ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Архитектура ЭВМ»

Выполнил: студент группы ИП-7

Проверил: доцент кафедры ВС

Майданов Ю. С.

Новосибирск, 2018

ОГЛАВЛЕНИЕ

[Задание на курсовую работу 3](#_Toc514502036)

[ПрограммнаямодельSimpleComputer 4](#_Toc514502037)

[Описание модели SimpleComputer 4](#_Toc514502038)

[Система команд SimpleComputer 5](#_Toc514502039)

[Консоль управления 6](#_Toc514502040)

[Описание работы SimpleComputer 7](#_Toc514502041)

[Блок-схемаработыSimpleComputer 9](#_Toc514502042)

[Транслятор SimpleAssembler 11](#_Toc514502043)

[Блок-схема работы транслятора 12](#_Toc514502044)

[Транслятор SimpleBasic 13](#_Toc514502045)

[Блок-схема работы транслятора 15](#_Toc514502046)

[Исходный код SimpleComputer 17](#_Toc514502047)

[Исходный код Translator 41](#_Toc514502048)

[Выводы 78](#_Toc514502049)

[Список использованной литературы 79](#_Toc514502050)

# Задание на курсовую работу

1. Реализовать программную модель вычислительной системы SimpleComputer;
2. Для управления программной моделью создать консоль управления, позволяющая определять начальные состояния узлов SimpleComputer, просматривать и изменять значение регистров и ячеек оперативной памяти, запускать программы на выполнение;
3. Реализовать транслятор, переводящий текст низкоуровневого языка программирования SimpleAssemblerв бинарный формат, которыйможет быть считан консолью Simple Computer;
4. Реализовать транслятор, переводящий текст высокоуровневого языка программирования SimpleBasicв текст низкоуровневого языка программирования SimpleAssembler;
5. Реализовать функции ALU(intcommand, intoperand) - арифметико-логическое устройство и CU() - устройство управления;
6. Реализовать пользовательскую функцию JP, которая осуществляет переход на указанную ячейку памяти, если результат последней операции четный.

# ПрограммнаямодельSimpleComputer

## Описание модели SimpleComputer

Архитектура SimpleComputerвключает следующие функциональные блоки:

* Оперативная память, состоящая из 100 ячеек, последовательно адресуемых целыми числами от 0 до 99, и имеющими разрядность 15 бит.
* Центральный процессор, выполняющий команды, хранящиеся в оперативной памяти, и осуществляющий функции взаимодействия с памятью и внешними устройствами.
* Внешние устройства, через которые осуществляется ввод-вывод информации.

Центральный процессор состоит из следующих функциональных блоков:

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

Аккумулятор имеет разрядность 15 бит, счетчик 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.

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»).

## Система команд SimpleComputer

Получив текущую команду из оперативной памяти, устройство управления декодирует ее с целью определить номер функции, которую надо выполнить и операнд. Формат команды следующий: старший разряд содержит признак команды (0 – команда), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд.



Рисунок 1. Формат команды центрального процессора 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

Для управления программной моделью SimpleComputer используется консоль управления. Интерфейс консоли управления представлен на рисунке 2. 

Рисунок 2. Консоль управления SimpleComputer

Она 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);
* Keys – подсказка по функциональным клавишам;
* Input/Output – область, используемая Simple Computer в процессе выполнения программы для ввода информации с клавиатуры и вывода ее на экран.

## Описание работы SimpleComputer

Сразу после запуска программной модели SimpleComputer производится инициализация оперативной памяти и регистров, запускается консоль управления. Так как во время инициализации флаг игнорирования тактовых импульсов устанавливается в 1, выполнение команд центрального процессора не производится и система ожидает нажатия функциональной клавиши.

Список функциональных клавиш и их назначение отображаются в области «Keys» консоли управления SimpleComputer. После нажатия функциональной клавиши производится выбор управляющей команды, соответствующей коду нажатой клавиши.

При нажатии функциональной клавиши «l» в поле Input/Outputпользователю предлагается ввести имя файла, содержащего код программы в бинарном формате. После ввода имени файла указанный файл загружается в оперативную память SimpleComputer, при этом оперативная память очищается. В случае если указан несуществующий файл или файл недоступен для чтения, в поле Input/Output выводится сообщение об ошибке.

При нажатии функциональной клавиши «s» в поле Input/Outputпользователю предлагается ввести имя файла для сохранения кода программы, хранящейся в оперативной памяти SimpleComputer. После ввода имени файла содержимое памяти выгружается в указанный файл. В случае если файл недоступен для записи, в поле Input/Output выводится сообщение об ошибке.

При нажатии функциональной клавиши «r» производится запуск программы, хранящейся в памяти SimpleComputer, на выполнение. При этом значения всех флагов и значение счетчика команд устанавливаются в 0. Чтение команд из оперативной памяти производится последовательно, начиная с ячейки, имеющей адрес 0. После чтения содержимого ячейки памяти, значение декодируется как команда. Если получена арифметическая или логическая операция, то вызывается функция ALU, иначе команда выполняется устройством управления (функция CU). После выполнения команды, устройство управления определяет, какая команда должна быть выполнена следующей и адрес ее ячейки памяти заносится в регистр счетчика команд. Затем цикл повторяется. В ходе выполнения программы производится контроль ошибок. В случае возникновения ошибки флаг игнорирования тактовых импульсов устанавливается в 1, и хотя бы еще один из флагов, и выполнение программы прекращается.

При нажатии на клавишу «t», производится выполнение шага программы, записанной в память SimpleComputer. Этот режим используется для отладки программ и позволяет отслеживать состояние ячеек памяти и регистров центрального процессора.

Сброс регистровв начальное состояние осуществляется при нажатии функциональной клавиши «i».

Консоль управления позволяет также задавать значение счетчика команд, аккумулятора и выделенной ячейки памяти с помощью клавиш F5, F6 и Enterсоответственно. Во всех этих случаях в поле Input/Output появляется приглашение ввода значения. При нажатии клавиши Enterпользователь предлагается ввести значение команды, затем значение операнда. Если же пользователь пропустил ввод значения команды, то тогда предлагается ввести произвольные данные (то есть ячейка не будет декодироваться как команда и операнд). Во время ввода значений производится контроль ошибок ввода.

## Блок-схемаработыSimpleComputer

Выполнение одного шага программы

Сброс регистров в начальное состояние

«l»

«s»

«r»

«t»

«i»

«F5»

«F6»

«Enter»

Загрузка программы из файла

Сохранение программы в файл

Начало

Конец

Инициализация памяти и регистров

Запуск консоли управления

1

Установка значения аккумулятора

Установка значения счетчика команд

Установка значения выделенной ячейки памяти

Отображение интерфейса консоли управления

Нажатие функциональной клавиши

2

1

Декодирование

Ошибка?

да

нет

да

Арифметико-логическая операция?

нет

Выполнение команды устройством управления

Передача команды в ALU

Ошибка?

Установка одного из флагов

E, O, М или P

да

нет

2

да

Установка флага T вединицу

нет

Определение следующей команды

Конец программы?

Считывание значения ячейки памяти

# Транслятор SimpleAssembler

SimpleAssembler – низкоуровневый язык программирования, предназначенный для разработки программ, выполняемых на программной модели SimpleComputer.

Для загрузки программ в SimpleComputer разработан транслятор, переводящий текст языка SimpleAssembler в бинарный формат. Команда запуска транслятора имеет вид: sat файл.sa файл.oo, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.oo – результат трансляции.

Транслятор работает следующим образом. При запуске транслятора предварительно проверяется наличие файла с расширением «sa». В случае обнаружения ошибки выводится сообщение об отсутствии файла, и транслятор прекращает свою работу. При отсутствии ошибок открывается файл с исходным кодом программы SimpleAssembler и в цикле производится построчное считывание строк исходного кода. Сначала считывается номер строки и команда в словесном виде. Если считали символ «=», то операнд – шестнадцатеричное число (значение ячейки в явном виде), иначе вызывается функция, преобразующая мнемонику команды в ее код согласно системе команд процессора SimpleComputer. Если преобразование невозможно, на экран выводится сообщение с указанием номера строки, содержащей ошибку, и работа транслятора завершается. При отсутствии ошибки код команды и значение операнда кодируются в бинарный формат и помещаются в выходной буфер.

При достижении конца файла трансляция прекращается, и выходной буфер записывается в файл с расширением «oo».

## Блок-схема работы транслятора

Конец

Формирование бинарного кода

да

нет

Проверка входных данных

Начало

Ошибка?

Отображение сообщения об ошибке

да

нет

Ошибка?

Выделение в строке адреса ячейки памяти, мнемоники команды и операнда

Ошибка?

да

Пока не достигнут конец файла

Запись выходного буфера в файл

Инициализация выходного буфера

Открытие файла с исходным кодом SA

Считывание

строки кода SimpleAssembler

Запись кода в выходной буфер

нет

Ввод параметров

# Транслятор SimpleBasic

SimpleBasic – высокоуровневый язык программирования, предназначенный для разработки программ, выполняемых на программной модели SimpleComputer.

Каждая строка программы состоит из номера строки, оператора SimpleBasic и параметров. Номера строк следуют в возрастающем порядке. Синтаксис языка SimpleBasic предусматривает написание программ только в верхнем регистре. Имена переменных могут состоять только из одной буквы. В языке используются следующие операции: «+», «-», «\*» и «/». Приоритет операций аналогичен языку программирования Си. SimpleBasic оперирует только с целыми числами, операции со строками не поддерживаются.

ДлязагрузкипрограммвSimpleComputerразработантранслятор, переводящийтекстязыкаSimpleBasicвисходныйкодSimpleAssembler. Команда запуска транслятора имеет вид: sbt файл.sb файл.sa, где файл.sb – имя файла, в котором содержится программа на Simple Basic, файл.sa – результат трансляции в исходном коде SimpleAssembler.

Транслятор работает следующим образом. При запуске транслятора предварительно проверяется наличие файла с расширением «sb». В случае обнаружения ошибки, на экран выводится сообщение об отсутствии файла, и транслятор прекращает свою работу. При отсутствии ошибок производится инициализация выходного буфера, промежуточного буфера и буфера переменных SimpleBasic,а также производится открытие файла исходного кода SimpleBasic.

Трансляция исходного кода SimpleBasicпроизводится в два этапа. На первом этапе производится формирование промежуточного кода, который включает в себя:

* номер строки SimpleBasic для последующего определения адресов перехода оператора GOTO;
* адрес размещения команды SimpleAssembler;
* мнемоника команды SimpleAssembler, соответствующая оператору SimpleBasic;
* имя переменной SimpleBasic, адрес которой будет определен на втором проходе.

Первый этап трансляций происходит путем построчного считывания исходного кода SimpleBasic. Каждая строка разбивается на составляющие элементы и производится проверка синтаксиса. В случае обнаружения ошибки на экран выводится сообщение с указанием номера строки, исходного кода SimpleBasic, содержащего ошибку, и транслятор завершает работу.

На втором этапе трансляции производится определение адресов переменных SimpleBasic с последующей заменой этими адресами имен переменных, хранящихся в промежуточном буфере, а также определение адресов перехода для команд условного и безусловного перехода языка SimpleAssembler. На втором этапе трансляции производится формирование выходного буфера.

После окончания второго этапа трансляции выходной буфер записывается в файл с расширением «sa»и работа транслятора прекращается.

## Блок-схема работы транслятора

Формирование промежуточного кода

да

нет

Ввод параметров командной строки

Проверка входных данных

Начало

Ошибка?

да

нет

Ошибка?

Считывание номера строки и оператора SimpleBasic

Пока не достигнут конец файла

Открытие файла с исходным кодом SB

Считывание

строки кода SimpleBasic

Запись кода в промежуточный буфер

Инициализация буфера промежуточного и выходного кода, а также буфера адресов переменных SB

да

Ошибка?

1

2

нет

Конец

Отображение сообщения об ошибке

Запись выходного буфера в файл

Считывание

строки промежуточного кода

1

2

Формирование выходного кода

Пока не достигнут конец буфера промежуточного кода

да

Ошибка?

Определение адреса размещения переменной SB (адреса перехода)

нет

# Исходный код SimpleComputer

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <sys/time.h>

#include "MySimpleComputer.h"

#include "MyTerm.h"

#include "Interface.h"

int sc\_reg\_flags;

int \*correct\_ops;

int ops\_num;

int instructionCounter;

int unsigned x, y, printcount;

int accumulator;

unsigned int Plus[2] = { 404232447, 4279769112 };

unsigned int Minus[2] = { 255, 4278190080 };

unsigned int Zero[2] = { 2118535762, 1382179454 };

unsigned int One[2] = { 236863102, 235802126 };

unsigned int Two[2] = { 1014908422, 202931838 };

unsigned int Three[2] = { 1013319228, 1007052348 };

unsigned int Four[2] = { 1717986942, 2114323974 };

unsigned int Five[2] = { 2122211452, 503717500 };

unsigned int Six[2] = { 236728416, 2120640126 };

unsigned int Seven[2] = { 2114061320, 2082488320 };

unsigned int Eight[2] = { 1010975292, 1111638588 };

unsigned int Nine[2] = { 2118271614, 33686142 };

void signalhandler1() {

reset();

}

void signalhandler2() {

int flag;

CU();

if (instructionCounter < 99) {

++instructionCounter;

}

sc\_regGet(FLAG\_HALT, &flag);

if (flag) {

// alarm(0);

struct itimerval nval;

nval.it\_interval.tv\_sec = 0;

nval.it\_interval.tv\_usec = 0;

nval.it\_value.tv\_sec = 0;

nval.it\_value.tv\_usec = 0;

setitimer(ITIMER\_REAL, &nval, NULL);

sc\_regSet(FLAG\_IGNORE\_SYSTEM\_TIMER, 1);

}

else {

// alarm(1);

}

info();

}

int main() {

int flag;

reset();

signal(SIGUSR1, signalhandler1);

signal(SIGALRM, signalhandler2);

mt\_clrscr();

sc\_interface();

while (1) {

sc\_regGet(FLAG\_IGNORE\_SYSTEM\_TIMER, &flag);

if (flag) {

info();

if (control()) {

break;

}

}

}

mt\_clrscr();

return 0;

}

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include "MyTerm.h"

#include "Interface.h"

#define MEMORY\_SIZE 100

#define FLAG\_WRONG\_ADDRESS 1

#define FLAG\_OPEN\_FILE 2

#define FLAG\_FILE\_SIZE 4

#define FLAG\_WRONG\_COMMAND 8

#define FLAG\_WRONG\_OPERAND 16

#define FLAG\_OVERFLOW 32

#define FLAG\_IGNORE\_SYSTEM\_TIMER 64

#define FLAG\_HALT 128

#define ERROR -1

extern int sc\_memory[MEMORY\_SIZE];

extern int sc\_reg\_flags;

extern int \*correct\_ops;

extern int ops\_num;

extern int instructionCounter;

extern unsigned int x, y, printcount;

extern int accumulator;

int reset();

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);

int ALU(int command, int operand);

void CU();

#include "MySimpleComputer.h"

int sc\_memory[MEMORY\_SIZE];

int reset() {

x = 0;

y = 0;

mt\_clrscr();

sc\_interface();

sc\_memoryInit();

sc\_regInit();

instructionCounter = 0;

accumulator = 0;

printcount = 31;

return 0;

}

int sc\_memoryInit() {

for (unsigned int i = 0; i < MEMORY\_SIZE; ++i) {

sc\_memory[i] = 0;

}

return 0;

}

int sc\_memorySet(int ADDRESS, int VALUE) {

if ((ADDRESS < 0) || (ADDRESS >= MEMORY\_SIZE)) {

sc\_regSet(FLAG\_WRONG\_ADDRESS, 1);

return ERROR;

}

sc\_memory[ADDRESS] = VALUE;

return 0;

}

int sc\_memoryGet(int ADDRESS, int \*VALUE) {

if ((ADDRESS < 0) || (ADDRESS >= MEMORY\_SIZE)) {

sc\_regSet(FLAG\_WRONG\_ADDRESS, 1);

return ERROR;

}

\*VALUE = sc\_memory[ADDRESS];

return 0;

}

int sc\_memorySave(char \*FILENAME) {

FILE \*file;

if ((file = fopen(FILENAME, "wb")) == NULL) {

sc\_regSet(FLAG\_OPEN\_FILE, 1);

return ERROR;

}

if (fwrite(sc\_memory, MEMORY\_SIZE \* sizeof(\*sc\_memory), 1, file) != 1) {

fclose(file);

sc\_regSet(FLAG\_FILE\_SIZE, 1);

return ERROR;

}

fclose(file);

return 0;

}

int sc\_memoryLoad(char \*FILENAME) {

FILE \*file;

if ((file = fopen(FILENAME, "rb")) == NULL) {

sc\_regSet(FLAG\_OPEN\_FILE, 1);

return ERROR;

}

if (fread(sc\_memory, sizeof(\*sc\_memory) \* MEMORY\_SIZE, 1, file) != 1) {

fclose(file);

sc\_regSet(FLAG\_FILE\_SIZE, 1);

return ERROR;

}

fclose(file);

return 0;

}

int sc\_regInit() {

sc\_reg\_flags = FLAG\_IGNORE\_SYSTEM\_TIMER;

return 0;

}

int sc\_regSet(int REGISTER, int VALUE) {

if ((REGISTER < 0) || (REGISTER > 128)) {

return ERROR;

}

if (VALUE) {

sc\_reg\_flags |= REGISTER;

}

else {

sc\_reg\_flags &= ~REGISTER;

}

return 0;

}

int sc\_regGet(int REGISTER, int \*VALUE) {

if ((REGISTER < 0) || (REGISTER > 128)) {

return ERROR;

}

\*VALUE = sc\_reg\_flags & REGISTER;

return 0;

}

int command\_check(char COMMAND) {

switch (COMMAND) {

case 10: /\*READ\*/

break;

case 11: /\*WRITE\*/

break;

case 20: /\*LOAD\*/

break;

case 21: /\*STORE\*/

break;

case 30: /\*ADD\*/

break;

case 31: /\*SUB\*/

break;

case 32: /\*DIVIDE\*/

break;

case 33: /\*MUL\*/

break;

case 40: /\*JUMP\*/

break;

case 41: /\*JNEG\*/

break;

case 42: /\*JZ\*/

break;

case 43: /\*HALT\*/

break;

case 51: /\*NOT\*/

break;

default:

return ERROR;

break;

}

return 0;

}

int sc\_commandEncode(int COMMAND, int OPERAND, int \*VALUE) {

if (OPERAND >= 128 || OPERAND < 0) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return ERROR;

}

if (command\_check(COMMAND)) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return ERROR;

}

\*VALUE = COMMAND;

\*VALUE <<= 7;

\*VALUE |= OPERAND & 127;

return 0;

}

int sc\_commandDecode(int VALUE, int \*COMMAND, int \*OPERAND) {

\*OPERAND = VALUE & 127;

if (\*OPERAND >= 128 || \*OPERAND < 0) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return ERROR;

}

VALUE >>= 7;

\*COMMAND = VALUE;

if (command\_check(\*COMMAND)) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return ERROR;

}

return 0;

}

int ALU(int command, int operand) {

switch (command) {

case 30: /\* ADD \*/

if (sc\_memory[operand] > 0) {

if (accumulator + sc\_memory[operand] < 2147483648) {

accumulator += sc\_memory[operand];

}

else {

sc\_regSet(FLAG\_OVERFLOW, 1);

accumulator = 2147483647;

}

}

else {

if (accumulator + sc\_memory[operand] > -2147483649) {

accumulator += sc\_memory[operand];

}

else {

sc\_regSet(FLAG\_OVERFLOW, 1);

accumulator = -2147483648;

}

}

break;

case 31: /\* SUB \*/

if (sc\_memory[operand] > 0) {

if (accumulator - sc\_memory[operand] > -2147483649) {

accumulator -= sc\_memory[operand];

}

else {

accumulator = -2147483648;

}

}

else {

if (accumulator - sc\_memory[operand] < 2147483648) {

accumulator -= sc\_memory[operand];

}

else {

accumulator = 2147483647;

}

}

break;

case 33: /\* MUL \*/

if (sc\_memory[operand] > 0) {

if (accumulator > 0) {

if (accumulator \* sc\_memory[operand] < 2147483648) {

accumulator \*= sc\_memory[operand];

}

else {

accumulator = 2147483647;

}

}

else {

if (accumulator \* sc\_memory[operand] > -2147483649) {

accumulator \*= sc\_memory[operand];

}

else {

accumulator = -2147483648;

}

}

}

else {

if (accumulator > 0) {

if (accumulator \* sc\_memory[operand] > -2147483649) {

accumulator \*= sc\_memory[operand];

}

else {

accumulator = -2147483648;

}

}

else {

if (accumulator \* sc\_memory[operand] < 2147483648) {

accumulator \*= sc\_memory[operand];

}

else {

accumulator = 2147483647;

}

}

}

break;

case 32: /\* DIV \*/

if (sc\_memory[operand] != 0) {

accumulator /= sc\_memory[operand];

}

else {

sc\_regSet(FLAG\_WRONG\_OPERAND, 1);

return ERROR;

}

break;

}

return 0;

}

void CU() {

int command, operand;

int flag;

long long int value;

if (instructionCounter >= MEMORY\_SIZE) {

sc\_regSet(FLAG\_WRONG\_ADDRESS, 1);

return;

}

if (sc\_commandDecode(sc\_memory[instructionCounter], &command, &operand)) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return;

}

if ((operand < 0) && (operand >= MEMORY\_SIZE)) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

return;

}

if ((command >= 30) && (command <= 33)) {

if (ALU(command, operand) != 0) {

sc\_regSet(FLAG\_WRONG\_COMMAND, 1);

}

}

else {

switch (command) {

case 10: /\* READ \*/

mt\_gotoXY(0, printcount);

mt\_setfgcolor(Default);

mt\_setbgcolor(Default);

printf("Read: ");

scanf("%lld", &value);

++printcount;

if (value > 2147483647 || value < -2147483648) {

value = 0;

}

sc\_memorySet(operand, value);

break;

case 11: /\* WRITE \*/

mt\_gotoXY(0, printcount);

mt\_setfgcolor(Default);

mt\_setbgcolor(Default);

printf("Write: %d\n", sc\_memory[operand]);

++printcount;

break;

case 20: /\* LOAD \*/

accumulator = sc\_memory[operand];

break;

case 21: /\* STORE \*/

sc\_memory[operand] = accumulator;

break;

case 40: /\* JUMP \*/

instructionCounter = operand - 1;

break;

case 41: /\* JNEG \*/

if (accumulator < 0) {

instructionCounter = operand - 1;

}

break;

case 42: /\* JZ \*/

if (accumulator == 0) {

instructionCounter = operand - 1;

}

break;

case 43: /\* HALT \*/

sc\_regSet(FLAG\_HALT, 1);

mt\_setfgcolor(Default);

mt\_setbgcolor(Default);

mt\_gotoXY(0, printcount);

printf("Program is over\n");

break;

case 51: /\* NOT \*/

accumulator = ~operand;

sc\_memory[operand] = accumulator;

break;

} }

}

}

#pragma once

#include <sys/ioctl.h>

#include <stdio.h>

#include <unistd.h>

#include "MySimpleComputer.h"

enum Color {

Black,

Red,

Green,

Brown,

Blue,

Magneta,

Cyan,

LGray,

Default

};

int mt\_clrscr();

int mt\_gotoXY(int X, int Y);

int mt\_getscreensize(int \*ROWS, int \*COLUMNS);

int mt\_setfgcolor(enum Color COLOR);

int mt\_setbgcolor(enum Color COLOR);

#include "MyTerm.h"

int mt\_clrscr() {

printf("\E[H\E[2J");

return 0;

}

int mt\_gotoXY(int X, int Y) {

int max\_x, max\_y;

mt\_getscreensize(&max\_y, &max\_x);

if ((Y >= max\_y) || (X >= max\_x) || (X < 0) || (Y < 0)) {

return ERROR;

}

printf("\E[%d;%dH", Y, X);

return 0;

}

int mt\_getscreensize(int \*MAX\_ROWS, int \*MAX\_COLUMNS) {

struct winsize buffer;

if (ioctl(STDOUT\_FILENO, TIOCGWINSZ, &buffer)) {

return ERROR;

}

\*MAX\_ROWS = buffer.ws\_row;

\*MAX\_COLUMNS = buffer.ws\_col;

return 0;

}

int mt\_setfgcolor(enum Color COLOR) {

switch (COLOR) {

case Black:

printf("\E[30m");

break;

case Red:

printf("\E[31m");

break;

case Green:

printf("\E[32m");

break;

case Brown:

printf("\E[33m");

break;

case Blue:

printf("\E[34m");

break;

case Magneta:

printf("\E[35m");

break;

case Cyan:

printf("\E[36m");

break;

case LGray:

printf("\E[37m");

break;

case Default:

printf("\E[39m");

break;

default:

return ERROR;

}

return 0;

}

int mt\_setbgcolor(enum Color COLOR) {

switch (COLOR) {

case Black:

printf("\E[40m");

break;

case Red:

printf("\E[41m");

break;

case Green:

printf("\E[42m");

break;

case Brown:

printf("\E[43m");

break;

case Blue:

printf("\E[44m");

break;

case Magneta:

printf("\E[45m");

break;

case Cyan:

printf("\E[46m");

break;

case LGray:

printf("\E[47m");

break;

case Default:

printf("\E[49m");

break;

default:

return ERROR;

}

return 0;

}

#pragma once

#include <stdio.h>

#include <sys/types.h>

#include <fcntl.h>

#include <string.h>

#include <unistd.h>

#include <termios.h>

#include "MySimpleComputer.h"

enum Key {

KEY\_l,

KEY\_s,

KEY\_r,

KEY\_t,

KEY\_i,

KEY\_q,

KEY\_f5,

KEY\_f6,

KEY\_up,

KEY\_down,

KEY\_left,

KEY\_right,

KEY\_enter,

KEY\_other

};

int rk\_readkey(enum Key \*KEY);

int rk\_mytermsave();

int rk\_mytermrestore();

int rk\_mytermregime(int regime, int vtime, int vmin, int echo, int sigint);

#include "MyReadKeys.h"

int rk\_readkey(enum Key \*KEY) {

struct termios orig\_options;

char buf[16];

int num\_read;

if (tcgetattr(STDIN\_FILENO, &orig\_options)) {

return ERROR;

}

if (rk\_mytermregime(0, 1, 0, 0, 0)) {

return ERROR;

}

num\_read = read(STDIN\_FILENO, buf, 15);

if (num\_read < 0) {

return ERROR;

}

buf[num\_read] = '\0';

if (strcmp(buf, "l") == 0) {

\*KEY = KEY\_l;

}

else {

if (strcmp(buf, "s") == 0) {

\*KEY = KEY\_s;

}

else {

if (strcmp(buf, "r") == 0) {

\*KEY = KEY\_r;

}

else {

if (strcmp(buf, "t") == 0) {

\*KEY = KEY\_t;

}

else {

if (strcmp(buf, "i") == 0) {

\*KEY = KEY\_i;

}

else {

if (strcmp(buf, "q") == 0) {

\*KEY = KEY\_q;

}

else {

if (strcmp(buf, "\n") == 0) {

\*KEY = KEY\_enter;

}

else {

if (strcmp(buf, "\033[15~") == 0) {

\*KEY = KEY\_f5;

}

else {

if (strcmp(buf, "\033[17~") == 0) {

\*KEY = KEY\_f6;

}

else {

if (strcmp(buf, "\033[A") == 0) {

\*KEY = KEY\_up;

}

else {

if (strcmp(buf, "\033[B") == 0) {

\*KEY = KEY\_down;

}

else {

if (strcmp(buf, "\033[C") == 0) {

\*KEY = KEY\_right;

}

else {

if (strcmp(buf, "\033[D") == 0) {

\*KEY = KEY\_left;

}

else {

\*KEY = KEY\_other;

}

}

}

}

}

}

}

}

}

}

}

}

}

if (tcsetattr(STDIN\_FILENO, TCSANOW, &orig\_options)) {

return ERROR;

}

return 0;

}

int rk\_mytermsave() {

struct termios options;

FILE \*file;

if (tcgetattr(STDIN\_FILENO, &options)) {

return ERROR;

}

if ((file = fopen("termsettings", "wb")) == NULL) {

return ERROR;

}

fwrite(&options, sizeof(options), 1, file);

fclose(file);

return 0;

}

int rk\_mytermrestore() {

struct termios options;

FILE \*file;

if ((file = fopen("termsettings", "rb")) == NULL) {

return ERROR;

}

if (fread(&options, sizeof(options), 1, file) == 0) {

return ERROR;

}

if (tcsetattr(STDIN\_FILENO, TCSAFLUSH, &options)) {

return ERROR;

}

return 0;

}

int rk\_mytermregime(int regime, int vtime, int vmin, int echo, int sigint) {

struct termios options;

if (tcgetattr(STDIN\_FILENO, &options)) {

return ERROR;

}

if (regime == 1) {

options.c\_lflag |= ICANON;

}

else {

if (regime == 0) {

options.c\_lflag &= ~ICANON;

options.c\_cc[vtime] = vtime;

options.c\_cc[vmin] = vmin;

if (echo == 1) {

options.c\_lflag |= echo;

}

else {

if (echo == 0) {

options.c\_lflag &= ~echo;

}

else {

return ERROR;

}

}

if (sigint == 1) {

options.c\_lflag |= ISIG;

}

else {

if (sigint == 0) {

options.c\_lflag &= ~ISIG;

}

else {

return ERROR;

}

}

}

else {

return ERROR;

}

}

if (tcsetattr(STDIN\_FILENO, TCSANOW, &options)) {

return ERROR;

}

return 0;

}

#pragma once

#include "MyTerm.h"

#define BOXCHAR\_FULL 'a'

#define BOXCHAR\_DOWN\_RIGHT "j"

#define BOXCHAR\_DOWN\_LEFT "m"

#define BOXCHAR\_UP\_RIGHT "k"

#define BOXCHAR\_UP\_LEFT "l"

#define BOXCHAR\_VERTICAL "x"

#define BOXCHAR\_HORIZONTAL "q"

int bc\_printA(char \*STR);

int bc\_box(int X1, int Y1, int X2, int Y2);

int bc\_printbigchar(unsigned int \*BIG, int X, int Y, enum Color FG, enum Color BG);

int bc\_setbigcharpos(int \*BIG, int X, int Y, int VALUE);

int bc\_getbigcharpos(int \*BIG, int X, int Y, int \*VALUE);

int bc\_bigcharwrite(int FD, int \*BIG, int COUNT);

int bc\_bigcharread(int FD, int \*BIG, int NEED\_COUNT, int \*COUNT);

#include "MyBigChar.h"

int bc\_printA(char \*STR) {

printf("\E(0%s\E(B", STR);

return 0;

}

int bc\_box(int X1, int Y1, int X2, int Y2) {

int tmp;

int max\_x, max\_y;

int i;

if (X1 > X2) {

tmp = X1;

X1 = X2;

X2 = tmp;

}

if (Y1 > Y2) {

tmp = Y1;

Y1 = Y2;

Y2 = tmp;

}

if (X1 < 0 || Y1 < 0 || X2 - X1 < 2 || Y2 - Y1 < 2) {

return ERROR;

}

else {

mt\_getscreensize(&max\_y, &max\_x);

if (X2 > max\_x || Y2 > max\_y) {

return ERROR;

}

}

mt\_gotoXY(X1, Y1);

bc\_printA(BOXCHAR\_UP\_LEFT);

for (i = X1 + 1; i < X2 - 1; ++i) {

bc\_printA(BOXCHAR\_HORIZONTAL);

}

mt\_gotoXY(X2 - 1, Y1);

bc\_printA(BOXCHAR\_UP\_RIGHT);

for (i = Y1 + 1; i < Y2; ++i) {

mt\_gotoXY(X2 - 1, i);

bc\_printA(BOXCHAR\_VERTICAL);

}

for (i = Y1 + 1; i < Y2; ++i) {

mt\_gotoXY(X1, i);

bc\_printA(BOXCHAR\_VERTICAL);

}

mt\_gotoXY(X1, Y2);

bc\_printA(BOXCHAR\_DOWN\_LEFT);

for (i = X1 + 1; i < X2 - 1; ++i) {

bc\_printA(BOXCHAR\_HORIZONTAL);

}

bc\_printA(BOXCHAR\_DOWN\_RIGHT);

return 0;

}

int bc\_printbigchar(unsigned int \*BIG, int X, int Y, enum Color FG, enum Color BG) {

int maxx, maxy;

unsigned int bit;

int i, j, k;

char string[9];

string[8] = '\0';

if (X < 0 || Y < 0) {

return ERROR;

}

else {

mt\_getscreensize(&maxy, &maxx);

if (X > maxx || Y > maxy) {

return ERROR;

}

}

mt\_setfgcolor(FG);

mt\_setbgcolor(BG);

for (i = 1; i >= 0; --i) {

bit = BIG[i];

for (j = 3; j >= 0; --j) {

mt\_gotoXY(X, Y + i \* 4 + j);

for (k = 7; k >= 0; --k) {

if (bit & 1) {

string[k] = BOXCHAR\_FULL;

}

else {

string[k] = ' ';

}

bit >>= 1;

}

bc\_printA(string);

}

}

mt\_setfgcolor(Default);

mt\_setbgcolor(Default);

return 0;

}

int bc\_setbigcharpos(int \*BIG, int X, int Y, int VALUE) {

int position;

if (X < 0 || Y < 0 || X > 7 || Y > 7 || VALUE < 0 || VALUE > 1) {

return ERROR;

}

if (Y <= 3) {

position = 0;

}

else {

position = 1;

}

Y = Y % 4;

if (VALUE) {

BIG[position] |= 1 << (Y \* 8 + X);

}

else {

BIG[position] &= ~(1 << (Y \* 8 + X));

}

return 0;

}

int bc\_getbigcharpos(int \*BIG, int X, int Y, int \*VALUE) {

int position;

if (X < 0 || Y < 0 || X > 7 || Y > 7) {

return ERROR;

}

if (Y <= 3) {

position = 0;

}

else {

position = 1;

}

Y = Y % 4;

\*VALUE = (BIG[position] >> (Y \* 8 + X)) & 1;

return 0;

}

int bc\_bigcharwrite(int FD, int \*BIG, int COUNT) {

if (write(FD, &COUNT, sizeof(COUNT)) == -1) {

return ERROR;

}

if (write(FD, BIG, COUNT \* (sizeof(int)) \* 2) == -1) {

return ERROR;

}

return 0;

}

int bc\_bigcharread(int FD, int \*BIG, int NEED\_COUNT, int \*COUNT) {

int n, outcome;

outcome = read(FD, &n, sizeof(n));

if (outcome == -1 || (outcome != sizeof(n))) {

return ERROR;

}

outcome = read(FD, BIG, NEED\_COUNT \* sizeof(int) \* 2);

if (outcome == -1) {

return ERROR;

}

\*COUNT = outcome / (sizeof(int) \* 2);

return 0;

}

#pragma once

#include <math.h>

#include <signal.h>

#include <sys/time.h>

#include "MySimpleComputer.h"

#include "MyBigChar.h"

#include "MyReadKeys.h"

extern unsigned int Plus[2];

extern unsigned int Minus[2];

extern unsigned int Zero[2];

extern unsigned int One[2];

extern unsigned int Two[2];

extern unsigned int Three[2];

extern unsigned int Four[2];

extern unsigned int Five[2];

extern unsigned int Six[2];

extern unsigned int Seven[2];

extern unsigned int Eight[2];

extern unsigned int Nine[2];

extern int instructionCounter;

void print\_simbol(int X, int Y, int SIMBOL, enum Color FG, enum Color BG);

void print\_big\_char(int X, int Y, int NUMBER, enum Color FG, enum Color BG);

void sc\_interface();

void info();

void instrcnt2xy();

int control();

#include "Interface.h"

void print\_simbol(int X, int Y, int SIMBOL, enum Color FG, enum Color BG) {

switch (SIMBOL) {

case 0:

bc\_printbigchar(Zero, X, Y, FG, BG);

break;

case 1:

bc\_printbigchar(One, X, Y, FG, BG);

break;

case 2:

bc\_printbigchar(Two, X, Y, FG, BG);

break;

case 3:

bc\_printbigchar(Three, X, Y, FG, BG);

break;

case 4:

bc\_printbigchar(Four, X, Y, FG, BG);

break;

case 5:

bc\_printbigchar(Five, X, Y, FG, BG);

break;

case 6:

bc\_printbigchar(Six, X, Y, FG, BG);

break;

case 7:

bc\_printbigchar(Seven, X, Y, FG, BG);

break;

case 8:

bc\_printbigchar(Eight, X, Y, FG, BG);

break;

case 9:

bc\_printbigchar(Nine, X, Y, FG, BG);

break;

}

}

void print\_big\_char(int X, int Y, int NUMBER, enum Color FG, enum Color BG) {

mt\_gotoXY(X, Y);

if (NUMBER < 0) {

bc\_printbigchar(Minus, X, Y, FG, BG);

}

else {

bc\_printbigchar(Plus, X, Y, FG, BG);

}

X += 8;

unsigned short int i = 9, n;

do {

mt\_gotoXY(X, Y);

n = abs(NUMBER) / (int)pow(10, i) % 10;

print\_simbol(X, Y, n, FG, BG);

X += 8;

--i;

} while (i > 0);

n = abs(NUMBER) % 10;

print\_simbol(X, Y, n, FG, BG);

}

void sc\_interface() {

int max\_rows, max\_columns;

mt\_getscreensize(&max\_rows, &max\_columns);

bc\_box(1, 1, max\_columns - 1, 12);

bc\_box(1, 13, 21, 26);

bc\_box(22, 13, max\_columns - 1, 22);

bc\_box(22, 23, max\_columns - 1, 26);

mt\_gotoXY(2, 14);

printf("Enter. Set value");

mt\_gotoXY(2, 15);

printf("l. Load memory");

mt\_gotoXY(2, 16);

printf("s. Save memory");

mt\_gotoXY(2, 17);

printf("r. Run");

mt\_gotoXY(2, 18);

printf("t. Step");

mt\_gotoXY(2, 19);

printf("i. Reset");

mt\_gotoXY(2, 20);

printf("F5. Accumulator");

mt\_gotoXY(2, 21);

printf("F6. Instr Counter");

mt\_gotoXY(2, 22);

printf("q. Quit");

mt\_gotoXY(0, 27);

printf("Size of window: %d x %d\n\n\n", max\_columns, max\_rows);

printf("Input/Output\n");

}

void instrcnt2xy() {

x = instructionCounter % 10;

y = instructionCounter / 10;

}

void info() {

int flag;

instrcnt2xy();

for (unsigned int i = 0; i < MEMORY\_SIZE / 10; ++i) {

mt\_gotoXY(2, i + 2);

for (unsigned int j = 0; j < 10; ++j) {

mt\_setbgcolor(Magneta);

printf("%3d. ", i \* 10 + j);

mt\_setbgcolor(Default);

if (i == y && j == x) {

mt\_setbgcolor(Green);

printf("%10d ", sc\_memory[i \* 10 + j]);

mt\_setbgcolor(Default);

}

else {

printf("%10d ", sc\_memory[i \* 10 + j]);

}

}

}

print\_big\_char(24, 14, sc\_memory[x + y \* 10], Green, Default);

mt\_gotoXY(23, 24);

printf(" ");

mt\_gotoXY(23, 24);

sc\_regGet(FLAG\_WRONG\_ADDRESS, &flag);

if (flag) {

mt\_setfgcolor(Red);

printf("WRONG\_ADDRESS ");

printf("NOT\_OK");

}

else {

printf("WRONG\_ADDRESS ");

sc\_regGet(FLAG\_OPEN\_FILE, &flag);

if (flag) {

mt\_setfgcolor(Red);

printf("CAN'T\_OPEN\_FILE ");

printf("NOT\_OK");

}

else {

printf("CAN'T\_OPEN\_FILE ");

sc\_regGet(FLAG\_FILE\_SIZE, &flag);

if (flag) {

mt\_setfgcolor(Red);

printf("WRONG\_FILE\_SIZE ");

printf("NOT\_OK");

}

else {

printf("WRONG\_FILE\_SIZE ");

sc\_regGet(FLAG\_IGNORE\_SYSTEM\_TIMER, &flag);

if (flag) {

mt\_setfgcolor(Green);

printf("IGNORE\_SYSTEM\_TIMER ");

}

else {

mt\_setfgcolor(Default);

printf("NOT\_IGNORE\_SYSTEM\_TIMER ");

}

mt\_setfgcolor(Green);

printf("OK");

}

}

}

mt\_setfgcolor(Default);

mt\_gotoXY(0, 28);

printf(" ");

mt\_gotoXY(0, 28);

printf("instructionCounter %d | accumulator %d\n", instructionCounter, accumulator);

}

int control() {

char filename[21] = { "operative\_memory.dat" };

int value, flag;

enum Key KEY;

mt\_gotoXY(0, 30);

while (flag) {

rk\_readkey(&KEY);

flag = 0;

switch (KEY) {

case KEY\_l:

sc\_memoryLoad(filename);

break;

case KEY\_s:

sc\_memorySave(filename);

break;

case KEY\_r:

sc\_regSet(FLAG\_IGNORE\_SYSTEM\_TIMER, 0);

x = 0;

y = 0;

// alarm(1);

struct itimerval nval;

nval.it\_interval.tv\_sec = 0;

nval.it\_interval.tv\_usec = 100;

nval.it\_value.tv\_sec = 0;

nval.it\_value.tv\_usec = 100;

setitimer(ITIMER\_REAL, &nval, NULL);

break;

case KEY\_t:

sc\_regGet(FLAG\_IGNORE\_SYSTEM\_TIMER, &flag);

if (flag) {

sc\_regGet(FLAG\_HALT, &flag);

if (flag == 0) {

CU();

if (instructionCounter < 99) {

++instructionCounter;

}

else {

instructionCounter = 99;

sc\_regSet(FLAG\_HALT, 1);

}

}

}

flag = 0;

break;

case KEY\_i:

raise(SIGUSR1);

break;

case KEY\_q:

return 1;

break;

case KEY\_f5:

mt\_gotoXY(0, printcount);

printf("Enter accumulator:");

scanf("%d", &accumulator);

break;

case KEY\_f6:

mt\_gotoXY(0, printcount);

printf("Enter instructionCounter:");

scanf("%d", &instructionCounter);

break;

case KEY\_up:

if (y > 0) {

--y;

}

break;

case KEY\_down:

if (y < 9) {

++y;

}

break;

case KEY\_left:

if (x > 0) {

--x;

}

break;

case KEY\_right:

if (x < 9) {

++x;

}

break;

case KEY\_enter:

mt\_gotoXY(0, printcount);

printf("Enter value:");

scanf("%d", &value);

sc\_memorySet(x + y \* 10, value);

break;

case KEY\_other:

flag = 1;

break;

}

}

return 0;

}

# Исходный код Translator

#include "Basic2Asm.h"

#include "Asm2Bin.h"

int main() {

char filename1[] = { "program.bas" };

char filename2[] = { "program.asm" };

char filename3[] = { "operative\_memory.dat" };

basic2asm(filename1, filename2);

asm2bin(filename2, filename3);

return 0;

}

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <list>

#include <stack>

#include <iterator>

enum BasicComand {

INPUT,

PRINT,

LET,

IF,

GOTO,

REM,

END

};

struct Variable {

char Var;

int str\_num;

};

struct Constant {

int Const;

int str\_num;

};

struct Address {

int Addr;

int str\_num;

};

struct Link {

int str\_num;

int Lnk;

};

int chk\_command(enum BasicComand \*key, char \*Comm);

int print(FILE \* fout, std::list<Variable> varList,

std::list<Constant>& constList, char \* Variables, int \* Constants,

int & StrCounter);

int input(FILE \* fout, std::list<Variable> &varList, char \* Variables,

int & StrCounter);

int let(FILE \* fout, std::list<Variable>& varList,

std::list<Constant>& constList, int \* Constants, char \* Variables,

int & StrCounter, int & max\_tmp\_num);

int go\_to(FILE \* fout, std::list<Address>& addrList, int & StrCounter);

int basic2asm(char \*filename1, char \*filename2);

void add\_var(char variable, std::list<Variable> &varList);

void add\_const(int Const, std::list<Constant> &constList);

void add\_addr(int Addr, int str\_num, std::list<Address> &addrList);

int chk\_var(int Var, std::list<Variable> varList);

int chk\_const(int Const, std::list<Constant> constList);

int is\_var(char simbol);

int get\_prior(char c);

int rpn(char \*rpn\_str, std::list<Constant> &constList);

int rpn\_pars(FILE \*fout, int \*Constants, char \*Variables, int &StrCounter,

char \*rpn, int &max\_tmp\_num);

void initialization(FILE \*fout, std::list<Variable> &varList,

std::list<Constant> &constList, int max\_tmp\_num, int &StrCounter);

void linking(std::list<Link> &lnkList, std::list<Variable> varList,

std::list<Constant> constList, std::list<Address> addrList,

char \*Variables, int \*Constants, int \*Bas\_str\_nums);

void second\_run(FILE \*fout, std::list<Link> &lnkList);

bool lnk\_sort\_pred(Link FIRST, Link SECOND);

#include "Basic2Asm.h"

int chk\_command(enum BasicComand \*key, char \*Comm) {

if (strcmp(Comm, "REM") == 0) {

\*key = REM;

}

else {

if (strcmp(Comm, "INPUT") == 0) {

\*key = INPUT;

}

else {

if (strcmp(Comm, "PRINT") == 0) {

\*key = PRINT;

}

else {

if (strcmp(Comm, "GOTO") == 0) {

\*key = GOTO;

}

else {

if (strcmp(Comm, "IF") == 0) {

\*key = IF;

}

else {

if (strcmp(Comm, "LET") == 0) {

\*key = LET;

}

else {

if (strcmp(Comm, "END") == 0) {

\*key = END;

}

else {

return -1;

}

}

}

}

}

}

}

return 0;

}

int print(FILE \*fout, std::list<Variable> varList,

std::list<Constant> &constList, char \*Variables, int \*Constants,

int &StrCounter) {

int number\_buff;

char \*ptr;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (is\_var(\*ptr)) {

if (!chk\_var(\*ptr, varList)) {

return 1;

}

Variables[StrCounter] = \*ptr;

}

else {

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d WRITE \n", StrCounter);

++StrCounter;

return 0;

}

int input(FILE \*fout, std::list<Variable> &varList, char \*Variables, int &StrCounter) {

char \*ptr;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (!chk\_var(\*ptr, varList)) {

add\_var(\*ptr, varList);

}

Variables[StrCounter] = \*ptr;

fprintf(fout, "%d READ \n", StrCounter);

++StrCounter;

return 0;

}

int let(FILE \*fout, std::list<Variable> &varList,

std::list<Constant> &constList, int \*Constants, char \*Variables,

int &StrCounter, int &max\_tmp\_num) {

char \*ptr, rpn\_str[100];

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (!chk\_var(\*ptr, varList)) {

add\_var(\*ptr, varList);

}

if (rpn(rpn\_str, constList)) {

return 1;

}

if (rpn\_pars(fout, Constants, Variables, StrCounter, rpn\_str, max\_tmp\_num)) {

return 1;

}

fprintf(fout, "%d STORE \n", StrCounter);

Variables[StrCounter] = \*ptr;

++StrCounter;

return 0;

}

int go\_to(FILE \*fout, std::list<Address> &addrList, int &StrCounter) {

int number\_buff;

char \*ptr;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

add\_addr(number\_buff, StrCounter, addrList);

fprintf(fout, "%d JUMP \n", StrCounter);

++StrCounter;

return 0;

}

void add\_var(char variable, std::list<Variable> &varList) {

Variable buff;

buff.Var = variable;

varList.push\_back(buff);

}

void add\_const(int Const, std::list<Constant> &constList) {

Constant buff;

buff.Const = Const;

constList.push\_back(buff);

}

void add\_addr(int Addr, int str\_num, std::list<Address> &addrList) {

Address buff;

buff.Addr = Addr;

buff.str\_num = str\_num;

addrList.push\_back(buff);

}

int chk\_var(int Var, std::list<Variable> varList) {

for (auto &element : varList) {

if (element.Var == Var) {

return 1;

}

}

return 0;

}

int chk\_const(int Const, std::list<Constant> constList) {

for (auto &element : constList) {

if (element.Const == Const) {

return 1;

}

}

return 0;

}

int is\_var(char simbol) {

if (simbol >= 'A' && simbol <= 'Z') {

return 1;

}

return 0;

}

int get\_prior(char c) {

switch (c) {

case '\*':

return 3;

case '/':

return 3;

case '-':

return 2;

case '+':

return 2;

case '(':

return 1;

}

return 0;

}

int rpn(char \*rpn\_str, std::list<Constant> &constList) {

int str\_pos = 0, rpn\_pos = 0, number\_buff;

char \*ptr;

std::stack<char> rpn\_stack;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

while (ptr = strtok(NULL, " \t")) {

if (\*ptr == ')') {

while (rpn\_stack.top() != '(') {

rpn\_str[rpn\_pos++] = rpn\_stack.top();

rpn\_str[rpn\_pos++] = ' ';

rpn\_stack.pop();

}

rpn\_stack.pop();

}

else {

if (is\_var(\*ptr)) {

rpn\_str[rpn\_pos++] = \*ptr;

rpn\_str[rpn\_pos++] = ' ';

}

else {

if (\*ptr == '(') {

rpn\_stack.push('(');

}

else {

if (\*ptr == '+' || \*ptr == '-' || \*ptr == '/' || \*ptr == '\*') {

if (rpn\_stack.empty()) {

rpn\_stack.push(\*ptr);

}

else {

if (get\_prior(rpn\_stack.top()) < get\_prior(\*ptr)) {

rpn\_stack.push(\*ptr);

}

else {

while (!rpn\_stack.empty() && (get\_prior(rpn\_stack.top()) >= get\_prior(\*ptr))) {

rpn\_str[rpn\_pos++] = rpn\_stack.top();

rpn\_str[rpn\_pos++] = ' ';

rpn\_stack.pop();

}

rpn\_stack.push(\*ptr);

}

}

}

else {

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

while (\*ptr != '\0' && \*ptr != '\n') {

rpn\_str[rpn\_pos++] = \*ptr;

++ptr;

}

rpn\_str[rpn\_pos++] = ' ';

}

}

}

}

}

while (!rpn\_stack.empty()) {

rpn\_str[rpn\_pos++] = rpn\_stack.top();

rpn\_str[rpn\_pos++] = ' ';

rpn\_stack.pop();

}

rpn\_str[rpn\_pos] = '\0';

return 0;

}

int rpn\_pars(FILE \*fout, int \*Constants, char \*Variables, int &StrCounter,

char \*rpn, int &max\_tmp\_num) {

char \*ptr;

int tmp\_num = 0;

int k, buff\_number;

struct operand {

bool is\_var;

union {

char variable;

int constant;

};

};

operand buff, second\_operand;

std::stack<operand> rpn\_stack;

ptr = strtok(rpn, " \t");

if (ptr == NULL) {

return 1;

}

if (is\_var(\*ptr)) {

Variables[StrCounter] = \*ptr;

buff.is\_var = true;

buff.variable = \*ptr;

rpn\_stack.push(buff);

}

else {

if (sscanf(ptr, "%d", &buff\_number) != 1) {

return 1;

}

Constants[StrCounter] = buff\_number;

buff.is\_var = false;

buff.constant = \*ptr;

rpn\_stack.push(buff);

}

k = 0;

fprintf(fout, "%d LOAD \n", StrCounter);

++StrCounter;

while (ptr = strtok(NULL, " \t")) {

if (ptr == NULL) {

break;

}

if (is\_var(\*ptr)) {

buff.is\_var = true;

buff.variable = \*ptr;

rpn\_stack.push(buff);

}

else {

if ((\*ptr == '+') || (\*ptr == '-') || (\*ptr == '\*') || (\*ptr == '/')) {

if (tmp\_num < int(rpn\_stack.size() - 2)) {

k = 1;

}

else {

if (tmp\_num > int(rpn\_stack.size() - 2)) {

k = 2;

}

else {

k = 0;

}

}

switch (k) {

case 0:

{

if (rpn\_stack.top().is\_var) {

second\_operand.variable = rpn\_stack.top().variable;

second\_operand.is\_var = true;

}

else {

second\_operand.constant = rpn\_stack.top().constant;

second\_operand.is\_var = false;

}

rpn\_stack.pop();

rpn\_stack.pop();

}

break;

case 1:

{

fprintf(fout, "%d STORE \n", StrCounter);

Variables[StrCounter] = tmp\_num;

++StrCounter;

++tmp\_num;

if (tmp\_num > max\_tmp\_num) {

max\_tmp\_num = tmp\_num;

}

if (rpn\_stack.top().is\_var) {

second\_operand.variable = rpn\_stack.top().variable;

second\_operand.is\_var = true;

}

else {

second\_operand.constant = rpn\_stack.top().constant;

second\_operand.is\_var = false;

}

rpn\_stack.pop();

fprintf(fout, "%d LOAD \n", StrCounter);

if (rpn\_stack.top().is\_var) {

Variables[StrCounter] = rpn\_stack.top().variable;

}

else {

Constants[StrCounter] = rpn\_stack.top().constant;

}

++StrCounter;

rpn\_stack.pop();

}

break;

case 2:

{

fprintf(fout, "%d STORE \n", StrCounter);

Variables[StrCounter] = tmp\_num;

++StrCounter;

--tmp\_num;

if (rpn\_stack.top().is\_var) {

second\_operand.variable = rpn\_stack.top().variable;

second\_operand.is\_var = true;

}

else {

second\_operand.constant = rpn\_stack.top().constant;

second\_operand.is\_var = false;

}

rpn\_stack.pop();

fprintf(fout, "%d LOAD \n", StrCounter);

if (rpn\_stack.top().is\_var) {

Variables[StrCounter] = rpn\_stack.top().variable;

}

else {

Constants[StrCounter] = rpn\_stack.top().constant;

}

++StrCounter;

rpn\_stack.pop();

}

break;

}

switch (\*ptr) {

case '+':

fprintf(fout, "%d ADD \n", StrCounter);

break;

case '-':

fprintf(fout, "%d SUB \n", StrCounter);

break;

case '/':

fprintf(fout, "%d DIVIDE \n", StrCounter);

break;

case '\*':

fprintf(fout, "%d MUL \n", StrCounter);

break;

}

if (second\_operand.is\_var) {

Variables[StrCounter] = second\_operand.variable;

}

else {

Constants[StrCounter] = second\_operand.constant;

}

++StrCounter;

buff.variable = tmp\_num;

buff.is\_var = true;

rpn\_stack.push(buff);

}

else {

if (sscanf(ptr, "%d", &buff\_number) != 1) {

return 1;

}

buff.constant = buff\_number;

buff.is\_var = false;

rpn\_stack.push(buff);

}

}

}

return 0;

}

void initialization(FILE \*fout, std::list<Variable> &varList,

std::list<Constant> &constList, int max\_tmp\_num, int &StrCounter) {

for (auto &element : constList) {

fprintf(fout, "%d = %d\n", StrCounter, element.Const);

element.str\_num = StrCounter;

++StrCounter;

}

for (auto &element : varList) {

element.str\_num = StrCounter;

++StrCounter;

}

for (int i = 0; i <= max\_tmp\_num; ++i) {

add\_var(i, varList);

varList.back().str\_num = StrCounter;

++StrCounter;

}

}

void linking(std::list<Link> &lnkList, std::list<Variable> varList,

std::list<Constant> constList, std::list<Address> addrList,

char \*Variables, int \*Constants, int \*Bas\_str\_nums) {

int i;

Link buff;

for (auto &element : varList) {

buff.Lnk = element.str\_num;

for (i = 0; i < 100; ++i) {

if (element.Var == Variables[i]) {

buff.str\_num = i;

lnkList.push\_back(buff);

}

}

}

for (auto &element : constList) {

buff.Lnk = element.str\_num;

for (i = 0; i < 100; ++i) {

if (element.Const == Constants[i]) {

buff.str\_num = i;

lnkList.push\_back(buff);

}

}

}

for (auto &element : addrList) {

buff.str\_num = element.str\_num;

for (i = 0; i < 100; ++i) {

if (element.Addr == Bas\_str\_nums[i]) {

buff.Lnk = i;

lnkList.push\_back(buff);

}

}

}

}

void second\_run(FILE \*fout, std::list<Link> &lnkList) {

int StrCounter = -1;

char link[2], Str[100];

std::list<Link>::iterator it = lnkList.begin();

while (fgets(Str, 100, fout) && it != lnkList.end()) {

++StrCounter;

if (StrCounter == it->str\_num) {

fseek(fout, -3, SEEK\_CUR);

link[0] = (it->Lnk / 10) + '0';

link[1] = (it->Lnk % 10) + '0';

fprintf(fout, "%c", link[0]);

fprintf(fout, "%c", link[1]);

fseek(fout, 1, SEEK\_CUR);

++it;

}

}

}

bool lnk\_sort\_pred(Link FIRST, Link SECOND) {

return FIRST.str\_num < SECOND.str\_num;

}

int basic2asm(char \*filename1, char \*filename2) {

int StrCounter = 0, ifjmp = -1;

int Bas\_str\_nums[100], Constants[100];

int basic\_str\_num, i, number\_buff, max\_tmp\_num = 0;

char Str[100], \*ptr;

char Variables[100];

std::list<Variable> varList;

std::list<Constant> constList;

std::list<Address> addrList;

std::list<Link> lnkList;

FILE \*fin;

FILE \*fout;

enum BasicComand Comm;

if ((fin = fopen(filename1, "r")) == NULL) {

return 1;

}

if ((fout = fopen(filename2, "w+")) == NULL) {

return 1;

}

for (i = 0; i < 100; ++i) {

Variables[i] = ' ';

}

for (i = 0; i < 100; ++i) {

Constants[i] = -1;

}

while (fgets(Str, 100, fin)) {

ptr = strtok(Str, " \t\n");

if (ptr == NULL) {

continue;

}

if (sscanf(ptr, "%d", &basic\_str\_num) != 1) {

return 1;

}

Bas\_str\_nums[StrCounter] = basic\_str\_num;

if (ifjmp != -1) {

add\_addr(basic\_str\_num, ifjmp, addrList);

ifjmp = -1;

}

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

chk\_command(&Comm, ptr);

switch (Comm) {

case REM:

{

continue;

}

break;

case PRINT:

{

if (print(fout, varList, constList, Variables, Constants,

StrCounter)) {

return 1;

}

}

break;

case INPUT:

{

if (input(fout, varList, Variables, StrCounter)) {

return 1;

}

}

break;

case LET:

{

if (let(fout, varList, constList, Constants, Variables,

StrCounter, max\_tmp\_num)) {

return 1;

}

}

break;

case END:

{

fprintf(fout, "%d HALT 00\n", StrCounter);

++StrCounter;

}

break;

case GOTO:

{

if (go\_to(fout, addrList, StrCounter)) {

return 1;

}

}

break;

case IF:

{

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

char \*first\_var = ptr;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

switch (\*ptr) {

case '<':

{

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (is\_var(\*ptr)) {

if (!chk\_var(\*ptr, varList)) {

return 1;

}

Variables[StrCounter] = \*ptr;

}

else {

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d LOAD \n", StrCounter);

++StrCounter;

if (is\_var(\*first\_var)) {

if (!chk\_var(\*first\_var, varList)) {

return 1;

}

Variables[StrCounter] = \*first\_var;

}

else {

if (sscanf(first\_var, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d SUB \n", StrCounter);

++StrCounter;

if (!chk\_const(1, constList)) {

add\_const(1, constList);

}

Constants[StrCounter] = 1;

fprintf(fout, "%d SUB \n", StrCounter);

++StrCounter;

ifjmp = StrCounter;

fprintf(fout, "%d JNEG \n", StrCounter);

++StrCounter;

}

break;

case '>':

{

if (is\_var(\*first\_var)) {

if (!chk\_var(\*first\_var, varList)) {

return 1;

}

Variables[StrCounter] = \*first\_var;

}

else {

if (sscanf(first\_var, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d LOAD \n", StrCounter);

++StrCounter;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (is\_var(\*ptr)) {

if (!chk\_var(\*ptr, varList)) {

return 1;

}

Variables[StrCounter] = \*ptr;

}

else {

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d SUB \n", StrCounter);

++StrCounter;

if (!chk\_const(1, constList)) {

add\_const(1, constList);

}

Constants[StrCounter] = 1;

fprintf(fout, "%d SUB \n", StrCounter);

++StrCounter;

ifjmp = StrCounter;

fprintf(fout, "%d JNEG \n", StrCounter);

++StrCounter;

}

break;

case '=':

{

if (is\_var(\*first\_var)) {

if (!chk\_var(\*first\_var, varList)) {

return 1;

}

Variables[StrCounter] = \*first\_var;

}

else {

if (sscanf(first\_var, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d LOAD \n", StrCounter);

++StrCounter;

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (is\_var(\*ptr)) {

if (!chk\_var(\*ptr, varList)) {

return 1;

}

Variables[StrCounter] = \*ptr;

}

else {

if (sscanf(ptr, "%d", &number\_buff) != 1) {

return 1;

}

if (!chk\_const(number\_buff, constList)) {

add\_const(number\_buff, constList);

}

Constants[StrCounter] = number\_buff;

}

fprintf(fout, "%d SUB \n", StrCounter);

++StrCounter;

fprintf(fout, "%d JZ %d\n", StrCounter, StrCounter + 2);

++StrCounter;

ifjmp = StrCounter;

fprintf(fout, "%d JUMP \n", StrCounter);

++StrCounter;

}

break;

}

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

chk\_command(&Comm, ptr);

switch (Comm) {

case PRINT:

{

if (print(fout, varList, constList, Variables, Constants,

StrCounter)) {

return 1;

}

}

break;

case INPUT:

{

if (input(fout, varList, Variables, StrCounter)) {

return 1;

}

}

break;

case LET:

{

if (let(fout, varList, constList, Constants, Variables,

StrCounter, max\_tmp\_num)) {

return 1;

}

}

break;

case END:

{

fprintf(fout, "%d HALT 00\n", StrCounter);

++StrCounter;

}

break;

case GOTO:

{

if (go\_to(fout, addrList, StrCounter)) {

return 1;

}

}

break;

}

}

break;

}

}

initialization(fout, varList, constList, max\_tmp\_num, StrCounter);

if (StrCounter >= 100) {

return 1;

}

fclose(fin);

rewind(fout);

linking(lnkList, varList, constList, addrList, Variables, Constants, Bas\_str\_nums);

lnkList.sort(lnk\_sort\_pred);

second\_run(fout, lnkList);

fclose(fout);

system("PAUSE");

return 0;

}

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MEMORY\_SIZE 100

int command\_check(char COMMAND);

int sc\_commandEncode(int COMMAND, int OPERAND, int \*VALUE);

int str2command(char \*str);

int pars\_line(char \*str, int \*addr, int \*value);

int asm2bin(char \*filename1, char \*filename2);

#include "Asm2Bin.h"

int command\_check(char COMMAND) {

switch (COMMAND) {

case 10: //READ

break;

case 11: //WRITE

break;

case 20: //LOAD

break;

case 21: //STORE

break;

case 30: //ADD

break;

case 31: //SUB

break;

case 32: //DIVIDE

break;

case 33: //MUL

break;

case 40: // JUMP

break;

case 41: //JNEG

break;

case 42: //JZ

break;

case 43: // HALT

break;

case 57: // JNC

break;

default:

return 1;

break;

}

return 0;

}

int sc\_commandEncode(int COMMAND, int OPERAND, int \*VALUE) {

if (OPERAND >= 128 || OPERAND < 0) {

return 1;

}

if (command\_check(COMMAND)) {

return 1;

}

\*VALUE = COMMAND;

\*VALUE <<= 7;

\*VALUE |= OPERAND & 127;

return 0;

}

int str2command(char \*str) {

int result;

if (strcmp(str, "READ") == 0) {

result = 10;

}

else {

if (strcmp(str, "WRITE") == 0) {

result = 11;

}

else {

if (strcmp(str, "LOAD") == 0) {

result = 20;

}

else {

if (strcmp(str, "STORE") == 0) {

result = 21;

}

else {

if (strcmp(str, "ADD") == 0) {

result = 30;

}

else {

if (strcmp(str, "SUB") == 0) {

result = 31;

}

else {

if (strcmp(str, "DIVIDE") == 0) {

result = 32;

}

else {

if (strcmp(str, "MUL") == 0) {

result = 33;

}

else {

if (strcmp(str, "JUMP") == 0) {

result = 40;

}

else {

if (strcmp(str, "JNEG") == 0) {

result = 41;

}

else {

if (strcmp(str, "JZ") == 0) {

result = 42;

}

else {

if (strcmp(str, "HALT") == 0) {

result = 43;

}

else {

if (strcmp(str, "NOT") == 0) {

result = 51;

}

else {

result = -1;

}

}

}

}

}

}

}

}

}

}

}

}

}

return result;

}

int pars\_line(char \*str, int \*addr, int \*value) {

char \*ptr;

int operand, command;

int flag\_assign = 0;

ptr = strchr(str, ';');

if (ptr != NULL) {

\*ptr = '\0';

}

else {

ptr = strchr(str, '\n');

if (ptr != NULL) {

\*ptr = '\0';

}

}

ptr = strtok(str, " \t");

if (ptr == NULL) {

return 1;

}

if (sscanf(ptr, "%d", addr) != 1) {

return 1;

}

if ((\*addr < 0) || (\*addr >= MEMORY\_SIZE)) {

return 1;

}

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (strcmp(ptr, "=") == 0) {

ptr = strtok(NULL, " \t");

\*value = atoi(ptr);

return 0;

}

else {

command = str2command(ptr);

if (command == -1) {

return 1;

}

}

ptr = strtok(NULL, " \t");

if (ptr == NULL) {

return 1;

}

if (sscanf(ptr, "%d", &operand) != 1) {

return 1;

}

if ((operand < 0) || (operand >= MEMORY\_SIZE)) {

return 1;

}

ptr = strtok(NULL, " \t");

if (ptr != NULL) {

return 1;

}

sc\_commandEncode(command, operand, value);

return 0;

}

int asm2bin(char \*filename1, char \*filename2) {

char Str[100];

int value, addr;

int sc\_memory[MEMORY\_SIZE];

FILE \*fin, \*fout;

if ((fin = fopen(filename1, "r")) == NULL) {

return 1;

}

if ((fout = fopen(filename2, "wb")) == NULL) {

return 1;

}

for (int i = 0; i < MEMORY\_SIZE; ++i) {

sc\_memory[i] = 0;

}

while (fgets(Str, 100, fin)) {

if (pars\_line(Str, &addr, &value) == 0) {

sc\_memory[addr] = value;

}

else {

break;

}

}

fwrite(sc\_memory, 1, MEMORY\_SIZE \* sizeof(int), fout);

fclose(fin);

fclose(fout);

return 0;

}

# Выводы

В ходе выполнения курсовой работы была разработана модель вычислительной машины SimpleComputer. Для управления этой моделью была создана консоль управления, позволяющая изменять значение ячеек памяти и регистров, исполнять программы и т.д.

Для написания программ, которые может выполнять SimpleComputer, был написан транслятор с низкоуровневого языкаSimpleAssembler, переводящий текст языка в бинарный формат. Также написан транслятор с высокоуровневого языка SimpleBasic, переводящий текст языка в текст SimpleAssembler.

Были программно реализованы арифметико-логическое устройство и устройство управления в виде функций ALU(intcommand, intoperand) и CU().

Быланаписанаоднаизпользовательскихфункций – JP, которая выполняет переход к указанной ячейке, если результат последней операции четный.

# Список использованной литературы

1. Организация ЭВМ и систем. Практикум // С.Н. Мамойленко, Новосибирск: ГОУ ВПО «СибГУТИ», 2005 г.
2. Справочник по Си/Си++ [Электронный ресурс]. -http://en.cppreference.com/w/
3. Википедия [Электронный ресурс]. - https://ru.wikipedia.org/