);
  if ((value >> 14) > 0)
    {
      sc_icounterSet (cell_number);
    }
}

void
JZ_C (int cell_number) // 0x2A
{
  int value;
  sc_accumulatorGet (&value);
  if ((value & 0x3FFF) == 0)
    {
      sc_icounterSet (cell_number);
    }
}

void
HALT_C (void)
{
  sc_regSet (SC_THROTTLE, 1);
}

void
JNS_C (int cell_number) // 0x37
{
  int value;
  sc_accumulatorGet (&value);
  if (((value >> 14) == 0) && (value & 0x3FFF) != 0)
    {
      sc_icounterSet (cell_number);
    }
}

void
JC_C (int cell_number) // 0x38
{
  int value = 0;
  sc_regGet (SC_OVERFLOW, &value);
  if (value)
```

```
    {
      sc_icounterSet (cell_number);
    }
}

void
JNC_C (int cell_number) // 0x39
{
  int value = 0;
  sc_regGet (SC_OVERFLOW, &value);
  if (!value)
    {
      sc_icounterSet (cell_number);
    }
}

void
JP_C (int cell_number) // 0x3A
{
  int value;
  sc_accumulatorGet (&value);
  if ((value & 0x3FFF) % 2 == 0)
    {
      sc_icounterSet (cell_number);
    }
}

void
JNP_C (int cell_number) // 0x3B
{
  int value;
  sc_accumulatorGet (&value);
  if ((value & 0x3FFF) % 2 != 0)
    {
      sc_icounterSet (cell_number);
    }
}

int
MOVA_C (int cell_number) // 0x47
{
  int value;
  int address;
  sc_accumulatorGet (&address);
  if (sc_memoryGet (cell_number, &value) == -2)
    return -2;
  if (sc_memorySet (address & 0x7F, value) == -2)
    return -2;
  return 0;
}

int
MOVR_C (int cell_number) // 0x48
{
  int value;
  int address;
  sc_accumulatorGet (&address);
  if (sc_memoryGet (address & 0x7F, &value) == -2)
    return -2;
  if (sc_memorySet (cell_number, value) == -2)
    return -2;
  return 0;
}

int
MOVCA_C (int cell_number) // 0x49
{
  int value;
  int address_from;
  int address_to;
  sc_accumulatorGet (&address_from);
  if (sc_memoryGet (address_from & 0x7F, &address_to) == -2)
    return -2;
  if (sc_memoryGet (cell_number, &value) == -2)
    return -2;
  if (sc_memorySet (address_to & 0x7F, value) == -2)
    return -2;
  return 0;
}

void
CU (void)
{
  int command_number;
  int memory_value;
  int sign;
  int value;
  int operand;
  int returned;
  sc_icounterGet (&command_number);

  sc_setIgnoreCache (0);

  returned = sc_memoryGet (command_number, &memory_value);
  if (returned == -1)
    {
      sc_regSet (SC_OUT_OF_MEMORY, 1);
      sc_regSet (SC_THROTTLE, 1);
      return;
    }
  if (returned == -2)
    return;
  if (sc_commandDecode (memory_value, &sign, &value, &operand))
    {
      sc_regSet (SC_INVALID_COMMAND, 1);
      sc_regSet (SC_THROTTLE, 1);
      return;
    }
  if (sc_commandValidate (value) || sign == 1)
    {
      sc_regSet (SC_INVALID_COMMAND, 1);
      sc_regSet (SC_THROTTLE, 1);
      return;
    }
  switch (value)
    {
    case NOP:
      break;
    case CPUINFO:
      CPUINFO_C ();
      break;
    case READ:
```

```
    READ_C (operand);
    break;
  case WRITE:
    if (WRITE_C (operand) == -2)
      return;
    break;
  case LOAD:
    if (LOAD_C (operand) == -2)
      return;
    break;
  case STORE:
    if (STORE_C (operand) == -2)
      return;
    break;
  case JUMP:
    JUMP_C (operand);
    break;
  case JNEG:
    JNEG_C (operand);
    break;
  case JZ:
    JZ_C (operand);
    break;
  case HALT:
    HALT_C ();
    break;
  case JNS:
    JNS_C (operand);
    break;
  case JC:
    JC_C (operand);
    break;
  case JNC:
    JNC_C (operand);
    break;
  case JP:
    JP_C (operand);
    break;
  case JNP:
    JNP_C (operand);
    break;
  case MOVA:
    if (MOVA_C (operand) == -2)
      return;
    break;
  case MOVR:
    if (MOVR_C (operand) == -2)
      return;
    break;
  case MOVCA:
    if (MOVCA_C (operand) == -2)
      return;
    break;
  case NOT:
    if (sc_memorySet (operand, alu (value, operand)) == -2)
      return;
    break;
  default:
    returned = alu (value, operand);
    if (returned == 1000000)
      return;
      sc_accumulatorSet (returned);
    }
  int new_command_number;
  sc_icounterGet (&new_command_number);
  if (new_command_number == command_number)
    if (sc_icounterSet (command_number + 1))
      sc_regSet (SC_THROTTLE, 1);
}

default_state.c
#include "console.h"
#include <mySimpleComputer.h>

void
default_state (void)
{
  cell = 0;
  sc_accumulatorSet (0);
  sc_icounterSet (0);
  draw_boxes ();
  sc_setIgnoreCache (1);
  sc_memoryInit ();
  sc_setIgnoreCache (0);
  sc_regInit ();
  sc_cacheInit ();
  printFlags ();
  printAccumulator ();
  printCounters ();
  printInfo ();
}
draw_boxes.c
#include <myBigChars.h>
#include <myTerm.h>

void
draw_boxes (void)
{
  mt_clrscr ();
  bc_box (1, 1, 13, 59, DEFAULT, DEFAULT, " Оперативная память ", RED,
          DEFAULT);
  bc_box (16, 1, 1, 59, DEFAULT, DEFAULT, " Редактируемая ячейка (формат) ",
          RED, WHITE);
  bc_box (1, 62, 1, 21, DEFAULT, DEFAULT, " Аккумулятор ", RED, DEFAULT);
  bc_box (1, 85, 1, 22, DEFAULT, DEFAULT, " Регистр  флагов ", RED, DEFAULT);
  bc_box (4, 62, 1, 21, DEFAULT, DEFAULT, " Счётчик команд ", RED, DEFAULT);
  bc_box (4, 85, 1, 22, DEFAULT, DEFAULT, " Команда ", RED, DEFAULT);
  bc_box (7, 62, 10, 45, DEFAULT, DEFAULT,
          " Редактируемая ячейка (увеличено) ", RED, WHITE);
  bc_box (19, 1, 5, 64, DEFAULT, DEFAULT, " Кеш процессора ", GREEN, WHITE);
  bc_box (19, 67, 5, 9, DEFAULT, DEFAULT, " IN--OUT ", GREEN, WHITE);
  bc_box (19, 78, 5, 29, DEFAULT, DEFAULT, " Клавиши ", GREEN, WHITE);
}

draw_interface.c
#include "console.h"
#include <mySimpleComputer.h>
#include <myTerm.h>

void
draw_interface (void)
{
  int value;
  sc_setIgnoreCache (1);
  sc_memoryGet (cell, &value);
  print_all_mem_cells_def ();
  printCell (cell, BLACK, WHITE);
  printDecodedCommand (value);
  printBigCell ();
  printAccumulator ();
  printFlags ();
  printCache ();
  printCounters ();
  printCommand ();
  sc_setIgnoreCache (0);
  mt_gotoXY (27, 1);
}

font.c
#include <myBigChars.h>

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void
print_bin (int n)
{
  int j = 0;
  for (int i = 0; i < 32; i++)
    {
      printf ("%d", (n >> i) & 1);
      j++;
      if (j == 8)
        {
          printf ("\n");
          j = 0;
        }
    }
}

void
make_zero (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 3; i <= 6; i++)
    {
      bc_setbigcharpos (big, i, 4, 0);
      bc_setbigcharpos (big, i, 5, 0);
    }
}

void
make_one (int *big)
{
  big[0] = 0;
  big[1] = 0;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 5, 1);
      bc_setbigcharpos (big, i, 6, 1);
    }
  bc_setbigcharpos (big, 2, 4, 1);
  bc_setbigcharpos (big, 3, 3, 1);
  bc_setbigcharpos (big, 3, 4, 1);
  bc_setbigcharpos (big, 7, 4, 1);
  bc_setbigcharpos (big, 8, 3, 1);
  bc_setbigcharpos (big, 8, 4, 1);
  bc_setbigcharpos (big, 8, 7, 1);
}

void
make_two (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 2; i <= 5; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
      bc_setbigcharpos (big, 6, i + 2, 0);
    }
}

void
make_three (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 3; i <= 5; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
      bc_setbigcharpos (big, 6, i, 0);
      bc_setbigcharpos (big, 4, i - 2, 0);
      bc_setbigcharpos (big, 5, i - 2, 0);
    }
}

void
make_four (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 1; i <= 4; i++)
    {
      bc_setbigcharpos (big, i, 4, 0);
      bc_setbigcharpos (big, i, 5, 0);
      bc_setbigcharpos (big, 7, i + 1, 0);
      bc_setbigcharpos (big, 8, i + 1, 0);
    }
}

void
make_five (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 2; i <= 5; i++)
    {
      bc_setbigcharpos (big, 3, i + 2, 0);
      bc_setbigcharpos (big, 6, i, 0);
    }
}

void
make_six (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 4; i <= 7; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
    }
  bc_setbigcharpos (big, 6, 4, 0);
  bc_setbigcharpos (big, 6, 5, 0);
}

void
make_seven (int *big)
{
  big[0] = 0;
  big[1] = 0;
  for (int i = 2; i <= 7; i++)
    {
      bc_setbigcharpos (big, 1, i, 1);
      bc_setbigcharpos (big, 2, i, 1);
    }
  for (int i = 3; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 6, 1);
      bc_setbigcharpos (big, i, 7, 1);
    }
}
```

```
void
make_eight (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  bc_setbigcharpos (big, 3, 4, 0);
  bc_setbigcharpos (big, 3, 5, 0);
  bc_setbigcharpos (big, 6, 4, 0);
  bc_setbigcharpos (big, 6, 5, 0);
}

void
make_nine (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 2; i <= 5; i++)
    {
      bc_setbigcharpos (big, 6, i, 0);
    }
  bc_setbigcharpos (big, 3, 4, 0);
  bc_setbigcharpos (big, 3, 5, 0);
}

void
make_A (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 4; i <= 5; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
      bc_setbigcharpos (big, 4, i, 0);
      bc_setbigcharpos (big, 7, i, 0);
      bc_setbigcharpos (big, 8, i, 0);
    }
}

void
make_B (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
```

```
        bc_setbigcharpos (big, i, 8, 0);
      }
    for (int i = 4; i <= 5; i++)
      {
        bc_setbigcharpos (big, 3, i, 0);
        bc_setbigcharpos (big, 6, i, 0);
        bc_setbigcharpos (big, i, 7, 0);
      }
}

void
make_C (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 4; i <= 7; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
      bc_setbigcharpos (big, 4, i, 0);
      bc_setbigcharpos (big, 5, i, 0);
      bc_setbigcharpos (big, 6, i, 0);
    }
}

void
make_D (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 3; i <= 6; i++)
    {
      bc_setbigcharpos (big, i, 4, 0);
    }
  bc_setbigcharpos (big, 4, 5, 0);
  bc_setbigcharpos (big, 5, 5, 0);
  bc_setbigcharpos (big, 1, 7, 0);
  bc_setbigcharpos (big, 8, 7, 0);
}

void
make_E (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 4; i <= 7; i++)
    {
        bc_setbigcharpos (big, 3, i, 0);
        bc_setbigcharpos (big, 6, i, 0);
      }
}

void
make_F (int *big)
{
  big[0] = 0xffffffff;
  big[1] = 0xffffffff;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 1, 0);
      bc_setbigcharpos (big, i, 8, 0);
    }
  for (int i = 4; i <= 7; i++)
    {
      bc_setbigcharpos (big, 3, i, 0);
      bc_setbigcharpos (big, 6, i, 0);
      bc_setbigcharpos (big, 7, i, 0);
      bc_setbigcharpos (big, 8, i, 0);
    }
}

void
make_plus (int *big)
{
  big[0] = 0;
  big[1] = 0;
  for (int i = 1; i <= 8; i++)
    {
      bc_setbigcharpos (big, i, 4, 1);
      bc_setbigcharpos (big, i, 5, 1);
    }
  for (int i = 2; i <= 7; i++)
    {
      bc_setbigcharpos (big, 4, i, 1);
      bc_setbigcharpos (big, 5, i, 1);
    }
}

void
make_minus (int *big)
{
  big[0] = 0;
  big[1] = 0;
  for (int i = 2; i <= 7; i++)
    {
      bc_setbigcharpos (big, 4, i, 1);
      bc_setbigcharpos (big, 5, i, 1);
    }
}

int
main ()
{
  int *big = malloc (36 * sizeof (int));
  if (!big)
    return -1;
  int index = 0;
  make_zero (big + index);
  index += 2;
  make_one (big + index);
  index += 2;
  make_two (big + index);
  index += 2;
  make_three (big + index);
  index += 2;
  make_four (big + index);
  index += 2;
  make_five (big + index);
  index += 2;
  make_six (big + index);
  index += 2;
  make_seven (big + index);
  index += 2;
  make_eight (big + index);
  index += 2;
  make_nine (big + index);
  index += 2;
  make_A (big + index);
  index += 2;
  make_B (big + index);
  index += 2;
  make_C (big + index);
  index += 2;
  make_D (big + index);
  index += 2;
  make_E (big + index);
  index += 2;
  make_F (big + index);
  index += 2;
  make_plus (big + index);
  index += 2;
  make_minus (big + index);

  int fd = open ("font.bin", O_CREAT | O_WRONLY | O_TRUNC, 0644);
  if (fd == -1)
    {
      printf ("1\n");
      return -1;
    }
  if (bc_bigcharwrite (fd, big, 18))
    {
      printf ("2\n");

      return -1;
    }
  close (fd);
  free (big);
}

get_font.c
#include "console.h"
#include <myBigChars.h>

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int
get_font (char *filename)
```

```
{
  int fd;
  fd = open (filename, O_RDONLY);
  if (fd == -1)
    {
      printf ("Can't open font :(\n");
      return -1;
    }
  int count;
  bc_bigcharread (fd, big, 18, &count);
  if (count != 18)
    {
      printf ("Something wrong with bc_bigcharread\n");
      close (fd);
      return -1;
    }
  close (fd);
  return 0;
}

IRC.c
#include <signal.h>
#include <unistd.h>

#include "console.h"
#include <mySimpleComputer.h>

void
IRC (int signum)
{
  if (signum == SIGUSR1)
    {
      sc_memoryInit ();
      sc_regInit ();
      sc_accumulatorInit ();
      sc_icounterSet (0);
      sc_regSet (SC_THROTTLE, 1);
    }
  if (signum == SIGALRM)
    {
      int flag;
      sc_regGet (SC_THROTTLE, &flag);
      unsigned char tcounter;
      sc_tcounterGet (&tcounter);
      if (tcounter)
        {
          sc_tcounterSet (--tcounter);

          sc_setIgnoreCache (1);
          print_all_mem_cells_def ();
          printBigCell ();
          printAccumulator ();
          printFlags ();
          printCounters ();
          printCommand ();
          printCache ();
          sc_setIgnoreCache (0);
          mt_gotoXY (27, 1);

          if (!tcounter)
            {
              sc_regSet (SC_THROTTLE, 0);
              flag = 0;
            }
          else
            {
              sc_regSet (SC_THROTTLE, 1);
              return;
            }
        }
      if (flag)
        return;
      CU ();
      sc_setIgnoreCache (1);
      print_all_mem_cells_def ();
      printBigCell ();
      printAccumulator ();
      printFlags ();
      printCounters ();
      printCommand ();
      printCache ();
      sc_setIgnoreCache (0);
      mt_gotoXY (27, 1);
    }
}
```

```
main.c
#include "console.h"
#include <mySimpleComputer.h>

#include <signal.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

int cell = 0;
int big[36];

int
main (int argc, char *argv[])
{
  if (argc > 1)
    {
      if (get_font (argv[1]))
        return 1;
    }
  else
    {
      if (get_font ("./console/font.bin"))
        return 1;
    }

  if (!isatty (STDIN_FILENO))
    {
      printf ("Can't reach terminal\n");
      return 1;
    }

  if (check_terminal_size ())
    return 1;

  sc_setIgnoreCache (1);
  default_state ();
  draw_interface ();
  sc_setIgnoreCache (0);

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

  struct itimerval nval, oval;

  nval.it_interval.tv_sec = 0;
  nval.it_interval.tv_usec = 500000;
  nval.it_value.tv_sec = 1;
  nval.it_value.tv_usec = 0;

  setitimer (ITIMER_REAL, &nval, &oval);

  running_application ();

  mt_print ("\n");
  mt_gotoXY (50, 1);
}
```

makefile

```
APP_NAME = app

SRC_EXT = c

APP_SOURCES = $(filter-out font.c, $(wildcard *.$(SRC_EXT)))
APP_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(APP_OBJECTS:.o=.d)

.PHONY: all
all: $(APP_NAME) font.bin

-include $(DEPS)

$(APP_NAME): $(APP_OBJECTS) $(LIBS)
	$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@ $(LFLAGS)

/%.o: /%.$(SRC_EXT)
	$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@ $(LFLAGS)

font.bin: font
	./font
	rm font

font: font.c $(LIBS)
	$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@ $(LFLAGS)

.PHONY: clean
clean:
	rm -rf $(APP_OBJECTS) $(DEPS) $(APP_NAME) font.d font.bin
```

print_all_mem_cells_def.c

```
#include "console.h"

void
print_all_mem_cells_def (void)
{
  for (int i = 0; i < 128; i++)
    printCell (i, DEFAULT, DEFAULT);
}

printAccumulator.c
#include <mySimpleComputer.h>
#include <myTerm.h>
#include <stdio.h>

void
printAccumulator (void)
{
  int value;
  if (sc_accumulatorGet (&value))
    {
      mt_print ("Error!\n");
      return;
    }
  mt_gotoXY (2, 64);
  mt_print ("sc: ");
  if (value >> 14)
    {
      if (value & 0x3FFF)
        {
          mt_print ("-");
          value = (~value & 0x3FFF) + 1;
        }
      else
        {
          mt_print ("-7F80");
          mt_print (" hex: %04X", value);
          return;
        }
    }
  else
    mt_print ("+");
  mt_print ("%02X", value >> 7 & 0b1111111);
  mt_print ("%02X", value & 0b1111111);
  sc_accumulatorGet (&value);
  mt_print (" hex: %04X", value);
}

printBigCell.c
#include "console.h"
#include <myBigChars.h>
#include <mySimpleComputer.h>

void
printBigCell (void)
{
  int value;
  sc_memoryGet (cell, &value);
  if (value >> 14)
    {
      bc_printbigchar (&big[34], 9, 64, DEFAULT, DEFAULT);
      value = ((~value & 0x3FFF) + 1) | 0x4000;
    }
  else
    bc_printbigchar (&big[32], 9, 64, DEFAULT, DEFAULT);

  if ((value >> 14) && ((value & 0x3FFF) == 0))
    {
      bc_printbigchar (&big[14], 9, 72, DEFAULT, DEFAULT);
      bc_printbigchar (&big[30], 9, 80, DEFAULT, DEFAULT);
      bc_printbigchar (&big[16], 9, 88, DEFAULT, DEFAULT);
      bc_printbigchar (&big[0], 9, 96, DEFAULT, DEFAULT);
    }
  else
    {
      bc_printbigchar (&big[((value >> 11) & 0b111) * 2], 9, 72, DEFAULT,
                       DEFAULT);
      bc_printbigchar (&big[((value >> 7) & 0b1111) * 2], 9, 80, DEFAULT,
                       DEFAULT);
      bc_printbigchar (&big[((value >> 4) & 0b111) * 2], 9, 88, DEFAULT,
                       DEFAULT);
      bc_printbigchar (&big[(value & 0b1111) * 2], 9, 96, DEFAULT, DEFAULT);
    }
  mt_gotoXY (17, 64);
  mt_setfgcolor (BLUE);
  mt_print ("Номер редактируемой ячейки: %03d", cell);
  mt_setfgcolor (DEFAULT);
}

printCache.c
#include "mySimpleComputer.h"
#include "myTerm.h"
#include <stdio.h>

void
printCache (void)
{
  int cacheline[10];
  int line_size;
  int address;
  for (int line = 0; line < 5; line++)
    {
      address = sc_cachelineGet (line, cacheline);
      if (address != -1)
        {
          line_size = address == 120 ? 8 : 10;
          mt_gotoXY (20 + line, 2);
          mt_print ("%d:                                                     "
                    "      ",
                    address);
          for (int i = 0; i < line_size; i++)
            {
              mt_gotoXY (20 + line, i * 6 + 7);

              if (cacheline[i] >> 14)
                {
                  mt_print ("-");
                  cacheline[i] = (~cacheline[i] & 0x3FFF) + 1;
                }
              else
                mt_print ("+");
              mt_print ("%02X", cacheline[i] >> 7 & 0b1111111);
              mt_print ("%02X", cacheline[i] & 0b1111111);
            }
        }
    }
}
```

printCell.c

```
#include "mySimpleComputer.h"
#include "myTerm.h"
#include <stdio.h>

void
printCell (int address, enum colors fg, enum colors bg)
{
  int value;
  if (sc_memoryGet (address, &value) == -1)
    return;

  mt_setbgcolor (bg);
  mt_setfgcolor (fg);
  int row = 1, col = 0;
  col = address % 10;
  int tmp_address = address;
  while (tmp_address > 9)
    {
      tmp_address -= 10;
      row++;
    }

  mt_gotoXY (row + 1, col * 6 + 2);

  if (value >> 14)
    {
      mt_print ("-");
      value = (~value & 0x3FFF) + 1;
    }
  else
    mt_print ("+");
  mt_print ("%02X", value >> 7 & 0b1111111);
  mt_print ("%02X", value & 0b1111111);
  if (value >> 14 && (value & 0x3FFF) == 0)
    {
      mt_gotoXY (row + 1, col * 6 + 2);
      mt_print ("-7F80");
    }
  mt_setdefaultcolor ();
}
```

printCommand.c

```
#include <mySimpleComputer.h>
#include <myTerm.h>

void
printCommand (void)
{
  int value, cell_number;
  sc_icounterGet (&cell_number);
  mt_gotoXY (5, 90);
  if (cell_number < 0 || cell_number > 127)
    {
      mt_print ("! FF : FF");
      return;
    }
  sc_memoryGet (cell_number, &value);
  if (value >> 14 > 0)
    mt_print ("- ");
  else
    mt_print ("+ ");
  mt_print ("%02X : %02X", (value >> 7) & 0x7F, value & 0x7F);
}
printCounters.c
#include <mySimpleComputer.h>
#include <myTerm.h>
#include <stdio.h>

void
printCounters (void)
{
  int IC;
  unsigned char TC;
  if (sc_icounterGet (&IC))
    {
      mt_print ("Error!\n");
      return;
    }
  if (sc_tcounterGet (&TC))
    {
      mt_print ("Error!\n");
      return;
    }
  mt_gotoXY (5, 63);
  mt_print ("T: %02d     IC: ", TC);
  mt_gotoXY (5, 77);
  if (IC >> 14)
    {
      if (IC & 0x3FFF)
        {
          mt_print ("-");
          IC = (~IC & 0x3FFF) + 1;
        }
      else
        {
          mt_print ("-7F80");
          return;
        }
    }
  else
    mt_print ("+");
  mt_print ("%02X", IC >> 7 & 0b1111111);
  mt_print ("%02X", IC & 0b1111111);
}

printDecodedCommand.c
#include <mySimpleComputer.h>
#include <myTerm.h>
#include <stdio.h>

void
printDecodedCommand (int value)
{
  mt_gotoXY (17, 2);
  mt_print ("dec: %05d | ", value);
  mt_print ("oct: %05o | ", value);
  mt_print ("hex: %04X    ", value);
  mt_print ("bin: ");
  for (int i = 14; i >= 0; i--)
    mt_print ("%d", (value >> i) & 1);
  mt_print ("\n");
}

printFlags.c
#include <mySimpleComputer.h>
#include <myTerm.h>
#include <stdio.h>

void
printFlags (void)
{
  int P, ZERO, M, T, E;
  if (sc_regGet (SC_OVERFLOW, &P))
    {
      mt_print ("Error!\n");
      return;
    }
  if (sc_regGet (SC_DIVIDING_BY_ZERO, &ZERO))
    {
      mt_print ("Error!\n");
      return;
    }
  if (sc_regGet (SC_OUT_OF_MEMORY, &M))
    {
      mt_print ("Error!\n");
      return;
    }
  if (sc_regGet (SC_THROTTLE, &T))
    {
      mt_print ("Error!\n");
      return;
    }
  if (sc_regGet (SC_INVALID_COMMAND, &E))
    {
      mt_print ("Error!\n");
      return;
    }

  mt_gotoXY (2, 90);
  if (P == 0)
    mt_print ("_ ");
  else
    mt_print ("P ");

  mt_gotoXY (2, 93);
  if (ZERO == 0)
    mt_print ("_ ");
  else
    mt_print ("0 ");

  mt_gotoXY (2, 96);
  if (M == 0)
    mt_print ("_ ");
  else
    mt_print ("M ");

  mt_gotoXY (2, 99);
  if (T == 0)
    mt_print ("_ ");
  else
    mt_print ("T ");

  mt_gotoXY (2, 102);
  if (E == 0)
    mt_print ("_ ");
  else
    mt_print ("E ");
}



printInfo.c
#include <myTerm.h>

void
printInfo ()
{
  mt_gotoXY (20, 79);
  mt_print ("l - load  s - save  i - reset");
  mt_gotoXY (21, 79);
  mt_print ("r - run  t - step");
  mt_gotoXY (22, 79);
  mt_print ("ESC - выход");
  mt_gotoXY (23, 79);
  mt_print ("F5 - accumulator");
  mt_gotoXY (24, 79);
  mt_print ("F6 - instruction counter");
}

printTerm.c
#include <myReadKey.h>
#include <mySimpleComputer.h>
#include <myTerm.h>

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

char terms[5][10];

int
printTerm (int address, int input)
{
  for (int i = 4; i > 0; i--)
    strcpy (terms[i], terms[i - 1]);
  int value = 0;
  int returned;
  if (input == 0)
    {
      returned = sc_memoryGet (address, &value);
      if (returned)
        {
          return returned;
        }
      if (value >> 14)
        {
          value = (~value & 0x3FFF) + 1;
          snprintf (terms[0], 10, "%02X> -%02X%02X", address,
                    (value >> 7) & 0x7F, value & 0x7F);
        }
      else
        snprintf (terms[0], 10, "%02X> +%02X%02X", address,
                  (value >> 7) & 0x7F, value & 0x7F);
    }
  else
    {
      int row = 20;
      for (int i = 4; i >= 0; i--)
        {
          mt_gotoXY (row++, 68);
          mt_print (terms[i]);
        }

      mt_gotoXY (24, 68);
      mt_print ("%02X<       ", address);
      mt_gotoXY (24, 72);
      rk_readvalue (&value, 1000);
      sc_setIgnoreCache (1);
      sc_memorySet (address, value);
      if (value >> 14)
        {
          if (value & 0x3FFF)
            {
              value = (~value & 0x3FFF) + 1;
              snprintf (terms[0], 10, "%02X< -%02X%02X", address,
                        (value >> 7) & 0x7F, value & 0x7F);
            }
          else
            {
              snprintf (terms[0], 10, "%02X< -7F80", address);
            }
        }
      else
        {
          snprintf (terms[0], 10, "%02X< +%02X%02X", address,
                    (value >> 7) & 0x7F, value & 0x7F);
        }
    }
  int row = 20;
  for (int i = 4; i >= 0; i--)
    {
      mt_gotoXY (row++, 68);
      mt_print (terms[i]);
    }
  return 0;
}
```

```
running_application.c
#include "console.h"
#include <myReadKey.h>
#include <mySimpleComputer.h>
#include <myTerm.h>

void
running_application ()
{
  rk_mytermsave ();
  int value;
  int running = 1;
  enum keys key;
```

```
int throttle = 1;
unsigned char tc = 0;

draw_interface ();

while (running)
  {

    rk_mytermregime (0, 1, 0, 0, 0);
    sc_regGet (SC_THROTTLE, &throttle);
    sc_tcounterGet (&tc);
    if (throttle && tc == 0)
      {
        if (key)
          draw_interface ();
        key = 0;
        rk_readkey (&key);
      }
    if (key == key_ESC)
      running = 0;
    if (key == key_RIGHT)
      {
        cell++;
        if (cell % 10 == 0)
          cell -= 10;
        if (cell == 128)
          cell = 120;

        sc_setIgnoreCache (1);
        print_all_mem_cells_def ();
        printCell (cell, BLACK, WHITE);
        sc_setIgnoreCache (0);
      }
    if (key == key_DOWN)
      {
        cell += 10;
        if (cell > 127)
          {
            cell -= 130;
            if (cell < 0)
              cell += 10;
          }

        sc_setIgnoreCache (1);
        print_all_mem_cells_def ();
        printCell (cell, BLACK, WHITE);
        sc_setIgnoreCache (0);
      }
    if (key == key_LEFT)
      {
        cell--;
        if ((cell + 1) % 10 == 0)
          {
            cell += 10;
            if (cell == 129)
              cell -= 2;
          }

        sc_setIgnoreCache (1);
        print_all_mem_cells_def ();
        printCell (cell, BLACK, WHITE);
        sc_setIgnoreCache (0);
      }
    if (key == key_UP)
      {
        cell -= 10;
        if (cell < 0)
          {
            cell += 130;
            if (cell > 127)
              cell -= 10;
          }

        sc_setIgnoreCache (1);
        print_all_mem_cells_def ();
        printCell (cell, BLACK, WHITE);
        sc_setIgnoreCache (0);
      }
    if (key == key_ENTER)
      {

        mt_gotoXY (2 + cell / 10, 2 + (cell % 10) * 6);
        mt_print ("     ");
        mt_gotoXY (2 + cell / 10, 2 + (cell % 10) * 6);
        sc_setIgnoreCache (1);
        if (!rk_readvalue (&value, 100))
          sc_memorySet (cell, value);
        print_all_mem_cells_def ();
        printCell (cell, BLACK, WHITE);
        sc_setIgnoreCache (0);
      }
    if (key == key_F5)
      {
        mt_gotoXY (2, 68);
        mt_print ("     ");
        mt_gotoXY (2, 68);
        if (!rk_readvalue (&value, 100))
          sc_accumulatorSet (value);
        printAccumulator ();
      }
    if (key == key_F6)
      {
        mt_gotoXY (5, 77);
        mt_print ("     ");
        mt_gotoXY (5, 77);
        if (!rk_readvalue (&value, 100))
          sc_icounterSet (value);
        printCounters ();
      }
    if (key == key_L)
      {
        rk_mytermrestore ();
        mt_gotoXY (26, 1);
        mt_print ("Введите имя файла для загрузки: ");
        char filename[128];
        ssize_t size;
        size = read (STDIN_FILENO, filename, 127);
        filename[size - 1] = '\0';
        mt_gotoXY (26, 1);
        mt_print ("%*c", 108, ' ');
        mt_gotoXY (26, 1);
        if (sc_memoryLoad (filename))
```

```
            {
              mt_print ("Не удаётся загрузить память из файла \"%s\"",
                        filename);
              sleep (2);
            }
          else
            {
              mt_print ("Память из файла \"%s\" успешно загружена", filename);
              sleep (2);
            }

          mt_gotoXY (26, 1);
          mt_print ("%*c", 108, ' ');
          rk_mytermregime (0, 0, 1, 0, 0);
        }
      if (key == key_S)
        {
          rk_mytermrestore ();
          mt_gotoXY (26, 1);
          mt_print ("Введите имя файла для сохранения: ");
          char filename[128];
          ssize_t size;
          size = read (STDIN_FILENO, filename, 127);
          filename[size - 1] = '\0';
          mt_gotoXY (26, 1);
          mt_print ("%*c", 108, ' ');
          mt_gotoXY (26, 1);
          if (sc_memorySave (filename))
            {
              mt_print ("Не удаётся сохранить память в файл \"%s\"", filename);
              sleep (2);
            }
          else
            {
              mt_print ("Память успешно сохранена в файл \"%s\"", filename);
              sleep (2);
            }

          mt_gotoXY (26, 1);
          mt_print ("%*c", 108, ' ');
          rk_mytermregime (0, 0, 1, 0, 0);
        }
      if (key == key_I)
        {
          default_state ();
        }
      if (key == key_R)
        {
          sc_regSet (SC_THROTTLE, 0);
          key = 0;
        }
      if (key == key_T)
        {
          CU ();
        }
    }
  rk_mytermrestore ();
}
```

include

```
myBigChars.h
#pragma once
#include <myTerm.h>

int bc_strlen (char *str);
int bc_printA (char *str);
int bc_box (int x1, int y1, int x2, int y2, enum colors box_fg,
            enum colors box_bg, char *header, enum colors header_fg,
            enum colors header_bg);
int bc_setbigcharpos (int *big, int x, int y, int value);
int bc_getbigcharpos (int *big, int x, int y, int *value);
int bc_printbigchar (int *big, int x, int y, enum colors, enum colors);
int bc_bigcharwrite (int fd, int *big, int count);
int bc_bigcharread (int fd, int *big, int need_count, int *count);

myReadKey.h
#pragma once

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

extern struct termios current, backup;

enum keys
{
  key_UNDEFINED,
  key_0,
  key_1,
  key_2,
  key_3,
  key_4,
  key_5,
  key_6,
  key_7,
  key_8,
  key_9,
  key_A,
  key_B,
  key_C,
  key_D,
  key_E,
  key_F,
  key_plus,
  key_minus,
  key_L,
  key_S,
  key_I,
  key_R,
  key_T,
  key_F5,
  key_F6,
  key_ESC,
  key_ENTER,
  key_UP,
  key_RIGHT,
  key_DOWN,
  key_LEFT
};

int rk_readkey (enum keys *key);
int rk_mytermsave (void);
int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint);
int rk_mytermrestore (void);
int rk_readvalue (int *value, int timeout);
```

```
mySimpleComputer.h
#define GET_BIT_VALUE(REGISTER, NUMBER) ((REGISTER >> (NUMBER - 1)) & 1)
#define SET_BIT_ZERO(REGISTER, NUMBER)                                    \
  REGISTER = (REGISTER & (~(1 << (NUMBER - 1))))
#define SET_BIT(REGISTER, NUMBER) REGISTER = (REGISTER | (1 << (NUMBER - 1)))

#define SC_OVERFLOW 16
#define SC_DIVIDING_BY_ZERO 8
#define SC_OUT_OF_MEMORY 4
#define SC_INVALID_COMMAND 2
#define SC_THROTTLE 1

int sc_accumulatorGet (int *value);
int sc_accumulatorInit (void);
int sc_accumulatorSet (int value);
int sc_commandDecode (int value, int *sign, int *command, int *operand);
int sc_commandEncode (int sign, int command, int operand, int *value);
int sc_commandValidate (int command);
int sc_icounterGet (int *value);
int sc_icounterInit (void);
int sc_icounterSet (int value);
int sc_memoryGet (int address, int *value);
int sc_memoryInit (void);
int sc_memoryLoad (char *filename);
int sc_memorySave (char *filename);
int sc_memorySet (int adress, int value);
int sc_regGet (int sc_register, int *value);
int sc_regInit (void);
int sc_regSet (int sc_register, int value);
int sc_cacheGet (int address, int *value);
int sc_cacheSet (int address, int value);
int sc_cacheInit (void);
int sc_tcounterSet (unsigned char value);
int sc_tcounterGet (unsigned char *value);
int sc_tcounterInit (void);
int sc_cachelineGet (int line_number, int *cacheline);
int sc_setIgnoreCache (int value);
```

```
myTerm.h
#pragma once

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

int mt_clrscr (void);
int mt_delline (void);
int mt_getscreensize (int *rows, int *cols);
int mt_gotoXY (int x, int y);
int mt_setbgcolor (enum colors color);
int mt_setcursorvisible (int value);
int mt_setdefaultcolor (void);
int mt_setfgcolor (enum colors color);
int mt_print (char *format, ...);
```

myBigChars

```
bc_bigcharread.c
#include <unistd.h>

int
bc_bigcharread (int fd, int *big, int need_count, int *count)
{
  if (!big)
    return -1;
  if (!count)
    return -1;
  ssize_t size = need_count * 2 * sizeof (int);
  *count = read (fd, big, size) / 2 / sizeof (int);
  if (*count != need_count)
    return -1;
  return 0;
}

bc_bigcharwrite.c
#include <unistd.h>

int
bc_bigcharwrite (int fd, int *big, int count)
{
  if (!big)
    return -1;
  ssize_t size = count * sizeof (int) * 2;
  if (write (fd, big, size) != size)
    return -1;
  return 0;
}

bc_box.c
#include <myBigChars.h>
#include <myTerm.h>

int
bc_box (int x1, int y1, int x2, int y2, enum colors box_fg, enum colors box_bg,
        char *header, enum colors header_fg, enum colors header_bg)
{
  if (!header)
    return -1;
  mt_setbgcolor (box_bg);
  mt_setfgcolor (box_fg);
  mt_gotoXY (x1, y1);
  bc_printA ("l");
  mt_gotoXY (x1, ++y1);
  for (int i = 0; i < y2; i++)
    {
      bc_printA ("q");
      mt_gotoXY (x1, ++y1);
    }

  bc_printA ("k");
  mt_gotoXY (++x1, y1);
  for (int i = 0; i < x2; i++)
    {
      bc_printA ("x");
      mt_gotoXY (++x1, y1);
    }

  bc_printA ("j");
  mt_gotoXY (x1, --y1);
  for (int i = 0; i < y2; i++)
    {
      bc_printA ("q");
      mt_gotoXY (x1, --y1);
    }

  bc_printA ("m");
  mt_gotoXY (--x1, y1);
  for (int i = 0; i < x2; i++)
    {
      bc_printA ("x");
      mt_gotoXY (--x1, y1);
    }
  mt_setbgcolor (header_bg);
  mt_setfgcolor (header_fg);
  mt_gotoXY (x1, y1 + (y2 - bc_strlen (header)) / 2 + 1);
  mt_print ("%s", header);
  mt_setdefaultcolor ();
  return 0;
}

bc_getbigcharpos.c
#include <mySimpleComputer.h>

int
bc_getbigcharpos (int *big, int x, int y, int *value)
{
  int bit_number = x < 5 ? y + (x - 1) * 8 : y + (x - 5) * 8;

  if (x < 5)
    *value = GET_BIT_VALUE (big[0], bit_number);
  else
    *value = GET_BIT_VALUE (big[1], bit_number);

  return 0;
}

bc_printA.c
#include <myTerm.h>

int
bc_printA (char *str)
{
  if (!str)
    return -1;
  mt_print ("\e(0%s\e(B", str);
  return 0;
}
```

```
bc_printbigchar.c
#include <myBigChars.h>
#include <myTerm.h>

int
bc_printbigchar (int *big, int x, int y, enum colors fg, enum colors bg)
{
  if (!big)
    return -1;
  mt_setfgcolor (fg);
  mt_setbgcolor (bg);
  for (int i = 0; i < 8; i++)
    {
      for (int j = 0; j < 8; j++)
        {
          int value;
          mt_gotoXY (x + i, y + j);
          bc_getbigcharpos (big, i + 1, j + 1, &value);
          if (value)
            bc_printA ("a");
          else
            mt_print (" ");
        }
    }
  mt_setdefaultcolor ();
  return 0;
}

bc_setbigcharpos.c
#include <mySimpleComputer.h>

int
bc_setbigcharpos (int *big, int x, int y, int value)
{
  int bit_number = x < 5 ? y + (x - 1) * 8 : y + (x - 5) * 8;
  if (x < 5)
    {
      if (value)
        SET_BIT (big[0], bit_number);
      else
        SET_BIT_ZERO (big[0], bit_number);
    }
  else
    {
      if (value)
        SET_BIT (big[1], bit_number);
      else
        SET_BIT_ZERO (big[1], bit_number);
    }
  return 0;
}

bc_strlen.c
int
bc_strlen (char *str)
{
  int length = 0;
  while (*str != '\0')
    {
      length++;
      if ((*str & 128) > 0)
```

```
        str += 2;
      else
        str++;
    }
  return length;
}
```

```
makefile
LIB_PATH = libmyBigChars.a

SRC_EXT = c

APP_SOURCES = $(wildcard *.$(SRC_EXT))
LIB_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(LIB_OBJECTS:.o=.d)

.PHONY: all
all: $(LIB_PATH)

-include $(DEPS)

$(LIB_PATH): $(LIB_OBJECTS)
	ar rcs $@ $^

/%.o: /%.$(SRC_EXT)
	$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean
clean:
	rm -rf $(LIB_OBJECTS)
	rm -rf $(DEPS)
	rm -rf $(LIB_PATH)
```

myReadKey

```
makefile
LIB_PATH = libmyReadKey.a

SRC_EXT = c

APP_SOURCES = $(wildcard *.$(SRC_EXT))
LIB_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(LIB_OBJECTS:.o=.d)

.PHONY: all
all: $(LIB_PATH)

-include $(DEPS)

$(LIB_PATH): $(LIB_OBJECTS)
	ar rcs $@ $^

/%.o: /%.$(SRC_EXT)
	$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean
clean:
	rm -rf $(LIB_OBJECTS)
```

```
      rm -rf $(DEPS)
      rm -rf $(LIB_PATH)
```

```
rk_mytermregime.c
#include <myReadKey.h>

int
rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)
{
  struct termios term;
  if (tcgetattr (STDIN_FILENO, &term))
    return -1;

  if (regime)
    term.c_lflag |= ICANON;
  else
    term.c_lflag &= ~ICANON;

  if (echo)
    term.c_lflag |= ECHO;
  else
    term.c_lflag &= ~ECHO;

  if (sigint)
    term.c_lflag |= ISIG;
  else
    term.c_lflag &= ~ISIG;

  term.c_cc[VTIME] = vtime;
  term.c_cc[VMIN] = vmin;
  return tcsetattr (STDIN_FILENO, TCSANOW, &term);
}
```

```
rk_mytermrestore.c
#include <myReadKey.h>

int
rk_mytermrestore (void)
{
  return tcsetattr (STDIN_FILENO, TCSANOW, &backup);
}
```

```
rk_mytermsave.c
#include <myReadKey.h>

struct termios backup;

int
rk_mytermsave (void)
{
  return tcgetattr (STDOUT_FILENO, &backup);
}
```

```
rk_readkey.c
#include <myReadKey.h>

int
rk_readkey (enum keys *key)
{
  char buf[16];
  ssize_t n;
```

```
n = read (STDIN_FILENO, buf, 15);
if (n == 0)
  return -1;
buf[n] = '\0';
if (strcmp (buf, "l") == 0)
  *key = key_L;
else if (strcmp (buf, "s") == 0)
  *key = key_S;
else if (strcmp (buf, "i") == 0)
  *key = key_I;
else if (strcmp (buf, "r") == 0)
  *key = key_R;
else if (strcmp (buf, "t") == 0)
  *key = key_T;
else if (strcmp (buf, "\e[15~") == 0)
  *key = key_F5;
else if (strcmp (buf, "\e[17~") == 0)
  *key = key_F6;
else if (strcmp (buf, "\e") == 0)
  *key = key_ESC;
else if (strcmp (buf, "\n") == 0)
  *key = key_ENTER;
else if (strcmp (buf, "\e[A") == 0)
  *key = key_UP;
else if (strcmp (buf, "\e[C") == 0)
  *key = key_RIGHT;
else if (strcmp (buf, "\e[B") == 0)
  *key = key_DOWN;
else if (strcmp (buf, "\e[D") == 0)
  *key = key_LEFT;
else if (strcmp (buf, "0") == 0)
  *key = key_0;
else if (strcmp (buf, "1") == 0)
  *key = key_1;
else if (strcmp (buf, "2") == 0)
  *key = key_2;
else if (strcmp (buf, "3") == 0)
  *key = key_3;
else if (strcmp (buf, "4") == 0)
  *key = key_4;
else if (strcmp (buf, "5") == 0)
  *key = key_5;
else if (strcmp (buf, "6") == 0)
  *key = key_6;
else if (strcmp (buf, "7") == 0)
  *key = key_7;
else if (strcmp (buf, "8") == 0)
  *key = key_8;
else if (strcmp (buf, "9") == 0)
  *key = key_9;
else if (strcmp (buf, "a") == 0)
  *key = key_A;
else if (strcmp (buf, "b") == 0)
  *key = key_B;
else if (strcmp (buf, "c") == 0)
  *key = key_C;
else if (strcmp (buf, "d") == 0)
  *key = key_D;
else if (strcmp (buf, "e") == 0)
  *key = key_E;
else if (strcmp (buf, "f") == 0)
```

```
    *key = key_F;
  else if (strcmp (buf, "+") == 0)
    *key = key_plus;
  else if (strcmp (buf, "-") == 0)
    *key = key_minus;
  return 0;
}

rk_readvalue.c
#include <myReadKey.h>
#include <myTerm.h>

#include <stdlib.h>

int
rk_readvalue (int *value, int timeout)
{
  rk_mytermregime (0, timeout, 0, 0, 0);
  char buf[16] = "";
  int is_completed = 0;
  int n_symbol = 0;
  while (!is_completed)
    {
      enum keys key = key_UNDEFINED;
      if (rk_readkey (&key))
        return -1;
      if (key == key_ESC)
        return -1;
      if (n_symbol == 0)
        {
          if (key == key_plus)
            {
              buf[0] = '+';
              mt_print ("+");
              n_symbol++;
            }
          else if (key == key_minus)
            {
              buf[0] = '-';
              mt_print ("-");
              n_symbol++;
            }
        }
      else
        {
          if (key >= key_0 && key <= key_9)
            {
              buf[n_symbol] = key - key_0 + '0';
              mt_print ("%c", key - key_0 + '0');
              n_symbol++;
            }
          else if (key >= key_A && key <= key_F)
            {
              buf[n_symbol] = key - key_A + 'A';
              mt_print ("%c", key - key_A + 'A');
              n_symbol++;
            }
        }
      key = key_UNDEFINED;
      if (n_symbol == 5)
        is_completed++;
    }
  buf[5] = '\0';
  int sign = buf[0] == '+' ? 0 : 1;
  int right_value = strtol (&buf[3], NULL, 16);
  buf[3] = '\0';
  int left_value = strtol (&buf[1], NULL, 16);
  if (sign && right_value > 127 && left_value > 126)
    {
      *value = 0b100000000000000;
      return 0;
    }
  right_value = right_value > 127 ? 127 : right_value;
  left_value = left_value > 127 ? 127 << 7 : left_value << 7;
  *value = 0;
  *value |= (sign << 14) | right_value | left_value;
  if (sign)
    *value = ((~(*value - 1) & 0x3FFF) | (sign << 14));
  if (sign && !right_value && !left_value)
    *value = 0;
  return 0;
}
```

mySimpleComputer

```
makefile
LIB_PATH = libmySimpleComputer.a

SRC_EXT = c

APP_SOURCES = $(wildcard *.$(SRC_EXT))
LIB_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(LIB_OBJECTS:.o=.d)

.PHONY: all
all: $(LIB_PATH)

-include $(DEPS)

$(LIB_PATH): $(LIB_OBJECTS)
	ar rcs $@ $^

/%.o: /%.$(SRC_EXT)
	$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean
clean:
	rm -rf $(LIB_OBJECTS)
	rm -rf $(DEPS)
	rm -rf $(LIB_PATH)
```

```
sc_accumulatorGet.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_accumulatorGet (int *value)
```

```
{
  if (value == NULL)
    return -1;
  *value = SC_ACCUMULATOR;
  return 0;
}

sc_accumulatorInit.c
#include "sc_variables.h"

int
sc_accumulatorInit (void)
{
  SC_ACCUMULATOR = 0;
  return 0;
}

sc_accumulatorSet.c
#include "sc_variables.h"

int
sc_accumulatorSet (int value)
{
  if (value < 0 || value > 32767)
    return -1;
  SC_ACCUMULATOR = value;
  return 0;
}

sc_cacheGet.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_cacheGet (int address, int *value)
{
  if (address < 0 || address >= SC_MEMARR_SIZE || value == NULL)
    return -1;
  int cacheline_address;

  for (int i = 0; i < SC_CACHE_SIZE; i++)
    if (cache[i].address != -1)
      cache[i].downtime++;

  for (int i = 0; i < SC_CACHE_SIZE; i++)
    {
      if (cache[i].address != -1)
        {
          cacheline_address = address - cache[i].address;
          if (cacheline_address >= 0 && cacheline_address < 10)
            {
              cache[i].downtime = 0;
              *value = cache[i].line[cacheline_address];
              return 0;
            }
        }
    }

  return -1;
}
```

```
sc_cacheInit.c
#include "sc_variables.h"

int
sc_cacheInit (void)
{
  for (int i = 0; i < SC_CACHE_SIZE; i++)
    {
      cache[i].address = -1;
      cache[i].downtime = 0;
      for (int j = 0; j < 10; j++)
        {
          cache[i].line[j] = 0;
        }
    }
  return 0;
}

sc_cachelineGet.c
#include "sc_variables.h"

#include <stdio.h>

int
sc_cachelineGet (int line_number, int *cacheline)
{
  if (cacheline == NULL)
    return -1;
  if (line_number < 0 || line_number > 4)
    return -1;

  int line_size = cache[line_number].address == 120 ? 8 : 10;
  if (cache[line_number].address != -1)
    for (int i = 0; i < line_size; i++)
      cacheline[i] = cache[line_number].line[i];

  return cache[line_number].address;
}

sc_cacheSet.c
#include "sc_variables.h"
#include <myTerm.h>
#include <stdio.h>

int
sc_cacheSet (int address, int value)
{
  if (address < 0 || address >= SC_MEMARR_SIZE)
    return -1;

  int i;
  int max_downtime = 0;
  int displacement = 0;
  int line_size = 10;
  int hit = 0;

  for (i = 0; i < SC_CACHE_SIZE; i++)
    {
      if (cache[i].address == -1)
        {
          break;
        }
      if (cache[i].downtime > max_downtime)
        {
          max_downtime = cache[i].downtime;
        }
      if (cache[i].address == (address - (address % 10)))
        {
          hit = 1;
          break;
        }
    }
  if (hit)
    {
      cache[i].line[address % 10] = value;
      return 0;
    }
  else
    SC_TCOUNTER = 10;
  if (i == SC_CACHE_SIZE)
    {
      displacement = 1;
      for (i = 0; i < SC_CACHE_SIZE; i++)
        if (cache[i].downtime == max_downtime)
          break;
    }
  line_size = cache[i].address == 120 ? 8 : 10;
  if (displacement)
    for (int j = 0; j < line_size; j++)
      {
        SC_MEMARR[cache[i].address + j] = cache[i].line[j];
      }
  cache[i].address = address - (address % 10);
  line_size = cache[i].address == 120 ? 8 : 10;
  for (int j = 0; j < line_size; j++)
    {
      cache[i].line[j] = SC_MEMARR[cache[i].address + j];
    }
  return -2;
}

sc_commandDecode.c
#include <stdio.h>

int
sc_commandDecode (int value, int *sign, int *command, int *operand)
{
  if (sign == NULL)
    return -1;
  if (command == NULL)
    return -1;
  if (operand == NULL)
    return -1;
  if (value < 0 || value > 32767)
    return -1;
  int mask = 0x7f;
  *sign = value >> 14;
  *command = (value >> 7) & mask;
  *operand = value & mask;
  return 0;
}
```

```
sc_commandEncode.c
#include "mySimpleComputer.h"
#include <stdio.h>

int
sc_commandEncode (int sign, int command, int operand, int *value)
{
  if (sign != 0 && sign != 1)
    return -1;
  if (sc_commandValidate (command))
    return -1;
  if (operand < 0 || operand > 127)
    return -1;
  if (value == NULL)
    return -1;
  sign <<= 14;
  command <<= 7;
  *value = operand;
  *value |= command;
  *value |= sign;
  return 0;
}

sc_commandValidate.c
int
sc_commandValidate (int command)
{
  if (command < 0 || command > 32767)
    return 1;
  int mask = 0x7f;
  command = (command >> 7) & mask;
  if (command != 0 && command != 1 && command != 10 && command != 20
      && command != 21 && command != 30 && command != 31 && command != 32
      && command != 33 && command != 40 && command != 41 && command != 42
      && command != 43 && command < 51 && command > 67)
    return 1;
  return 0;
}

sc_icounterGet.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_icounterGet (int *value)
{
  if (value == NULL)
    return -1;
  *value = SC_ICOUNTER;
  return 0;
}

sc_icounterInit.c
#include "sc_variables.h"

int
sc_icounterInit (void)
{
  SC_ICOUNTER = 0;
  return 0;
}



sc_icounterSet.c
#include "sc_variables.h"

int
sc_icounterSet (int value)
{
  if (value < 0 || value > 127)
    return -1;
  SC_ICOUNTER = value;
  return 0;
}

sc_memoryGet.c
#include "sc_variables.h"
#include <mySimpleComputer.h>
#include <stdio.h>

int
sc_memoryGet (int address, int *value)
{
  if (address < 0 || address >= SC_MEMARR_SIZE || value == NULL)
    return -1;

  if (SC_IGNORE_CACHE)
    {
      *value = SC_MEMARR[address];
      return 0;
    }

  if (sc_cacheGet (address, value))
    {
      sc_cacheSet (address, *value);
      return -2;
    }
  return 0;
}

sc_memoryInit.c
#include "sc_variables.h"

int
sc_memoryInit (void)
{
  for (int i = 0; i < SC_MEMARR_SIZE; i++)
    SC_MEMARR[i] = 0;
  return 0;
}

sc_memoryLoad.c
#include "sc_variables.h"
#include <stdio.h>
#include <string.h>

int
sc_memoryLoad (char *filename)
```

```
{
  if (filename == NULL)
    return -1;
  FILE *file = fopen (filename, "rb");
  if (file == NULL)
    return -2;
  int tmp_SC_MEMARR[SC_MEMARR_SIZE] = { 0 };
  if (fread (tmp_SC_MEMARR, sizeof (int), SC_MEMARR_SIZE, file)
      != SC_MEMARR_SIZE)
    {
      fclose (file);
      return -3;
    }
  memcpy (SC_MEMARR, tmp_SC_MEMARR, sizeof (int) * SC_MEMARR_SIZE);
  fclose (file);

  return 0;
}

sc_memorySave.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_memorySave (char *filename)
{
  if (filename == NULL)
    return -1;
  FILE *file = fopen (filename, "wb");
  if (file == NULL)
    return -1;
  if (fwrite (SC_MEMARR, sizeof (*SC_MEMARR), SC_MEMARR_SIZE, file)
      != SC_MEMARR_SIZE)
    {
      fclose (file);
      return -1;
    }
  fclose (file);
  return 0;
}

sc_memorySet.c
#include "sc_variables.h"
#include <mySimpleComputer.h>

int
sc_memorySet (int address, int value)
{
  if (address < 0 || address >= SC_MEMARR_SIZE || value < 0 || value > 32767)
    return -1;
  if (SC_IGNORE_CACHE)
    {
      SC_MEMARR[address] = value;
      return 0;
    }
  if (sc_cacheSet (address, value))
    return -2;
  return 0;
}

sc_regGet.c
```

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

int
sc_regGet (int sc_register, int *value)
{
  if (value == NULL)
    return -1;
  if (sc_register != SC_THROTTLE && sc_register != SC_INVALID_COMMAND
      && sc_register != SC_OUT_OF_MEMORY && sc_register != SC_DIVIDING_BY_ZERO
      && sc_register != SC_OVERFLOW)
    return -1;
  *value = SC_FLAGS & sc_register;
  if (*value > 0)
    *value = 1;
  return 0;
}

sc_regInit.c
#include "sc_variables.h"

int
sc_regInit (void)
{
  SC_FLAGS = 1;
  return 0;
}

sc_regSet.c
#include "sc_variables.h"

int
sc_regSet (int sc_register, int value)
{
  if (sc_register != SC_THROTTLE && sc_register != SC_INVALID_COMMAND
      && sc_register != SC_OUT_OF_MEMORY && sc_register != SC_DIVIDING_BY_ZERO
      && sc_register != SC_OVERFLOW)
    return -1;
  switch (value)
    {
    case 1:
      SC_FLAGS |= sc_register;
      return 0;
    case 0:
      SC_FLAGS &= ~sc_register;
      return 0;
    }
  return -1;
}

sc_setIgnoreCache.c
#include "sc_variables.h"

int
sc_setIgnoreCache (int value)
{
  if (value != 0 && value != 1)
    return -1;
  SC_IGNORE_CACHE = value;
  return 0;
}
```

```
sc_tcounterGet.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_tcounterGet (unsigned char *value)
{
  if (value == NULL)
    return -1;
  *value = SC_TCOUNTER;
  return 0;
}


sc_tcounterInit.c
#include "sc_variables.h"
#include <stdio.h>

int
sc_tcounterInit (void)
{
  SC_TCOUNTER = 0;
  return 0;
}

sc_tcounterSet.c
#include "sc_variables.h"

int
sc_tcounterSet (unsigned char value)
{
  SC_TCOUNTER = value;
  return 0;
}

sc_variables.c
#include "sc_variables.h"
int SC_MEMARR[SC_MEMARR_SIZE];
cacheline cache[SC_CACHE_SIZE];
int SC_ACCUMULATOR;
int SC_ICOUNTER;
unsigned char SC_TCOUNTER;
int SC_FLAGS;
int SC_IGNORE_CACHE;

sc_variables.h
#pragma once
#define SC_MEMARR_SIZE 128
#define SC_CACHE_SIZE 5
#define SC_OVERFLOW 16
#define SC_DIVIDING_BY_ZERO 8
#define SC_OUT_OF_MEMORY 4
#define SC_INVALID_COMMAND 2
#define SC_THROTTLE 1
extern int SC_MEMARR[SC_MEMARR_SIZE];
extern int SC_ACCUMULATOR;
extern int SC_ICOUNTER;
extern unsigned char SC_TCOUNTER;
extern int SC_FLAGS;
extern int SC_IGNORE_CACHE;
```

```
typedef struct sc_cache_line
{
  int address;
  int downtime;
  int line[10];
} cacheline;

extern cacheline cache[SC_CACHE_SIZE];
```

myTerm

```
colors.h
enum colors
{
  BLACK,
  RED,
  GREEN,
  YELLOW,
  BLUE,
  PURPLE,
  CYAN,
  WHITE,
  DEFAULT
};

makefile
LIB_PATH = libmyTerm.a

SRC_EXT = c

APP_SOURCES = $(wildcard *.$(SRC_EXT))
LIB_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(LIB_OBJECTS:.o=.d)

.PHONY: all
all: $(LIB_PATH)

-include $(DEPS)

$(LIB_PATH): $(LIB_OBJECTS)
      ar rcs $@ $^

/%.o: /%.$(SRC_EXT)
      $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean
clean:
      rm -rf $(LIB_OBJECTS) $(DEPS) $(LIB_PATH)

mt_clrscr.c
#include <string.h>
#include <unistd.h>

int
mt_clrscr (void)
{
  const char *str = "\E[H\E[2J\E[0;0H";
  ssize_t bytes_written = write (STDOUT_FILENO, str, strlen (str));
  if (bytes_written == -1)
    return -1;
  return 0;
}

mt_delline.c
#include <string.h>
#include <unistd.h>

int
mt_delline (void)
{
  const char *esc = "\E[M";
  if (write (STDOUT_FILENO, esc, strlen (esc)) == -1)
    return -1;
  return 0;
}

mt_getscreensize.c
#include <sys/ioctl.h>
#include <unistd.h>

int
mt_getscreensize (int *rows, int *cols)
{
  struct winsize ws;

  if (ioctl (STDOUT_FILENO, TIOCGWINSZ, &ws))
    return -1;
  *rows = ws.ws_row;
  *cols = ws.ws_col;
  return 0;
}

mt_gotoXY.c
#include "mt_itoa.h"
#include <string.h>
#include <unistd.h>

int
mt_gotoXY (int x, int y)
{
  char buf[32];
  char *ptr = buf;
  const char *esc = "\033[";
  const char *sep = ";";
  const char *end = "H";

  strcpy (ptr, esc);
  ptr += strlen (esc);

  char x_str[16];
  mt_itoa (x, x_str);
  strcpy (ptr, x_str);
  ptr += strlen (x_str);
  *ptr++ = sep[0];

  char y_str[16];
  mt_itoa (y, y_str);
  strcpy (ptr, y_str);
  ptr += strlen (y_str);
  *ptr++ = end[0];

  if (write (STDOUT_FILENO, buf, ptr - buf) == -1)
    {
      return -1;
    }

  return 0;
}
```

```
mt_itoa.c
#include <string.h>

void
mt_itoa (int n, char *buf)
{
  int i = 0;
  if (n == 0)
    {
      buf[i++] = '0';
    }
  else
    {
      while (n != 0)
        {
          int digit = n % 10;
          buf[i++] = digit + '0';
          n /= 10;
        }
    }
  buf[i] = '\0';

  int len = strlen (buf);
  for (int j = 0; j < len / 2; ++j)
    {
      char temp = buf[j];
      buf[j] = buf[len - j - 1];
      buf[len - j - 1] = temp;
    }
}
```

```
mt_itoa.h
void mt_itoa (int n, char *buf);
```

```
mt_print.c
#include <stdarg.h>
#include <stdio.h>
#include <unistd.h>

int
mt_print (char *format, ...)
{
  int buffer_size = 128;
  char buffer[buffer_size];
  size_t length = 0;
  va_list arguments;
  va_start (arguments, format);
  length = vsnprintf (buffer, buffer_size, format, arguments);
  va_end (arguments);
  if (write (STDOUT_FILENO, buffer, length))
    return -1;
  return 0;
}

mt_setbgcolor.c
#include "colors.h"
#include <string.h>
#include <unistd.h>

int
mt_setbgcolor (enum colors color)
{
  ssize_t bytes_written;
  char *esc = NULL;
  switch (color)
    {
    case BLACK:
      esc = "\E[48;5;0m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case RED:
      esc = "\E[48;5;1m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case GREEN:
      esc = "\E[48;5;2m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case YELLOW:
      esc = "\E[48;5;3m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case BLUE:
      esc = "\E[48;5;4m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case PURPLE:
      esc = "\E[48;5;5m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case CYAN:
      esc = "\E[48;5;6m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case WHITE:
      esc = "\E[48;5;7m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case DEFAULT:
      esc = "\E[0m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;
    }
  if (bytes_written == -1)
```

```
    {
      return -1;
    }

  return 0;
}

mt_setcursorvisible.c
#include <string.h>
#include <unistd.h>

int
mt_setcursorvisible (int value)
{
  ssize_t bytes_written;
  char *esc;
  switch (value)
    {
    case 0:
      esc = "\E[?25l";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case 1:
      esc = "\E[?12;25h";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;
    }
  if (bytes_written == -1)
    {
      return -1;
    }

  return 0;
}

mt_setdefaultcolor.c
#include <string.h>
#include <unistd.h>

int
mt_setdefaultcolor (void)
{
  const char *esc = "\E[0m";
  if (write (STDOUT_FILENO, esc, strlen (esc)) == -1)
    return -1;
  return 0;
}

mt_setfgcolor.c
#include "colors.h"
#include <string.h>
#include <unistd.h>

int
mt_setfgcolor (enum colors color)
{
  ssize_t bytes_written;
  char *esc = NULL;
  switch (color)
    {
```

```
    case BLACK:
      esc = "\E[38;5;0m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case RED:
      esc = "\E[38;5;1m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case GREEN:
      esc = "\E[38;5;2m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case YELLOW:
      esc = "\E[38;5;3m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case BLUE:
      esc = "\E[38;5;4m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case PURPLE:
      esc = "\E[38;5;5m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case CYAN:
      esc = "\E[38;5;6m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case WHITE:
      esc = "\E[38;5;7m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;

    case DEFAULT:
      esc = "\E[0m";
      bytes_written = write (STDOUT_FILENO, esc, strlen (esc));
      break;
    }
  if (bytes_written == -1)
    {
      return -1;
    }

  return 0;
}
```

simpleassembler

```
cell_number_error.cpp
#include "simpleassembler.h"

#include <iostream>
```

```
void
cell_number_error (code_line &line, int line_number)
{
  std::cerr << "Error in line " << line_number << ":\n"
            << line.line << std::endl;
  std::cerr << "Uncorrect address: " << line.address << std::endl;
  std::cerr << "Line must be like:" << std::endl;
  std::cerr << "<address> <command> <operand>;<comment (optional)>"
            << std::endl;
}
```

cell_value_error.cpp

```
#include "simpleassembler.h"

#include <iostream>

void
cell_value_error (code_line &line, int line_number)
{
  std::cerr << "Error in line " << line_number << ":\n"
            << line.line << std::endl;
  std::cerr << "Uncorrect operand: " << line.operand << std::endl;
  std::cerr << "Line must be like:" << std::endl;
  std::cerr << "<address> <command> <operand>;<comment (optional)>"
            << std::endl;
}
```

convert_code_lines_to_memory.cpp

```
#include "simpleassembler.h"

#include <iostream>

int *
convert_code_lines_to_memory (code_line *code_lines)
{
  int *memory = new int[128];
  for (int i = 0; i < 128; i++)
    memory[i] = 0;

  int cell_number;
  int command;
  int operand;
  for (int i = 0; i < code_line::counter; i++)
    {
      if (code_lines[i].empty_line)
        {
          std::cout << code_lines[i].line << std::endl;
          continue;
        }
      if (code_lines[i].error_line)
        {
          default_error_output (code_lines[i], i + 1);
          return NULL;
        }
      try
        {
          cell_number = std::stoi (code_lines[i].address);
        }
      catch (const std::invalid_argument &e)
        {
          default_error_output (code_lines[i], i + 1);
          return NULL;
        }
      if (cell_number < 0 || cell_number > 127)
        {
          cell_number_error (code_lines[i], i + 1);
          return NULL;
        }
      command = convert_string_to_command (code_lines[i].command);
      if (command == -1)
        {
          unknownown_command_error (code_lines[i], i + 1);
          return NULL;
        }

      try
        {
          operand = std::stoi (code_lines[i].operand);
        }
      catch (const std::invalid_argument &e)
        {
          default_error_output (code_lines[i], i + 1);
          return NULL;
        }

      if (command == EQ)
        {
          if (operand < -0x7F80 || operand > 0x7F7F)
            {
              cell_value_error (code_lines[i], i + 1);
              return NULL;
            }
          if (code_lines[i].operand.length () != 5)
            {
              cell_value_error (code_lines[i], i + 1);
              return NULL;
            }
          memory[cell_number]
              = convert_string_to_cell_value (code_lines[i].operand);
          continue;
        }
      else
        {
          if (operand < 0 || operand > 127)
            {
              default_error_output (code_lines[i], i + 1);
              return NULL;
            }
        }
      memory[cell_number] = 0;
      memory[cell_number] |= (command << 7) | operand;
    }
  return memory;
}

convert_string_to_cell_value.cpp
#include "simpleassembler.h"

#include <cstring>
#include <string>

int
```

```
convert_string_to_cell_value (std::string str)
{
  char buf[128];
  int value;
  strcpy (buf, str.c_str ());

  int sign = buf[0] == '+' ? 0 : 1;
  int right_value = strtol (&buf[3], NULL, 16);
  buf[3] = '\0';
  int left_value = strtol (&buf[1], NULL, 16);
  if (sign && right_value > 127 && left_value > 126)
    {
      value = 0b100000000000000;
      return value;
    }
  right_value = right_value > 127 ? 127 : right_value;
  left_value = left_value > 127 ? 127 << 7 : left_value << 7;
  value = 0;
  value |= (sign << 14) | right_value | left_value;
  if (sign)
    value = ((~(value - 1) & 0x3FFF) | (sign << 14));
  if (sign && !right_value && !left_value)
    value = 0;
  return value;
}

convert_string_to_command.cpp
#include "simpleassembler.h"

#include <string>

int
convert_string_to_command (std::string command)
{
  if (command == "NOP")
    return NOP;
  else if (command == "CPUINFO")
    return CPUINFO;
  else if (command == "READ")
    return READ;
  else if (command == "WRITE")
    return WRITE;
  else if (command == "LOAD")
    return LOAD;
  else if (command == "STORE")
    return STORE;
  else if (command == "ADD")
    return ADD;
  else if (command == "SUB")
    return SUB;
  else if (command == "DIVIDE")
    return DIVIDE;
  else if (command == "MUL")
    return MUL;
  else if (command == "JUMP")
    return JUMP;
  else if (command == "JNEG")
    return JNEG;
  else if (command == "JZ")
    return JZ;
  else if (command == "HALT")
```

```
    return HALT;
  else if (command == "NOT")
    return NOT;
  else if (command == "AND")
    return AND;
  else if (command == "OR")
    return OR;
  else if (command == "XOR")
    return XOR;
  else if (command == "JNS")
    return JNS;
  else if (command == "JC")
    return JC;
  else if (command == "JNC")
    return JNC;
  else if (command == "JP")
    return JP;
  else if (command == "JNP")
    return JNP;
  else if (command == "CHL")
    return CHL;
  else if (command == "SHR")
    return SHR;
  else if (command == "RCL")
    return RCL;
  else if (command == "RCR")
    return RCR;
  else if (command == "NEG")
    return NEG;
  else if (command == "ADDC")
    return ADDC;
  else if (command == "SUBC")
    return SUBC;
  else if (command == "LOGLC")
    return LOGLC;
  else if (command == "LOGRC")
    return LOGRC;
  else if (command == "=")
    return EQ;
  return -1;
}

convert_strings_to_code_line.cpp
#include "simpleassembler.h"

#include <cstring>
#include <string>
#include <vector>

code_line *
convert_strings_to_code_line (std::vector<std::string> lines)
{
  code_line *code_lines = new code_line[lines.size ()];
  code_line::counter = lines.size ();
  char char_string_for_line[128];
  char *char_string_for_token;
  for (size_t i = 0; i < lines.size (); i++)
    {
      code_lines[i].line = lines[i];
      code_lines[i].empty_line = false;
      code_lines[i].error_line = false;
      strcpy (char_string_for_line, lines[i].c_str ());

      char_string_for_token = strtok (char_string_for_line, " ");
      if (char_string_for_token == NULL)
        {
          code_lines[i].empty_line = true;
          continue;
        }
      code_lines[i].address = char_string_for_token;
      char_string_for_token = NULL;

      char_string_for_token = strtok (NULL, " ");
      if (char_string_for_token == NULL)
        {
          code_lines[i].error_line = true;
          continue;
        }
      code_lines[i].command = char_string_for_token;
      char_string_for_token = NULL;

      char_string_for_token = strtok (NULL, " ");
      if (char_string_for_token == NULL)
        {
          code_lines[i].error_line = true;
          continue;
        }
      code_lines[i].operand = char_string_for_token;
      char_string_for_token = NULL;

      strtok (NULL, " ");
      char_string_for_token = strtok (NULL, ";");
      if (char_string_for_token == NULL)
        continue;
      code_lines[i].comment = char_string_for_token;
    }
  return code_lines;
}

default_error_output.cpp
#include "simpleassembler.h"

#include <iostream>

void
default_error_output (code_line &line, int line_number)
{
  std::cerr << "Error in line " << line_number << ":\n"
            << line.line << std::endl;
  std::cerr << "Line must be like:" << std::endl;
  std::cerr << "<address> <command> <operand>;<comment (optional)>"
            << std::endl;
}

main.cpp
#include "simpleassembler.h"

#include <cstring>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
```

```
int code_line::counter = 0;

int
main (int argc, char *argv[])
{
  if (argc != 3)
    {
      std::cerr << "Usage: " << argv[0] << " <input file> <output file>"
                << std::endl;
      return 1;
    }
  char input_file_name[256];
  char *input_file_name_p;
  strcpy (input_file_name, argv[1]);
  input_file_name_p = strtok (input_file_name, ".");
  input_file_name_p = strtok (NULL, ".");
  if (strcmp (input_file_name_p, "sa") != 0)
    {
      std::cerr << "Uncorrect input file extension!" << std::endl;
      std::cerr << "It must be like: <filename>.sa" << std::endl;
      return 1;
    }
  std::vector<std::string> lines = read_file (argv[1]);
  code_line *code_lines = convert_strings_to_code_line (lines);
  int *memory = convert_code_lines_to_memory (code_lines);
  delete[] code_lines;
  if (memory == NULL)
    return 1;
  if (write_memory_to_file (memory, 128, argv[2]))
    {
      delete[] memory;
      return 1;
    }
  return 0;
}
```

makefile

```
APP_NAME = sat

SRC_EXT = cpp
CC = g++
CFLAGS = -Wall -Wextra -Werror
CPPFLAGS = -MMD

APP_SOURCES = $(wildcard *.$(SRC_EXT))
APP_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(APP_OBJECTS:.o=.d)

.PHONY: all
all: $(APP_NAME)

-include $(DEPS)

$(APP_NAME): $(APP_OBJECTS)
	$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@

%.o: %.$(SRC_EXT)
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o $@
```

```
.PHONY: clean
clean:
      rm -rf $(APP_OBJECTS) $(DEPS) $(APP_NAME)
```

read_file.cpp

```
#include "simpleassembler.h"

#include <fstream>
#include <string>
#include <vector>

std::vector<std::string>
read_file (char *filename)
{
  std::ifstream file (filename);
  std::vector<std::string> lines;
  std::string line;
  while (std::getline (file, line))
    {
      lines.push_back (line);
    }
  file.close ();
  return lines;
}
```

simpleassembler.h

```
#pragma once

#include <string>
#include <vector>

enum 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,
  ADDC2 = 0x4B,
  SUBC2 = 0x4C,
  EQ = 0x4D
};

class code_line
{
public:
  static int counter;
  bool empty_line;
  bool error_line;
  std::string line;
  std::string address;
  std::string command;
  std::string operand;
  std::string comment;
};

std::vector<std::string> read_file (char *filename);
code_line *convert_strings_to_code_line (std::vector<std::string> lines);
bool validate_code_lines (code_line *code_lines);
void default_error_output (code_line &line, int line_number);
void unknownown_command_error (code_line &line, int line_number);
void cell_number_error (code_line &line, int line_number);
void cell_value_error (code_line &line, int line_number);
int convert_string_to_command (std::string command);
int convert_string_to_cell_value (std::string str);
int *convert_code_lines_to_memory (code_line *code_lines);
int write_memory_to_file (int *memory, int size, std::string filename);
```

unknown_command_error.cpp

```
#include "simpleassembler.h"

#include <iostream>

void
unknownown_command_error (code_line &line, int line_number)
{
  std::cerr << "Error in line " << line_number << ":\n"
            << line.line << std::endl;
  std::cerr << "Unknown command: " << line.command << std::endl;
  std::cerr << "Line must be like:" << std::endl;
  std::cerr << "<address> <command> <operand>;<comment (optional)>"
            << std::endl;
}
```

validate_code_lines.cpp

```
#include "simpleassembler.h"

bool
validate_code_lines (code_line *code_lines)
```

```
{
  for (int i = 0; i < code_line::counter; i++)
    {
      if (code_lines[i].error_line)
        return true;
    }
  return false;
}
```

```
write_memory_to_file.cpp
#include <fstream>
#include <iostream>
#include <string>

int
write_memory_to_file (int *memory, int size, std::string filename)
{
  std::ofstream outputFile (filename, std::ios::binary);

  if (outputFile.is_open ())
    {
      outputFile.write (reinterpret_cast<const char *> (memory),
                        sizeof (int) * size);
      outputFile.close ();
      std::cout << "Массив успешно записан в файл в бинарном виде."
                << std::endl;
    }
  else
    {
      std::cerr << "Ошибка открытия файла для записи." << std::endl;
      return -1;
    }

  return 0;
}
```

simplebasic

```
check_code_lines_empty.cpp
#include "simplebasic.h"

#include <iostream>
#include <vector>

int
check_code_lines_empty (vector<code_line> &code_lines)
{
  int i = 0;
  for (auto &cl : code_lines)
    {
      i++;
      if (cl.is_empty ())
        {
          cerr << "Error in " << i << " line" << endl;
          cerr << "This is an empty line" << endl;
          return 1;
        }
    }
  return 0;
}
```

```
check_code_lines_spaces.cpp
#include "simplebasic.h"

#include <iostream>
#include <vector>

int
check_code_lines_spaces (vector<code_line> &code_lines)
{
  int i = 0;
  for (auto &cl : code_lines)
    {
      i++;
      if (!cl.is_there_any_spaces ())
        {
          cerr << "Error in " << i << " line" << endl;
          cerr << "Code line must have at least line number and command"
               << endl;
          cerr << "This is an empty line" << endl;
          return 1;
        }
    }
  return 0;
}

code_lines_to_tokens.cpp
#include "simplebasic.h"

#include <iostream>
#include <vector>

int
code_lines_to_tokens (vector<code_line> &code_lines)
{
  int i = 0;
  for (auto &cl : code_lines)
    {
      i++;
      if (cl.split ())
        {
          cerr << "Error in " << i << " line" << endl;
          cerr << cl.basic_code << endl;
          cerr << "This is an invalid code line" << endl;
          return 1;
        }
    }
  return 0;
}

convert_basic_code_line_to_assembler.cpp
#include "simplebasic.h"

string
convert_basic_code_line_to_assembler (code_line &cl, map<char, int> &table,
                                      map<int, int> &addresses)
{
  string result = "";
  int address = cl.assembler_line_number;
  if (cl.first_command == "INPUT")
    {
      result += int_to_address (address);
      result += " READ   ";
      result += int_to_address (table[cl.operand[0]]);
      result += '\n';
    }
  else if (cl.first_command == "OUTPUT")
    {
      result += int_to_address (address);
      result += " WRITE  ";
      result += int_to_address (table[cl.operand[0]]);
      result += '\n';
    }
  else if (cl.first_command == "GOTO")
    {
      result += int_to_address (address);
      result += " JUMP   ";
      result += int_to_address (addresses[stoi (cl.operand)]);
      result += '\n';
    }
  else if (cl.first_command == "LET")
    {
      result += convert_let_to_assembler (address, cl.postfix_expression,
                                           cl.expression_result, table);
    }
  else if (cl.first_command == "IF")
    {
      result += convert_if_to_assembler (cl, table, addresses);
    }
  else if (cl.first_command == "END")
    {
      result += int_to_address (address);
      result += " HALT   ";
      result += "00\n";
    }
  return result;
}

convert_basic_code_lines_addresses_to_assembler.cpp
#include "simplebasic.h"

map<int, int>
convert_basic_code_lines_adresses_to_assembler (vector<code_line> &code_lines)
{
  map<int, int> addresses;
  int address = 0;
  for (auto &cl : code_lines)
    {
      addresses[cl.line_number] = address;
      cl.assembler_line_number = address;
      if (cl.first_command == "LET")
        {
          address += count_commands (cl.first_command, cl.postfix_expression);
          continue;
        }
      if (cl.first_command == "LET")
        {
          address        +=        count_commands         (cl.second_command,
cl.postfix_expression);
          continue;
        }
      if (cl.first_command == "IF")
```

```
        {
          address += count_commands (cl.first_command, cl.first_expression);
          address += count_commands (cl.second_command, cl.second_expression);
          continue;
        }
      address += count_commands (cl.first_command, cl.first_expression);
    }
  return addresses;
}
```

convert_basic_code_lines_to_assembler.cpp

```
#include "simplebasic.h"

string
convert_basic_code_lines_to_assembler (vector<code_line> &code_lines,
                                       map<char, int> &table,
                                       map<int, int> &addresses,
                                       map<char, pair<int, int> > &constants)
{
  string result = "";
  for (auto &cl : code_lines)
    {
      result += convert_basic_code_line_to_assembler (cl, table, addresses);
    }
  for (auto c = constants.rbegin (); c != constants.rend (); ++c)
    {
      result += int_to_address (c->second.first);
      result += " =   ";
      result += int_to_sc_number (c->second.second);
      result += "\n";
    }
  return result;
}
```

convert_if_to_assembler.cpp

```
#include "simplebasic.h"

string
convert_if_to_assembler (code_line &cl, map<char, int> &table,
                         map<int, int> &addresses)
{
  string result = "";
  int address = cl.assembler_line_number;
  int left;
  int right;
  char left_operand;
  char right_operand;
  char sign;
  int jump_to;
  for (auto &ad : addresses)
    {
      if (ad.second > address)
        {
          jump_to = ad.second;
          break;
        }
    }
  tie (left, right) = is_there_numbers_in_if (cl.first_expression);
  if (left != 100000)
    {
      result += int_to_address (address++);
```

```
        result += " =   ";
        result += int_to_sc_number (left);
      }
    if (right != 100000)
      {
        result += int_to_address (address++);
        result += " =   ";
        result += int_to_sc_number (right);
      }

    result += int_to_address (address++);
    result += " LOAD   ";
    if (left != 100000)
      {
        if (right != 100000)
          result += int_to_address (address - 3);
        else
          result += int_to_address (address - 2);
      }
    else
      {
        left_operand += cl.first_expression[0];
        result += int_to_address (table[left_operand]);
      }
    result += "\n";

    result += int_to_address (address++);
    result += " SUB    ";
    if (right != 100000)
      result += int_to_address (address - 3);
    else
      {
        right_operand
            =  cl.first_expression[cl.first_expression.find_first_of  ("<>=")  +
  1];
        result += int_to_address (table[right_operand]);
      }
    result += "\n";
    sign = cl.first_expression[cl.first_expression.find_first_of ("<>=")];
    switch (sign)
      {
      case ('<'):
        result += int_to_address (address++);
        result += " JZ     ";
        result += int_to_address (jump_to);
        result += "\n";
        result += int_to_address (address++);
        result += " JNS    ";
        result += int_to_address (jump_to);
        result += "\n";
        break;
      case ('>'):
        result += int_to_address (address++);
        result += " JZ     ";
        result += int_to_address (jump_to);
        result += "\n";
        result += int_to_address (address++);
        result += " JNEG   ";
        result += int_to_address (jump_to);
        result += "\n";
        break;
      case ('='):
        result += int_to_address (address++);
        result += " JNEG   ";
        result += int_to_address (jump_to);
        result += "\n";
        result += int_to_address (address++);
        result += " JNS    ";
        result += int_to_address (jump_to);
        result += "\n";
        break;
      }
    if (cl.second_command == "LET")
      {
        result += convert_let_to_assembler (address, cl.postfix_expression,
                                            cl.expression_result, table);
      }
    else if (cl.second_command == "INPUT")
      {
        result += int_to_address (address);
        result += " READ   ";
        result += int_to_address (table[cl.operand[0]]);
        result += '\n';
      }
    else if (cl.second_command == "OUTPUT")
      {
        result += int_to_address (address);
        result += " WRITE  ";
        result += int_to_address (table[cl.operand[0]]);
        result += '\n';
      }
    else if (cl.second_command == "GOTO")
      {
        result += int_to_address (address);
        result += " JUMP   ";
        result += int_to_address (addresses[stoi (cl.operand)]);
        result += '\n';
      }
    return result;
  }

  convert_infix_to_postfix.cpp
  #include "simplebasic.h"

  void
  convert_infix_to_postfix (vector<code_line> &code_lines,
                            map<char, pair<int, int> > &constants)
  {
    for (auto &cl : code_lines)
      {
        if (cl.first_command == "LET")
          {
            cl.split_expression ();
            cl.postfix_expression
                = infix_to_postfix (cl.first_expression, constants);
          }
        if (cl.second_command == "LET")
          {
            cl.split_expression ();
            cl.postfix_expression
                = infix_to_postfix (cl.second_expression, constants);
          }
      }
  }

  convert_let_to_assembler.cpp
  #include "simplebasic.h"

  string
  convert_let_to_assembler (int address, string &expression,
                            string &expression_result, map<char, int> &table)
  {
    int n = 0;

    string left;
    string right;
    char sign;

    string result = "";
    for (auto c : expression)
      {
        if (isalpha (c))
          n++;
      }
    if (n == 1)
      {
        result += int_to_address (address++);
        result += " LOAD   ";
        result += int_to_address (table[expression[0]]);
        result += "\n";
        result += int_to_address (address++);
        result += " STORE  ";
        result += int_to_address (table[expression_result[0]]);
        result += "\n";
        return result;
      }
    n--;
    for (int i = 0; i < n; i++)
      {
        tie (left, right, sign)
            = get_one_operation (expression, expression_result);

        result += int_to_address (address++);
        result += " LOAD   ";
        result += int_to_address (table[left[0]]);
        result += "\n";

        result += int_to_address (address++);
        switch (sign)
          {
          case ('+'):
            result += " ADD    ";
            break;
          case ('-'):
            result += " SUB    ";
            break;
          case ('*'):
            result += " MUL    ";
            break;
          case ('/'):
            result += " DIVIDE ";
            break;
          }
        result += int_to_address (table[right[0]]);
        result += "\n";

        result += int_to_address (address++);
        result += " STORE  ";
        result += int_to_address (table[expression_result[0]]);
        result += "\n";
      }
    return result;
  }
```

count_commands.cpp

```
#include <string>

using namespace std;

int
count_commands (string &command, string &expression)
{
  bool inNumber = false;
  int count = 0;
  bool there_is_expression = false;
  if (command == "REM")
    return 0;
  if (command == "GOTO" || command == "INPUT" || command == "OUTPUT"
      || command == "END")
    return 1;
  if (command == "IF")
    {
      for (char c : expression)
        {
          if (isdigit (c))
            {
              if (!inNumber)
                {
                  inNumber = true;
                  count++;
                }
            }
          else
            {
              inNumber = false;
            }
        }
      return count + 4;
    }
  if (command == "LET")
    {
      for (char c : expression)
        {
          if (isalpha (c))
            count += 3;
          if (c == '+' || c == '-' || c == '*' || c == '/')
            there_is_expression = true;
        }
      if (there_is_expression)
        return count - 3;
      else
        return 2;
    }
  return 0;
```

```
}

get_one_operation.cpp
#include "simplebasic.h"

#include <string>
#include <tuple>

using namespace std;

tuple<string, string, char>
get_one_operation (string &expression, string &expression_result)
{
  bool there_is_expression = false;
  for (char c : expression)
    {
      if (c == '+' || c == '-' || c == '*' || c == '/')
        {
          there_is_expression = true;
          break;
        }
    }
  if (!there_is_expression)
    return make_tuple ("", "", 0);

  size_t position = expression.find_first_of ("+-*/");
  char sign = expression[position];
  string right_operand = take_operand (expression);
  string left_operand = take_operand (expression);
  position = expression.find_first_of ("+-*/");
  expression[position] = expression_result[0];
  return make_tuple (left_operand, right_operand, sign);
}

get_symbols_from_expression.cpp
#include <map>
#include <string>

using namespace std;

int
get_symbols_from_expression (map<char, int> &table, string &expression,
                             int address)
{
  for (char c : expression)
    if (isupper (c))
      if (table.count (c) == 0)
        table[c] = address--;
  return address;
}

give_addresses_to_constants.cpp
#include <map>
#include <utility>

using namespace std;

void
give_addresses_to_constants (map<char, int> &table,
                             map<char, pair<int, int> > &constants)
{
  auto last_address = table.end ();
  --last_address;
  int address = last_address->second - 1;
  for (auto &c : constants)
    c.second.first = address--;
}

infix_to_postfix.cpp
#include "simplebasic.h"

#include <map>
#include <stack>
#include <string>
#include <utility>

using namespace std;

string
infix_to_postfix (string &expression, map<char, pair<int, int> > &constants)
{
  std::stack<char> s;
  std::stack<int> s_prec;
  int increment = 0;
  int prec = 0;
  string result;
  char constant = 'a';
  int exist = false;
  int int_number;
  string number = "";
  if (constants.size () != 0)
    {
      auto last_constant = constants.end ();
      --last_constant;
      constant = last_constant->first + 1;
    }
  for (char c : expression)
    {
      if (isdigit (c))
        {
          number += c;
          continue;
        }
      if (number != "")
        {
          int_number = stoi (number);
          for (auto &c : constants)
            {
              if (c.second.second == int_number)
                {
                  constant = c.first;
                  exist = true;
                }
            }
          if (!exist)
            {
              if (constants.size () != 0)
                {
                  auto last_constant = constants.end ();
                  --last_constant;
                  constant = last_constant->first + 1;
                }
              constants[constant] = make_pair (0, int_number);
            }
          result += constant;

          exist = false;
          number = "";
        }
      prec = precedence (c);
      switch (prec)
        {
        case 0:
          result += c;
          break;
        case 3:
          increment += 2;
          break;
        case 4:
          increment -= 2;
          break;
        default:
          prec += increment;
          if (s.size () == 0)
            {
              s.push (c);
              s_prec.push (prec);
            }
          else if (s_prec.top () >= prec)
            {
              while (s.size () > 0)
                {
                  result += s.top ();
                  s.pop ();
                  s_prec.pop ();
                }
              s.push (c);
              s_prec.push (prec);
            }
          else
            {
              s.push (c);
              s_prec.push (prec);
            }
          break;
        }
    }

  if (number != "")
    {
      int_number = stoi (number);
      for (auto &c : constants)
        {
          if (c.second.second == int_number)
            {
              constant = c.first;
              exist = true;
            }
        }
      if (!exist)
        {
          if (constants.size () != 0)
            {
              auto last_constant = constants.end ();
              --last_constant;
              constant = last_constant->first + 1;
            }
          constants[constant] = make_pair (0, int_number);
        }
      result += constant++;

      exist = false;
      number = "";
    }

  while (s.size () > 0)
    {
      result += s.top ();
      s.pop ();
      s_prec.pop ();
    }
  return result;
}
```

```
int_to_address.cpp
#include <iomanip>
#include <sstream>
#include <string>

using namespace std;

string
int_to_address (int address)
{
  ostringstream oss;
  oss << setfill ('0') << setw (2) << address;
  return oss.str ();
}
```

```
int_to_sc_number.cpp
#include <iomanip>
#include <sstream>
#include <string>

using namespace std;

string
int_to_sc_number (int value)
{
  stringstream stream;
  stream << hex << uppercase << setfill ('0') << setw (4) << value;
  if (value < 0)
    {
      return '-' + stream.str ();
    }
  return '+' + stream.str ();
}
```

```
is_there_numbers_in_if.cpp
#include <string>
#include <utility>

using namespace std;
```

```
pair<int, int>
is_there_numbers_in_if (string &expression)
{
  int left = 100000;
  int right = 100000;
  int index = 0;
  string expression_copy = expression;

  if (expression_copy[0] == '-' || isdigit (expression_copy[0]))
    {
      left = stoi (expression_copy);
    }

  index = expression_copy.find_first_of ("><=");
  expression_copy.erase (0, index);

  if (expression[0] == '-' || isdigit (expression[0]))
    {
      left = stoi (expression_copy);
    }
  return make_pair (left, right);
}
```

```
main.cpp
#include "simplebasic.h"

#include <fstream>
#include <iostream>

int
main (int argc, char *argv[])
{
  if (argc != 3)
    {
      std::cerr << "Usage: " << argv[0] << " <input file> <output file>"
                << std::endl;
      return 1;
    }

  vector<code_line> code_lines = read_file (argv[1]);

  if (check_code_lines_empty (code_lines))
    return 1;
  if (check_code_lines_spaces (code_lines))
    return 1;

  if (code_lines_to_tokens (code_lines))
    return 1;

  map<char, int> table = make_symbolic_table (code_lines);

  map<char, pair<int, int> > constants;

  convert_infix_to_postfix (code_lines, constants);

  give_addresses_to_constants (table, constants);

  map<int, int> addresses
      = convert_basic_code_lines_adresses_to_assembler (code_lines);

  merge_tables (table, constants);
  string result = convert_basic_code_lines_to_assembler (code_lines, table,
                                                         addresses,
constants);

  ofstream out (argv[2]);
  if (!out.is_open ())
    {
      cerr << "Can't open file \"" << argv[2] << "\"" << endl;
      return 1;
    }
  out << result;
  out.close ();
  cout << "Success!" << endl;
}
```

```
make_symbolic_table.cpp
#include "simplebasic.h"

#include <map>
#include <string>
#include <vector>

map<char, int>
make_symbolic_table (vector<code_line> &code_lines)
{
  int address = 127;
  map<char, int> table;
  string first_expression;
  string second_expression;
  for (auto &cl : code_lines)
    {
      if (cl.first_command == "REM" || cl.first_command == "GOTO"
          || cl.first_command == "OUTPUT")
        continue;
      if (cl.first_command == "INPUT" || cl.second_command == "INPUT")
        {
          table[cl.operand[0]] = address--;
          continue;
        }
      if (cl.first_command == "IF")
        {
          address = get_symbols_from_expression (table, cl.first_expression,
                                                 address);
        }
      if (cl.first_command == "LET")
        address = get_symbols_from_expression (table, cl.first_expression,
                                               address);
      if (cl.second_command == "LET")
        address = get_symbols_from_expression (table, cl.second_expression,
                                               address);
    }
  return table;
}
```

```
makefile
APP_NAME = sbt

SRC_EXT = cpp
CC = g++
CFLAGS = -Wall -Wextra -Werror
```

```
CPPFLAGS = -MMD

APP_SOURCES = $(wildcard *.$(SRC_EXT))
APP_OBJECTS := $(patsubst %.$(SRC_EXT),%.o,$(APP_SOURCES))

DEPS = $(APP_OBJECTS:.o=.d)

.PHONY: all
all: $(APP_NAME)

-include $(DEPS)

$(APP_NAME): $(APP_OBJECTS)
	$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@

%.o: %.$(SRC_EXT)
	$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean
clean:
	rm -rf $(APP_OBJECTS) $(DEPS) $(APP_NAME)
```

```
merge_tables.cpp
#include <map>
#include <utility>

using namespace std;

void
merge_tables (map<char, int> &table, map<char, pair<int, int> > &constants)
{
  for (auto &c : constants)
    {
      table[c.first] = c.second.first;
    }
}
```

```
precedence.cpp
int
precedence (char op)
{
  if (op == '+' || op == '-')
    return 1;
  if (op == '*' || op == '/')
    return 2;
  if (op == '(')
    return 3;
  if (op == ')')
    return 4;
  return 0;
}
```

```
read_file.cpp
#include "simplebasic.h"

#include <fstream>
#include <iostream>
#include <vector>

vector<code_line>
read_file (string filename)
```

```
{
  ifstream file (filename);
  if (!file.is_open ())
    {
      cerr << "Can't open file \"" << filename << "\"" << endl;
      exit (1);
    }
  string line;
  vector<code_line> code_lines;
  while (getline (file, line))
    {
      code_line cl;
      cl.basic_code = line;
      code_lines.push_back (cl);
    }
  file.close ();
  return code_lines;
}

simplebasic.h
#pragma once

#include <map>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

class code_line
{
public:
  string basic_code;
  string first_command;
  string second_command;
  string first_expression;
  string second_expression;
  string operand;
  string postfix_expression;
  string expression_result;
  int line_number;
  int assembler_line_number;

  code_line ()
  {
    basic_code = "";
    first_command = "";
    second_command = "";
    first_expression = "";
    second_expression = "";
    operand = "";
    postfix_expression = "";
    expression_result = "";
    line_number = 0;
    assembler_line_number = 0;
  }

  bool
  is_empty ()
  {
    return basic_code.empty ();
```

```
}

bool
is_there_any_spaces ()
{
  return basic_code.find (' ') != string::npos;
}

bool
token_is_command (string token)
{
  return token == "REM" || token == "INPUT" || token == "OUTPUT"
         || token == "GOTO" || token == "IF" || token == "LET"
         || token == "END";
}

int
split ()
{
  istringstream iss (basic_code);
  string token;
  iss >> token;
  if (token.empty ())
    return 1;
  line_number = stoi (token);

  iss >> token;
  if (!token_is_command (token))
    return 1;
  first_command = token;
  if (first_command == "REM")
    {
      return 0;
    }
  else if (first_command == "INPUT" || first_command == "OUTPUT"
           || first_command == "GOTO")
    {
      iss >> token;
      if (token.empty ())
        return 1;
      operand = token;
      return 0;
    }
  else if (first_command == "IF")
    {
      iss >> token;
      while (!token_is_command (token))
        {
          first_expression += token;
          iss >> token;
        }
      second_command = token;
      if (second_command == "INPUT" || second_command == "OUTPUT"
          || second_command == "GOTO")
        {
          iss >> token;
          if (token.empty ())
            return 1;
          operand = token;
          return 0;
        }
        else if (second_command == "LET")
          {
            token = "";
            iss >> token;
            while (!token.empty ())
              {
                second_expression += token;
                token = "";
                iss >> token;
              }
            return 0;
          }
      }
    else if (first_command == "LET")
      {
        token = "";
        iss >> token;
        while (!token.empty ())
          {
            first_expression += token;
            token = "";
            iss >> token;
          }
        return 0;
      }
    else if (first_command == "END")
      {
        return 0;
      }
    return 1;
  }

  void
  split_expression ()
  {
    if (first_command == "LET")
      {
        expression_result += first_expression[0];
        first_expression.erase (0, 2);
      }
    if (second_command == "LET")
      {
        expression_result += second_expression[0];
        second_expression.erase (0, 2);
      }
  }
};

vector<code_line> read_file (string filename);
int check_code_lines_empty (vector<code_line> &code_lines);
int check_code_lines_spaces (vector<code_line> &code_lines);
int code_lines_to_tokens (vector<code_line> &code_lines);
int get_symbols_from_expression (map<char, int> &table, string &expression,
                                 int address);
map<char, int> make_symbolic_table (vector<code_line> &code_lines);
int precedence (char op);
string infix_to_postfix (string &expression,
                         map<char, pair<int, int> > &constants);
void convert_infix_to_postfix (vector<code_line> &code_lines,
                               map<char, pair<int, int> > &constants);
void give_addresses_to_constants (map<char, int> &table,
                                  map<char, pair<int, int> > &constants);
int count_commands (string &command, string &expression);
map<int, int>
convert_basic_code_lines_adresses_to_assembler              (vector<code_line>
&code_lines);
string int_to_address (int address);
string take_operand (string &expression);
tuple<string, string, char> get_one_operation (string &expression,
                                               string &expression_result);
void merge_tables (map<char, int> &table,
                   map<char, pair<int, int> > &constants);
string convert_let_to_assembler (int address, string &expression,
                                 string &expression_result,
                                 map<char, int> &table);
pair<int, int> is_there_numbers_in_if (string &expression);
string int_to_sc_number (int value);
string convert_if_to_assembler (code_line &cl, map<char, int> &table,
                                map<int, int> &addresses);
string convert_basic_code_line_to_assembler (code_line &cl,
                                             map<char, int> &table,
                                             map<int, int> &addresses);
string convert_basic_code_lines_to_assembler (
    vector<code_line> &code_lines, map<char, int> &table,
    map<int, int> &addresses, map<char, pair<int, int> > &constants);

take_operand.cpp
#include <string>

using namespace std;

string
take_operand (string &expression)
{
  size_t right_position = expression.find_first_of ("+-*/");
  size_t left_position = right_position - 1;
  string operand
      = expression.substr (left_position, right_position - left_position);
  expression.erase (left_position, right_position - left_position);
  return operand;
}
```

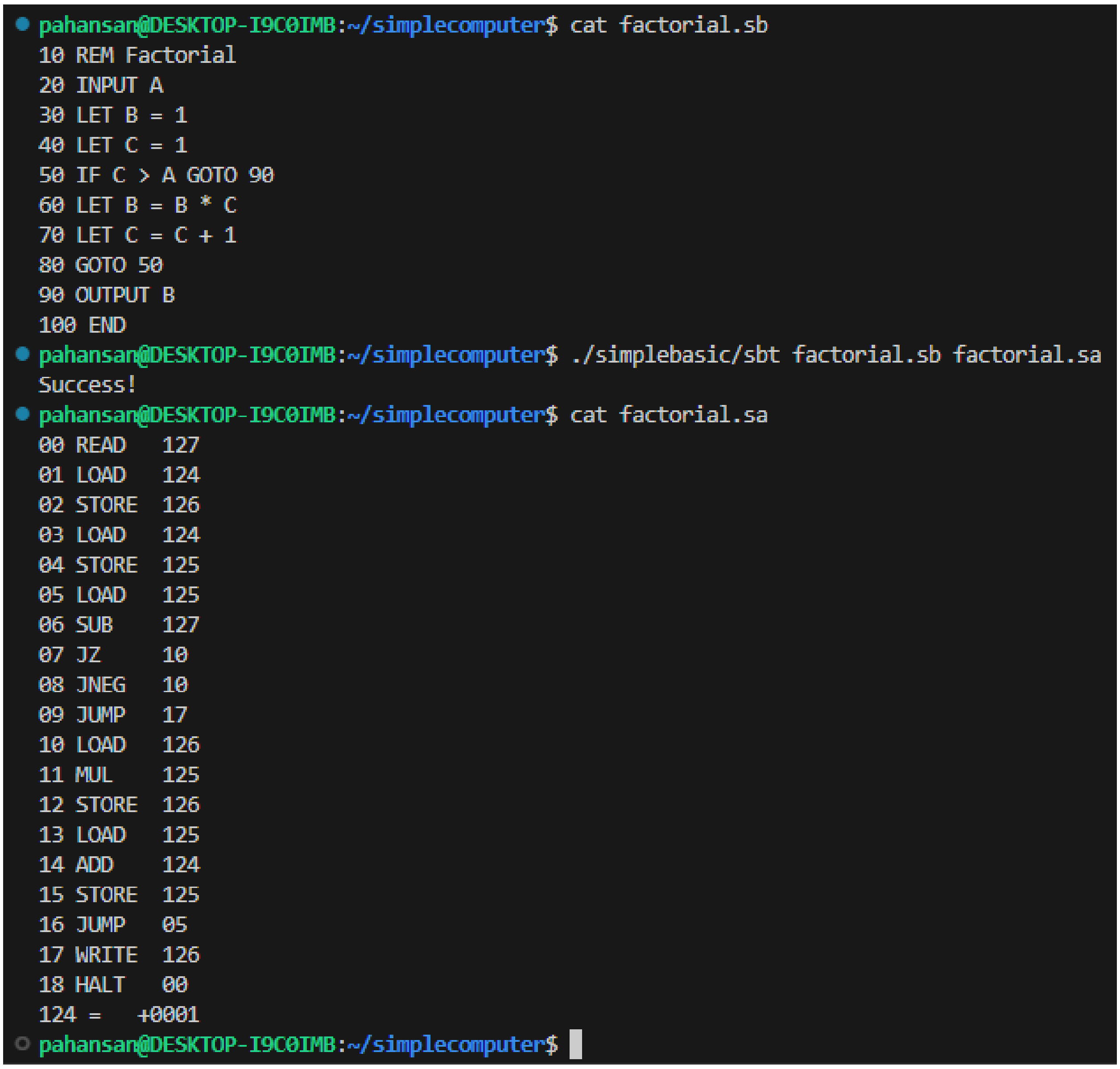
simplecomputer

```
makefile
export CFLAGS = -Wall -Wextra -Werror
export CPPFLAGS = -I$(PWD)/include -L$(PWD)/mySimpleComputer -L$(PWD)/myTerm -
L$(PWD)/myBigChars -L$(PWD)/myReadKey -MMD
export CC = gcc

SUBDIRS = myTerm mySimpleComputer myBigChars myReadKey console
```

```
LFLAGS = -lmySimpleComputer -lmyTerm -lmyBigChars -lmyReadKey
LIBS = $(PWD)/mySimpleComputer/libmySimpleComputer.a $(PWD)/myTerm/libmyTerm.a
$(PWD)/myBigChars/libmyBigChars.a $(PWD)/myReadKey/libmyReadKey.a

export LFLAGS
export LIBS

all: $(SUBDIRS)

$(SUBDIRS):
	@$(MAKE) --no-print-directory -C $@

clean:
	@for dir in $(SUBDIRS); do \
		$(MAKE) --no-print-directory -C $$dir clean; \
	done

.PHONY: all clean $(SUBDIRS)

format:
	find . -type f -name *.[ch] | xargs clang-format --style GNU -i -verbose
```

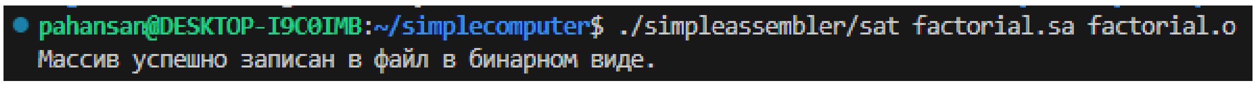
РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Трансляция с языка simple basic в simple assembler



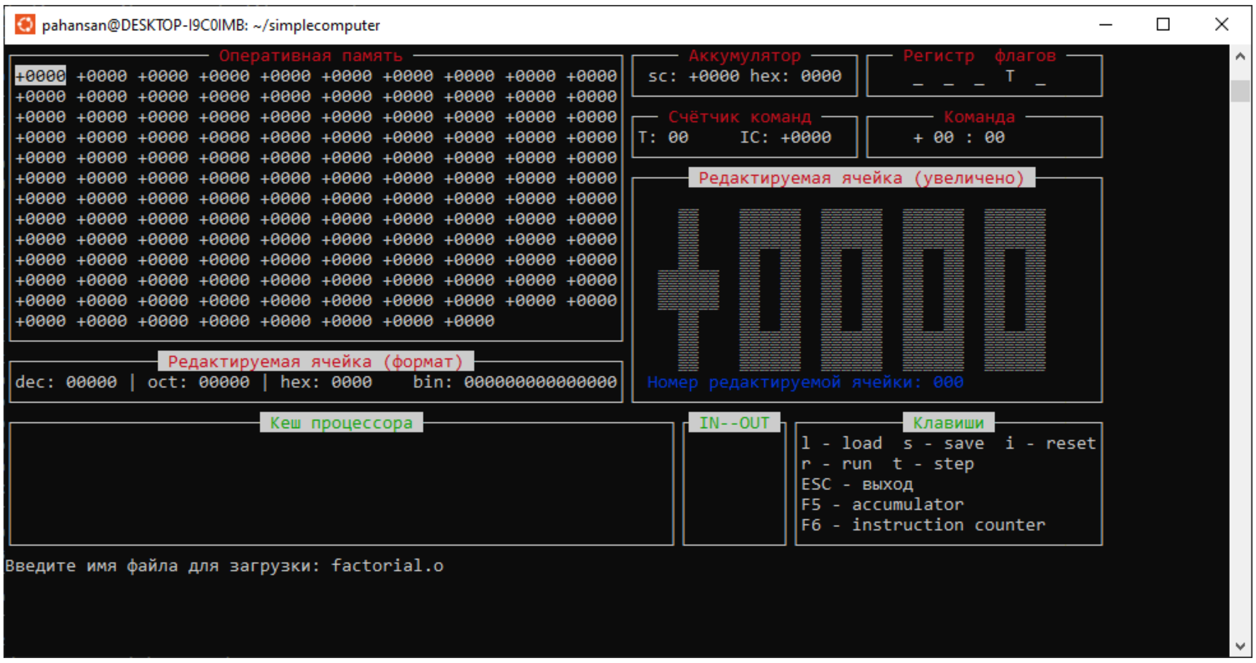
```
pahansan@DESKTOP-I9C0IMB:~/simplecomputer$ cat factorial.sb
10 REM Factorial
20 INPUT A
30 LET B = 1
40 LET C = 1
50 IF C > A GOTO 90
60 LET B = B * C
70 LET C = C + 1
80 GOTO 50
90 OUTPUT B
100 END
pahansan@DESKTOP-I9C0IMB:~/simplecomputer$ ./simplebasic/sbt factorial.sb factorial.sa
Success!
pahansan@DESKTOP-I9C0IMB:~/simplecomputer$ cat factorial.sa
00 READ   127
01 LOAD   124
02 STORE  126
03 LOAD   124
04 STORE  125
05 LOAD   125
06 SUB    127
07 JZ     10
08 JNEG   10
09 JUMP   17
10 LOAD   126
11 MUL    125
12 STORE  126
13 LOAD   125
14 ADD    124
15 STORE  125
16 JUMP   05
17 WRITE  126
18 HALT   00
124 =   +0001
pahansan@DESKTOP-I9C0IMB:~/simplecomputer$
```

Трансляция с языка simple assembler в образ оперативной памяти

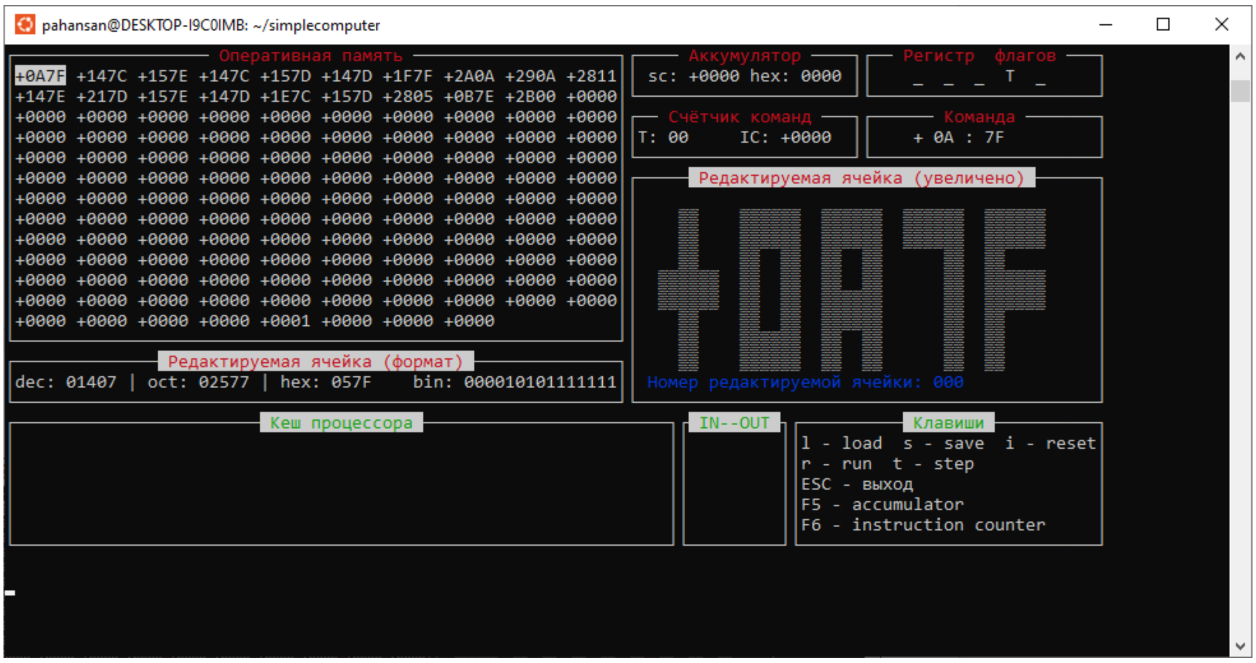


```
pahansan@DESKTOP-I9C0IMB:~/simplecomputer$ ./simpleassembler/sat factorial.sa factorial.o
Массив успешно записан в файл в бинарном виде.
```

Загрузка образа оперативной памяти



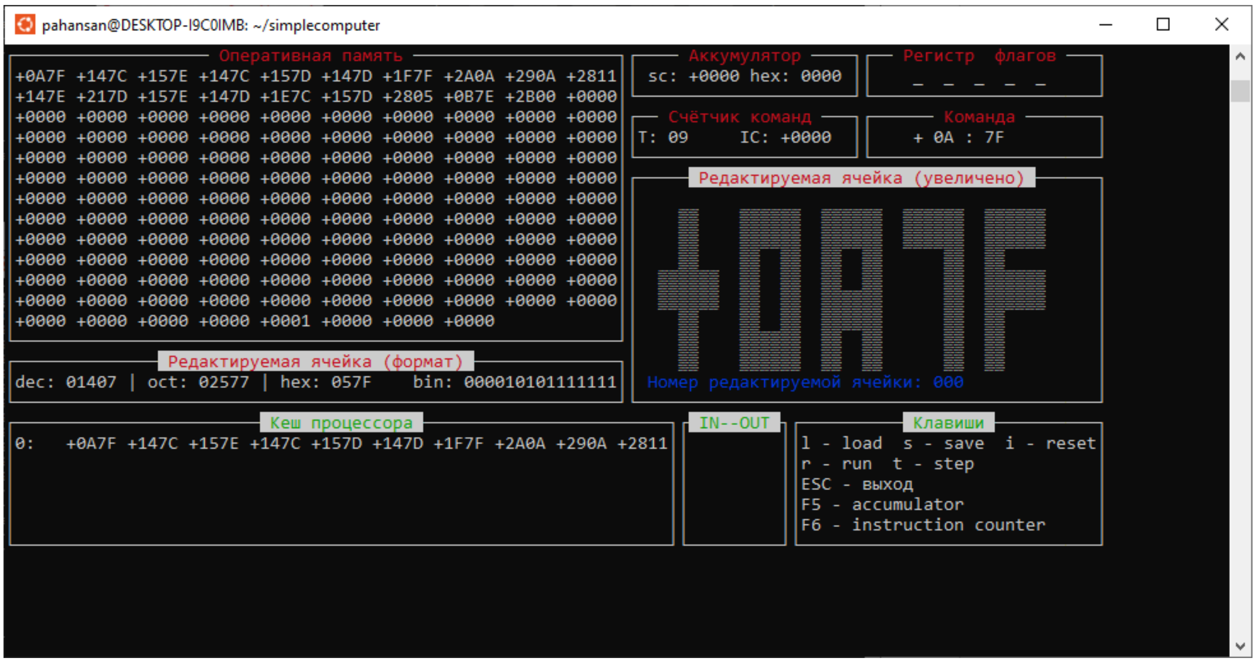
```
pahansan@DESKTOP-I9C0IMB: ~/simplecomputer

Оперативная память
+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 +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 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
+0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

Аккумулятор          sc: +0000 hex: 0000
Регистр флагов       _ _ _ T _
Счётчик команд       T: 00     IC: +0000
Команда              + 00 : 00

Редактируемая ячейка (увеличено)
+0000
Номер редактируемой ячейки: 000

Редактируемая ячейка (формат)
dec: 00000 | oct: 00000 | hex: 0000     bin: 000000000000000

Кеш процессора
IN--OUT
Клавиши
l - load  s - save  i - reset
r - run  t - step
ESC - выход
F5 - accumulator
F6 - instruction counter

Введите имя файла для загрузки: factorial.o
```



```
pahansan@DESKTOP-I9C0IMB: ~/simplecomputer

Оперативная память
+0A7F +147C +157E +147C +157D +147D +1F7F +2A0A +290A +2811
+147E +217D +157E +147D +1E7C +157D +2805 +0B7E +2B00 +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 +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 +0001 +0000 +0000 +0000

Аккумулятор          sc: +0000 hex: 0000
Регистр флагов       _ _ _ T _
Счётчик команд       T: 00     IC: +0000
Команда              + 0A : 7F

Редактируемая ячейка (увеличено)
+0A7F
Номер редактируемой ячейки: 000

Редактируемая ячейка (формат)
dec: 01407 | oct: 02577 | hex: 057F     bin: 000010101111111

Кеш процессора
IN--OUT
Клавиши
l - load  s - save  i - reset
r - run  t - step
ESC - выход
F5 - accumulator
F6 - instruction counter
```

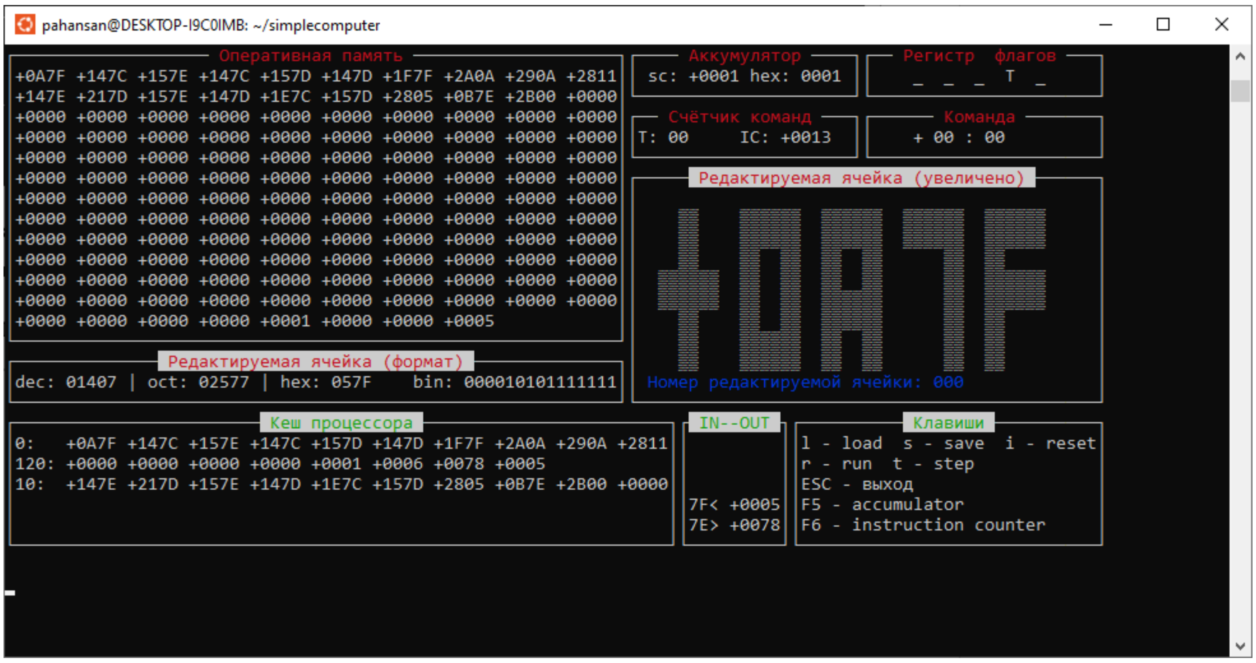
Память успешно загрузилась. Программа, которая записана в памяти, может вычислить значение факториала от 0 до 7.

При запуске программы произошёл cache miss, поэтому строка из оперативной памяти была загружена в кэш процессора и счётчик тактов простоя установился в 10.



```
pahansan@DESKTOP-I9C0IMB: ~/simplecomputer

Оперативная память
+0A7F +147C +157E +147C +157D +147D +1F7F +2A0A +290A +2811
+147E +217D +157E +147D +1E7C +157D +2805 +0B7E +2B00 +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 +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 +0001 +0000 +0000 +0000

Аккумулятор
sc: +0000 hex: 0000

Регистр  флагов
_ _ _ _ _

Счётчик команд
T: 09      IC: +0000

Команда
+ 0A : 7F

Редактируемая ячейка (увеличено)
Номер редактируемой ячейки: 000

Редактируемая ячейка (формат)
dec: 01407 | oct: 02577 | hex: 057F     bin: 000010101111111

Кеш процессора
0:   +0A7F +147C +157E +147C +157D +147D +1F7F +2A0A +290A +2811

IN--OUT

Клавиши
l - load  s - save  i - reset
r - run  t - step
ESC - выход
F5 - accumulator
F6 - instruction counter
```

Результат выполнения программы



```
pahansan@DESKTOP-I9C0IMB: ~/simplecomputer

Оперативная память
+0A7F +147C +157E +147C +157D +147D +1F7F +2A0A +290A +2811
+147E +217D +157E +147D +1E7C +157D +2805 +0B7E +2B00 +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 +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 +0001 +0000 +0000 +0005

Аккумулятор
sc: +0001 hex: 0001

Регистр  флагов
_ _ _ T _

Счётчик команд
T: 00      IC: +0013

Команда
+ 00 : 00

Редактируемая ячейка (увеличено)
Номер редактируемой ячейки: 000

Редактируемая ячейка (формат)
dec: 01407 | oct: 02577 | hex: 057F     bin: 000010101111111

Кеш процессора
0:   +0A7F +147C +157E +147C +157D +147D +1F7F +2A0A +290A +2811
120: +0000 +0000 +0000 +0000 +0001 +0006 +0078 +0005
10:  +147E +217D +157E +147D +1E7C +157D +2805 +0B7E +2B00 +0000

IN--OUT
7F< +0005
7E> +0078

Клавиши
l - load  s - save  i - reset
r - run  t - step
ESC - выход
F5 - accumulator
F6 - instruction counter
```

Программа правильно вычислила факториал 5, запрошенный пользователем. 78 в шестнадцатеричной системе счисления – это 120 в десятичной. Видно также, что все данные, которые потребовались в ходе выполнения программы, были загружены из оперативной памяти в кэш процессора.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была доработано модель Simple Computer таким образом, чтобы имитировать работу кэша процессора при выполнении программ. В модели был сымитирован алгоритм замещения кэша LRU, при котором в случае переполнения кэша из него вытесняется самая невостребованная строка.

Помимо работы кэша также были реализованы 2 транслятора: транслятор с языка simple assembler, который позволяет превратить программу на simple assembler в образ оперативной памяти simple computer, и транслятор simple basic, который превращает программу, написанную на языке более высокого уровня simple basic в программу на языке simple assembler.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Мамойленко С.Н., Молдованова О.В. ЭВМ и периферийные устройства: Учебное пособие. – Новосибирск: СибГУТИ, 2012. – 106 с.