

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Сибирский Государственный Университет Телекоммуникаций и
Информатики

Факультет Информатики и Вычислительной Техники

Кафедра вычислительных систем

Курсовая работа

По дисциплине «Архитектура ЭВМ»

Вариант 25 (SUBC)

**Программная модель простейшей вычислительной машины Simple
Computer. Транслятор с языка Simple Assembler и Simple Basic.**

Работу выполнил:

Студент 2 курса

группы ИП-114

Яворский Д.И.

Научный

руководитель:

Майданов Ю.С.

Новосибирск, 2023

Содержание:

Задание курсовой работы.....	3
Архитектура вычислительной машины Simple Computer.....	6
Постановка задачи исследования.....	10
Блок-схемы используемых алгоритмов	10
Программная реализация.....	13
Результаты проведенного исследования.....	16
Приложение (листинг программы)	17
Выводы	74
Ссылка на проект	75

Задание курсовой работы:

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

Обработка команд центральным процессором

Для выполнения программ моделью Simple Computer необходимо реализовать две функции:

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

Обработку команд осуществляет устройство управления. Функция CU вызывается либо обработчиком сигнала от системного таймера, если не установлен флаг «игнорирование тактовых импульсов», либо при нажатии на клавишу “t”. Алгоритм работы функции следующий:

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

Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления. Пример программы на Simple Assembler:

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

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

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

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.

В рамках выполнения практических заданий и курсового проектирования необходимо разработать программную модель простейшей вычислительной машины Simple Computer. Архитектура Simple Computer представлена ниже.

Для управления моделью (определения начальных состояний узлов Simple Computer, запуска программ на выполнения, отражения хода выполнения программ) требуется создать консоль (см. рисунок 1). Необходимо реализовать трансляторы с языков Simple Assembler и Simple Basic для программирования Simple Computer.

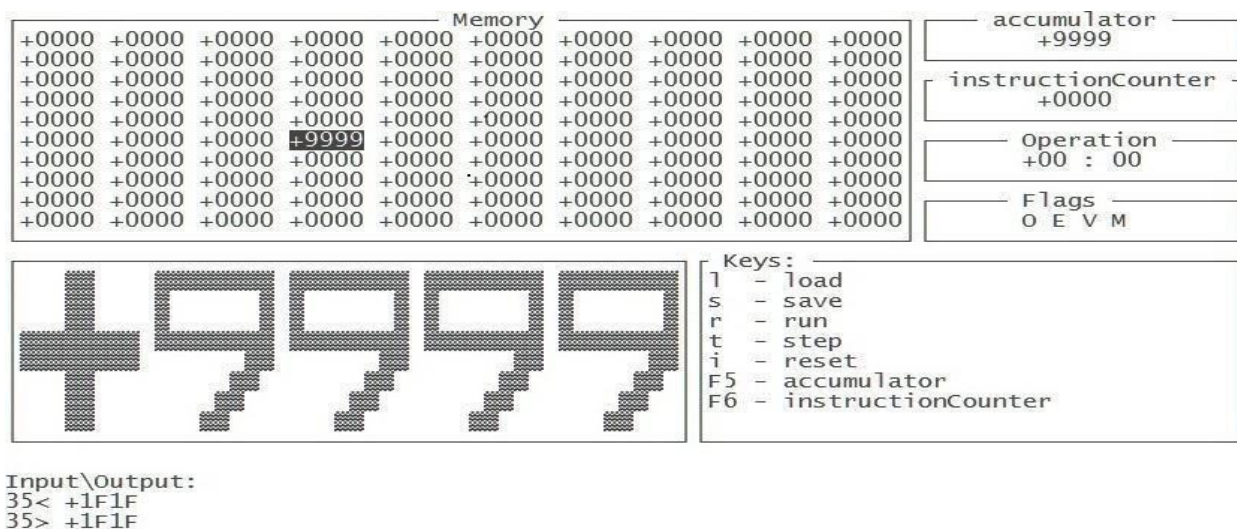


Рис. 1 – интерфейс консоли управления моделью Simple Computer

Архитектура вычислительной машины Simple Computer

Архитектура Simple Computer представлена на рисунке 2 и включает следующие функциональные блоки:

- оперативную память;
- внешние устройства;
- центральный процессор.



Рис. 2 – Архитектура вычислительной машины Simple Computer

Оперативная память

Оперативная память – это часть Simple Computer, где хранятся программа и данные. Память состоит из ячеек (массив), каждая из которых хранит 15 двоичных разрядов. Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является её адресом и задается 7-миразрядным числом. Simple Computer оборудован памятью из 100 ячеек (с адресами от 0 до 99₁₀).

Внешние устройства

Внешние устройства включают: клавиатуру и монитор, используемые для

взаимодействия с пользователем, системный таймер, задающий такты работы Simple Computer и кнопку «Reset», позволяющую сбросить Simple Computer в исходное состояние.

Центральный процессор

Выполнение программ осуществляется центральным процессором Simple Computer.

Процессор состоит из следующих функциональных блоков:

- регистры (аккумулятор, счетчик команд, регистр флагов);
- арифметико-логическое устройство (АЛУ);
- управляющее устройство (УУ);
- обработчик прерываний от внешних устройств (ОП);
- интерфейс доступа к оперативной памяти.

Регистры являются внутренней памятью процессора. Центральный процессор Simple Computer имеет: аккумулятор, используемый для временного хранения данных и результатов операций, счетчик команд, указывающий на адрес ячейки памяти, в которой хранится текущая выполняемая команда и регистр флагов, сигнализирующий об определённых событиях. Аккумулятор имеет разрядность 15 бит, счетчика команд – 7 бит. Регистр флагов содержит 5 разрядов: переполнение при выполнении операции, ошибка деления на 0, ошибка выхода за границы памяти, игнорирование тактовых импульсов, указана неверная команда.

Арифметико-логическое устройство (англ. arithmetic and logic unit, ALU) — блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде команды или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в аккумуляторе или может помещаться в оперативную память. В ходе выполнения операций АЛУ устанавливает значения флагов «деление на 0» и «переполнение». Управляющее устройство (англ. control unit, CU) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. В его функции входит: чтение текущей команды из памяти, её декодирование, передача номера команды и операнда в АЛУ, определение следующей выполняемой команды и реализации взаимодействий с клавиатурой и монитором. Выбор очередной команды из оперативной памяти производится по сигналу от системного таймера. Если установлен флаг «игнорирование тактовых импульсов», то эти сигналы устройством управления игнорируются. В ходе выполнения операций устройство управления устанавливает значения флагов «указана неверная команда» и «игнорирование тактовых импульсов».

Обработчик прерываний реагирует на сигналы от системного таймера и кнопки «Reset». При поступлении сигнала от кнопки «Reset» состояние процессора сбрасывается в начальное (значения всех регистров обнуляется и устанавливается флаг «игнорирование сигналов от таймера»). При поступлении сигнала от системного таймера, работать начинает устройство управления.

Система команд Simple Computer

Получив текущую команду из оперативной памяти, устройство управления декодирует её с целью определить номер функции, которую надо выполнить и операнд. Формат команды, следующий (см. рисунок 3): старший разряд содержит признак команды (0 – команда), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд. Коды операций, их назначение и обозначение в Simple Assembler и приведены в таблице 1.



Рис. 3 – Формат команды центрального процессора Simple Computer

Таблица 1. Команды центрального процессора Simple Computer

Операция		Значение
Обозначение	Код	
Операции ввода/вывода		
READ	10	Ввод с терминала в указанную ячейку памяти с контролем переполнения
WRITE	11	Вывод на терминал значение указанной ячейки памяти
Операции загрузки/выгрузки в аккумулятор		
LOAD	20	Загрузка в аккумулятор значения из указанного адреса памяти
STORE	21	Выгружает значение из аккумулятора по указанному адресу памяти
Арифметические операции		
ADD	30	Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе)
SUB	31	Вычитает из слова в аккумуляторе слово из указанной ячейки памяти (результат в аккумуляторе)
DIVIDE	32	Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
MUL	33	Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
Операции передачи управления		
JUMP	40	Переход к указанному адресу памяти
JNEG	41	Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число
JZ	42	Переход к указанному адресу памяти, если в аккумуляторе находится ноль
HALT	43	Останов, выполняется при завершении работы программы
Пользовательские функции		
SUBC	76	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)

Выполнение команд центральным процессором Simple Computer

Команды выполняются последовательно. Адрес ячейки памяти, в которой находится текущая выполняемая команда, задается в регистре «Счетчик команд».

Устройство управления запрашивает содержимое указанной ячейки памяти и декодирует его согласно используемому формату команд. Получив код операции, устройство управления определяет, является ли эта операция арифметико-логической. Если да, то выполнение операции передается в АЛУ. В противном случае операция выполняется устройством управления. Процедура выполняется до тех пор, пока флаг «останов» не будет равен 1.

Консоль управления

Интерфейс консоли управления представлен на рисунке 1. Он содержит следующие области:

- “Memory” – содержимое оперативной памяти Simple Computer.
- “Accumulator” – значение, находящееся в аккумуляторе;
- “instructionCounter” – значение регистра «счетчик команд»;
- “Operation” – результат декодирования операции;
- “Flags” – состояние регистра флагов («П» - переполнение при выполнении операции,
«0» - ошибка деления на 0, «М» - ошибка выхода за границы памяти,
«Т» - игнорирование тактовых импульсов, «Е» - указана неверная команда);
- “Cell” – значение выделенной ячейки памяти в области “Memory” (используется для редактирования);
- “Keys” – подсказка по функциональным клавишам;
- “Input/Output” – область, используемая Simple Computer в процессе выполнения программы для ввода информации с клавиатуры и вывода её на экран.

Содержимое ячеек памяти и регистров центрального процессора выводится в декодированном виде. При этом, знак «+» соответствует значению 0 в поле «признак команды», следующие две цифры – номер команды и затем операнд в шестнадцатеричной системе счисления.

Пользователь имеет возможность с помощью клавиш управления курсора выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу “F5”, пользователь может задать значение аккумулятору, “F6” – регистру «счетчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может, нажав на клавиши «l», «s» соответственно (после нажатия в поле Input/Output пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорировать такты таймера» в 0) можно с помощью клавиши “r”. В процессе выполнения программы, редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу “t”. Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу “i”.

Постановка задачи исследования

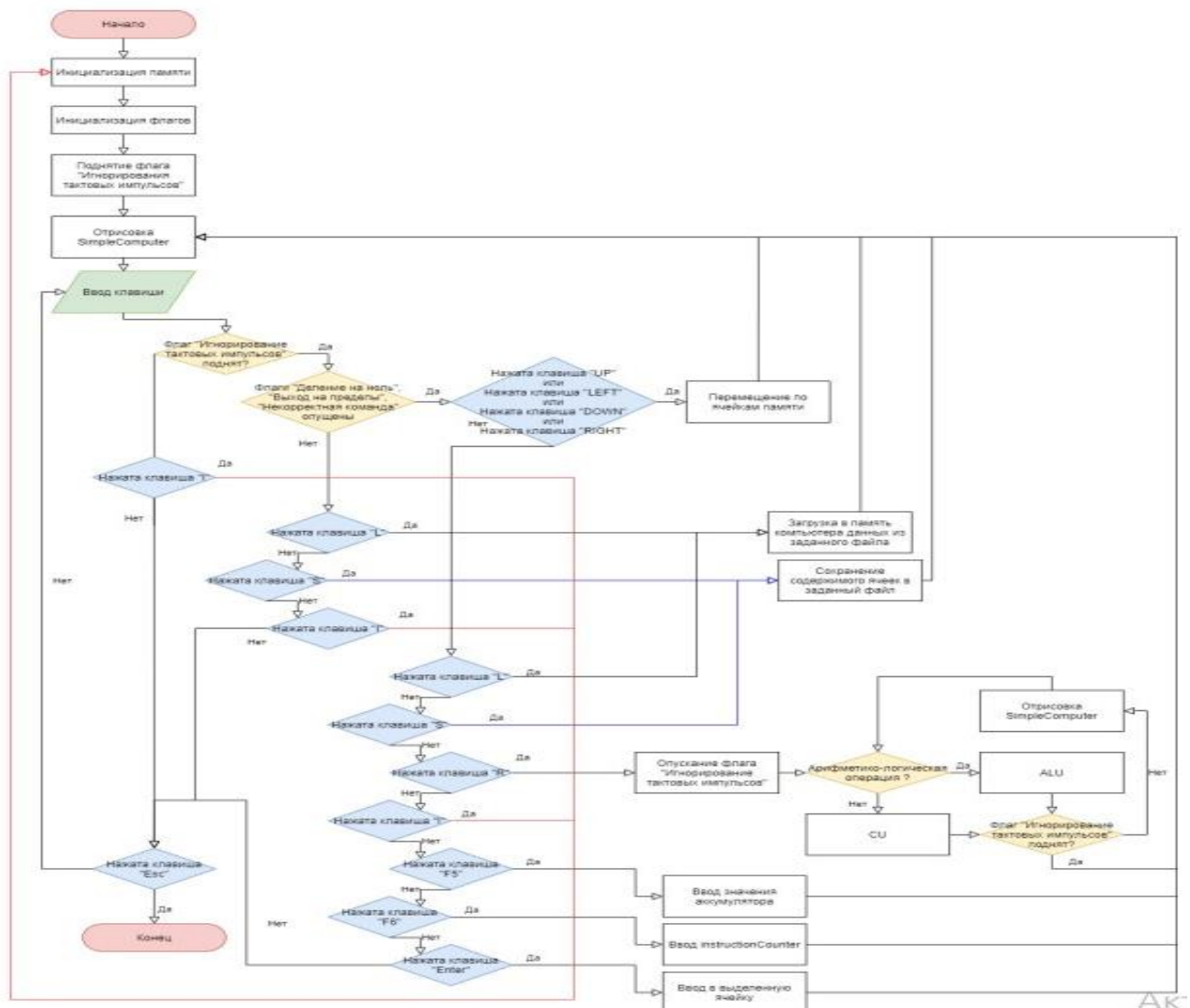
Курс “Архитектура ЭВМ” состоит из двух этапов:

- 1) Разработка Simple Computer;
- 2) Разработка трансляторов (с языка Simple Assembler и с языка Simple Basic).

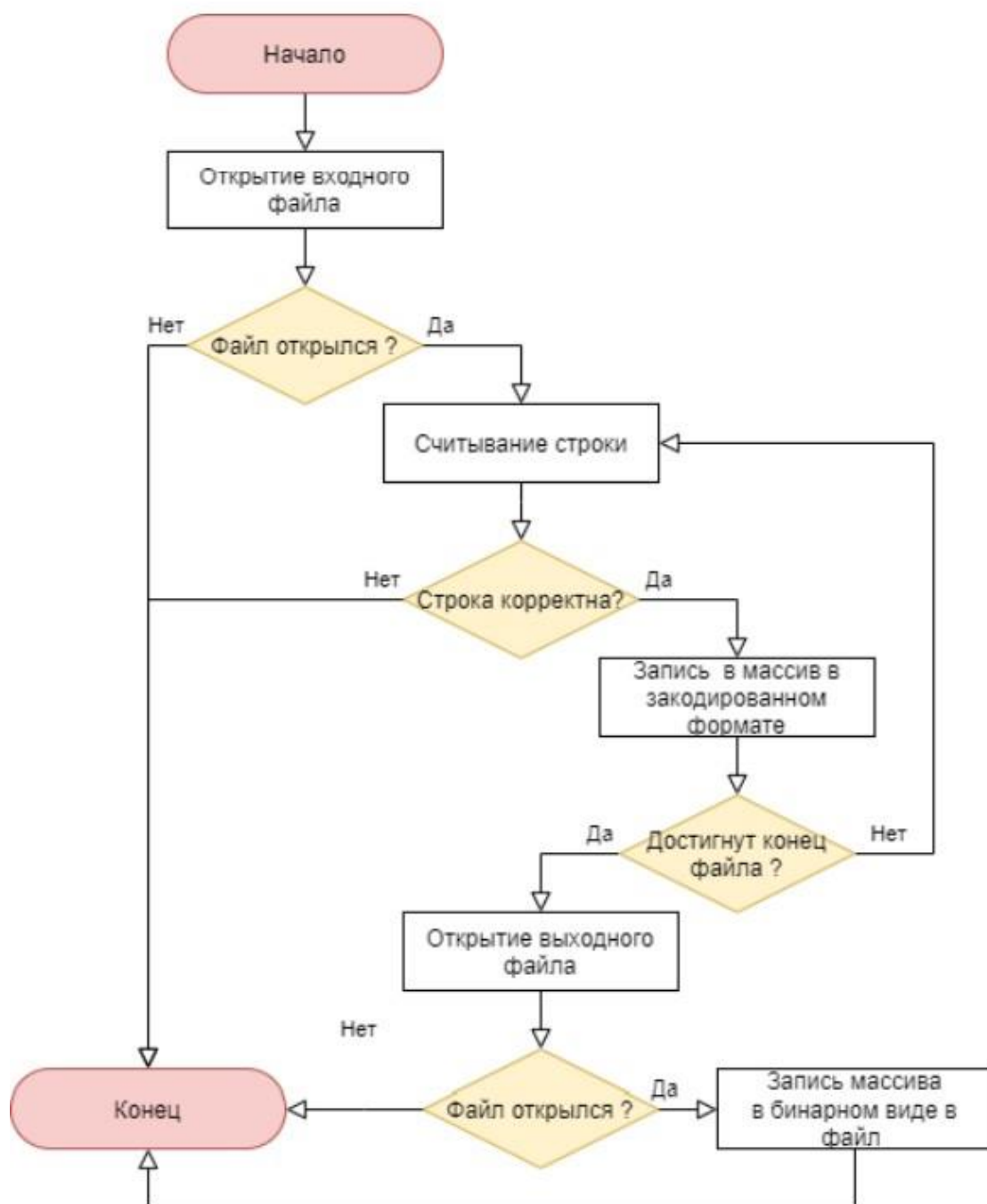
Первый этап - сдаётся практику в формате лабораторных, второй - лектору в формате курсового проекта. Второй этап – реализация трансляторов. Задача первого этапа подробно расписана в разделе “Архитектура вычислительной машины Simple Computer”. Задача второго этапа подробно расписана в разделе “Задание к курсовой работе”.

Блок-схемы используемых алгоритмов

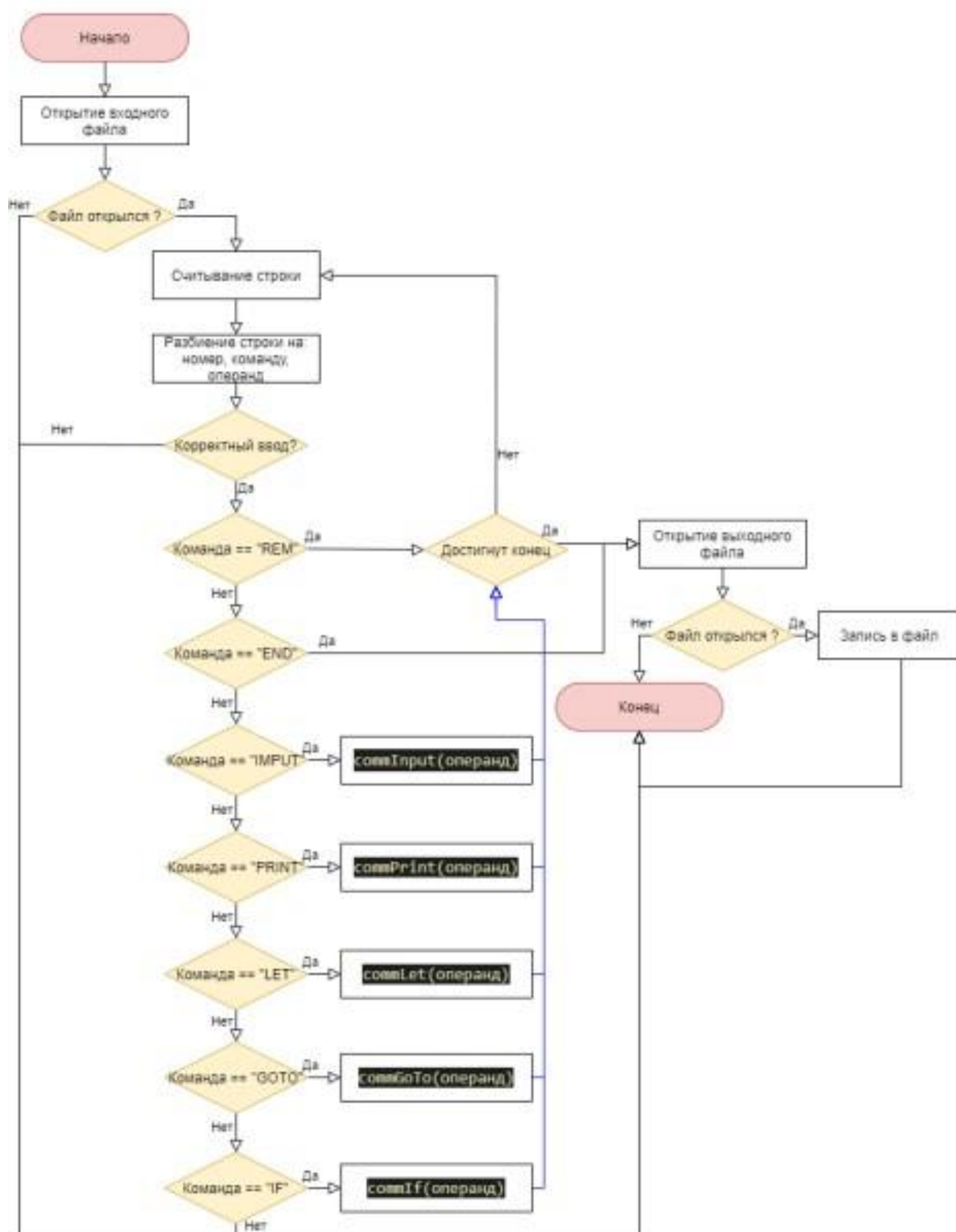
SimpleComputer



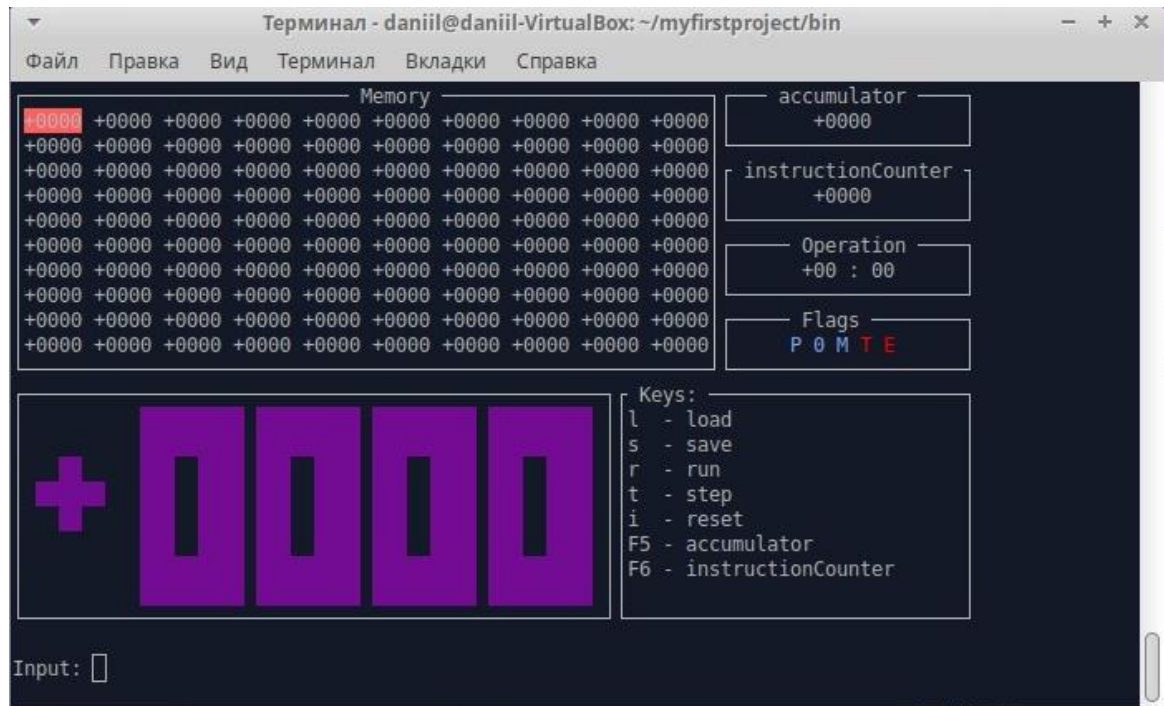
SimpleAssembler



SimpleBasic



Программная реализация: Интерфейс SimpleComputer



fact.bs - программа, считающая факториал введённого числа

```
~/myfirstproject/bin/fact.bs - M
Файл  Правка  Поиск  Вид  Доку
10 INPUT A
20 LET B = 1
30 IF A = 0 GOTO 70
40 LET B = B * A
50 LET A = A - 1
60 GOTO 30
70 PRINT B
80 END
```

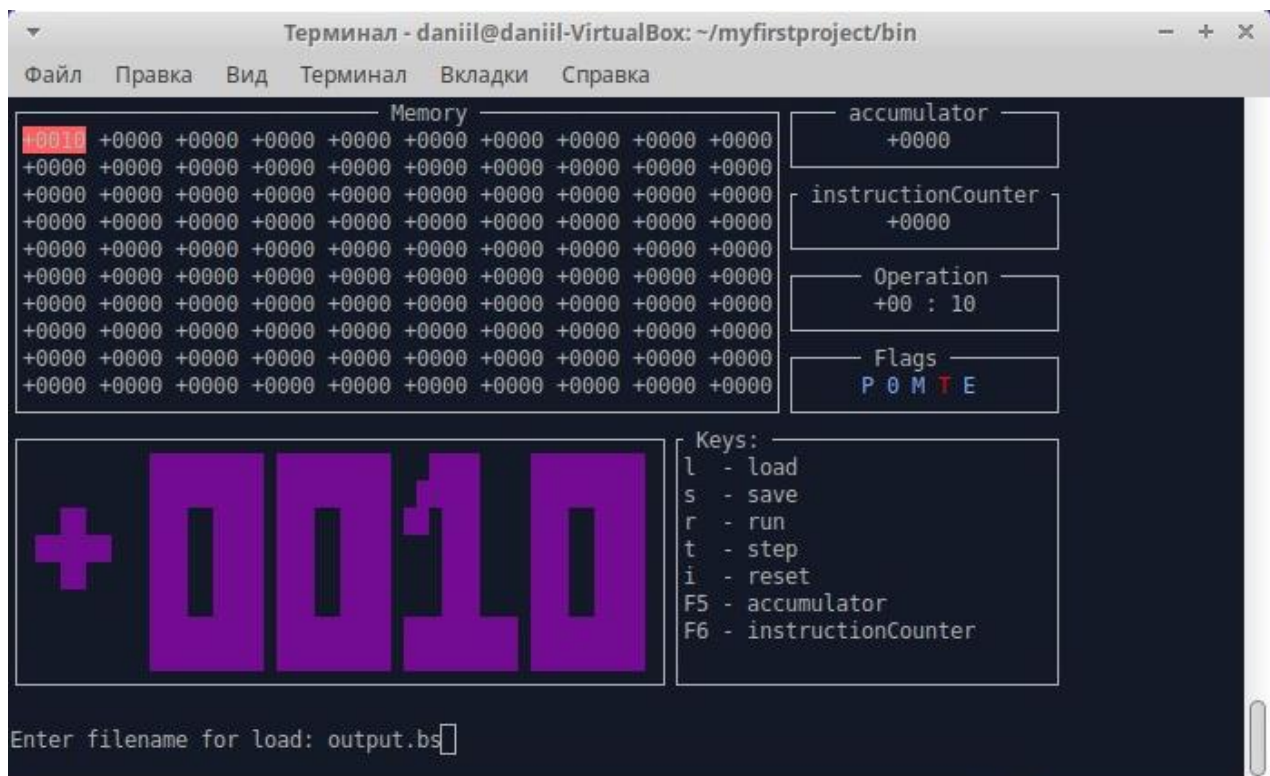
1) Запуск трансляции:

```
daniil@daniil-VirtualBox:~/myfirstproject/bin$ ./sbt fact.bs output.bs
```

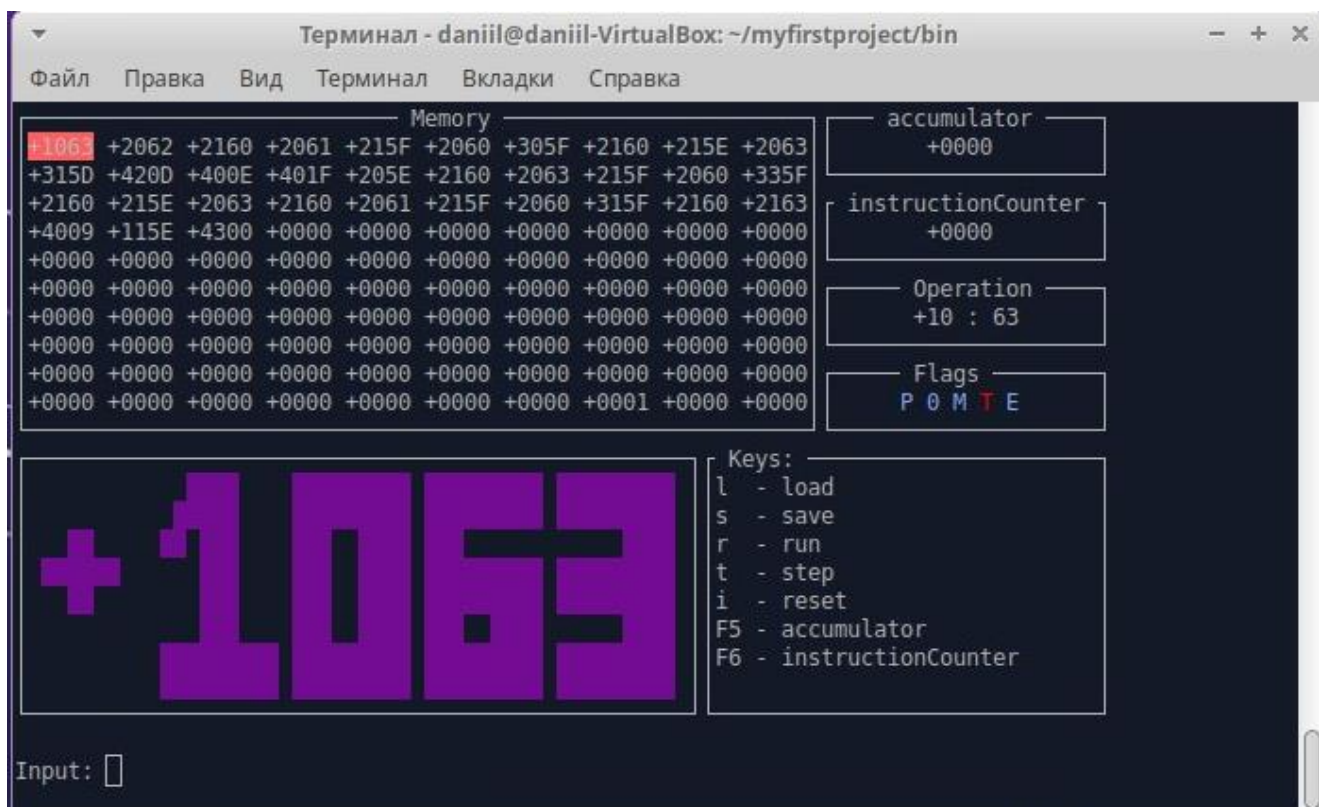
Появился файл:



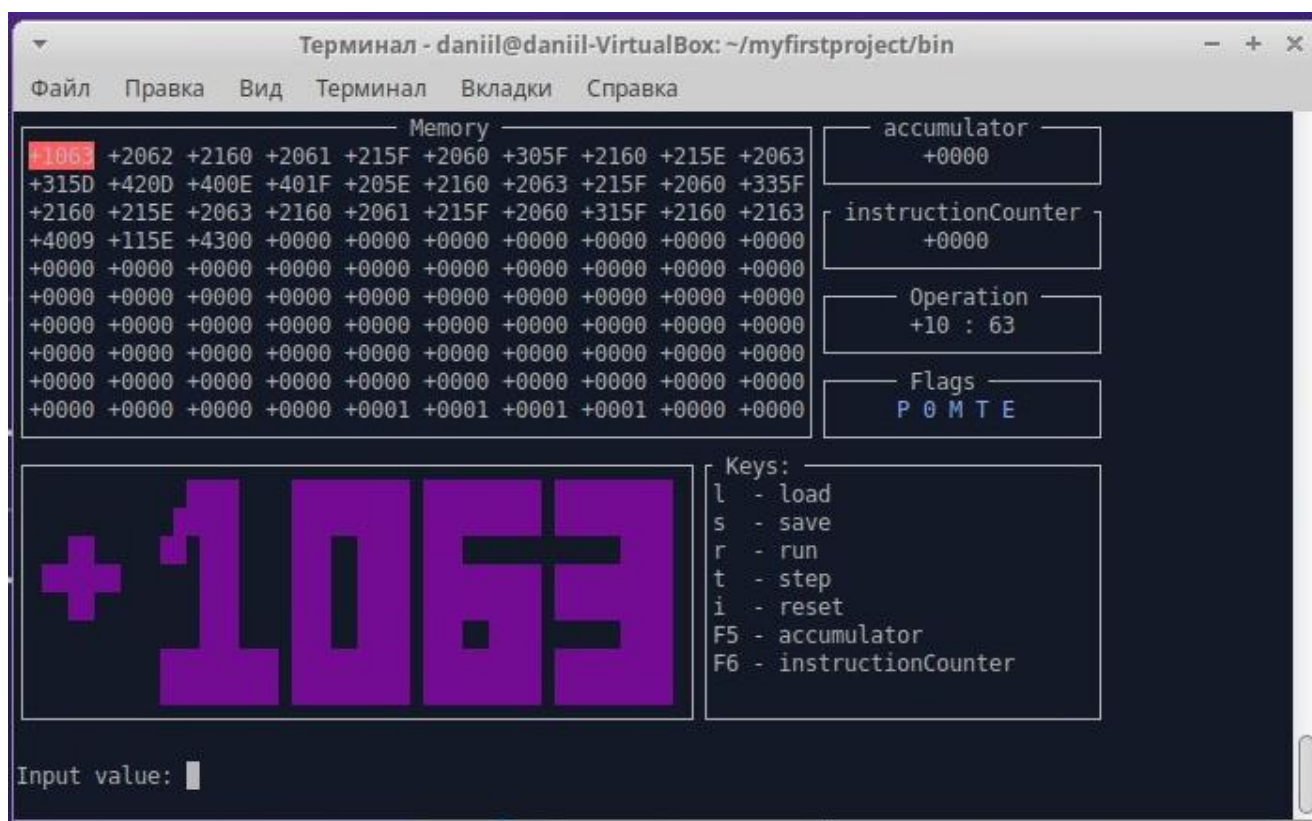
2) Жмём клавишу l (load) и вводим название бинарного файла с результатом трансляции:



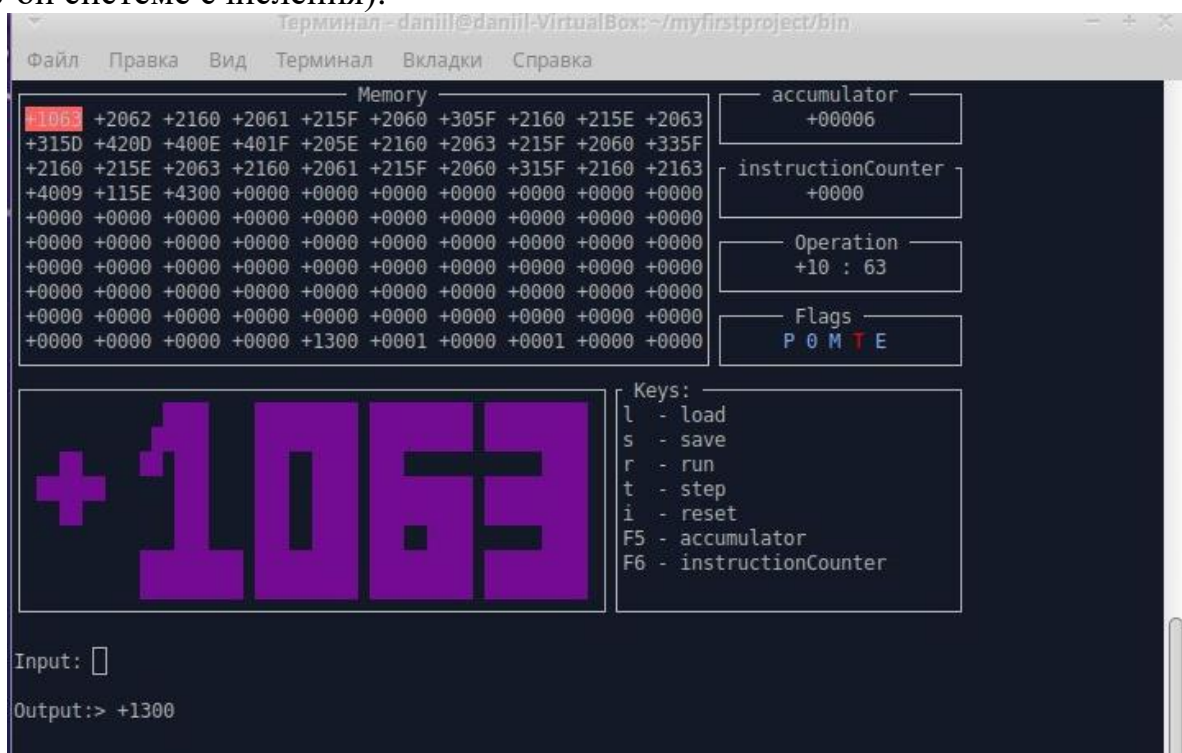
3) Бинарный файл загружен:



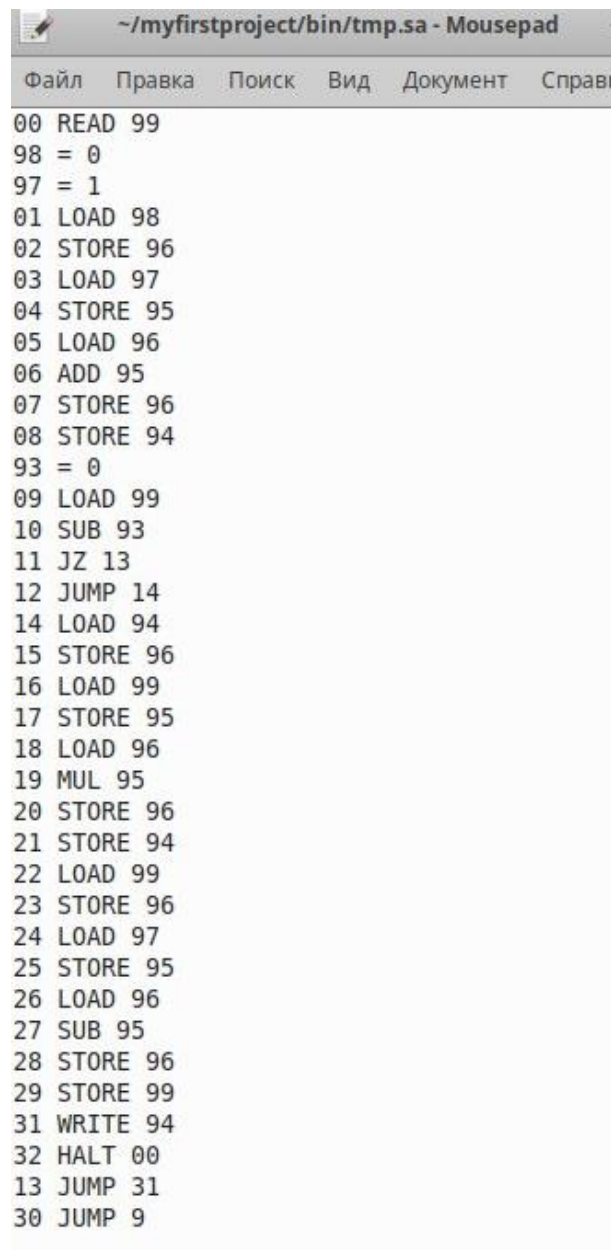
Теперь нужно нажать клавишу `r` (run) для запуска:



4) Введём, например 9 (число вводится в защищённом режиме, поэтому в консоли не отображается). Программа прошла по всем допустимым ячейкам памяти и напечатала «1300» (значение хранится в ячейке «94» в 16-ой системе счисления):



Код, получившийся в результате трансляции из Бейсика на Ассемблер. Он автоматически загружается после трансляции языка Бейсик в файл output.bs.

A screenshot of a text editor window titled "~/myfirstproject/bin/tmp.sa - Mousepad". The window has a menu bar with options: "Файл", "Правка", "Поиск", "Вид", "Документ", and "Справка". The main text area contains assembly code with line numbers on the left and instructions on the right. The instructions include READ, =, LOAD, STORE, ADD, SUB, JZ, JUMP, and WRITE, with various numeric operands.

```
00 READ 99
98 = 0
97 = 1
01 LOAD 98
02 STORE 96
03 LOAD 97
04 STORE 95
05 LOAD 96
06 ADD 95
07 STORE 96
08 STORE 94
93 = 0
09 LOAD 99
10 SUB 93
11 JZ 13
12 JUMP 14
14 LOAD 94
15 STORE 96
16 LOAD 99
17 STORE 95
18 LOAD 96
19 MUL 95
20 STORE 96
21 STORE 94
22 LOAD 99
23 STORE 96
24 LOAD 97
25 STORE 95
26 LOAD 96
27 SUB 95
28 STORE 96
29 STORE 99
31 WRITE 94
32 HALT 00
13 JUMP 31
30 JUMP 9
```

Результаты проведенного исследования

При выполнении данной курсовой работы я изучил терминальные управляющие последовательности, режимы работы терминала, псевдографики, а также принципы работы подсистемы прерываний ЭВМ. В ходе курсовой работы я реализовал базовые команды УУ и АЛУ. Были написаны трансляторы: с Basic в Assembler; с Assembler в бинарный формат.

Приложение (листинг программы)

Файл Makefile

```
CC = gcc

CFLAGS = -Wall
INC_DIR = include
OBJ_DIR = obj
SRC_DIR = src
BIN_DIR = bin
TRANS_DIR = translators
TEST_DIR = test

all: $(BIN_DIR)/console $(BIN_DIR)/sat $(BIN_DIR)/sbt $(BIN_DIR)/test_comp
$(BIN_DIR)/test_term $(BIN_DIR)/test_bigchars $(BIN_DIR)/test_readkey

$(BIN_DIR)/console: $(OBJ_DIR)/main.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/sat: $(OBJ_DIR)/sat.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/sbt: $(OBJ_DIR)/sbt.o $(INC_DIR)/libmy.a $(TRANS_DIR)/rpnTranslator.h
$(SRC_DIR)/*.h
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/test_comp: $(OBJ_DIR)/test_comp.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/test_term: $(OBJ_DIR)/test_term.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/test_bigchars: $(OBJ_DIR)/test_bigchars.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(BIN_DIR)/test_readkey: $(OBJ_DIR)/test_readkey.o $(INC_DIR)/libmy.a
$(CC) -o $@ $< -L$(INC_DIR) -lmy

$(INC_DIR)/libmy.a: $(OBJ_DIR)/myInterface.o $(OBJ_DIR)/myTerm.o
$(OBJ_DIR)/mySimpleComputer.o $(OBJ_DIR)/myBigChars.o $(OBJ_DIR)/myReadKey.o
$(OBJ_DIR)/mySignal.o $(OBJ_DIR)/myCU.o $(OBJ_DIR)/myALU.o $(OBJ_DIR)/sbt.o
$(OBJ_DIR)/rpnTranslator.o $(OBJ_DIR)/sat.o
ar cr $@ $^

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.c
$(CC) $(CFLAGS) -c $< -o $@
```

```

$(OBJ_DIR)/test_comp.o: $(TEST_DIR)/test_comp.c
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/test_bigchars.o: $(TEST_DIR)/test_bigchars.c
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/test_readkey.o: $(TEST_DIR)/test_readkey.c
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/test_term.o: $(TEST_DIR)/test_term.c
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/sat.o: $(TRANS_DIR)/sat.c
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/sbt.o: $(TRANS_DIR)/sbt.c $(TRANS_DIR)/rpnTranslator.h $(SRC_DIR)/*.h
    $(CC) $(CFLAGS) -c $< -o $@

$(OBJ_DIR)/rpnTranslator.o: $(TRANS_DIR)/rpnTranslator.c
$(TRANS_DIR)/rpnTranslator.h $(SRC_DIR)/*.h
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(OBJ_DIR)/*.o $(INC_DIR)/*.a bin/console bin/sat bin/sbt
    bin/test_bigchars bin/test_comp bin/test_readkey bin/test_term bin/tmp.sa

```

Файл main.c

```

#include "myBigChars.h"
#include "myInterface.h"
#include "myReadKey.h"
#include "mySignal.h"
#include "mySimpleComputer.h"
#include "myTerm.h"
#include <fcntl.h>
#include <stdlib.h>
#include <sys/time.h>

#include <unistd.h>

bool on = 1;
// make clean && make && ./sat sat.sa sat.o
int
main ()
{
    int valueR;
    rk_myTermSave ();
    sc_memoryInit ();
    int rows, cols;

```

```

sc_memorySet (0, 5);
sc_memorySet (0, 0x10);

mt_getscreensize (&rows, &cols);
if (rows < 24 || cols < 24)
    return -1;
ui_initial ();
sc_regSet (3, 1);

struct itimerval nval, oval;
nval.it_interval.tv_sec = 1;
nval.it_interval.tv_usec = 0;
nval.it_value.tv_sec = 1;
nval.it_value.tv_usec = 0;
ms_setSignals ();
enum keys key;
// sc_memorySet(0, 32767);
sc_memoryLoad ("sat.o");
do
{
    ui_update ();
    rk_readKey (&key);
    switch (key)
    {
        case UP_KEY:
            (currMemCell <= 9) ? (currMemCell = 90 + currMemCell)
                               : (currMemCell -= 10);

            break;
        case RIGHT_KEY:
            (!((currMemCell + 1) % 10)) ? (currMemCell -= 9)
                                         : (currMemCell += 1);

            break;
        case DOWN_KEY:
            (currMemCell >= 90) ? (currMemCell = currMemCell - 90)
                               : (currMemCell += 10);

            break;
        case LEFT_KEY:
            (! (currMemCell % 10)) ? (currMemCell += 9) : (currMemCell -= 1);
            break;

        case L_KEY:
            ui_loadfile ();
            break;
        case S_KEY:
            ui_savefile ();
            break;

        case R_KEY:
            sc_regGet (3, &valueR);
            if (valueR)
            {

```

```

        sc_regSet (3, 0);
        setitimer (ITIMER_REAL, &nval, &oval);
    }
    else
    {
        alarm (0);
        sc_regSet (3, 1);
    }
    drawing_IC ();
    break;
case T_KEY:
    sc_regSet (3, 0);
    ms_timerHandler (SIGALRM);
    sc_regSet (3, 1);
    drawing_IC ();
    break;
case I_KEY:
    raise (SIGUSR1);
    accumulator = 0;
    instruction_counter = 0;
    currMemCell = 0;
    drawing_IC ();
    sc_memoryInit ();
    sc_regSet (3, 1);
    ui_update ();
    break;

case F5_KEY:
    ui_Accum ();
    break;

case F6_KEY:
    ui_Counter ();
    break;

case ENTER_KEY:
    if (ui_setValue () == -1)
        continue;
    ui_update ();
    break;
case ESC_KEY:
    on = 0;
    break;
}
}
while (on);
mt_gotoXY (1, 24);
return 0;
}

```

Файл myALU.c

```
#include "myALU.h"

int
ALU (int command, int operand)
{
    int tmp;
    sc_memoryGet (operand, &tmp);
    switch (command)
    {
        case 0x30: //сложение, результат в акк
            accumulator += tmp;
            break;
        case 0x31: //вычитание
            accumulator -= tmp;
            break;
        case 0x32: //деление
            if (tmp == 0)
            {
                sc_regSet (1, 1);
                return -1;
            }
            sc_regSet (1, 0);
            accumulator /= tmp;
            break;
        case 0x33: //произведение
            accumulator *= tmp;
            break;
        case 0x52: //логич и
            accumulator &= tmp;
            break;
    }
    return 0;
}
```

Файл myALU.h

```
#ifndef MY_ALU_H

#define MY_ALU_H

#include "mySimpleComputer.h"

int ALU (int command, int operand);

#endif
```

Файл myBigChars.c

```
#include "myBigChars.h"
unsigned int big_chars[][2] = {
    { 0xE7E7FFFF, 0xFFFFE7E7 }, // 0
    { 0x1CDC7C3C, 0xFFFF1C1C }, // 1
    { 0xFF07FFFF, 0xFFFFE0FF }, // 2
    { 0xFF07FFFF, 0xFFFF07FF }, // 3
    { 0xFFE7E7E7, 0x070707FF }, // 4
    { 0xFFE0FFFF, 0xFFFF07FF }, // 5
    { 0xFFE0FFFF, 0xFFFFE7FF }, // 6
    { 0x1C0EFFFF, 0x3838FE38 }, // 7
    { 0x7EE7FF7E, 0x7EFFE77E }, // 8
    { 0xFFE7FFFF, 0xFFFF07FF }, // 9
    { 0xFFE7FF7E, 0xE7E7E7FF }, // A
    { 0xFEE7FFFF, 0xFEFFE7FE }, // B
    { 0xE0E7FF7E, 0x7EFFE7E0 }, // C
    { 0xE7E7FFF8, 0xF8FFE7E7 }, // D
    { 0xFFE0FFFF, 0xFFFFE0FF }, // E
    { 0xFFE0FFFF, 0xE0E0E0FF }, // F
    { 0x7E180000, 0x00000018 }, // +
    { 0x7E000000, 0x00000000 }, // -
};

char buf[512];

int
bc_printA (char charr)
{
    sprintf (buf, "\033(0%c\033(B", charr);
    write (1, buf, strlen (buf));
    return 0;
}

int
bc_box (int x, int y, int width, int height)
{
    int rows, cols;
    mt_getscreenSize (&rows, &cols);
    if ((x <= 0) || (y <= 0) || (x + width - 1 > cols) || (y + height - 1 > rows)
        || (width <= 1) || (height <= 1))
        return -1;

    mt_gotoXY (x, y);
    bc_printA ((char)ACS_ULCORNER);
    mt_gotoXY (x + width - 1, y);
    bc_printA ((char)ACS_URCORNER);
    mt_gotoXY (x + width - 1, y + height - 1);
    bc_printA ((char)ACS_LRCORNER);
    mt_gotoXY (x, y + height - 1);
    bc_printA ((char)ACS_LLCORNER);
}
```

```

/* Горизонтальные линии */
for (int i = 1; i < width - 1; ++i)
{
    // верхняя
    mt_gotoXY (x + i, y);
    bc_printA ((char)ACS_HLINE);
    // нижняя
    mt_gotoXY (x + i, y + height - 1);
    bc_printA ((char)ACS_HLINE);
}

/* Вертикальные линии */
for (int i = 1; i < height - 1; ++i)
{
    // верхняя
    mt_gotoXY (x, y + i);
    bc_printA ((char)ACS_VLINE);
    // нижняя
    mt_gotoXY (x + width - 1, y + i);
    bc_printA ((char)ACS_VLINE);
}
return 0;
}

int
bc_printBigChar (unsigned int *big, int x, int y, enum colors colorFG,
                 enum colors colorBG)
{
    if (colorFG != DEFAULT)
        mt_setfgcolor (colorFG);
    if (colorBG != DEFAULT)
        mt_setbgcolor (colorBG);

    for (int i = 0; i < 8; ++i)
        for (int j = 0; j < 8; ++j)
        {
            mt_gotoXY (x + i, y + j);
            int value;
            if (bc_getbigCharPos (big, i, j, &value))
                return -1;
            if (value)
                bc_printA (ACS_CKBOARD);
            if (!value)
            {
                snprintf (buf, 6, "%c ", ' ');
            }
            write (1, buf, strlen (buf));
        }
    set_default_color ();
    return 0;
}

```

```

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

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

int
bc_bigCharWrite (int fd, int *big, int count)
{
    if (write (fd, big, count * 2 * sizeof (unsigned int)))
        return -1;
    return 0;
}

int
bc_bigCharRead (int fd, int *big, int need_count, int *count)
{
    *count = 0;
    for (int i = 0; i < need_count * 2; ++i)
    {
        if (read (fd, &big[i], sizeof (unsigned int)) == -1)
            return -1;
        if (!((i + 1) % 2))
            (*count)++;
    }

    return 0;
}

```


Файл myBigChars.h

```
#ifndef MYBIGCHARS_H
#define MYBIGCHARS_H

#include "myTerm.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
/* Псевдографика */
#define ACS_CKBOARD 'a' // Штриховка
#define ACS_ULCORNER 'l' // Левый верхний угол
#define ACS_URCORNER 'k' // Правый верхний угол
#define ACS_LRCORNER 'j' // Правый нижний угол
#define ACS_LLCORNER 'm' // Левый нижний угол
#define ACS_HLINE 'q' // Горизонтальная линия
#define ACS_VLINE 'x' // Вертикальная линия

extern unsigned int big_chars[][2];

int bc_printA (char charr);
int bc_box (int x1, int y1, int x2, int y2);
int bc_printBigChar (unsigned int *big, int x, int y, enum colors colorFG,
                    enum colors colorBG);
int bc_setBigCharPos (int *big, int x, int y, int value);
int bc_getbigCharPos (unsigned int *big, int x, int y, int *value);
int bc_bigCharWrite (int fd, int *big, int count);
int bc_bigCharRead (int fd, int *big, int need_count, int *count);

#endif
```

Файл myCU.c

```
#include "myCU.h"

int
READ (int operand)
{
    Char buff[32];
    mt_gotoXY (1, 24);
    sprintf (buff, "Input value: ");
    write (STDOUT_FILENO, buff, 14);
    char buffAf[32];
    fgets (buffAf, 32, stdin);
    sc_memorySet (operand, atoi (buffAf));
    return 0;
}

int
```

```

WRITE (int operand)
{
    int value;
    int command;
    char buf[20];

    if (sc_memoryGet (operand, &value)
        || sc_commandDecode (value & 0x3FFF, &command, &operand))
    {
        sc_regSet (4, 1);
        return -1;
    }

    snprintf (buf, 20, "Output:> %c%02X%02X", (value & 0x4000) ? '-' : '+',
              command, operand);

    mt_gotoXY (1, 26);
    write (1, buf, 15);

    return 0;
}

int
LOAD (int operand)
{
    int value = 0;
    sc_memoryGet (operand, &value);
    accumulator = value;
    return 0;
}

int
STORE (int operand)
{
    sc_memorySet (operand, accumulator);
    return 0;
}

int
JUMP (int operand)
{
    instruction_counter = operand;
    CU ();
    return 0;
}

int
JNEG (int operand)
{
    if (accumulator < 0)
    {

```

```

        instruction_counter = operand;
        CU ();
    }
    return 0;
}

int
JZ (int operand)
{
    if (accumulator == 0)
    {
        instruction_counter = operand;
        CU ();
    }
    return 0;
}

int
MOVR (int operand)
{
    int value;
    sc_memoryGet (accumulator, &value);
    sc_memorySet (operand, value);
    return 0;
}

int
HALT ()
{
    sc_regSet (3, 1);
    instruction_counter = 0;

    return 0;
}

int
JNS (int operand)
{
    if (accumulator > 0)
    {
        instruction_counter = operand;
    }
    return 0;
}

// int NOT(int operand)
// {
//     accumulator
// }
int
CU ()

```

```

{
    int value = 0;
    sc_memoryGet (instruction_counter, &value);
    int command, operand;
    if (sc_commandDecode (value & 0x3FFF, &command, &operand))
    {
        sc_regSet (4, 1);
        return -1;
    }
    if (((command >= 0x30) && (command <= 0x33)) || (command == 0x52))
    {
        ALU (command, operand);
    }
    else {
        switch (command)
        {
            {
                case 0x10:
                    READ (operand);
                    break;
                case 0x11:
                    WRITE (operand);
                    break;
                case 0x20:
                    LOAD (operand);
                    break;
                case 0x21:
                    STORE (operand);
                    break;
                case 0x40:
                    JUMP (operand);
                    break;
                case 0x41:
                    JNEG (operand);
                    break;
                case 0x42:
                    JZ (operand);
                    break;
                case 0x43:
                    HALT ();
                    break;
                case 0x72:
                    MOVR (operand);
                    break;
                case 0x55:
                    JNS (operand);
                    break;
            }
        }
    }
    return 0;
}

```

Файл myCU.h

```
#ifndef MY_CU_H
#define MY_CU_H
#include "myALU.h"
#include "myInterface.h"
#include "myReadKey.h"
#include "mySimpleComputer.h"
#include "myTerm.h"
#include "string.h"

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

#endif
```

Файл myInterface.c

```
#include "myInterface.h"
#include "myBigChars.h"
#include "myReadKey.h"
#include "mySimpleComputer.h"
#include "myTerm.h"

int instruction_counter;
int accum;
// extern currMemCell;
char buffer[512];
short ind;
```

```
int
drawingBoxes ()
{
    if (bc_box (1, 1, 61, 12)) // Окно Memory
        return -1;
    if (bc_box (62, 1, 22, 3)) // Окно accumulator
        return -1;
    if (bc_box (62, 4, 22, 3)) // Окно instructionCounter
        return -1;
    if (bc_box (62, 7, 22, 3)) // Окно Operation
```

```

        return -1;
    if (bc_box (62, 10, 22, 3)) // Окно Flags
        return -1;
    if (bc_box (1, 13, 52, 10)) // Окно BigChars
        return -1;
    if (bc_box (53, 13, 31, 10)) // Окно Keys
        return -1;

    return 0;
}

int
drawing_texts ()
{
    mt_gotoXY (30, 1);
    write (1, " Memory ", 8);

    mt_gotoXY (66, 1);
    write (1, " accumulator ", 13);

    mt_gotoXY (63, 4);
    write (1, " instructionCounter ", 21);

    mt_gotoXY (68, 7);
    write (1, " Operation ", 11);

    mt_gotoXY (68, 10);
    write (1, " Flags ", 7);

    mt_gotoXY (54, 13);
    write (1, " Keys: ", 7);

    /* HotKeys */
    char *hot_keys[7] = { (char *)"l - load",
                          (char *)"s - save",
                          (char *)"r - run",
                          (char *)"t - step",
                          (char *)"i - reset",
                          (char *)"F5 - accumulator",
                          (char *)"F6 - instructionCounter" };

    for (int i = 0; i < 7; ++i)
    {
        mt_gotoXY (54, i + 14);
        write (1, hot_keys[i], strlen (hot_keys[i]));
    }
    mt_gotoXY (1, 24);

    return 0;
}

```

```

int
drawingBigChars ()
{
    // int tmp = memory_arr[instruction_counter];
    int value, command, operand;
    sc_memoryGet (currMemCell, &value);
    if (sc_memoryGet (currMemCell, &value) < 0)
    {
        return -1; // обработка ошибки чтения из памяти
    }
    if (sc_commandDecode (value & 0x3FFF, &command, &operand) < 0)
    {
        return -2; // обработка ошибки декодирования команды
    }
    if (!(value & 0x4000))
    {
        ind = 16;
        bc_printBigChar (big_chars[ind], 2, 14, VIOLET, 0);
    }

    else
    {
        ind = 17;
        bc_printBigChar (big_chars[ind], 2, 14, VIOLET, 0);
    }

    int ch;
    for (int i = 0; i < 4; ++i)
    {
        switch (i)
        {
            {
            case 0:
                ch = (command >> 4) & 0xF;
                break;
            case 1:
                ch = (command)&0xF;
                break;
            case 2:
                ch = (operand >> 4) & 0xF;
                break;
            case 3:
                ch = (operand)&0xF;
                break;
            }
            bc_printBigChar (big_chars[ch], 2 + 8 * (i + 1) + 2 * (i + 1), 14, VIOLET,
                            0);
        }
    }
    return 0;
}

```

```

int
drawing_memory ()
{
    int value, command, operand;
    char buf[6];
    for (int i = 0; i < 10; ++i)
    {
        for (int j = 0; j < 10; ++j)
        {
            int address = i * 10 + j;
            if (sc_memoryGet (address, &value) < 0)
            {
                return -1; // обработка ошибки чтения из памяти
            }
            if (sc_commandDecode (value & 0x3FFF, &command, &operand) < 0)
            {
                return -2; // обработка ошибки декодирования команды
            }
            mt_gotoXY (2 + (5 * j + j), 2 + i);
            if (currMemCell == address)
            {
                mt_setbgcolor (PEACH);
            }
            snprintf (buf, 6, "%c%02X%02X", (value & 0x4000) ? '-' : '+',
                    command, operand);
            write (STDOUT_FILENO, buf, strlen (buf));
            set_default_color ();
        }
    }
    return 0;
}

int
drawing_OP ()
{
    int value;
    int command;
    int operand;
    sc_memoryGet (instruction_counter, &value);
    mt_gotoXY (69, 8);
    char buf[32];
    if (sc_commandDecode (value, &command, &operand) > 0)
    {
        sc_regSet (4, 1);
        sprintf (buf, "+00 : 00");
        write (STDOUT_FILENO, buf, 9);
        return -1;
    }
    mt_gotoXY (65, 8);

```



```

sprintf (buf, "          ");
write (STDOUT_FILENO, buf, strlen (buf));
mt_gotoXY (68, 8);
sprintf (buf, " +%02X : %02X", command, operand);

write (STDOUT_FILENO, buf, strlen (buf));
return 0;
}

//«P» – переполнение при выполнении операции, «0» – ошибка деления на 0, «M» –
//ошибка выхода за границы памяти, «T» – игнорирование тактовых импульсов, «E»
//- указана неверная команда
int
drawing_flags ()
{
    char tmp[] = { 'P', '0', 'M', 'T', 'E' };

    for (int i = 0; i < REGISTER_SIZE; ++i)
    {
        int value, value2;

        sc_memoryGet (instruction_counter, &value2);
        (value2 & 0x4000) || value2 == 0 ? sc_regSet (4, 1) : sc_regSet (4, 0);
        if (sc_regGet (i, &value))
            return -1;
        mt_gotoXY (68 + (i * 2), 11);

        if (value)
        {
            mt_setfgcolor (RED);
            write (STDOUT_FILENO, &tmp[i], 1);
        }
        else
        {
            mt_setfgcolor (LIGHT_BLUE);
            write (STDOUT_FILENO, &tmp[i], 1);
        }

        set_default_color ();
    }
    return 0;
}

int
ui_savefile ()
{
    char filename[256];
    write (1, "\033[2K", 4);
    mt_gotoXY (1, 24);
    write (1, "Enter filename for save: ", 26);

```

```

fgets (filename, 256, stdin);
filename[strlen (filename) - 1] = 0;
sc_memorySave (filename);
return 0;
}

int
ui_loadfile ()
{
    char filename[256];
    write (1, "\033[2K", 4);
    mt_gotoXY (1, 24);
    write (1, "Enter filename for load: ", 26);
    fgets (filename, 256, stdin);
    filename[strlen (filename) - 1] = 0;
    sc_memoryLoad (filename);
    return 0;
}

int
ui_Counter ()
{
    char buffer[32];
    write (1, "\033[2K", 4);
    mt_gotoXY (1, 24);
    write (STDOUT_FILENO, "Enter instruction_counter: ", 28);
    fgets (buffer, 32, stdin);
    instruction_counter = atoi (buffer);
    if (instruction_counter < 0 || instruction_counter > 99)
    {
        instruction_counter = 0;
        mt_clrscr ();
        write (STDOUT_FILENO, "\033[38;5;196mError\n", 17);
        return 1;
    }
    drawing_IC ();
    return 0;
}

int
drawing_IC ()
{
    mt_gotoXY (70, 5);
    char buff[32];
    sprintf (buff, "+%04d", instruction_counter);
    write (STDOUT_FILENO, buff, 7);
    return 0;
}

int
ui_Accum ()

```

```

{
    char buffer[32];
    write (1, "\033[2K", 5);
    mt_gotoXY (1, 24);
    write (STDOUT_FILENO, "Enter Accumulator: ", 20);
    fgets (buffer, 32, stdin);
    accumulator = atoi (buffer);
    drawing_Accum ();
    return 0;
}

int
drawing_Accum ()
{
    mt_gotoXY (70, 2);
    char buff[32];
    accumulator < 0 ? sprintf (buff, "-%04d", -accumulator)
                    : sprintf (buff, "+%04d", accumulator);
    write (STDOUT_FILENO, buff, 6);
    return 0;
}

int
ui_initial ()
{
    mt_clrscr ();

    currMemCell = 0;
    instruction_counter = 0;
    mt_gotoXY (70, 5);
    write (STDOUT_FILENO, "+0000", 5);
    mt_gotoXY (70, 2);
    write (STDOUT_FILENO, "+0000", 5);
    if (drawingBoxes ())
        return -1;
    if (drawing_texts ())
        return -1;
    if (drawing_memory ())
        return -1;
    mt_gotoXY (1, 24);

    return 0;
}

int
ui_update ()
{
    if (drawingBoxes ())
        return -1;
    if (drawing_texts ())

```

```

    return -1;
if (drawing_memory ())
    return -1;
if (drawing_flags ())
    return -1;
if (drawingBigChars ())
    return -1;

drawing_OP ();
drawing_Accum ();
mt_gotoXY (1, 24);
write (1, "\033[2K", 4);
write (1, "Input: ", 7);

return 0;
}

int
ui_setValue ()
{
    char buffer[11];
    fgets (buffer, 10, stdin);

    mt_gotoXY (8, 24);
    write (1, "\033[2K", 4);

    if (buffer[strlen (buffer) - 1] == '\n')
        buffer[strlen (buffer) - 1] = 0;

    long int number;
    char *tmp;

    if (buffer[0] == '+')
    {
        number = strtol (&buffer[1], &tmp, 10);
        if (number > 0x7FFF)
        {
            return -1;
        }
        sc_memorySet (currMemCell, number);
    }
    else if (buffer[0] != '-')
    {
        number = strtol (buffer, &tmp, 10);
        if (number > 0x7FFF)
        {
            return -1;
        }
        sc_memorySet (currMemCell, number);
    }
}

```

```

    }
    else if (buffer[0] == '-')
    {
        number = strtol (&buffer[1], &tmp, 10);
        if (number > 0x7FFF)
        {
            return -1;
        }
        number = number | 0x4000;

        sc_memorySet (currMemCell, number);
    }

    return 0;
}

bool
checkCorrectInput (const char buffer[10])
{
    int i;
    if (buffer[0] == '+' || buffer[0] == '-')
    {
        if (strlen (buffer) == 2 || strlen (buffer) > 6)
            return false;
        i = 1;
    }
    else
    {
        i = 0;
        if (strlen (buffer) == 1 || strlen (buffer) > 5)
            return false;
    }
    // for (i; i < strlen (buffer) - 1; ++i)
    if (!(isdigit (buffer[i])))
        return false;

    return true;
}

int
ui_messageOutput (char *str, enum colors color)
{
    char buff[32];
    sprintf (buff, "\033[38;5;%dm%s\033[0m", color, str);
    write (STDOUT_FILENO, buff, 32);
    sleep (1.5);
    write (1, "\033[2K", 4);
    return 0;
}

```

```

int
clearBuffIn ()
{
    int c;
    do
    {
        c = getchar ();
    }
    while (c != '\n' && c != '\0');
    return 0;
}

```

Файл myInterface.h

```

#ifndef INTERFACE_H
#define INTERFACE_H
#include "myBigChars.h"
#include "mySimpleComputer.h"
#include <ctype.h>
#include <string.h>
#include <unistd.h>
extern int instruction_counter;
int drawingBigChars ();
int ui_initial ();
int ui_update ();
int ui_setValue ();
int drawingBoxes ();

int drawing_texts ();
int ui_Counter ();
int drawing_memory ();
int drawing_flags ();
int drawing_IC ();
int ui_Accum ();
int drawing_Accum ();
int ui_setValue ();
bool checkCorrectInput (const char buffer[10]);
int ui_messageOutput (char *str, enum colors color);
int clearBuffIn ();
int ui_savefile ();
int ui_loadfile ();
#endif

```

Файл myReadKey.c

```

#include "myReadKey.h"

struct termios save;

int
rk_readKey (enum keys *key)

```

```

{
    fflush (stdout); // очистка потока вывода
    fflush (stdin);
    char buffer[5] = { 0 };
    rk_myTermRegime (0, 30, 0, 0, 0);
    read (fileno (stdin), buffer, sizeof (buffer));
    rk_myTermRestore ();

    if (key == NULL)
    {
        return -1;
    }

    if (buffer[0] == '\033')
    { // Esc-последовательности
        switch (buffer[1])
        {
            case '\0':
                *key = ESC_KEY;
                break;
            case '[':
                switch (buffer[2])
                {
                    case 'A':
                        *key = UP_KEY;
                        break;
                    case 'B':
                        *key = DOWN_KEY;
                        break;
                    case 'C':
                        *key = RIGHT_KEY;
                        break;
                    case 'D':
                        *key = LEFT_KEY;
                        break;
                    case '5':
                        *key = F5_KEY;
                        break;
                    case '1':
                        if (buffer[3] == '7' && buffer[4] == '~')
                            *key = F6_KEY;
                        if (buffer[3] == '5' && buffer[4] == '~')
                            *key = F5_KEY;
                        break;
                    default:
                        break;
                }
                break;
            default:
                break;
        }
    }
}

```

```

    }
}
else if (buffer[0] == '\n' && buffer[1] == '\0')
{ // клавиша Enter
    *key = ENTER_KEY;
}
else
{ // прочие клавиши
    switch (buffer[0])
    {
        case 'l':
        case 'L':
            *key = L_KEY;
            break;

        case 's':
        case 'S':
            *key = S_KEY;
            break;

        case 'r':
        case 'R':
            *key = R_KEY;
            break;

        case 't':
        case 'T':
            *key = T_KEY;
            break;

        case 'i':
        case 'I':
            *key = I_KEY;
            break;

        default:
            break;
    }
}
return 0;
}

int
rk_myTermSave ()
{
    if (tcgetattr (fileno (stdin), &save))
        return -1;
    return 0;
}

int
rk_myTermRestore ()
{
    tcsetattr (fileno (stdin), TCSAFLUSH, &save);
    return 0;
}

```



```

}

int
rk_myTermRegime (int regime, int vtime, int vmin, int echo, int sigint)
{
    struct termios curr;

    if (tcgetattr (fileno (stdin), &curr) == -1)
    {
        perror ("tcgetattr");
        return -1;
    }

    if (regime == false)
    {
        curr.c_lflag &= ~ICANON;
        curr.c_lflag &= ~ECHO;
        curr.c_lflag &= ~ISIG;
        curr.c_cc[VMIN] = vmin > 0 ? vmin : 1;
        curr.c_cc[VTIME] = vtime;
    }
    else
    {
        curr.c_lflag |= ICANON;
    }

    if (tcsetattr (fileno (stdin), TCSAFLUSH, &curr) == -1)
    {
        perror ("tcsetattr");
        return -1;
    }

    return 0;
}

```

Файл myReadKey.h

```

#ifndef MYREADKEY_H
#define MYREADKEY_H

#include <stdbool.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

enum keys
{
    ESC_KEY,
    L_KEY,
    S_KEY,

```

```

    R_KEY,
    T_KEY,
    I_KEY,
    F5_KEY,
    F6_KEY,
    UP_KEY,
    DOWN_KEY,
    LEFT_KEY,
    RIGHT_KEY,
    ENTER_KEY,
};

extern struct termios save;

int rk_readKey (enum keys *key);
int rk_myTermSave ();
int rk_myTermRestore ();
int rk_myTermRegime (int regime, int vtime, int vmin, int echo, int sigit);

#endif

```

mySignal.c

```

#include "mySignal.h"

void
ms_setSignals ()
{
    signal (SIGALRM, ms_timerHandler);
    signal (SIGUSR1, ms_userSignal);
}

void
ms_timerHandler (int sig)
{
    int check;
    sc_regGet (4, &check);
    if (check)
    {
        sc_regSet (3, 1);
        return;
    }
    CU ();
    int value;
    sc_regGet (3, &value);
    if (!value && instruction_counter <= 99)
    {
        instruction_counter++;
        ualarm (100000, 0);
    }
    else if (instruction_counter > 99)

```

```

    {
        instruction_counter = 0;
        sc_regSet (0, 0);
    }

    drawing_IC ();
    currMemCell = instruction_counter;
    ui_update ();
}

void
ms_userSignal (int sig)
{
    alarm (0);
    sc_regInit ();
    sc_regSet (3, 1);
    instruction_counter = 0;
    accumulator = 0;
    drawing_IC ();
}

```

Файл mySignal.h

```

#ifndef MYSIGNAL_H
#define MYSIGNAL_H

#include "myALU.h"
#include "myCU.h"
#include "myInterface.h"
#include "mySimpleComputer.h"
#include <signal.h>

void ms_setSignals ();
void ms_timerHandler (int sig);
void ms_userSignal (int sig);

#endif // MYSIGNAL_H

```

Файл mySimpleComputer.c

```

#include "mySimpleComputer.h"

short int memory_arr[N];
char flag = 0;
short currMemCell;
int accumulator;

int

```

```

sc_memoryInit ()
{
    currMemCell = 0;
    for (int i = 0; i < N; i++)
        memory_arr[i] = 0;
    return 0;
}

int
sc_memorySet (int address, int value)
{
    if (0 > address || address > 99)
    {
        flag = 1;
        return 1;
    }
    memory_arr[address] = value;
    return 0;
}

int
sc_memoryGet (int address, int *value)
{
    if (0 > address || address > 99)
    {
        flag = 1;
        return -1;
    }
    *value = memory_arr[address];
    return 0;
}

int
sc_memorySave (char *filename)
{
    FILE *out = fopen (filename, "wb");
    if (out == NULL)
    {
        printf ("not ok");
        return 1;
    }

    fwrite (memory_arr, sizeof (memory_arr), 1, out);
    fclose (out);
    return 0;
}

int
sc_memoryLoad (char *filename)
{
    FILE *in = fopen (filename, "rb");

```

```

    if (in == NULL)
        return 1;
    fread (memory_arr, sizeof (memory_arr), 1, in);
    fclose (in);
    return 0;
}

int
sc_regInit ()
{
    flag = 0;
    return 0;
}

int
sc_regSet (int _register, int value)
{
    if (_register < 0 || _register > 4)
    {
        write (1, "Error. Wrong _register\n", 24);
        return -1;
    }
    if (value < 0 || value > 1)
    {
        write (1, "Error. Wrong Value\n", 19);
        return -2;
    }
    if (value == 1)
    {
        flag = flag | (1 << _register);
    }
    else
    {
        flag = flag & ~(1 << _register);
    }
    return 0;
}

int
sc_regGet (int _register, int *value)
{
    if (_register < 0 || _register > 4)
    {
        return 1;
    }
    if (flag & (1 << _register))
    {
        *value = 1;
    }
    else

```

```

    {
        *value = 0;
    }
    return 0;
}

int
sc_commandEncode (int command, int operand, int *value)
{
    if (operand < 0 || operand > 127)
    {
        return -1;
    }

    *value = 0;
    *value = *value | operand;
    *value = *value | (command << 7);
    return 0;
}

int
sc_commandDecode (int value, int *command, int *operand)
{
    *operand = 0;
    *command = 0;

    if (value & 0x4000)
        return 1;

    *command = (value & 0x3F80) >> 7; // 16256
    *operand = (value & 0x7F);        // 127

    return 0;
}

```

Файл mySimpleComputer.h

```

#ifndef SIMPLECOMPUTER_H
#define SIMPLECOMPUTER_H

#include <inttypes.h>
#include <math.h>
#include <stdio.h>
#include <unistd.h> // for write function
#define N 100

#define REGISTER_SIZE 5

#define OVERFLOW 0 // переполнение
#define DIVISION_ERROR_BY_ZERO 1 // ошибка деления на 0

```

```

#define OUT_OF_MEMORY 2          // выход за границы памяти
#define IGNORING_CLOCK_PULSES 3 // игнорирование тактовых импульсов
#define INVALID_COMMAND 4 // неверная команда

extern short int memory_arr[N];
extern char flag;
extern short currMemCell;
extern int accumulator;

int sc_memoryInit ();
int sc_memorySet (int address, int value);
int sc_memoryGet (int address, int *value);
int sc_memorySave (char *filename);
int sc_memoryLoad (char *filename);
int sc_regInit ();
int sc_regSet (int _register, int value);
int sc_regGet (int _register, int *value);
int sc_commandEncode (int command, int operand, int *value);
int sc_commandDecode (int value, int *command, int *operand);

#endif

```

Файл myTerm.c

```

#include "myTerm.h"

#include <unistd.h> // for write and dup2

int
mt_clrscr ()
{
    write (1, "\E[H\E[2J", 7);
    return 0;
}

int
set_default_color ()
{
    write (1, "\033[0m", 4);
    return 0;
}

int
mt_gotoXY (int _x, int _y)
{
    int count_rows, count_columns;

    if (mt_getscreensize (&count_rows, &count_columns) == -1)
    {
        return -1;
    }
}

```

```

    if ((_x > count_columns) || (_x <= 0) || (_y > count_rows) || (_y <= 0))
    {
        return -1;
    }

    char buf[16];
    int len = snprintf (buf, 16, "\033[%d;%dH", _y, _x);
    write (1, buf, len);

    return 0;
}

int
mt_getscreensize (int *_rows, int *_columns)
{
    struct winsize window_size;

    if (ioctl (1, TIOCGWINSZ, &window_size))
    {
        return -1;
    }

    *_rows = window_size.ws_row;
    *_columns = window_size.ws_col;

    return 0;
}

int
mt_setfgcolor (enum colors color)
{
    char buf[16];
    int len = snprintf (buf, 16, "\033[38;5;%dm", color);
    write (1, buf, len);
    return 0;
}

int
mt_setbgcolor (enum colors color)
{
    char buf[16];
    int len = snprintf (buf, 16, "\033[48;5;%dm", color);
    write (1, buf, len);
    return 0;
}


```

Файл **myTerm.h**

```

#ifndef MYTERM_H
#define MYTERM_H

```



```

#include <stdio.h>
#include <sys/ioctl.h>
#include <unistd.h> // for write and dup2

enum colors
{
    PEACH = 203,
    RED = 196,
    SOFT_GREEN = 192,
    VIOLET = 165,
    LIGHT_BLUE = 111,
    BLUE = 20,
    BLACK = 16,
    WHITE = 15,
    GREEN = 10,
    DEFAULT = 0
};

int mt_clrscr ();
int mt_gotoXY (int _x, int _y);
int mt_getscreenize (int *rows, int *cols);
int mt_setfgcolor (enum colors);
int mt_setbgcolor (enum colors);
int set_default_color ();

#endif

```

Файл test_bigchars.c

```

#include "../src/myBigChars.h"
#include "../src/myInterface.h"
#include "../src/mySimpleComputer.h"
#include "../src/myTerm.h"
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>

extern int instruction_counter;

void
testBIGCHARS ()
{
    mt_clrscr ();
    bc_printBigChar (big_chars[8], 5, 10, 2, 1);
    mt_gotoXY (1, 24);
}

int
main ()
{
    // mySuperComputer();
    // myTerm();
}

```

```

testBIGCHARS ();

return 0;
}

```

Файл test_comp.c

```

#include "../src/myInterface.h"

#include "../src/mySimpleComputer.h"
#include "../src/myTerm.h"
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>

void
mySuperComputer ()
{

    for (int i = 0; i < 2500; ++i)
    {

        sc_commandDecode(i,&a,&b);
        printf("%d - %d %d\n", i, a, b);
    }
    int value;
    sc_memoryInit ();
    sc_regInit ();
    sc_memorySet (0, 16);
    sc_memorySet (1, 14);
    sc_memorySet (2, 12);
    sc_memorySet (3, 10);
    sc_memorySet (4, 8);
    sc_memorySet (5, 6);

    for (int i = 0; i < 6; ++i)
    {
        sc_memoryGet (i, &value);
        printf ("RAM [ %d ] : %d\n", i, value);
    }

    sc_memorySave ("savefile.bin");
    printf ("RAM was saved\n");
    sc_memorySet (1, 99);
    sc_memoryGet (1, &value);
    printf ("RAM [1] : %d\n", value);
    sc_memoryLoad ("savefile.bin");
    printf ("RAM was loaded\n");
    for (int i = 0; i < 6; ++i)
    {
        sc_memoryGet (i, &value);
    }
}

```

```

        printf ("RAM [ %d ] : %d\n", i, value);
    }

    sc_regSet (DIVISION_ERROR_BY_ZERO, 1);
    sc_regSet (OVERFLOW, 1);
    sc_regSet (INVALID_COMMAND, 1);

    sc_regGet (OVERFLOW, &value);
    printf ("[FLAG] OVERFLOW: %d\n", value);
    sc_regGet (DIVISION_ERROR_BY_ZERO, &value);
    printf ("[FLAG] DIVISION_ERROR_BY_ZERO: %d\n", value);
    sc_regGet (OUT_OF_MEMORY, &value);
    printf ("[FLAG] OUT_OF_MEMORY: %d\n", value);
    sc_regGet (IGNORING_CLOCK_PULSES, &value);
    printf ("[FLAG] IGNORING_CLOCK_PULSES: %d\n", value);
    sc_regGet (INVALID_COMMAND, &value);
    printf ("[FLAG] INVALID_COMMAND: %d\n", value);

    sc_commandEncode (65, 127, &value);
    printf ("Value: %d\n", value);
    int command = 0;
    int operand = 0;
    sc_commandDecode (value, &command, &operand);
    printf ("Command: %d  Operand: %d\n", command, operand);
}

int
main ()
{
    mySuperComputer ();
    return 0;
}

```

Файл test_readkey.c

```

#include "../src/myReadKey.h"
#include "../src/mySimpleComputer.h"

int
main ()
{
    rk_myTermSave ();
    enum keys key;
    int value;
    sc_memorySet (0, 5);
    sc_memoryGet (0, &value);
    printf ("address[0] = %d\n", value);
    printf ("Enter S_KEY\n");
    rk_readKey (&key);
    if (key == S_KEY)

```

```

    sc_memorySave ("savefile.bin");
    printf ("Successful save\n");
    sc_memorySet (0, 6);
    sc_memoryGet (0, &value);
    printf ("address[0] = %d\n", value);
    printf ("Value was changed! Enter L_KEY for load previous setup\n");
    rk_readKey (&key);
    if (key == L_KEY)
        sc_memoryLoad ("savefile.bin");
    sc_memoryGet (0, &value);
    printf ("address[0] = %d\n", value);
    key = ESC_KEY;
    return 0;
}

```

Файл test_term.c

```

#include "../src/myInterface.h"

#include "../src/mySimpleComputer.h"
#include "../src/myTerm.h"
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>

void
myTerm ()
{
    mt_clrscr ();
    mt_gotoXY (4, 4);
    mt_setfgcolor (BLUE);
    printf ("Hello with blue color\n");
    mt_setbgcolor (WHITE);
    printf ("Goodbye with white background color\n");
    printf ("\033[0m");
}

int
main ()
{
    // mySuperComputer();
    myTerm ();
    // srand (time (NULL));

    return 0;
}

```

Файл rpnTranslator.c

```

#include "rpnTranslator.h"
void
stack_push (char data, Node **top)

```

```

{
    Node *tmp = (Node *)malloc (sizeof (Node));
    tmp->data = data;
    tmp->next = *top;
    *top = tmp;
}

```

```

char
stack_pop (Node **top)
{
    Node *tmp;
    char d;
    if (*top == NULL)
    {
        // printf("Stack/Queue is underflow!\n");
        return -1;
    }
    else
    {
        tmp = *top;
        *top = tmp->next;
        d = tmp->data;
        free (tmp);
        return d;
    }
}

```

```

char
stack_top (Node *top)
{
    if (top != NULL)
    {
        return top->data;
    }
    return 0;
}

```

```

int
checkPriority (char sign)
{
    switch (sign)
    {
        case '*':
        case '/':
            return 4;
        case '+':
        case '-':
            return 2;
        case '(':
        case ')':

```

```

        return 1;
    }
    return 0;
}

// Функция translateToRPN принимает в качестве аргументов строку инфиксного
// выражения и строку для хранения обратной польской записи
char *
translateToRPN (char *inf, char *rpn)
{
    // Создание корневого узла стека
    Node *root = NULL;
    int i = 0;
    int j = 0;

    // Пока не достигнут конец строки и не встречен символ новой строки
    while (inf[i] != '\0' && inf[i] != '\n')
    {
        char x = inf[i];

        // Если встречен операнд, добавляем его в выходную строку
        if ((x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z'))
        {
            rpn[j] = x;
            j++;
        }

        // Если встречена открывающая скобка, добавляем ее в стек
        else if (x == '(')
        {
            stack_push (x, &root);
        }

        // Если встречена закрывающая скобка, извлекаем элементы из стека, пока
        // не достигнем открывающую скобку и добавляем их в выходную строку
        else if (x == ')')
        {
            while (stack_top (root) != '(')
            {
                char c = stack_pop (&root);
                if (c != 0)
                {
                    rpn[j] = c;
                    j++;
                }
            }
            // Извлекаем открывающую скобку из стека
            stack_pop (&root);
        }

        // Если встречен оператор, извлекаем операторы из стека, пока их

```

```

// приоритет не станет меньше приоритета текущего оператора, и добавляем
// их в выходную строку
else if (x == '+' || x == '-' || x == '*' || x == '/')
{
    while (root != NULL
           && checkPriority (root->data) >= checkPriority (x))
    {
        char c = stack_pop (&root);
        if (c != 0)
        {
            rpn[j] = c;
            j++;
        }
    }

    // Добавляем текущий оператор в стек
    stack_push (x, &root);
}

// Если встречен недопустимый символ, выводим сообщение об ошибке и
// выходим из функции
else if (x != ' ')
{
    // free(rpn); // этот код, возможно, лишний
    fprintf (stderr, "Wrong expression!\n");
    exit (EXIT_FAILURE);
}
i++;
}

// Извлекаем все оставшиеся операторы из стека и добавляем их в выходную
// строку
while (root != NULL)
{
    char c = stack_pop (&root);
    if (c != 0)
    {
        rpn[j] = c;
        j++;
    }
}

// Проверяем, что выходная строка корректно сформирована
for (int k = 0; k < j; k++)
{
    if (rpn[k] == '(' || rpn[k] == ')')
    {
        fprintf (stderr, "Check your expression!\n");
        exit (EXIT_FAILURE);
    }
}

```

```

    // Добавляем нуль-терминальный символ в конец выходной строки
    rpn[j] = '\\0';
    return rpn;
}

```

Файл **rpnTranslator.h**

```

#ifndef RPN_TRANSLATOR_H
#define RPN_TRANSLATOR_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct NODE
{
    char data;
    struct NODE *next;
} Node;

void stack_push (char data, Node **top);
void stack_print (Node *top);
char stack_pop (Node **top);
char stack_top (Node *top);
int checkPriority (char sign);
char *translateToRPN (char *inf, char rpn[]);

#endif // RPN_TRANSLATOR_H

```

Файл **sat.c**

```

#include "../src/mySimpleComputer.h" // Подключение заголовочного файла для
работы со SC (Simple Computer)

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int
asm_analis (char *comm) // Функция для определения кода операции из строки с
                        // мнемокодом операции
{
    if (strcmp (comm, "READ") == 0)
        return 0x10;
    if (strcmp (comm, "WRITE") == 0)
        return 0x11;
    if (strcmp (comm, "LOAD") == 0)
        return 0x20;
    if (strcmp (comm, "STORE") == 0)
        return 0x21;
}

```



```

    if (strcmp (comm, "ADD") == 0)
        return 0x30;
    if (strcmp (comm, "SUB") == 0)
        return 0x31;
    if (strcmp (comm, "DIVIDE") == 0)
        return 0x32;
    if (strcmp (comm, "MUL") == 0)
        return 0x33;
    if (strcmp (comm, "JUMP") == 0)
        return 0x40;
    if (strcmp (comm, "JNEG") == 0)
        return 0x41;
    if (strcmp (comm, "JZ") == 0)
        return 0x42;
    if (strcmp (comm, "HALT") == 0)
        return 0x43;
    if (strcmp (comm, "JC") == 0)
        return 0x56;
    if (strcmp (comm, "JNS") == 0)
        return 0x55;
    if (strcmp (comm, "KOR") == 0)
        return 0x61;
    if (strcmp (comm, "MOVR") == 0)
        return 0x72;
    if (comm[0] == '=')
        return 1;
    return -1;
}
int
main (int argc, char **argv) // Функция main
{
    char line[100],
        ch; // Строка для чтения данных, переменная ch для проверки знака
    char strComm[10],
        operand_str[10]; // Строки для чтения мнемокода операции и операнда
    int value, command, operand,
        address; // переменные для чтения значений из строки
    FILE *file; // Указатели на файлы
    if (argc != 3) // Если не переданы два параметра через командную строку
    {
        printf (
            "Usage: sat file.sa file.o\n "); // Выводим сообщение о правильном
            // использовании программы
        return -1; // Завершаем программу с ошибкой
    }
    else if ((file = fopen (argv[1], "rb"))
        <= 0) // Попытка открыть файл с исходным кодом для чтения
    {
        printf ("Can`t open '%s' file.\n",
            argv[1]); // Выводим сообщение о невозможности открыть файл
    }
}

```

```

        return -1; // Завершаем программу с ошибкой
    }
    sc_memoryInit (); // Инициализируем память Simple Computer'a
    do // Цикл трансляции, продолжающийся до конца файла
    {
        fgets (line, sizeof (line), file); // Читаем строку из файла
        if (sscanf (line, "%d %s %s", &address, strComm, operand_str)
            < 3) // Разбираем строку на составляющие
        {
            printf ("Translation error1\n"); // Выводим сообщение о невозможности
                                            // трансляции строки
            return -1; // Завершаем программу с ошибкой
        }
        command = asm_analis (strComm); // Определяем код операции из мнемокода
        if (command != -1) // Если код операции определен верно
        {
            if (command != 1) // Если операция не запись значения в ячейку памяти
            {
                if (sscanf (operand_str, "%d", &operand) != 1
                    || // Читаем операнд
                    sc_commandEncode (command, operand, &value) == -1
                    || // Кодировем операцию и операнд в машинный код
                    sc_memorySet (address, value)
                        == -1) // Записываем значение в ячейку памяти
                {
                    printf ("Translation error2\n"); // Выводим сообщение о
                                                    // невозможности трансляции строки
                    return -1; // Завершаем программу с ошибкой
                }
            }
        }
        else if (line[3] == '=')
        {
            int address = atoi (&line[0]);
            int value;
            if ('0' <= line[5] && line[5] <= '9')
                value = atoi (&line[5]);
            sc_memorySet (address, value);
        }
        else // Если операция записи значения в память
        {
            int a = 0;
            sscanf (operand_str, "%c%02d%02X", &ch, &command,
                &operand); // Читаем код операции и операнд
            if (ch == '-')
                a |= (1 << 14); // Если отрицательное значение
            value = a + (command << 7) + operand; // Формируем машинное слово
            printf ("Translation, %02d", command);
            if (sc_memorySet (address, value)
                == -1) // Записываем значение в ячейку памяти
            {

```

```

        printf ("Translation error3\n"); // Выводим сообщение о
                                           // невозможности трансляции строки
        return -1; // Завершаем программу с ошибкой
    }
}
}
else // Если код операции определен неверно
{
    printf ("Translation error4\n"); // Выводим сообщение о невозможности
                                     // трансляции строки
    return -1; // Завершаем программу с ошибкой
};
}
while (!feof (file)); // Цикл продолжается, пока не кончится файл
fclose (file); // Закрываем файл с исходным кодом

if (sc_memorySave (argv[2])
    == -1) // Попытка создать файл для записи машинного кода
{
    printf ("Can`t create '%s' file.\n",
            argv[2]); // Выводим ошибку о невозможности создания файла
    return -1; // Завершаем программу с ошибкой
}
// printf("Translation status: OK.\n"); // Выводим сообщение об успешном
// завершении трансляции
return 0; // Завершаем программу
}

```

Файл sbt.c

```

#include <stdio.h>

#include <stdlib.h>
#include <string.h>
//#include "mySimpleComputer.h"
#include "rpnTranslator.h"
void INPUT (char *arguments);
void PRINT (char *arguments);
void IF (char *arguments);
void GOTO (int address, int operand);
void function (char *command, char *arguments);
void parsRPN (char *rpn, char var);
void LET (char *arguments);
void END ();

#define ERR_EXPECTED_ADDRESS_OF_MEMORY_CELL -1
#define ERR_WRONG_COMMAND -2
#define ERR_WRONG_OPERAND -3
#define ERR_BY_ENCODE -4

struct variable

```

```

{
    char Name;
    int Address;
    int Value;
};

struct variable Variables[52];
int variableCount = 0;
char lastConstantName = 'a';

struct command
{
    int Number;
    char *Command;
    int Address;
};

FILE *input = NULL;
FILE *output = NULL;

const char commandID[][7]
    = { "REM", "INPUT", "PRINT", "GOTO", "IF", "LET", "END" };
int basicCommandCounter = 0;
int assemblerCommandCounter = 0;
struct command *program;

int gotoCounter = -1;
struct command *gotoRecords;

int
getVariableAddress (char name)
{
    // printf("%c ", name);
    for (int i = 0; i < 52; i++)
    {
        if (Variables[i].Name == name)
        {
            return Variables[i].Address;
        }

        if (!Variables[i].Name)
        {
            Variables[i].Name = name;
            Variables[i].Address = 99 - i;
            variableCount++;
            return Variables[i].Address;
        }
    }
    return 0;
}

```

```

char
intToConstant (int value)
{
    for (int i = 0; i < 52; i++)
    {
        if (!Variables[i].Name)
        {
            Variables[i].Name = lastConstantName;
            lastConstantName++;
            Variables[i].Address = 99 - i;
            Variables[i].Value = value;
            fprintf (output, "%.2i = %x\n", Variables[i].Address,
                    abs (Variables[i].Value));
            variableCount++;
            return Variables[i].Name;
        }
        if (Variables[i].Value)
        {
            if (Variables[i].Value == value)
            {
                return Variables[i].Name;
            }
        }
    }
    return 0;
}

```

```

char
preCalcProcessing (char *expr)
{
    char *ptr = strtok (expr, " =");
    char val;
    sscanf (ptr, "%c", &val);
    if (!((val) >= 'A' && (val) <= 'Z'))
    {
        fprintf (stderr, "NOT CORRECT!\n");
        exit (EXIT_FAILURE);
    }
    ptr = strtok (NULL, " ");
    char *eq = ptr;
    if (strcmp (eq, "=") != 0)
    {
        fprintf (stderr, "Wrong expression!\n");
        exit (EXIT_FAILURE);
    }
    ptr = strtok (NULL, "");
    char *exp = ptr;
    int i = 0;
    int pos = 0;
    int j = 0;

```

```

int operat = 0;
int flg = 1;
int m = 0;
char *assign = (char *)malloc (sizeof (char) * 255);
for (int k = 0; k < strlen (exp); k++)
{
    if (exp[k] == '-' && flg)
    {
        assign[m] = '0';
        m++;
    }
    flg = 0;
    if (exp[k] == '+' || exp[k] == '-' || exp[k] == '/' || exp[k] == '*')
    {
        operat++;
    }
    if (exp[k] == '+' || exp[k] == '-' || exp[k] == '/' || exp[k] == '*'
        || exp[k] == '(')
    {
        flg = 1;
    }
    assign[m] = exp[k];
    m++;
}
if (operat == 0) // 0+ перед ним, если перед минусом нет аргумента, то пишем
                // 0 перед минусом
{
    sprintf (exp, "0 + %s", assign);
}
else
{
    sprintf (exp, "%s", assign);
}
while (exp[i] != '\n' && exp[i] != '\0')
{
    if (exp[i] >= '0' && exp[i] <= '9')
    {
        char num[256];
        j = 0;
        num[j] = exp[i];
        j++;
        pos = i;
        exp[i] = ' ';
        i++;
        while (exp[i] >= '0' && exp[i] <= '9')
        {
            num[j] = exp[i];
            j++;
            exp[i] = ' ';
            i++;
        }
    }
}

```

```

        }
        num[j] = '\0';
        exp[pos] = intToConstant (atoi (num));
    }
    i++;
}
sprintf (expr, "%s", exp);

return val;
}

void
loadFile (const char *filename, const char *secondFilename)
{
    if ((input = fopen (filename, "r")) == NULL)
    {
        fprintf (stderr, "Can't open file: no such file.\n");
        exit (EXIT_FAILURE);
    }

    output = fopen (secondFilename, "w");
    return;
}

// Функция, отвечающая за перевод программы из языка BASIC в ассемблерный код
void
translation ()
{
    int instructionCounter = 1;
    // Считываем и подсчитываем количество команд в программе
    while (1)
    {
        char temp[255];
        fgets (temp, 255, input);
        if (feof (input))
        {
            break;
        }
        instructionCounter++;
    }
    basicCommandCounner = instructionCounter; // Записываем количество команд
    rewind (input); // Возвращаем указатель на начало файла

    // Выделяем память для массива команд программы и для массива gotoRecords
    program = (struct command *)malloc (sizeof (struct command)
                                         * instructionCounter);
    gotoRecords = (struct command *)malloc (sizeof (struct command)
                                             * instructionCounter);

    // Считываем команды программы и проверяем, удалось ли их считать
    for (int i = 0; i < instructionCounter; i++)

```

```

{
    program[i].Command = (char *)malloc (sizeof (char) * 255);
    if (!fgets (program[i].Command, 254, input))
    {
        if (feof (input))
        {
            return;
        }
        else
        {
            fprintf (stderr, "Line %d: can't read line from file.\n", i++);
            return;
        }
    }
}

// Проходимся по всем командам и разбираем их на номер строки, команду и
// аргументы
for (int i = 0; i < instructionCounter; i++)
{
    char *lin;
    char *thisCommand = (char *)malloc (sizeof (char) * 255);
    sprintf (thisCommand, "%s", program[i].Command);
    char *ptr = strtok (thisCommand, " ");
    lin = ptr;
    int line = atoi (lin);

    // Если не удалось прочитать номер строки, выводим ошибку и выходим из
    // функции
    if ((!line) && (strcmp (lin, "0") != 0) && (strcmp (lin, "00") != 0))
    {
        fprintf (stderr, "Line %d: expected line number.\n", i++);
        break;
    }

    // Проверяем, что номер строки больше, чем у предыдущей команды
    if (i - 1 >= 0)
    {
        if (line <= program[i - 1].Number)
        {
            fprintf (stderr, "Line %d: Wrong line number\n", i++);
            break;
        }
    }

    program[i].Number = line; // Записываем номер строки

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

```



```

ptr = strtok (NULL, "");
arguments = ptr;
program[i].Address
    = assemblerCommandCounter; // Записываем адрес команды помощью
                               // счетчика ассемблерного кода

// Если это не команда GOTO, вызываем соответствующую функцию с
// аргументами
if (strcmp (command, "GOTO") != 0)
{
    function (command, arguments);
}
// Иначе увеличиваем счетчик GOTO и сохраняем строку, адрес и номер
else
{
    gotoCounter++;
    gotoRecords[gotoCounter].Number = program[i].Number;
    gotoRecords[gotoCounter].Command = program[i].Command;
    gotoRecords[gotoCounter].Address = program[i].Address;
    assemblerCommandCounter++;
}
}
// Обрабатываем все команды GOTO в конце
for (int i = 0; i <= gotoCounter; i++)
{
    int address = gotoRecords[i].Address;
    char *ptr = strtok (gotoRecords[i].Command, " ");
    ptr = strtok (NULL, " ");
    ptr = strtok (NULL, "");
    int operand = atoi (ptr);
    GOTO (address, operand);
}
}

void
INPUT (char *arguments)
{
    if (!((strlen (arguments) == 2) && (arguments[0] >= 'A')
        && (arguments[0] <= 'Z'))))
    {
        fprintf (stderr, "Wrong variable name.\n");
        exit (EXIT_FAILURE);
    }

    fprintf (output, "%.2i READ %d\n", assemblerCommandCounter,
        getVariableAddress (arguments[0]));
    assemblerCommandCounter++;
}

void

```

```

PRINT (char *arguments)
{
    if (!((strlen (arguments) == 2) && (arguments[0] >= 'A')
        && (arguments[0] <= 'Z'))))
    {
        fprintf (stderr, "Wrong variable name.\n");
        exit (EXIT_FAILURE);
    }
    fprintf (output, "%.2i WRITE %d\n", assemblerCommandCounter,
        getVariableAddress (arguments[0]));
    assemblerCommandCounter++;
}

void
parsRPN (char *rpn, char var)
{
    int i = 0;
    int depth = 0;
    int operand1, operand2;
    char memoryCounter = '1';
    while (rpn[i] != '\0' && rpn[i] != '\n')
    {
        char x = rpn[i];
        if ((x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z'))
        {
            fprintf (output, "%.2i LOAD %d\n", assemblerCommandCounter,
                getVariableAddress (x));
            assemblerCommandCounter++;
            fprintf (output, "%.2i STORE %d\n", assemblerCommandCounter,
                getVariableAddress (memoryCounter));
            memoryCounter++;
            assemblerCommandCounter++;
            depth++;
        }
        if (x == '+' || x == '-' || x == '*' || x == '/')
        {
            if (depth < 2)
            {
                fprintf (stderr,
                    "Uncorrect LET statement, check your expression.\n");
                exit (EXIT_FAILURE);
            }
            else
            {
                operand1 = getVariableAddress (memoryCounter - 2);
                operand2 = getVariableAddress (memoryCounter - 1);
                fprintf (output, "%.2i LOAD %d\n", assemblerCommandCounter,
                    operand1); //закидываем самый правый операнд в акк
                assemblerCommandCounter++;
            }
        }
    }
}

```

```

        switch (x)
        {
            case '+':
                fprintf (output, "%.2i ADD %d\n", assemblerCommandCounter,
                    operand2);
                assemblerCommandCounter++;
                break;
            case '-': // SUB
                fprintf (output, "%.2i SUB %d\n", assemblerCommandCounter,
                    operand2);
                assemblerCommandCounter++;
                break;
            case '/': // DIVIDE
                fprintf (output, "%.2i DIVIDE %d\n", assemblerCommandCounter,
                    operand2);
                assemblerCommandCounter++;
                break;
            case '*': // MUL
                fprintf (output, "%.2i MUL %d\n", assemblerCommandCounter,
                    operand2);
                assemblerCommandCounter++;
                break;
        }
        fprintf (output, "%.2i STORE %d\n", assemblerCommandCounter,
            getVariableAddress (memoryCounter - 2));
        assemblerCommandCounter++;
        depth--;
        memoryCounter = memoryCounter - 1;
    }
    i++;
}
if (depth != 1)
{
    fprintf (stderr, "Uncorrect LET statement, check your expression.\n");
    exit (EXIT_FAILURE);
}
fprintf (output, "%.2i STORE %d\n", assemblerCommandCounter,
    getVariableAddress (var));
assemblerCommandCounter++;
}
void
GOTO (int address, int operand)
{
    for (int i = 0; i < basicCommandCounter; i++)
    {
        if (program[i].Number == operand)
        {
            fprintf (output, "%.2i JUMP %d\n", address, program[i].Address);
            return;
        }
    }
}

```

```

    }
}
fprintf (stderr, "Reference to an inspecified memory location.\n");
exit (EXIT_FAILURE);
}

void
IF (char *arguments)
{
    int mySign = -1;

    int before = 0;

    int after = 0;

    for (int i = 0; i < strlen (arguments); i++)
    {
        if ((arguments[i] == '>') || (arguments[i] == '<')
            || (arguments[i] == '='))
        {
            mySign = i;

            if (!(arguments[i - 1] == ' '))
            {
                before = 1;
            }

            if (!(arguments[i + 1] == ' '))
            {
                after = 1;
            }
            break;
        }
    }

    char *expression = (char *)malloc (sizeof (char) * 255);
    if (!(before) && !(after))
    {
        expression = strtok (arguments, "");
    }
    else
    {
        int j = 0;
        for (int i = 0; i < strlen (arguments); i++)
        {
            if (i == mySign)
            {
                if (before)
                {

```

```

        expression[j] = ' ';
        j++;
    }
    expression[j] = arguments[i];
    j++;
    if (after)
    {
        expression[j] = ' ';
        j++;
    }
}
else
{
    expression[j] = arguments[i];
    j++;
}
}
expression[j] = '\0';
}

char *ptr = strtok (expression, " ");
char *operand1 = ptr;
char operand1Name;

if (strlen (operand1) > 1)
{
    if (atoi (operand1))
    {
        operand1Name = intToConstant (atoi (operand1));
    }
}
else
{
    if ((operand1[0] >= '0') && (operand1[0] <= '9'))
    {
        operand1Name = intToConstant (atoi (operand1));
    }
    else if ((operand1[0] >= 'A') && (operand1[0] <= 'Z'))
    {
        operand1Name = operand1[0];
    }
    else
    {
        fprintf (stderr, "Wrong statement!\n");
        exit (EXIT_FAILURE);
    }
}

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

```

```

ptr = strtok (NULL, " ");
char *operand2 = ptr;
char operand2Name;
if (strlen (operand2) > 1)
{
    if (atoi (operand2))
    {
        operand2Name = intToConstant (atoi (operand2));
    }
}
else
{
    if ((operand2[0] >= '0') && (operand2[0] <= '9'))
    {
        operand2Name = intToConstant (atoi (operand2));
    }
    else if ((operand2[0] >= 'A') && (operand2[0] <= 'Z'))
    {
        operand2Name = operand2[0];
    }
    else
    {
        fprintf (stderr, "Wrong statement!\n");
        exit (EXIT_FAILURE);
    }
}

int falsePosition = -1;

if (logicalSign[0] == '<')
{
    fprintf (output, "%.2i LOAD %d\n", assemblerCommandCounter,
            getVariableAddress (operand1Name));
    assemblerCommandCounter++;
    fprintf (output, "%.2i SUB %d\n", assemblerCommandCounter,
            getVariableAddress (operand2Name));
    assemblerCommandCounter++;
    fprintf (output, "%.2i JNEG %d\n", assemblerCommandCounter,
            assemblerCommandCounter + 2);
    assemblerCommandCounter++;
    falsePosition = assemblerCommandCounter;
    assemblerCommandCounter++;
}
else if (logicalSign[0] == '>')
{
    fprintf (output, "%.2i LOAD %d\n", assemblerCommandCounter,
            getVariableAddress (operand2Name));
    assemblerCommandCounter++;
    fprintf (output, "%.2i SUB %d\n", assemblerCommandCounter,
            getVariableAddress (operand1Name));
}

```

```

        assemblerCommandCounter++;
        fprintf (output, "%.2i JNEG %d\n", assemblerCommandCounter,
                assemblerCommandCounter + 2);
        assemblerCommandCounter++;
        falsePosition = assemblerCommandCounter;
        assemblerCommandCounter++;
    }
else if (logicalSign[0] == '=')
{
    fprintf (output, "%.2i LOAD %d\n", assemblerCommandCounter,
            getVariableAddress (operand1Name));
    assemblerCommandCounter++;
    fprintf (output, "%.2i SUB %d\n", assemblerCommandCounter,
            getVariableAddress (operand2Name));
    assemblerCommandCounter++;
    fprintf (output, "%.2i JZ %d\n", assemblerCommandCounter,
            assemblerCommandCounter + 2);
    assemblerCommandCounter++;
    falsePosition = assemblerCommandCounter;
    assemblerCommandCounter++;
}

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

if (strcmp (command, "IF") == 0)
{
    fprintf (stderr, "Multiple IF!\n");
    exit (EXIT_FAILURE);
}
else if (strcmp (command, "GOTO") != 0)
{
    function (command, commandArguments);
}
else
{
    gotoCounter++;
    gotoRecords
        = realloc (gotoRecords, sizeof (struct command) * gotoCounter + 1);
    struct command gotoCommand;
    gotoCommand.Address = assemblerCommandCounter;
    char *buff = (char *)malloc (sizeof (char) * 255);
    sprintf (buff, "%d GOTO %s", falsePosition, commandArguments);
    gotoCommand.Command = buff;
    gotoRecords[gotoCounter] = gotoCommand;
    assemblerCommandCounter++;
}

fprintf (output, "%.2i JUMP %d\n", falsePosition, assemblerCommandCounter);

```

```

    // assemblerCommandCounter++;
}

void
LET (char *arguments)
{
    char fin[255];
    char var;
    var = preCalcProcessing (arguments);
    translateToRPN (arguments, fin);
    // printf("%c\n", var);
    parsRPN (fin, var);
}

void
END ()
{
    fprintf (output, "%.2i HALT 00\n", assemblerCommandCounter);
    assemblerCommandCounter++;
}

void
function (char *command, char *arguments)
{
    if (strcmp (command, "REM") == 0)
    {
    }
    else if (strcmp (command, "INPUT") == 0)
    {
        INPUT (arguments);
    }
    else if (strcmp (command, "PRINT") == 0)
    {
        PRINT (arguments);
    }
    else if (strcmp (command, "IF") == 0)
    {
        IF (arguments);
    }
    else if (strcmp (command, "LET") == 0)
    {
        LET (arguments);
    }
    else if (strcmp (command, "END") == 0)
    {
        END ();
    }
    else
    {
        fprintf (stderr, "%s is not a command.\n", command);
    }
}

```



```

    }
}

int
main (int argc, const char **argv)
{
    if (argc < 3)
    {
        fprintf (stderr, "Usage: %s input.sa output.o\n", argv[0]);
        exit (EXIT_FAILURE);
    }

    loadFile (argv[1], "tmp.sa");

    translation ();
    fclose (input);
    fclose (output);

    if (variableCount + assemblerCommandCounter > 100)
    {
        fprintf (stderr, "RAM overflow error!\n");
        system ("rm -rf tmp.sa");
    }

    char sat[255];
    sprintf (sat, "./sat tmp.sa %s\n", argv[2]);
    system (sat);

    if (argc >= 4)
    {
        if (argv[3][0] == '1')
        {
            return 0;
        }
    }
    // system("rm -rf tmp.sa");
    return 0;
}

```

Файл .gitignore

```

obj/*.o
include/*.a
.vscode
bin/console
bin/test_comp
bin/test_term
bin/test_bigchars
bin/test_readkey

```

Файл `git-lab-ci.yml`

```
image: registry.csc.sibsutis.ru/ci/git-clang-format:latest

stages:
  - check-format
  - build
  - test

check-format:
  stage: check-format
  script:
    - find . -type f -name '*.ch' | xargs clang-format --style GNU -i --verbose
    - git diff --exit-code

build:
  stage: build
  script:
    - make
```

Выводы

В результате выполнения данной курсовой работы были реализованы: SimpleComputer, интерфейс, Устройство управления(УУ), Арифметическо-логическое устройство(АЛУ). Позже разрабатывались трансляторы. Для Бейсика была реализована «обратная польская запись» или же RPN. Создающийся файл `tmp.sa` представляет собой переведенный код из Бейсика в Ассемблер.

Список используемой литературы

1. Мамоиленко С.Н. и Молдованова О.В., ЭВМ и периферийные устройства: учебное пособие, 2012. 106 с.

Ссылка на проект:

<https://git.csc.sibsutis.ru/ip114s27/myfirstproject>

Подпись и расшифровка: ____/____