Министерство цифрового развития,
связи и массовых коммуникаций Российской Федерации
СибГУТИ

Кафедра Вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине «Архитектура ЭВМ»
на тему Трансляция с языков Simple Assembler и Simple Basic

Выполнил студент группы ИП-012
Николаев А.Д.
Работу принял доцент Кафедры ВС
Ю. С. Майданов

Новосибирск – 2022 г.

Оглавление

Условие задачи

В рамках курсовой работы необходимо доработать модель Simple Computer так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице 1. Из пользовательских функций необходимо реализовать только одну согласно варианту задания (номеру вашей учетной записи). Для разработки программ требуется создать трансляторы с языков Simple Assembler 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 им-пульсов», либо при нажатии на клавишу – “t”. Алгоритм работы функции следующий:

1. из оперативной памяти считывается ячейка, адрес которой храниться 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.

Реферат

Объём пояснительной записки – страниц;

Количество таблиц в документе – 1;

Количество рисунков – 9;

Количество схем – 1;

Количество программ – 13;

Количество приложений – 0;

Результаты работы – были реализованы программы, переводящие текст с языка Simple Basic на язык Simple Assembler и с языка Simple Assembler в бинарный формат. Была доработана модель Simple Computer, созданная входе выполнения лабораторных работ, для обработки команд, записанных в оперативной памяти;

Содержание курсовой работы

Архитектура Simple Computer представлена на рисунке 1 и включает следующие функциональные блоки:

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

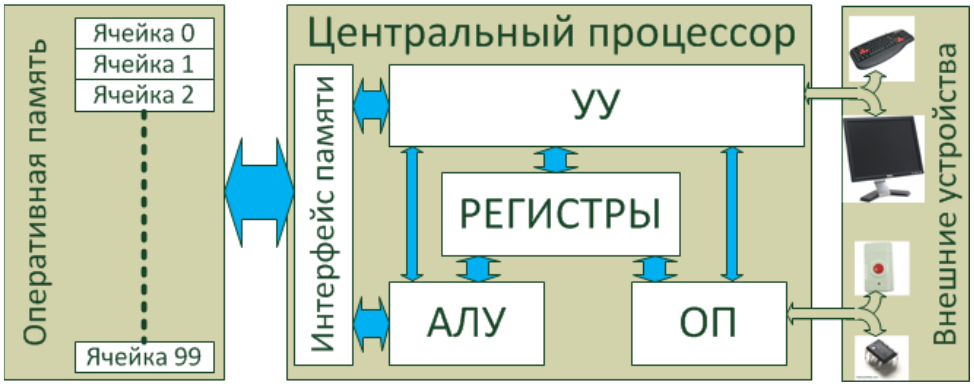

Рисунок 1. Архитектура вычислительной машины Simple Computer

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-разрядным числом. Предполагаем, что Simple Computer оборудован памятью из 100 ячеек (с адресами от 0 до 99_{10}).

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 – блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде ко-манды 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 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 2): 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.

Рисунок 2. Формат 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		
NOT	51	Двоичная инверсия слова в аккумуляторе и занесение результата в указанную ячейку памяти
AND	52	Логическая операция И между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
OR	53	Логическая операция ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
XOR	54	Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
JNS	55	Переход к указанному адресу памяти, если в аккумуляторе находится положительное число
JC	56	Переход к указанному адресу памяти, если при сложении произошло переполнение
JNC	57	Переход к указанному адресу памяти, если при сложении не произошло переполнение
JP	58	Переход к указанному адресу памяти, если результат предыдущей операции четный
JNP	59	Переход к указанному адресу памяти, если результат предыдущей операции нечетный
CHL	60	Логический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
SHR	61	Логический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)
RCL	62	Циклический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
RCR	63	Циклический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)
NEG	64	Получение дополнительного кода содержимого указанной ячейки памяти (результат в аккумуляторе)
ADDC	65	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
SUBC	66	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
LOGLC	67	Логический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
LOGRC	68	Логический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
RCCL	69	Циклический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
RCCR	70	Циклический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
MOVA	71	Перемещение содержимого указанной ячейки памяти в ячейку, адрес которой указан в аккумуляторе
MOVR	72	Перемещение содержимого ячейки памяти, адрес которой содержится в аккумуляторе в указанную ячейку памяти.
MOVCA	73	Перемещение содержимого указанной ячейки памяти в ячейку памяти, адрес которой находится в ячейке памяти, на которую указывает значение аккумулятора
MOVCR	74	Перемещение в указанный участок памяти содержимого участка памяти, адрес которого находится в участке памяти указанном в аккумуляторе
ADDC	75	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)
SUBC	76	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)

Таблица 1. Команды центрального процессора Simple Computer

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/Otput” – 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”. (см. схему 1) Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу “i”.

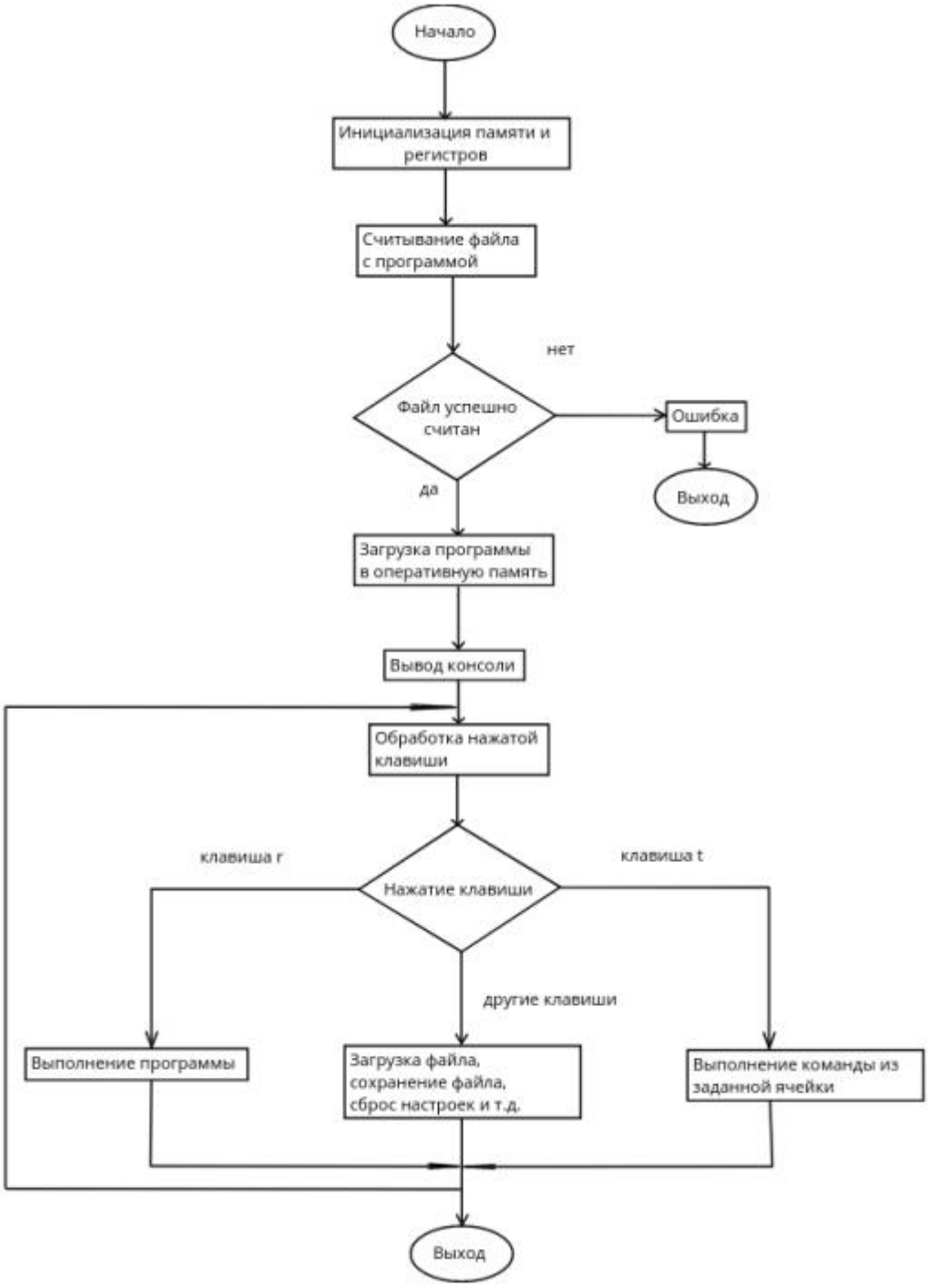

Схема 1. Блок-схема используемых алгоритмов

Программная реализация

Транслятор Simple Basic

```
#include<cstdio>
#include<cstdlib>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <regex>
#include <cstring>
#include <iomanip>

using namespace std;

struct SimpleAssembler {
    string command = "";
    int operand;
};

struct SimpleBasic {
    string lb = "";
    int pos;
};
SimpleAssembler Memory[100];
int counterStart = 0, counterEnd = 99, tagCounter = 0, con-
stCounter = 0, last = 0;
SimpleBasic Labels[100];
SimpleBasic ConstVars[100];
int Name_Address[26] = {
    0
};

int Prioritet(char c) {
    if (c < 0) return 4;
    switch (c) {
    case '(':
        return 1;
    case '-':
        return 2;
    case '+':
        return 2;
    case '*':
        return 3;
    case '/':
        return 3;
    }
```

```
    return -1;
}

int VarAddress(char c) {
    if ((c >= 'A') && (c <= 'Z')) {
        if (Name_Address[c - 'A'] < 0) {
            Name_Address[c - 'A'] = counterEnd;
            Memory[counterEnd].operand = 0;
            Memory[counterEnd].command = "=";
            counterEnd--;
        }
        return Name_Address[c - 'A'];
    } else
        return -1;
}

void SetMemory(string command, int operand) {
    Memory[counterStart].command = command;
    Memory[counterStart].operand = operand;
    counterStart++;
}

int ConstAddress(string s) {
    int a;
    try {
        a = stoi(s);
    } catch (const exception & e) {
        e.what();
        cout << "WRONG CONST 1" << s;
        exit(1);
    }
    for (int i = 0; i < constCounter; i++) {
        if (ConstVars[i].lb == s) {
            return ConstVars[i].pos;
        }
    }
    if (abs(a) > 0x1FFF) {
        cout << "WRONG CONST 2" << s;
        exit(1);
    }
    ConstVars[constCounter].lb = s;
    ConstVars[constCounter].pos = counterEnd;
    if (a < 0) a = abs(a) + (1 << 13);
    Memory[counterEnd].operand = a;
    Memory[counterEnd].command = "=";
    counterEnd--;
    constCounter++;
```

```
    return counterEnd + 1;
}

void RPN_to_SA(string in , char c) {
    int depth = 0;
    string tmp = "";
    string out = "";
    for (unsigned int i = 0; i < in .size(); i++) {
        if (last > counterEnd - depth + 1) {
            cout << "ERROR OVERFLOW MEMORY";
            exit(1);
        } //Check memory for temp operations
        if (( in [i] >= 'A' && in [i] <= 'Z')) {
            SetMemory("LOAD", VarAddress( in [i]));
            SetMemory("STORE", counterEnd - depth);
            depth++;
        } else if ( in [i] == ' ') {
            tmp = "";
            i++;
            while ( in [i] != ' ') {
                tmp += in [i];
                i++;
            }
            SetMemory("LOAD", ConstAddress(tmp));
            SetMemory("STORE", counterEnd - depth);
            depth++;
        } else {
            if ( in [i] < 0) {
                if (depth < 1) {
                    cout << "ERROR NOT ENOUGH OPERAND TO OPER-
STION IN LET";
                    exit(1);
                }
                if (- in [i] == '-') {
                    SetMemory("LOAD", ConstAddress("0"));
                    SetMemory("SUB", counterEnd - depth + 1);
                    SetMemory("STORE", counterEnd - depth +
1);
                }
            } else {
                if (depth < 2) {
                    cout << "ERROR NOT ENOUGH OPERAND TO OPER-
ATION IN LET";
                    exit(1);
                }
                int st1 = counterEnd - depth + 1;
                int st2 = st1 + 1;
```

```
                SetMemory("LOAD", st2);
                switch ( in [i]) {
                case '+':
                    SetMemory("ADD", st1);
                    break;

                case '-':
                    SetMemory("SUB", st1);
                    break;

                case '/':
                    SetMemory("DIVIDE", st1);
                    break;

                case '*':
                    SetMemory("MUL", st1);
                    break;
                }
                SetMemory("STORE", st2);
                depth--;
            }
        }
    }
    if (depth != 1) {
        cout << "ERROR TOO MANY OPERAND";
        exit(1);
    }
    SetMemory("LOAD", counterEnd);
    SetMemory("STORE", VarAddress(c));
}

int SearchLabel(string str) {
    for (int i = 0; i < tagCounter; i++) {
        if (Labels[i].lb == str) return Labels[i].pos;
    }
    return -1;
}

void SetLabel(string str1) {
    if (SearchLabel(str1) != -1) {
        cout << "ERROR CAN NOT FIND TAG " << str1;
        exit(1);
    }
    if (tagCounter > 0 && (Labels[tagCounter - 1].lb.length()
> str1.length() || (Labels[tagCounter - 1].lb.length() ==
str1.length() && Labels[tagCounter - 1].lb >= str1))) {
```

```
        cout << "ERROR WRONG TAG" << Labels[tagCounter - 1].lb
<< "|" << str1 << "|";
        exit(1);
    }
    Labels[tagCounter].lb = str1;
    Labels[tagCounter].pos = counterStart;
    tagCounter++;
    counterStart++;
}

int RPN(string & out, string in ) {
    int num_open = 0, num_close = 0;
    string opers = "";
    char prev = ' ';
    for (unsigned int i = 0; i < in .length(); i++) {
        if ( in [i] == ' ') continue;
        if ( in [i] == ')') {
            num_close++;
            if (num_open > num_close) {
                cout << "ERROR wrong operation";
                exit(1);
            }
            while (opers.back() != '(') {
                out += opers.back();
                opers.pop_back();
                if (opers.empty()) {
                    cout << "ERROR missing bracket \'(\'";
                    exit(1);
                }
            }
            opers.pop_back();
        }
        if ( in [i] >= 'A' && in [i] <= 'Z') {
            out += in [i];
        }
        if ( in [i] >= '0' && in [i] <= '9') {
            string tmp = "";
            while ( in [i] >= '0' && in [i] <= '9') {
                tmp += in [i];
                i++;
            }
            i--;
            int num;
            try {
                num = stoi(tmp);
            } catch (const exception & e) {
                e.what();
```

```
                cout << "Wrong: big number in LET";
                exit(1);
            }
            out += " " + to_string(num) + " ";
        }
        if ( in [i] == '(') {
            opers += "(";
            num_open++;
        }
        if (( in [i] == '+' || in [i] == '-') && (prev == '+'
|| prev == '-' || prev == '*' || prev == '/' || prev == '('))
{
            opers.push_back(- in [i]);
        } else if ( in [i] == '/' || in [i] == '*' || in [i]
== '+' || in [i] == '-') {
            if (opers.empty()) {
                opers += in [i];
            } else {
                if (Prioritet(opers.back()) < Prioritet( in
[i])) {
                    opers += in [i];
                } else {
                    while (!opers.empty() && (Prior-
itet(opers.back()) >= Prioritet( in [i]))) {
                        out += opers.back();
                        opers.pop_back();
                    }
                    opers += in [i];
                }
            }
        }
        prev = in [i];
    }
    if (num_open != num_close) {
        cout << "ERROR missing bracket";
        exit(1);
    }
    while (!opers.empty()) {
        out += opers.back();
        opers.pop_back();
    }
    return 0;
}

int SB_to_SA(string buf) {
    smatch sm;
```

```
    if (regex_search(buf, sm, regex("([0-9]+)\\s+REM"))) {
        return 1;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+LET\\s+([A-Z])\\s+=([A-Z0-9\\-+/*\\(\\) ]*)\\s*\n")))
{
        string rpnstr;
        if (RPN(rpnstr, sm[3].str()) == -1) cout << "ERROR";
        RPN_to_SA(rpnstr, sm[2].str()[0]);
        return 2;
    } else if (regex_search(buf, sm, regex("([0-9]+)\\s+IN-
PUT\\s+([A-Z])\\s*\n"))) {
        SetMemory("READ", VarAddress(sm[2].str()[0]));
        return 3;
    } else if (regex_search(buf, sm, regex("([0-9]+)\\s+OUT-
PUT\\s+([A-Z]\\s*)\n"))) {
        SetMemory("WRITE", VarAddress(sm[2].str()[0]));
        return 4;
    } else if (regex_search(buf, sm, regex("^(([0-
9]+)\\s+IF\\s+([-A-Z0-9]+)\\s*([<>=]{1,2})\\s*([-A-Z0-
9]+)\\s+GOTO\\s+([0-9]+)\\s*\n)$"))) {
        string s1, s2, s3;
        s1 = sm[3].str();
        s2 = sm[4].str();
        s3 = sm[5].str();
        int st1 = 0, st2 = 0;
        if (s1[0] >= 'A' && s1[0] <= 'Z') {
            if (s1.size() != 1) {
                cout << "ERROR WRONG FIRST ARGUMENT IN IF";
                exit(1);
            }
            st1 = VarAddress(s1[0]);
        } else {
            st1 = ConstAddress(s1);
        }
        if (s3[0] >= 'A' && s3[0] <= 'Z') {
            if (s3.size() != 1) {
                cout << "ERROR WRONG SECOND ARGUMENT IN IF";
                exit(1);
            }
            st2 = VarAddress(s3[0]);
        } else {
            st2 = ConstAddress(s3);
        }
        int d = -1;
        d = SearchLabel(sm[6].str());
        if (d == -1) {
            cout << "ERROR CAN NOT FIND TAG";
```

```
            exit(1);
        }
        if (s2 == ">") {
            SetMemory("LOAD", st2);
            SetMemory("SUB", st1);
            SetMemory("JNEG", d);
        } else if (s2 == "<") {
            SetMemory("LOAD", st1);
            SetMemory("SUB", st2);
            SetMemory("JNEG", d);
        } else if (s2 == "=") {
            SetMemory("LOAD", st2);
            SetMemory("SUB", st1);
            SetMemory("JZ", d);
        } else if (s2 == ">=") {
            SetMemory("LOAD", st2);
            SetMemory("SUB", st1);
            SetMemory("JNEG", d);
            SetMemory("JZ", d);
        } else if (s2 == "<=") {
            SetMemory("LOAD", st1);
            SetMemory("SUB", st2);
            SetMemory("JNEG", d);
            SetMemory("JZ", d);
        } else {
            cout << "ERROR";
            exit(1);
        }
        return 5;
    } else if (regex_search(buf, sm, regex("^(([0-
9]+)\\s+GOTO\\s+([0-9]+)\n)$"))) {
        int d = -1;
        d = SearchLabel(sm[3].str());
        if (d != -1) {
            SetMemory("JUMP", d);
            return 6;
        }
        return -1;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+END\\s*\n"))) {
        SetMemory("HALT", 0);
        return 7;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+PRINT\\s+([A-Z]\\s*)\n"))) {
        SetMemory("WRITE", VarAddress(sm[2].str()[0]));
        return 4;
    } else return -1;
```

```
    return 0;
}

int Check_SB(string buf) {
    smatch sm;
    if (regex_search(buf, sm, regex("([0-
9]+)\\s+REM\\s+(.*)\n"))) {
        Labels[tagCounter].lb = sm[1].str();
        Labels[tagCounter].pos = counterStart;
        tagCounter++;
        return 1;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+LET\\s+([A-Z])\\s+=([A-Z0-9\\-+/*\\(\\) ]*)\\s*\n")))
{
        SetLabel(sm[1].str());
        string rpnstr;
        if (RPN(rpnstr, sm[3].str()) == -1) {
            cout << "ERROR WRONG LET  in line " <<
sm[1].str();
            exit(1);
        }
        //cout << rpnstr << endl;
        for (unsigned int i = 0; i < rpnstr.size(); i++) {
            if (VarAddress(rpnstr[i]) != -1) counterStart +=
2;
            if (rpnstr[i] == ' ') {
                string tmp = "";
                i++;
                while (rpnstr[i] != ' ') {
                    tmp += rpnstr[i];
                    i++;
                }
                ConstAddress(tmp);
                counterStart += 2;
            }
            if (rpnstr[i] == '+' || rpnstr[i] == '-' ||
rpnstr[i] == '/' || rpnstr[i] == '*') counterStart += 3;
            if (rpnstr[i] == -'-') {
                ConstAddress("0");
                counterStart += 3;
            }
        }
        counterStart++;
        return 2;
    } else if (regex_search(buf, sm, regex("([0-9]+)\\s+IN-
PUT\\s+([A-Z])\\s*\n"))) {
        if (sm[2].str().size() != 1) {
```

```
            cout << "ERROR WRONG INPUT_COMMAND in line " <<
sm[1].str();
            exit(1);
        }
        VarAddress(sm[2].str()[0]);
        SetLabel(sm[1].str());
        return 3;
    } else if (regex_search(buf, sm, regex("([0-9]+)\\s+OUT-
PUT\\s+([A-Z]\\s*)\n"))) {
        if (sm[2].str().size() != 1) {
            cout << "ERROR WRONG OUTPUT_COMMAND  in line " <<
sm[1].str();
            exit(1);
        }
        VarAddress(sm[2].str()[0]);
        SetLabel(sm[1].str());
        return 4;
    } else if (regex_search(buf, sm, regex("^(([0-
9]+)\\s+IF\\s+([-A-Z0-9]+)\\s*([<>=]{1,2})\\s*([-A-Z0-
9]+)\\s+GOTO\\s+([0-9]+)\\s*\n)$"))) {
        SetLabel(sm[2].str());
        string s1, s2;
        s1 = sm[3].str();
        s2 = sm[5].str();
        if (s1[0] >= 'A' && s1[0] <= 'Z') {
            if (s1.size() != 1) {
                cout << "ERROR WRONG VALUE IN IF in line " <<
sm[2].str();
                exit(1);
            }
            VarAddress(s1[0]);
        } else {
            ConstAddress(s1);
        }
        if (s2[0] >= 'A' && s2[0] <= 'Z') {
            if (s2.size() != 1) {
                cout << "ERROR WRONG VALUE IN IF in line " <<
sm[2].str();
                exit(1);
            }
            VarAddress(s2[0]);
        } else {
            ConstAddress(s2);
        }
        if (sm[4].str().size() == 2) counterStart++;
        counterStart += 2;
        return 5;
```

```
    } else if (regex_search(buf, sm, regex("^(([0-
9]+)\\s+GOTO\\s+([0-9]+)\n)$"))) {
        SetLabel(sm[2].str());
        return 6;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+END\\s*\n"))) {
        SetLabel(sm[1].str());
        return 7;
    } else if (regex_search(buf, sm, regex("([0-
9]+)\\s+PRINT\\s+([A-Z]\\s*)\n"))) {
        if (sm[2].str().size() != 1) {
            cout << "ERROR WRONG PRINT_COMMAND";
            exit(1);
        }
        VarAddress(sm[2].str()[0]);
        SetLabel(sm[1].str());
        return 4;
    }
    if (regex_search(buf, sm, regex("\0"))) return 0;
    else return -1;
}

int main(int argc, char * argv[]) {
    bool endflag = 0;
    if (argc != 2) {
        cout << "Start command should looks like ./bin/main
<filename>\n";
        return -1;
    }
    string filename = argv[1];
    ifstream in (filename + ".sb");
    ofstream out(filename + ".sa");
    if (! in ) {
        cout << "CANNOT OPEN";
        exit(1);
    } else {
        for (int i = 0; i < 26; i++) {
            Name_Address[i] = -1;
        }
        while (! in .eof()) {
            string buf;
            getline( in , buf);
            buf += "\n";
            int tmp1 = Check_SB(buf);
            if (tmp1 == 7 && endflag) {
                cout << "ERROR MORE ONE END";
                exit(1);
```

```
            }
            if (tmp1 == 7) endflag = 1;
            if (tmp1 == 0) break;
            if (tmp1 == -1) {
                cout << "WRONG COMMAND IN LINE" << La-
bels[tagCounter].lb;
                exit(1);
            }
            if (counterStart > counterEnd + 1) {
                cout << "ERROR OVERFLOW MEMORY";
                exit(1);
            }
        }
        if (!endflag) {
            cout << "ERROR NO END";
            exit(1);
        }
        last = counterStart;
        counterStart = 0; in .clear(); in .seekg(0);
        while (! in .eof()) {
            string buf;
            getline( in , buf);
            buf += "\n";
            SB_to_SA(buf);
        }
    }
    for (int i = 0; i < 100; i++) {
        if (Memory[i].command == "") continue;
        if (Memory[i].command == "=")
            out << setfill('0') << setw(2) << i << " " <<
Memory[i].command << " " << (((Memory[i].operand & (1 << 13))
> 0) ? '-' : '+') << hex << uppercase << setfill('0') <<
setw(4) << (Memory[i].operand & 8191) << dec << '\n';
        else
            out << setfill('0') << setw(2) << i << " " <<
Memory[i].command << " " << setfill('0') << setw(2) <<
(Memory[i].operand & 8191) << '\n';
    }
    out << '\0';
    out.close(); in .close();
    return 0;
}
```

Транслятор Simple Assembler

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

```
#include <ctype.h>
#define countOper 39
#define size_array 100

int memory[100];
int listOper[countOper]
= {0x10, 0x11, 0x20, 0x21, 0x30, 0x31, 0x32, 0x33, 0x40, 0x41,
0x42, 0x43, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
0x59, 0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
0x69, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x00};


int getCommand(char* cmd){
    if(!strcmp(cmd,"READ"))return 0x10;
    if(!strcmp(cmd,"WRITE"))return 0x11;
    if(!strcmp(cmd,"LOAD"))return 0x20;
    if(!strcmp(cmd,"STORE"))return 0x21;
    if(!strcmp(cmd,"ADD"))return 0x30;
    if(!strcmp(cmd,"SUB"))return 0x31;
    if(!strcmp(cmd,"DIVIDE"))return 0x32;
    if(!strcmp(cmd,"MUL"))return 0x33;
    if(!strcmp(cmd,"JUMP"))return 0x40;
    if(!strcmp(cmd,"JNEG"))return 0x41;
    if(!strcmp(cmd,"JZ"))return 0x42;
    if(!strcmp(cmd,"HALT"))return 0x43;
    if(!strcmp(cmd,"LOGRC"))return 0x68;
    return -1;

}

int sc_commandEncode(int command, int operand, int* value)
{
    if(command>127||command<0) return -1;
    if(operand>127||operand<0) return -2;
    *value=0;
    *value=*value | (command<<(8-1));
    *value=*value | operand;
    return 0;
}


int main(int argc, char* argv[])
{
    if(argc!=2){
        printf("Command should look like ./bin/main <file-
name>\n");
        return -1;
```

```
    }
    char temp[]="";
    strcpy(temp,argv[1]);
    FILE *input=fopen(strcat(argv[1],".sa"),"r");
    if(input==NULL)
    {
        printf("File doesn't exist\n");
        return -1;
    }
    FILE *output=fopen(strcat(temp,".o"),"wb+");;
    int last = -1;
    int a,b;
    char str[10],in[3],out[10],s[1024];
    int data;
    while(!feof(input)){
        data=0;
        fgets(s,1024,input);
        int check = sscanf(s,"%s %s %s ",in,str,out);
        if(check == -1)break;

    if(check!=3||strlen(in)!=2||!(strlen(out)==2||strlen(out)=
=5)){printf("ERROR WRONG INPUT %d %s %s %s
\n",check,in,str,out);exit(1);}

        for(int i=0;i<strlen(in);i++)
if(!(in[i]>='0'&&in[i]<='9')){printf("ERROR FIRST ARGUMENT IN-
PUT\n");exit(1);}
        a=atoi(in);
        if(a<=last){printf("ERROR WRONG NUMERATION %s %d
\n",in,last);exit(1);}
        last=a;
        if(strlen(out)==2){
            if(getCommand(str)==-1){printf("ERROR COMMAND
%s\n",str);exit(1);}
            for(int i=0;i<strlen(out);i++)
if(!(out[i]>='0'&&out[i]<='9')){printf("ERROR WRONG
OUT1\n");exit(1);}
            b=atoi(out);
            sc_commandEncode(getCommand(str),b,&data);
            memory[a]=data;
        }else{
            if(strcmp(str,"=")){printf("ERROR DONT FIND
EQUEL\n");exit(1);}
            if(out[0]!='+'&&out[0]!='-'){printf("ERROR NOT
FIND +\n");exit(1);}
            for(int i=1;i<strlen(out);i++)
if(!isxdigit(out[i])){printf("ERROR WRON OUT2\n");exit(1);}
```

```
            b = (int)strtol(out,NULL,16);
            if(b<-0x1fff||b>0x1fff){printf("ERROR WRONG NUMBER
%s %d\n",out , b);exit(1);}
            if(b<0) b=abs(b)+(1<<13);
            b+=1<<14;
            memory[a]=b;
        }
    }

    fwrite(memory,sizeof(int),size_array,output);
    fclose(input);
    fclose(output);


    return 0;
}
```

Модель Simple Computer

main.c:

```
#include<stdio.h>
#include<stdlib.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>
#include "libproject/mySimpleComputer.h"
#include "libproject/myTerm.h"
#include "libproject/myUI.h"
#include "libproject/myReadkey.h"

void signalHandler(int signal) {
    switch (signal) {
    case SIGALRM:
        CU( & Memory_Position);
        break;
    case SIGUSR1:
        ui_restart();
        //ui_initial() ;
        break;
    case SIGUSR2:
        CU( & Memory_Position);
        break;
    default:
        break;
    }

}
```

```
int main() {
    ui_terminal_init();
    signal(SIGALRM, signalHandler);
    signal(SIGUSR1, signalHandler);
    signal(SIGUSR2, signalHandler);
    Memory_Position = 0;
    enum keys key;
    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;
    KeyFlag = 1;
    do {
        ui_terminal_update();
        rk_readkey( & key, KeyFlag);
        switch (key) {
        case key_up:
            ui_Change_Memory_Position(key_up, KeyFlag);
            break;
        case key_right:
            ui_Change_Memory_Position(key_right, KeyFlag);
            break;
        case key_down:
            ui_Change_Memory_Position(key_down, KeyFlag);
            break;
        case key_left:
            ui_Change_Memory_Position(key_left, KeyFlag);
            break;

        case key_load:
            ui_loadMemory();
            break;
        case key_save:
            ui_saveMemory();
            break;

        case key_run:
            while (CU( & Memory_Position) == 0) usleep(25000);
            break;
        case key_step:
            raise(SIGUSR2);
            break;
        case key_reset:
            raise(SIGUSR1);
            break;
```

```
        case key_F5:
            ui_set_accumulator();
            break;
        case key_F6:
            ui_set_instructionCounter( & Memory_Position);
            break;

        case key_enter:
            ui_set_memory_value();
            break;

        default:
            break;

        }
    } while (key != key_exit);

    return 0;
}
```

myBigChars.c:

```
#include "myBigChars.h"

int bc_printA(char * str) {
    if (str == NULL) return -1;
    printf("\E(0");
    for (int i = 0; i < strlen(str); i++) printf("%c",
str[i]);
    printf("\E(B");
    return 0;
}

int bc_box(int x1, int y1, int x2, int y2) {
    if (x2 <= 0 || y2 <= 0) return -1;
    int div_x = x2 - x1;
    int div_y = y2 - y1;
    if (div_x <= 0 || div_y <= 0) return -1;
    char * str1 = (char * ) malloc(sizeof(char) * (div_x +
1));
    char * str2 = (char * ) malloc(sizeof(char) * (div_x +
1));
    str1[0] = 'l';
    str2[0] = 'x';
    str1[div_x] = 'k';
    str2[div_x] = 'x';
    for (int i = 1; i < div_x; i++) {
```

```
        str1[i] = 'q';
        str2[i] = ' ';
    }
    mt_gotoXY(x1, y1);
    bc_printA(str1);
    for (int y = y1 + 1; y < y2; y++) {
        mt_gotoXY(x1, y);
        bc_printA(str2);
    }
    str1[0] = 'm';
    str1[div_x] = 'j';
    mt_gotoXY(x1, y2);
    bc_printA(str1);
    return 0;
}

int bc_printbigchar(long int * nums, int x, int y, enum colors
col_bg, enum colors col_fg) {
    int rows, cols;
    mt_getscreensize( & rows, & cols);
    if (x < 0 || x > cols - 7 || y < 0 || y > rows - 7) return
-1;
    long int num_0 = nums[0], num_1 = nums[1];
    printf("\E(0");
    mt_setfgcolor(col_fg);
    mt_setbgcolor(col_bg);
    for (int i = 0; i < 8; i++) {
        mt_gotoXY(x, y + i);
        for (int j = 0; j < 8; j++) {
            if (i < 4) {
                if (num_0 & 0x1) printf("a");
                else printf(" ");
                num_0 = num_0 >> 1;
            } else {
                if (num_1 & 0x1) printf("a");
                else printf(" ");
                num_1 = num_1 >> 1;
            }
        }
    }
    printf("\E(B");
    mt_setfgcolor(DEFAULT);
    mt_setbgcolor(DEFAULT);
    return 0;
}
```

```
int bc_setbigcharpos(long int * big, int x, int y, int value)
{
    if (x < 0 || x > 7 || y < 0 || y > 7) return -1;
    int ind = 1;
    if (y < 4) ind = 0;
    else y -= 4;
    if (value > 2 || value < 0) return -1;
    if (value) big[ind] = big[ind] | (1 << (x + y * 8));
    else big[ind] = big[ind] & (~(1 << (x + y * 8)));
    return 0;
}

int bc_getbigcharpos(long int * big, int x, int y, int *
value) {
    if (x < 0 || x > 7 || y < 0 || y > 7) return -1;
    int ind = 1;
    if (y < 4) ind = 0;
    else y -= 4;
    * value = (big[ind] >> (x + y * 8)) & 0x1;
    return 0;
}

int bc_bigcharwrite(char * filename, long int * big, int N,
int count) {
    FILE * fp = fopen(filename, "w+");
    int i = 0;
    if (count <= 0) return -1;
    if (N / 2 < count) return -1;
    while (count) {
        fwrite((char * ) & big[i], sizeof(big[i]), 1, fp);
        i++;
        fwrite((char * ) & big[i], sizeof(big[i]), 1, fp);
        i++;
        count--;
    }
    fclose(fp);
    return 0;
}

int bc_bigcharread(char * filename, long int * big, int N, int
need_count, int * count) {
    FILE * fp = fopen(filename, "r");
    if (fp == NULL) return -1;
    * count = 0;
    int i = 0;
    while (need_count > 0 && ( * count) + 1 <= N / 2) {
        fread((char * ) & big[i], sizeof(big[i]), 1, fp);
```

```
        i++;
        fread((char * ) & big[i], sizeof(big[i]), 1, fp);
        i++;
        need_count--;
        ( * count) ++;
    }
    fclose(fp);
    return 0;
}
```

myBigChars.c:

```
#ifndef MYBIGCHARS_H
#define MYBIGCHARS_H

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <termios.h>
#include <sys/ioctl.h>
#include "myTerm.h"


static const long int
zero[2]={0b10000001100000011000000111111111,
0b11111111100000011000000110000001 };
static const long int
one[2]={0b10011000101100001110000011000000,
0b10000000100000001000000010000000 };
static const long int
two[2]={0b11111111100000001000000011111111,
0b11111111000000010000000100000001 };
static const long int
three[2]={0b11111111100000001000000011111111,
0b11111111100000001000000010000000 };
static const long int
four[2]={0b11111111100000011000000110000001,
0b10000000100000001000000010000000 };
static const long int
five[2]={0b11111111100000001000000011111111,
0b11111111100000001000000010000000 };
static const long int
six[2]={0b11111111000000010000000111111111,
0b11111111100000011000000110000001 };
static const long int
seven[2]={0b00110000011000001100000011111111,
0b00000011000001100000110000011000 };
```

```
static const long int
eight[2]={0b11111111100000011000000111111111,
0b11111111100000011000000110000001 };
static const long int
nine[2]={0b11111111100000011000000111111111,
0b11111111100000001000000010000000 };
static const long int
ten[2]={0b11111111100000011000000111111111,
0b10000001100000011000000110000001 };
static const long int
eleven[2]={0b01111111100000011000000101111111,
0b01111111100000011000000110000001 };
static const long int
twelve[2]={0b00000001000000010000000111111111,
0b11111111000000010000000100000001 };
static const long int thir-
teen[2]={0b10000001100000011000000101111111,
0b01111111100000011000000110000001 };
static const long int four-
teen[2]={0b11111111000000010000000111111111,
0b11111111000000010000000100000001 };
static const long int fif-
teen[2]={0b11111111000000010000000111111111,
0b00000001000000010000000100000001 };
static const long int
SymbPlus[2]={0b11111111000110000001100000011000,
0b00011000000110000001100011111111 };
static const long int SymbMi-
nus[2]={0b11111111000000000000000000000000,
0b00000000000000000000000011111111};

static const long int
BigChars[18][2]={{zero[0],zero[1]},{one[0],one[1]},{two[0],two
[1]},{three[0],three[1]},{four[0],four[1]},{five[0],five[1]},{
six[0],six[1]},{seven[0],seven[1]},{eight[0],eight[1]},{nine[0
],nine[1]},{ten[0],ten[1]},{eleven[0],eleven[1]},{twelve[0],tw
elve[1]},{thirteen[0],thirteen[1]},{fourteen[0],four-
teen[1]},{fifteen[0],fif-
teen[1]},{SymbPlus[0],SymbPlus[1]},{SymbMinus[0],SymbMi-
nus[1]}};

int bc_printA(char * str);
int bc_box(int x1, int y1, int x2, int y2);
int bc_printbigchar (long int* nums, int x, int y, enum colors
col_bg, enum colors col_fg);
int bc_setbigcharpos(long int* big, int x, int y, int value);
int bc_getbigcharpos(long int* big, int x, int y, int* value);
```

```
int bc_bigcharwrite(char* fd, long int* big, int N, int
count);
int bc_bigcharread(char* fd, long int* big, int N, int
need_count, int* count);


#endif
```

myReadKey.c:

```
#include "myReadkey.h"

int rk_mytermsave() {
    return tcgetattr(fileno(stdin), & mysettings);
}

int rk_mytermrestore() {
    return tcsetattr(fileno(stdin), TCSAFLUSH, & mysettings);
}

int rk_mytermregime(int regime, int vtime, int vmin, int echo,
int sigint) {
    struct termios new_settings;
    tcgetattr(fileno(stdin), & new_settings);
    if (regime == 0) {
        new_settings.c_lflag &= (~ICANON);
        if (sigint == 0) new_settings.c_lflag &= (~ISIG);
        else new_settings.c_lflag |= ISIG;
        if (echo == 0) new_settings.c_lflag &= (~ECHO);
        else new_settings.c_lflag |= ECHO;
        new_settings.c_cc[VMIN] = vmin;
        new_settings.c_cc[VTIME] = vtime;
    } else {
        new_settings.c_lflag |= ICANON;
    }
    tcsetattr(0, TCSAFLUSH, & new_settings);
    return 0;
}

int rk_readkey(enum keys * key, int flag) {
    fflush(stdout);
    char buf[5] = "\0";
    rk_mytermregime(0, 1, 0, 0, 0);
    read(fileno(stdin), buf, 5);
    rk_mytermrestore();
    if (flag == 1) {
        if (!strcmp(buf, "\E[A")) * key = key_up;
        else if (!strcmp(buf, "\E[B")) * key = key_down;
```

```
        else if (!strcmp(buf, "\E[C")) * key = key_right;
        else if (!strcmp(buf, "\E[D")) * key = key_left;
        else if (!strcmp(buf, "\E[15~")) * key = key_F5;
        else if (!strcmp(buf, "\E[17~")) * key = key_F6;
        else if (buf[0] == 'l') * key = key_load;
        else if (buf[0] == 's') * key = key_save;
        else if (buf[0] == 'r') * key = key_run;
        else if (buf[0] == 't') * key = key_step;
        else if (buf[0] == 'i') * key = key_reset;
        else if (buf[0] == '\n') * key = key_enter;
        else if (buf[0] == 27) * key = key_exit;
        else *key = key_other;
    } else {
        if (buf[0] == 'i') * key = key_reset;
        else *key = key_other;
    }
    fflush(stdin);
    return 0;

}
```

myReadKey.h:

```
#ifndef MYREADKEY_H
#define MYREADKEY_H

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

static int KeyFlag;

enum keys
{
    key_load,
    key_save,
    key_run,
    key_step,
    key_reset,
    key_F5,
    key_F6,
    key_up,
    key_down,
    key_left,
    key_right,
    key_other,
    key_enter,
```

```
    key_exit
};

static struct termios mysettings;

int rk_mytermsave();

int rk_mytermrestore();

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

int rk_readkey(enum keys *key, int flag);

#endif
```

mySimpleComputer.c:

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


int sc_regSet(int flag, int value) {
    if (flag > 5 || flag < 1) return -1;
    if (value > 2 || value < 0) return -2;
    if (value) Flags = Flags | (1 << (flag - 1));
    else Flags = Flags & (~(1 << (flag - 1)));
    return 0;
}

int sc_memoryInit() {
    for (int i = 0; i < 100; i++) Memory[i] = 0;
    return 0;
}

int sc_memorySet(int address, int value) {
    if (address < 100 && address >= 0) {
        Memory[address] = value;
        return 0;
    } else {
        sc_regSet(MemoryOut, 1);
        return -1;
    }
}

int sc_memoryGet(int address, int * value) {
    if (address < 100 && address >= 0) {
```

```
        * value = Memory[address];
        return 0;
    } else {
        return -1;
    }
}

int sc_memorySave(char * filename) {
    FILE * fp = fopen(filename, "w+");
    for (int i = 0; i < 100; i++) {
        fwrite((char * ) & Memory[i], sizeof(int), 1, fp);
    }
    return 0;
}

int sc_memoryLoad(char * filename) {
    FILE * fp = fopen(filename, "r");
    if (fp == NULL) return -1;
    for (int i = 0; i < 100; i++) {
        fread((char * ) & Memory[i], sizeof(int), 1, fp);
    }
    return 0;
}

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

int sc_regGet(int flag, int * value) {
    if (flag > 5 || flag < 1) return -1;
    * value = (Flags >> (flag - 1)) & 0x1;
    return 0;
}

int sc_commandEncode(int command, int operand, int * value) {
    if (command > 127 || command < 0) {
        sc_regSet(ErrorCommand, 1);
        return -1;
    }
    if (operand > 127 || operand < 0) return -2;
    * value = 0;
    * value = * value | (command << (8 - 1));
    * value = * value | operand;
    return 0;
}
```

```
int sc_commandDecode(int value, int * command, int * operand)
{
    if (value >> (15 - 1) > 1) return -1;
    if (value >> (15 - 1) & 0x1) return 1;
    * command = value >> (8 - 1);
    * operand = value & 127;
    return 0;
}
```

mySimpleComputer.h:

```
#ifndef MYSIMPLECOMPUTER_H
#define MYSIMPLECOMPUTER_H

#define OverFlow 1
#define ZeroDivision 2
#define MemoryOut 3
#define TactIgnore 4
#define ErrorCommand 5

#define READ 0x10
#define WRITE 0x11
#define LOAD 0x20
#define STORE 0x21
#define ADD 0x30
#define SUB 0x31
#define DIVIDE 0x32
#define MUL 0x33
#define JUMP 0x40
#define JNEG 0x41
#define JZ 0x42
#define HALT 0x43
#define NOT 0x51
#define AND 0x52
#define OR 0x53
#define XOR 0x54
#define JNS 0x55
#define JC 0x56
#define JNC 0x57
#define JP 0x58
#define JNP 0x59
#define CHL 0x60
#define SHR 0x61
#define RCL 0x62
#define RCR 0x63
#define NEG 0x64
#define ADDC 0x65
#define SUBC 0x66
```

```
#define LOGLC 0x67
#define LOGRC 0x68
#define RCCL 0x69
#define RCCR 0x70
#define MOVA 0x71
#define MOVR 0x72
#define MOVCA 0x73
#define MOVCR 0x74

static int Memory[100];
static int Flags;
static int sc_accumulator;
static int sc_instructionCounter;


int sc_regSet(int flag, int value);
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_regGet(int flag, int* value);
int sc_commandEncode(int command, int operand, int* value);
int sc_commandDecode(int value, int* command, int* operand);

#endif
```

myTerm.c:

```
#include<stdio.h>
#include<stdlib.h>
#include<termios.h>
#include<sys/ioctl.h>
#include"myTerm.h"

int mt_clrscr()
{
    printf("\E[H\E[J");
    mt_gotoXY(1,1);
    return 0;
}

int mt_gotoXY(int x, int y)
{
    int rows, cols;
    mt_getscreensize(&rows, &cols);
    if(x<0||x>cols||y<0||y>rows) return -1;
```

```
    printf("\E[%i;%iH",y,x);
    return 0;
}

int mt_getscreensize(int *rows, int *cols)
{
    struct winsize ws;
    if(!ioctl(1,TIOCGWINSZ,&ws))
    {
        *rows=ws.ws_row;
        *cols=ws.ws_col;
        return 0;
    }
    return -1;
}

int mt_setfgcolor(enum colors col)
{
    printf("\E[3%im",col);
    return 0;
}

int mt_setbgcolor(enum colors col)
{
    printf("\E[4%im",col);
    return 0;
}
```

myTerm.h:

```
#ifndef MYTERM_H
#define MYTERM_H

enum colors
{
    BLACK,
    RED,
    GREEN,
    YELLOW,
    BLUE,
    MAGENTA,
    CYAN,
    WHITE,
    DEFAULT=9,
};

int mt_clrscr();
int mt_gotoXY(int x, int y);
```

```
int mt_getscreensize(int *rows, int *cols);
int mt_setfgcolor(enum colors);
int mt_setbgcolor(enum colors);

#endif
```

myUI.c:

```
#include "myUI.h"

char input[100][1024];
char output[100][1024];
int num_in, num_out;

char * ui_s10_To_S16(int x) {
    int mod;
    char * str = (char * ) malloc(sizeof(char) * 5);
    if (((x >> 14) & 0x1) == 0) str[0] = '+';
    else if (((x >> 13) & 0x1) == 0) {
        str[0] = ' ';
        x &= ~(1 << 14);
    } else {
        str[0] = '-';
        x &= ~(1 << 14);
        x &= ~(1 << 13);
    }
    for (int i = 4; i > 0; i--) {
        mod = x % 16;
        if (mod < 10) str[i] = '0' + mod;
        else str[i] = 'A' + mod - 10;
        x /= 16;
    }
    return str;
}

int ui_CreateBigChar(long int Big[], int num) {
    Big[0] = BigChars[num][0];
    Big[1] = BigChars[num][1];
    return 0;
}

int ui_input(int operand) {
    mt_gotoXY(1, 26);
    printf("> ");
    long int value;
    scanf("%li", & value);
    if (value > 0x1FFF || value < -0x1FFF) return -1;
    char * str;
```

```
    sprintf(str, "%li", value);
    strcpy(input[num_in], str);
    num_in++;
    if (value < 0) {
        value *= -1;
        value += 0x2000;
    }
    value += 0x4000;
    sc_memorySet(operand, (int) value);
    return 0;
}

int ui_output(int operand) {
    int value;
    sc_memoryGet(operand, & value);
    mt_gotoXY(1, 26);
    printf("< ");
    value -= 0x4000;
    if (value >= 0x2000) {
        value -= 0x2000;
        value *= -1;
    }
    char * str;
    sprintf(str, "%li", value);
    strcpy(output[num_out], str);
    printf("< ");
    ui_messageOut(output[num_out]);
    num_out++;
    return 0;
}

int ui_s16_To_S10(char * str, int * value) {
    * value = 0;
    if (!(str[0] == '+' || str[0] == '-')) {
        if (strlen(str) != 5) {
            ui_messageOut((char * )(str + 5));
            return -1;
        }
        for (int i = 5; i > 0; i--) {
            str[i] = str[i - 1];
        }
        str[0] = ' ';
    }
    if (strlen(str) != 6) {
        return -1;
    }
    for (int i = 4; i > 0; i--) {
```

```
        if (str[5 - i] >= 'A' && str[5 - i] <= 'F') * value +=
(str[5 - i] - 'A' + 10) << ((i - 1) * 4);
        else if (str[5 - i] >= '0' && str[5 - i] <= '9') *
value += (str[5 - i] - '0') << ((i - 1) * 4);
        else return -1;
    }
    if (str[0] != '+') * value |= 1 << 14;
    if (str[0] == '-') * value |= 1 << 13;
    return 0;
}

int ui_print_number(int address) {
    int value;
    char * str;
    mt_gotoXY(2 + (address % 10) * 6, 2 + (address / 10));
    sc_memoryGet(address, & value);
    str = ui_s10_To_S16(value);
    for (int j = 0; j < 5; j++) {
        printf("%c", str[j]);
    }
    return 0;
}

int ui_print_memory() {
    int value;
    char * str;
    for (int i = 0; i < 100; i++) ui_print_number(i);
    mt_gotoXY(2 + (Memory_Position % 10) * 6, 2 + (Memory_Po-
sition / 10));
    sc_memoryGet(Memory_Position, & value);
    str = ui_s10_To_S16(value);
    mt_setbgcolor(YELLOW);
    for (int j = 0; j < 5; j++) {
        printf("%c", str[j]);
    }
    mt_setbgcolor(DEFAULT);
    return 0;
}

int ui_print_accumulator() {
    mt_gotoXY(71, 2);
    char * str;
    str = ui_s10_To_S16(sc_accumulator);
    for (int j = 0; j < 5; j++) {
        printf("%c", str[j]);
    }
    return 0;
```

```
}

int ui_print_instructionCounter() {
    mt_gotoXY(71, 5);
    printf("%04X", sc_instructionCounter);
}

int ui_print_operation() {
    int value;
    mt_gotoXY(68, 8);
    int command = -1, operand;
    sc_memoryGet(Memory_Position, & value);
    sc_commandDecode(value, & command, & operand);
    if (command != -1) printf("%04X:%04X", command, operand);
    return 0;
}

int ui_print_flags() {
    int value;
    for (int i = 1; i <= 5; i++) {
        if (sc_regGet(i, & value)) return -1;
        mt_gotoXY(68 + (i * 2), 11);
        if (value) printf("%c", fl[i - 1]);
        else printf(" ");
    }
}

int ui_set_accumulator() {
    int value;
    sc_memoryGet(Memory_Position, & value);
    sc_accumulator = value & (~(1 << 14));
    return 0;
}

int ui_terminal_init() {
    Flags = 0;
    Memory_Position = 0;
    sc_instructionCounter = 0;
    if (rk_mytermsave()) return -1;
    sc_memoryInit();
    sc_regInit();
    return 0;
}

int ui_print_bigNumber() {
    long int Big[10];
    int temp, mod;
```

```
    sc_memoryGet(Memory_Position, & temp);
    if (temp >= 0) ui_CreateBigChar(Big, 16);
    else ui_CreateBigChar(Big, 17);
    for (int i = 4; i > 0; i--) {
        mod = temp % 16;
        ui_CreateBigChar((Big + i * 2), mod);
        temp /= 16;
    }
    for (int i = 0; i < 5; i++) bc_printbigchar((Big + 2 * i),
2 + i * 9, 14, DEFAULT, YELLOW);
    return 0;
}

void ui_SetMemoryPosition(int ind) {
    Memory_Position = ind;
    sc_instructionCounter = Memory_Position;
}

int ui_terminal_update() {
    mt_clrscr();
    bc_box(1, 1, 61, 12);
    ui_print_memory();
    mt_gotoXY(29, 1);
    printf(" Memory ");
    bc_box(62, 1, 83, 3);
    bc_box(62, 4, 83, 6);
    bc_box(62, 7, 83, 9);
    bc_box(62, 10, 83, 12);
    mt_gotoXY(66, 1);
    printf(" accumulator ");
    ui_print_accumulator();
    mt_gotoXY(63, 4);
    printf(" instructionCounter ");
    ui_print_instructionCounter();
    mt_gotoXY(68, 7);
    printf(" Operation ");
    ui_print_operation();
    mt_gotoXY(68, 10);
    printf(" Flags ");
    ui_print_flags();
    bc_box(1, 13, 46, 22);
    bc_box(47, 13, 83, 22);
    mt_gotoXY(48, 13);
    printf(" Keys: ");
    mt_gotoXY(48, 14);
    printf("l  - load");
    mt_gotoXY(48, 15);
```

```
    printf("s  - save");
    mt_gotoXY(48, 16);
    printf("r  - run");
    mt_gotoXY(48, 17);
    printf("t  - step");
    mt_gotoXY(48, 18);
    printf("i  - reset");
    mt_gotoXY(48, 19);
    printf("F5 - accumulator");
    mt_gotoXY(48, 20);
    printf("F6 - instructionCounter");
    ui_print_bigNumber();
    mt_gotoXY(1, 23);
    printf("Input/Output\n>");
    for (int i = 0; i < num_in; i++) printf("%s ", input[i]);
    printf("\n<");
    for (int i = 0; i < num_out; i++) printf("%s ", out-
put[i]);
    mt_gotoXY(1, 26);
    return 0;
}

int ui_Change_Memory_Position(enum keys key, int flag) {
    if (flag == 1) {
        if (key == key_up) {
            if (Memory_Position < 10) Memory_Position += 90;
            else Memory_Position -= 10;
            return 0;
        }
        if (key == key_down) {
            if (Memory_Position >= 90) Memory_Position -= 90;
            else Memory_Position += 10;
            return 0;
        }
        if (key == key_left) {
            if ((Memory_Position % 10) == 0) Memory_Position
+= 9;
            else Memory_Position--;
            return 0;
        }
        if (key == key_right) {
            if ((Memory_Position % 10) == 9) Memory_Position -
= 9;
            else Memory_Position++;
            return 0;
        }
    }
```

```
    return -1;
}

int ui_set_memory_value() {
    int value;
    char buffer[10];
    printf("Set the value of the cell under the number \n");
    printf("Enter value in HEX format > ");
    fgets(buffer, 10, stdin);
    printf("%s", buffer);
    if (buffer[strlen(buffer) - 1] != '\n') ui_clearBuffIn();
// Ð¾Ñ‡Ð¸Ñ‚Ð¸Ð¼ Ð¿Ð¾Ñ‚Ð¾Ðº Ð²Ð²Ð¾Ð´Ð°
    if (ui_s16_To_S10(buffer, & value) == -1) {
        ui_messageOut((char * )
            "Invalid input");
        return -1;
    }
    if (value > 0x7FFF) {
        ui_messageOut((char * )
            "Number should be lesser than 0x7FFF");
        return -1;
    }
    if (value < -0x3FFF) {
        ui_messageOut((char * )
            "Number should be bigger than 0x3FFF");
        return -1;
    }
    sc_memorySet(Memory_Position, value);
    return 0;
}

int ui_set_instructionCounter(int * value) {
    int value2;
    char buffer[10];
    printf("Set a value InstructionCounter between 0x0000 and
0x0063 inclusive\n");
    printf("Enter value in HEX format > ");
    fgets(buffer, 10, stdin);
    if (ui_s16_To_S10(buffer, & value2) == -1) {
        ui_messageOut((char * )
            "Invalid input");
        return -1;
    }
    if (value2 >= 100 || value2 < 0) {
        ui_messageOut((char * )
            "Number should be lesser than 0x0063 and more then
0x0000");
```

```
        return -1;
    }
    sc_instructionCounter = value2;
    Memory_Position = sc_instructionCounter;
    * value = sc_instructionCounter;
    return 0;
}

int ui_set_MemoryPosition() {
    int value;
    char buffer[10];
    printf("Set a value InstructionCounter between 0x0000 and
0x0063 inclusive\n");
    printf("Enter value in HEX format > ");
    fgets(buffer, 10, stdin);
    if (ui_s16_To_S10(buffer, & value) == -1) {
        ui_messageOut((char * )
            "Invalid input");
        return -1;
    }
    if (value >= 100 || value < 0) {
        ui_messageOut((char * )
            "Number should be lesser than 0x0063 and more then
0x0000");
        return -1;
    }
    Memory_Position = value;
    return 0;
}

/*int ui_run(int step)
{
    enum keys key_ex;
    struct itimerval nval, oval;
    nval.it_interval.tv_sec = 2;
    nval.it_interval.tv_usec = 0;
    nval.it_value.tv_sec = 0;
    nval.it_value.tv_usec = 0;

    if(!rk_readkey(&key_ex))
    {
        ui_terminal_update() ;
        Memory_Position += step ;
        if (Memory_Position >= 100)
        {
            Memory_Position = 0 ;
        }
```

```
        sc_instructionCounter=Memory_Position;
        //alarm(1);
        setitimer(ITIMER_REAL, &nval, &oval);
    }
    return 0;
}*/

int ui_pause(int time) {
    fflush(stdout); // Ð¾Ñ‡Ð¸Ñ‰Ð°ÐµÐ¼ Ð¿Ð¾Ñ‚Ð¾Ðº
Ð²Ñ‹Ð²Ð¾Ð´Ð°
    char buffer[5] = "\0";
    rk_mytermregime(0, time, 0, 0, 0);
    read(fileno(stdin), buffer, 5);
    rk_mytermrestore();
    return 0;
}

int ui_messageOut(char * str) {
    printf("%s", str);
    ui_pause(25);
    return 0;
}

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

int ui_saveMemory() {
    char filename[102];
    printf("Saving file...\n");
    printf("Enter the file name to save > ");
    mt_setfgcolor(GREEN);
    fgets(filename, 102, stdin);
    mt_setfgcolor(DEFAULT);
    if (filename[strlen(filename) - 1] != '\n') {
        printf("The file name is too long. The length is
trimmed to the first 100 characters.\n");
        ui_clearBuffIn();
    } else filename[strlen(filename) - 1] = '\0';

    if (sc_memorySave(filename)) {
        ui_messageOut((char * )
            "Failed to save memory");
```

```
        return -1;
    } else
        ui_messageOut((char * )
            "Successful save");
    return 0;
}

int ui_loadMemory() {
    char filename[102];
    printf("Loading file...\n");
    printf("Enter the file name to load > ");
    mt_setfgcolor(GREEN);
    fgets(filename, 102, stdin);
    mt_setfgcolor(DEFAULT);
    if (filename[strlen(filename) - 1] != '\n') {
        ui_messageOut((char * )
            "The name of the file to open is too long (up to
100 characters are allowed)");
        ui_clearBuffIn(); // Ð¾Ñ‡Ð¸Ñ‰Ð°ÐµÐ¼ Ð¿Ð¾Ñ‚Ð¾Ðº
Ð²Ð²Ð¾Ð´Ð°
        return -1;
    }
    filename[strlen(filename) - 1] = '\0';
    if (sc_memoryLoad(filename)) {
        ui_messageOut((char * )
            "Failed to load memory");
        return -1;
    } else {
        ui_messageOut((char * )
            "Successful load");
        ui_restart();
    }

    return 0;
}

int main_setICounter() {
    sc_regSet(TactIgnore, 1);
    sc_instructionCounter = 0;
    return 0;
}

int ui_restart() {
    sc_regInit();
    sc_instructionCounter = 0;
    Memory_Position = 0;
    for (int i = 0; i < 100; i++) {
```

```
        strcpy(input[i], "");
        strcpy(output[i], "");
    }
    num_in = 0;
    num_out = 0;
    return 0;
}

int CU(int * value) {
    Memory_Position = sc_instructionCounter;
    ui_terminal_update();
    int command = 0, operand = 0;
    int val;
    sc_memoryGet(sc_instructionCounter, & val);
    int res = sc_commandDecode(val, & command, & operand);
    if (res == -1) {
        sc_regSet(ErrorCommand, 1);
        sc_regSet(TactIgnore, 1);
        return -1;
    } else if (res == 0) {
        if (operand >= 100) {
            printf("Wrond address of memory");
            sc_regSet(MemoryOut, 1);
            return -1;
        }
        if (command == ADD || command == SUB || command == DI-
VIDE || command == MUL || command == LOGRC) {
            if (ALU(command, operand) == -1) {
                printf("ERROR\n");
                return -1;
            }
            sc_instructionCounter++;
            * value = sc_instructionCounter;
            return 0;
        }
        int value;
        switch (command) {
        case READ:
            if (ui_input(operand) != 0) {
                printf("Wrong value for memory\n");
                sc_regSet(OverFlow, 1);
                return -1;
            }
            sc_instructionCounter++;
            break;
        case WRITE:
            ui_output(operand);
```

```
            sc_instructionCounter++;
            break;
        case LOAD:
            sc_memoryGet(operand, & sc_accumulator);
            sc_instructionCounter++;
            break;
        case STORE:
            sc_memorySet(operand, sc_accumulator);
            sc_instructionCounter++;
            break;
        case JUMP:
            ui_SetMemoryPosition(operand);
            break;
        case JNEG:
            if (((sc_accumulator >> 13) & 0x1) == 1) ui_Set-
MemoryPosition(operand);
            else sc_instructionCounter++;
            break;
        case JZ:
            if (sc_accumulator == 0) ui_SetMemoryPosition(op-
erand);
            else sc_instructionCounter++;
            break;
        case HALT:
            sc_regSet(TactIgnore, 1);
            return -1;
        default:
            sc_instructionCounter++;
            break;
        }
    } else sc_instructionCounter++;
    * value = sc_instructionCounter;
    return 0;
}

int ALU(int command, int operand) {
    int value;
    sc_memoryGet(operand, & value);
    if (((value >> 14) & 0x1) && ((sc_accumulator >> 14) &
0x1)) {
        value &= ~(1 << 14);
        if (((value >> 13) & 0x1) == 1) {
            value &= ~(1 << 13);
            value *= -1;
        }
        sc_accumulator &= ~(1 << 14);
        if (((sc_accumulator >> 13) & 0x1) == 1) {
```

```
            sc_accumulator &= ~(1 << 13);
            sc_accumulator *= -1;
        }
        switch (command) {
        case ADD:
            sc_accumulator += value;
            if (sc_accumulator > 0x1FFF || sc_accumulator < -
0x1FFF) {
                printf("Modul of Sum of two numbers is too
big\n");
                sc_regSet(OverFlow, 1);
                return -1;
            }
            break;
        case SUB:
            sc_accumulator -= value;
            break;
        case DIVIDE:
            if (value == 0) {
                printf("Divide by zero!!!\n");
                sc_regSet(ZeroDivision, 1);
                return -1;
            }
            sc_accumulator /= value;
            break;
        case MUL:
            sc_accumulator *= value;
            if (sc_accumulator > 0x1FFF || sc_accumulator < -
0x1FFF) {
                printf("Modul of Mul of two numbers is too
big\n");
                sc_regSet(OverFlow, 1);
                return -1;
            }
            break;
        case LOGRC:
            if (value < 0) {
                value *= -1;
                sc_accumulator--;
            }
            for (int i = 0; i < sc_accumulator; i++) {
                value /= 2;
            }
            sc_accumulator = value;
            break;
        }
        if (sc_accumulator < 0) {
```

```
            sc_accumulator *= -1;
            sc_accumulator += 0x2000;
        }
        sc_accumulator += 0x4000;
    } else {
        ui_messageOut("Incorrect numbers\n");
        return -1;
    }
    return 0;
}
```

myUI.h:

```
#ifndef MYUI_H
#define MYUI_H
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>
#include<malloc.h>
#include "myBigChars.h"
#include "mySimpleComputer.h"
#include "myTerm.h"
#include "myReadkey.h"

static const char fl[5]={'O','Z','M','T','E'};
static int Memory_Position;

char* ui_s10_To_S16(int x);
int ui_CreateBigChar(long int Big[], int num);
int ui_input(int operand);
int ui_output(int operand);
int ui_s16_To_S10(char* str, int *value);
int ui_print_number(int address);
int ui_print_memory();
int ui_print_accumulator();
int ui_print_instructionCounter();
int ui_print_operation();
int ui_print_flags();
int ui_set_accumulator();
int ui_terminal_init();
int ui_print_bigNumber();
void ui_SetMemoryPosition(int ind);
int ui_terminal_update();
int ui_Change_Memory_Position(enum keys key, int flag);
```

```
int ui_set_memory_value();
int ui_set_instructionCounter(int *value);
int ui_run(int step);
int ui_pause(int time);
int ui_messageOut(char *str);
int ui_clearBuffIn();
int ui_saveMemory();
int ui_loadMemory();
int CU(int *value);
int main_setICounter();
int ALU();
int ui_restart();

#endif
```

Результаты работы программ

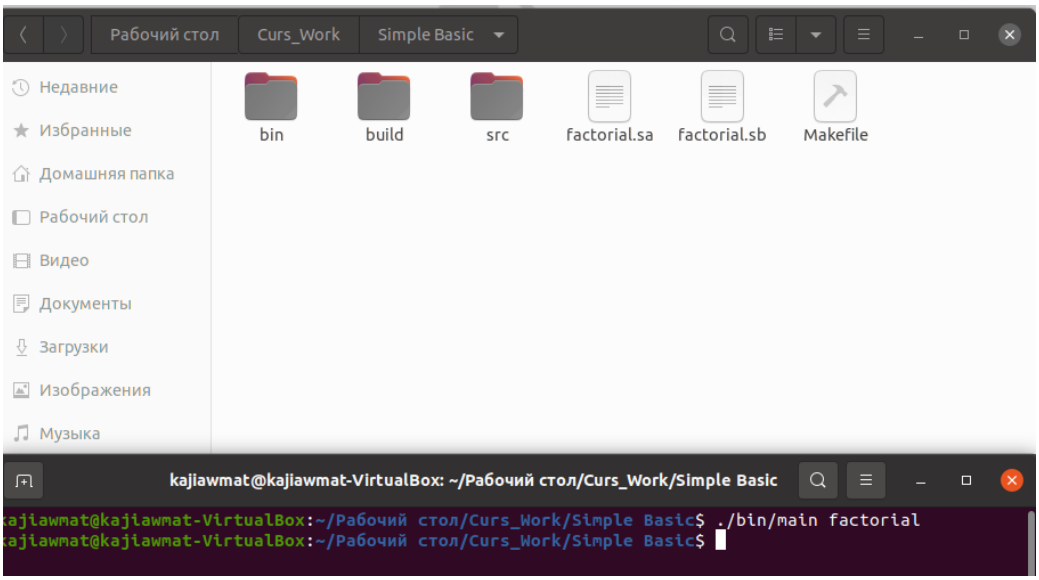

Рисунок 3. Перевод программы с языка Simple Basic на Simple Assembler

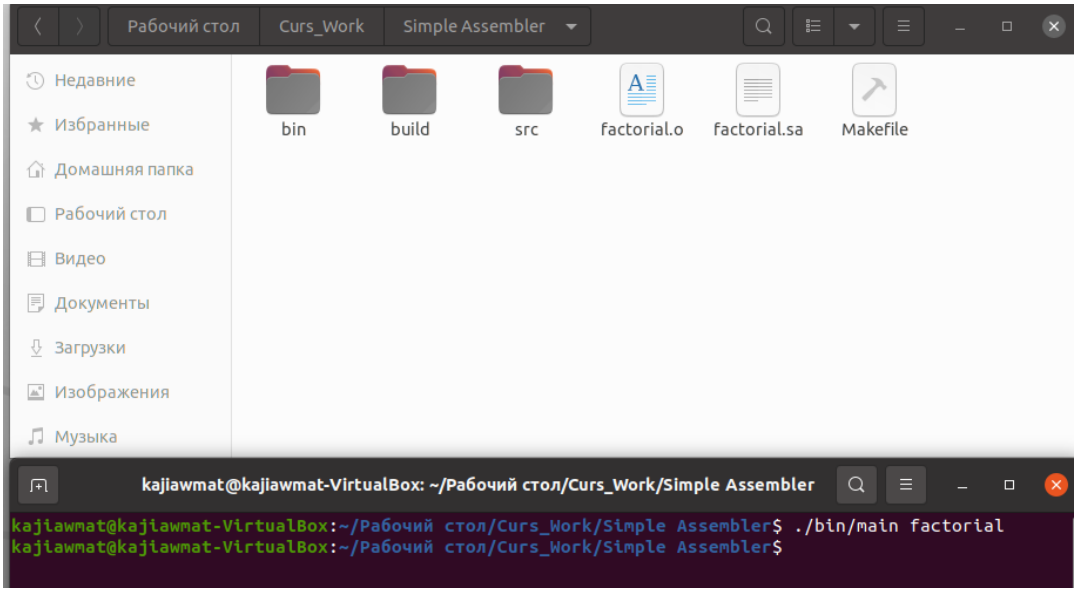

Рисунок 4. Перевод программы с языка Simple Assembler в бинарный формат

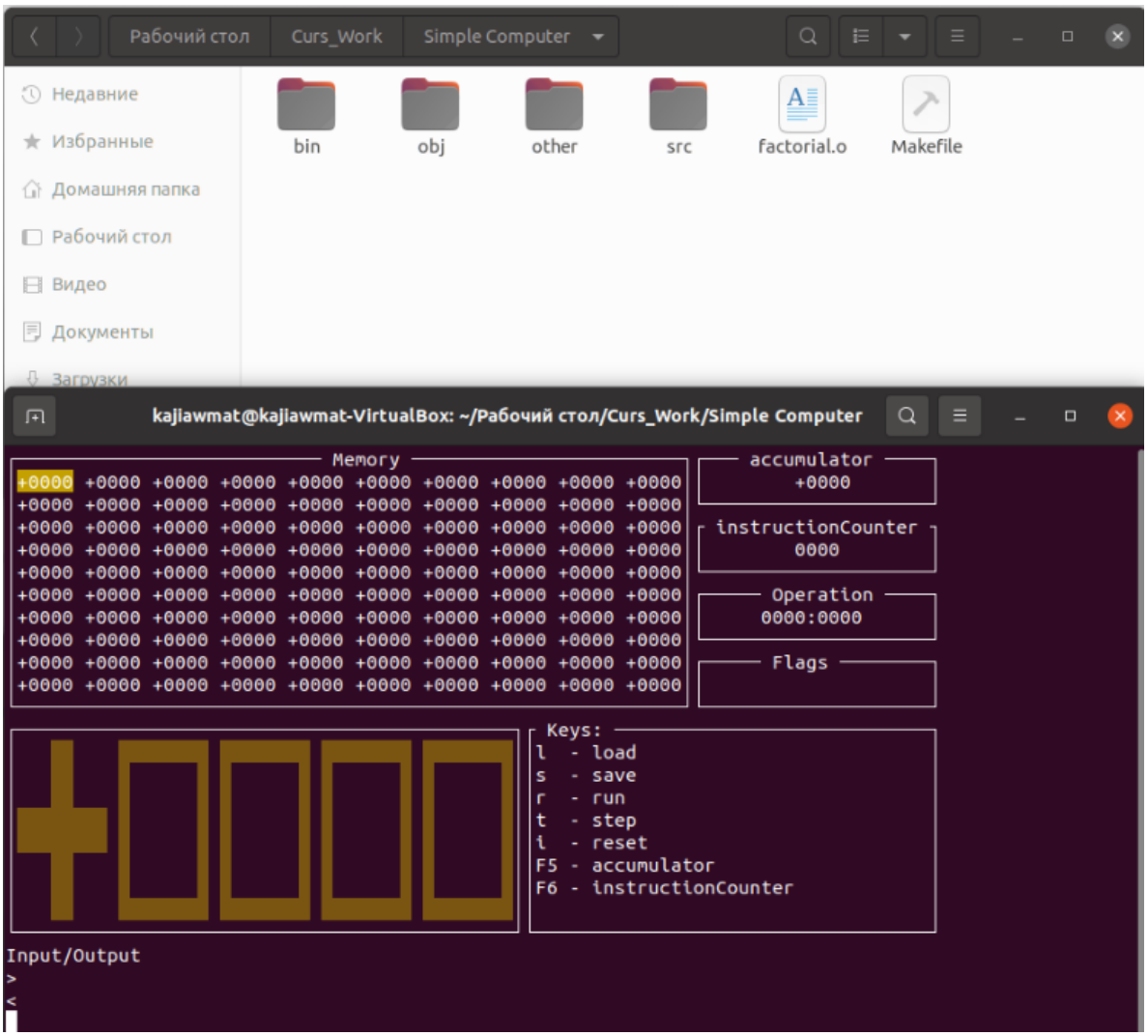

Рисунок 5. Инициализация работы Simple Computer

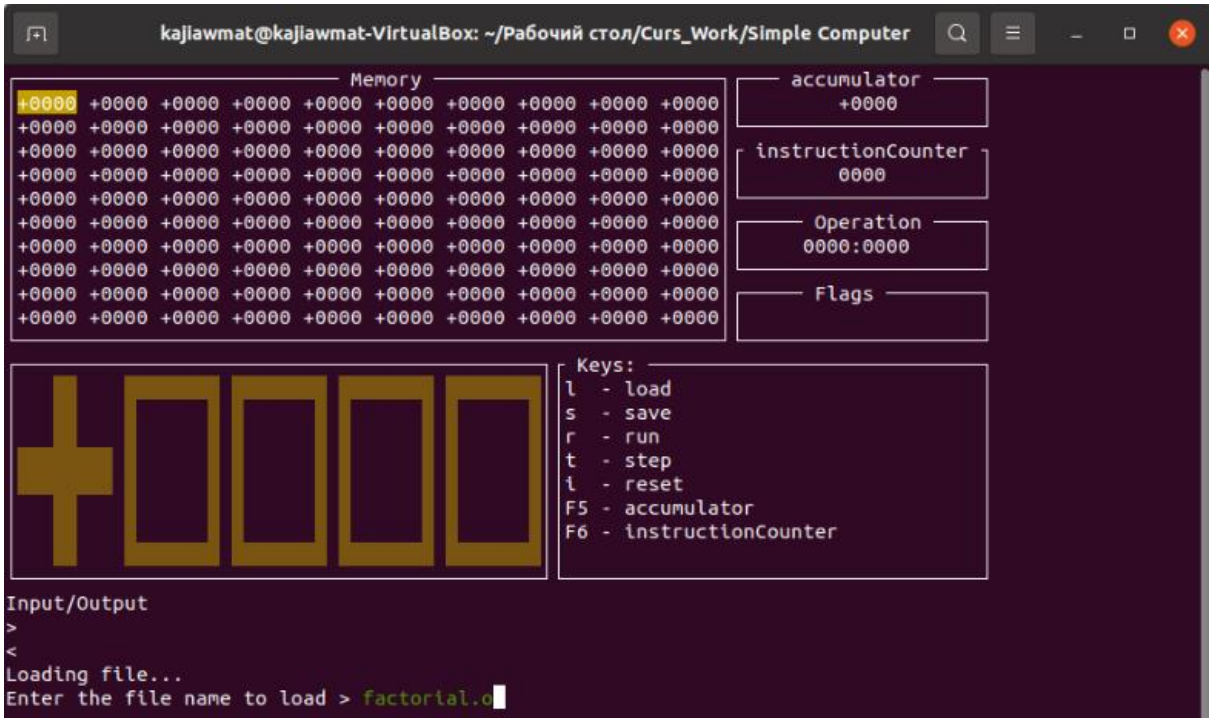

Рисунок 6. Процесс загрузки программы из бинарного файла

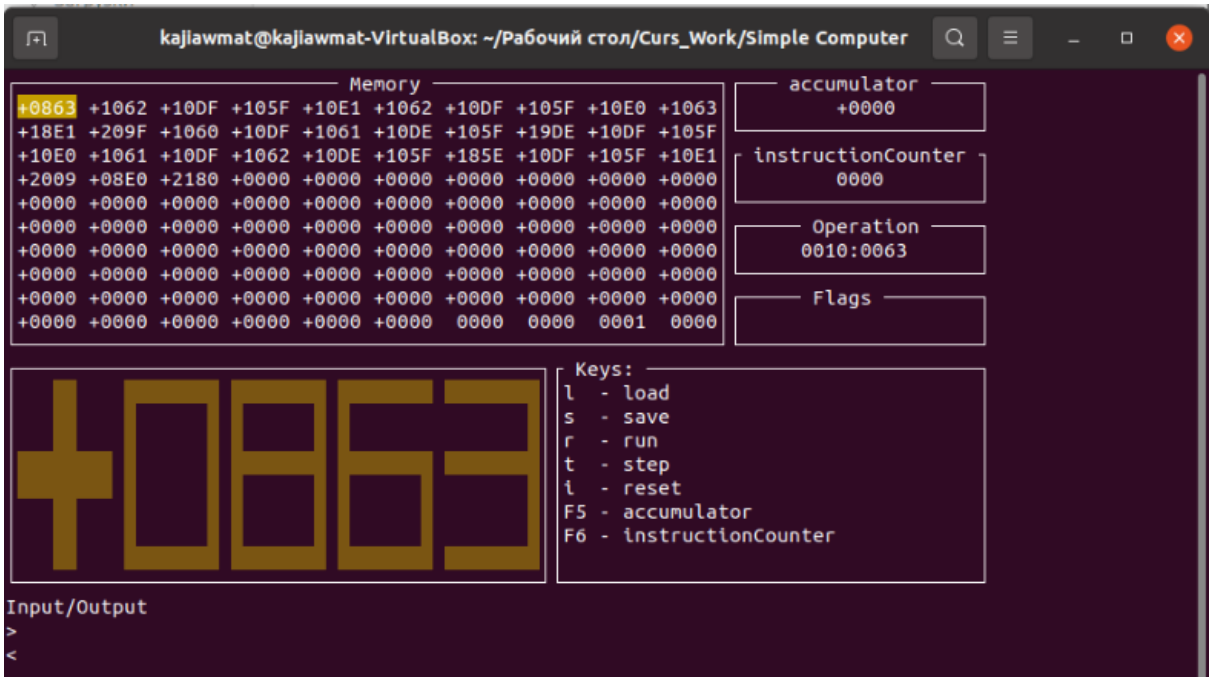

Рисунок 7. Simple Computer, загрузивший программу из бинарного файла

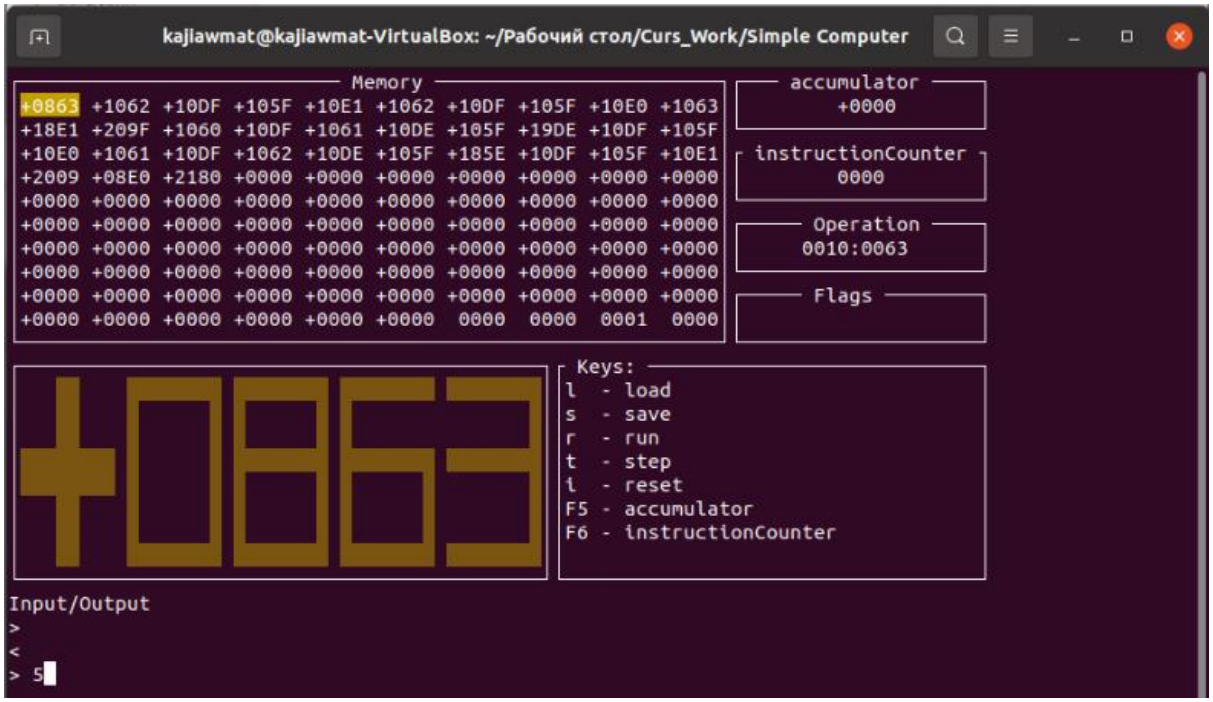

Рисунок 8. Начало работы программы по вычислению факториала и ввод данных

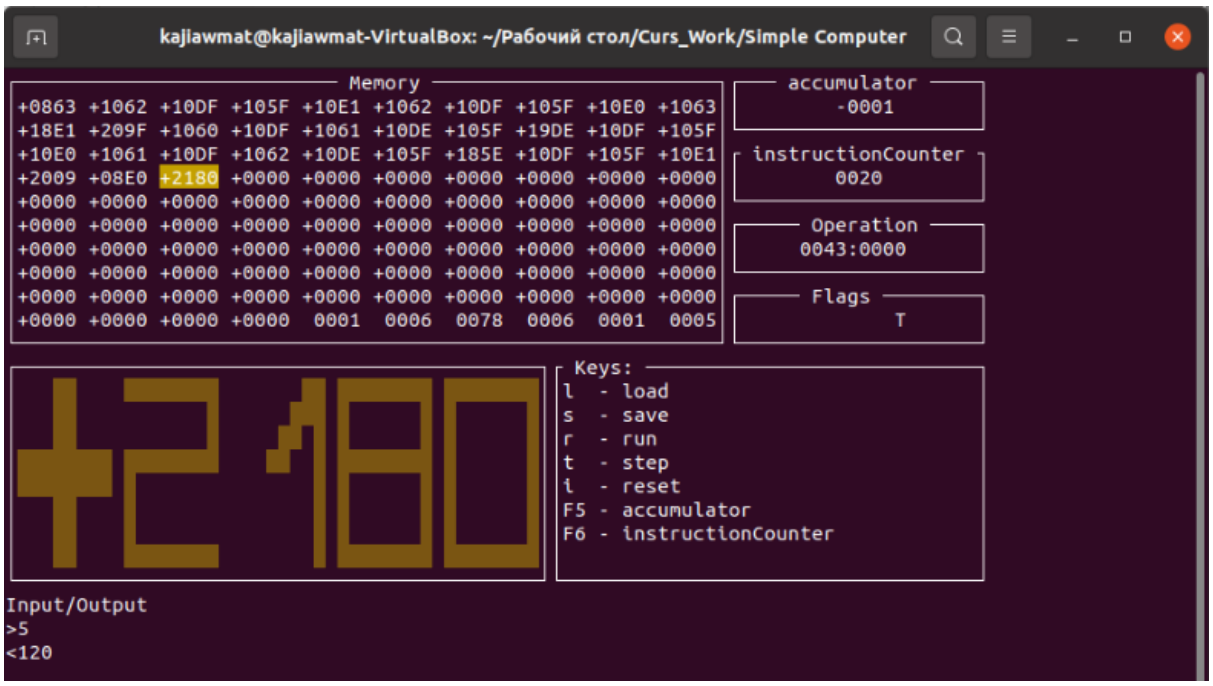

Рисунок 9. Завершение работы программы и вывод результата

Список использованной литературы

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

Дата _____ Подпись_____