**СОДЕРЖАНИЕ**

[Введение 6](#_Toc474319763)

[1 Технический проект 7](#_Toc474319764)

[1.1 Анализ предметной области 7](#_Toc474319765)

[1.1.1 Общие сведения о виртуальных машинах RISC 7](#_Toc474319766)

[1.1.2 Представление процессора 8](#_Toc474319767)

[1.1.3 Команды RISC 8](#_Toc474319768)

[1.2 Технология обработки информации 9](#_Toc474319769)

[1.2.1 Диаграмма классов 9](#_Toc474319770)

[1.2.2 Формат представления файла 10](#_Toc474319771)

[1.2.3 Алгоритм загрузки программы в память 10](#_Toc474319772)

[1.2.4 Алгоритм работы виртуального процессора 11](#_Toc474319773)

[1.2.5 Алгоритм выполнения команд 11](#_Toc474319774)

[1.3 Входные и выходные данные 13](#_Toc474319775)

[1.4 Системные требования 13](#_Toc474319776)

[2 Рабочий проект 14](#_Toc474319777)

[2.1 Общие сведения о работе системы 14](#_Toc474319778)

[2.2 Функциональное назначение программного продукта 14](#_Toc474319779)

[2.3 Инсталляция и выполнение программного продукта 14](#_Toc474319780)

[2.4 Общий алгоритм программного продукта 14](#_Toc474319781)

[2.5 Разработанные меню и интерфейсы 17](#_Toc474319782)

[2.6 Сообщения системы 17](#_Toc474319783)

[3 Программа и методика испытаний 18](#_Toc474319784)

[Заключение 20](#_Toc474319785)

[Список использованных источников 21](#_Toc474319786)

[Приложение Тестовые программы 22](#_Toc474319787)

# ВВЕДЕНИЕ

Иногда возникает необходимость получить второй компьютер, на котором можно установить другую операционную систему или безопасно протестировать программы. С этой задачей вам поможет справиться виртуальная машина.

Виртуальная машина – программа, которая эмулирует реальный (физический) компьютер со всем его компонентами (жёсткий диск, привод, BIOS, сетевые адаптеры и т.д.). На такой виртуальный компьютер можно установить операционную систему, драйверы, программы и т.д. Таким образом, Вы можете запустить на своем реальном компьютере  еще несколько виртуальных компьютеров, с такой же или другой операционной системой. Вы можете без проблем осуществить обмен данными между Вашим реальным и виртуальным компьютером.

Не каждому пользователя ПК нужна виртуальная машина, но продвинутые пользователи довольно часто используют ее.  Виртуальную машину используют для различных целей и задач:

* Установка второй/другой операционной системы;
* Тестирование программного обеспечения;
* Безопасный запуск подозрительных программ;
* Эмуляция компьютерной сети;
* Запуск приложений, которые нельзя запустить из Вашей операционной системы.

На сегодняшний день виртуальные машины находят широкое применение не только в обучении, но и в практике реального промышленного программирования. Одной из основных проблем в системном программировании является проблема *переносимости* (мобильности) программного обеспечения. Поэтому эта тема актуальна.

Данная программа реализует один из вариантов виртуальной машины, трехадресный, с регистрами общего назначения и тремя операндам в регистрах.

Цель: Разработать виртуальную машину

Назначение: Повышение качества знаний студентов.

# 1 ТЕХНИЧЕСКИЙ ПРОЕКТ

# 1.1 Анализ предметной области

# 1.1.1 Общие сведения о виртуальных машинах RISC

Поскольку виртуальная машина моделирует работу реальной ЭВМ, то она должна содержать те же составляющие, что и реальная ЭВМ.

ЭВМ состоит из связанных между собой элементов: процессоров, модулей памяти, устройств ввода-вывода. Структура ЭВМ показана на рисунке 1.1.

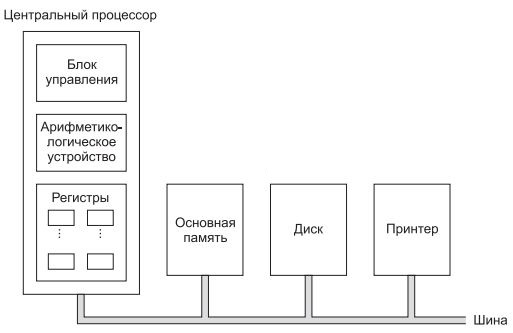


Рисунок 1.1 – Схема компьютера с одним центральным процессором

RISC (reduced instruction set computer — «компьютер с сокращённым [набором команд](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4)») — [архитектура процессора](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0), в котором быстродействие увеличивается за счёт упрощения [инструкций](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), чтобы их декодирование было более простым, а время выполнения — меньшим.

Интерпретатор — это программная модель исполнителя команд абстрактной (виртуальной) машины.

Загрузчик — это программа, выполняющая загрузку массива слов (программ и данных) с внешнего устройства в память компьютера.

Программа для виртуальной машины (вместе с данными) хранится в файле и перед интерпретацией считывается в память. Затем интерпретатор начинает выполнять команды, выбирая их из памяти.

Загружаемый в память массив слов может содержать команды, вещественные числа, целые числа и должен заканчиваться маркером конца.

Массив слов представляет собой текстовый файл, его нужно набрать в текстовом редакторе (например, в Блокноте).

# 1.1.2 Представление процессора

Процессор – центральная часть компьютера, в котором хранится IP, массив памяти, массив регистров, флаги состояния, массив команд.

1. IP — регистр, содержащий адрес-смещение следующей команды, подлежащей исполнению. Находится в процессоре.
2. Массив памяти – массив, содержащий структуры различных типов команд. Всего памяти предоставляется 64 кбайта. Массив разбит на 32-х битные слова, в каждом из которых может содержаться число или команда.
3. Флагов состояния всего 2: Z и N. Они используются при использовании операций сравнения и условных переходах.
4. Массив регистров – массив с целочисленным и вещественным полем. Оба поля используются для хранения соответствующих переменных.

# 1.1.3 Команды RISC

1) Формат представления команд представлена на рисунке 1.1

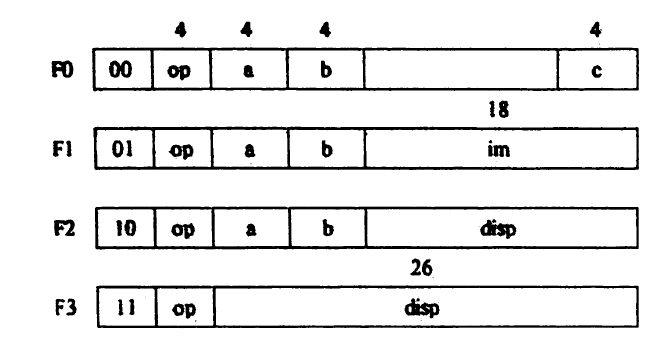


Рисунок 1.1 – Виды команд

2) Первые 6 бит файла – код выполняемой операции. Список кодов операций с их номерами представлен в таблице 1.1. Полный список кодов операций и их реализаций находится в массиве команд.

Таблица 1.1 – Список кодов операций

|  |  |
| --- | --- |
| Тип | Коды операций |
| Арифметика c регистрами | 0 – 9 |
| Сравнение регистров | 10, 11 |
| Арифметика с числами | 16 - 25 |
| Сравнение регистров с числами | 26, 27 |
| Команды перехода | 48 - 58 |
| Команда окончания работы программы(STOP) | 59 |

3) Четырехбитные a, b и c – номера регистров массива регистров. Всего в массиве 16 регистров [R0 – R15]. Счетчик команд по умолчанию содержится в регистре R15. При использовании команд перехода к подпрограммам, неявно используется регистр R14 для хранения адреса возврата.

4) im – 18-битовая константа.

5) disp – смещение. Используется для условного\безусловного перехода от одной команды в массиве памяти к другой.

6) При выполнении команды сравнения регистров изменяются флаги состояния N и Z. Если регистр R[b] = R[c], Z – истинно. Если регистр R[b] < R[c], N – истинно.

7) Соответственно при сравнении регистров с числами. Если регистр R[b] = im, Z – истинно. Если регистр R[b] < im, N – истинно.

# 1.2 Технология обработки информации

# 1.2.1 Диаграмма классов

На рисунке 1.2 представлена диаграмма классов

MOVI

MOVF

…

…

…

Commands

CPU



STOP

Рисунок 1.2 – Диаграмма классов

На диаграмме классов представлены элементы виртуальной машины. Процессор представлен классом CPU. Команды представляют иерархию классов, корнем которого является абстрактный класс Commands. В классе CPU хранится массив указателей на обьект команд. От класса Commands наследуются все команды виртуальной машины.

# 1.2.2 Формат представления файла

1. Первым числом в каждой строке должен быть индекс массива памяти, куда запишется вся информация данной строки
2. Построчно в строке файла может быть задана команда, целое или дробное число.
3. Формат строки: адрес\_тег\_содержание. Адрес – число в десятичной системе счисления.
4. После, через пробел должна следовать буква:

Тег с - если в строке написана команда в десятичной системе счисления.

Тег i - если в строке написано целое число в десятичной системе счисления.

Тег f - если в строке написано вещественное число в десятичной системе счисления.

Тег e - если в строке написан адрес начала работы программы

1. Последняя строка файла: e\_адрес начала программы в десятичной системе счисления.
2. Если в строке команда
   1. После буквы, через пробел должен следовать код типа выполняемой операции;
   2. Далее, через пробел пишется код, выполняемой операции;
   3. После кода операции следует один из видов команды: F0, F1, F2, F3;
   4. Строка команды заполняется согласно рисунку 1.1.
3. Если в строке тег i
   1. Через пробел стоит целое число;
4. Если в строке f
   1. Через пробел стоит вещественное число с фиксированной точкой;
5. Если в строке e

6.1 Через пробел стоит целое число – адрес начала работы программы;

# 1.2.3 Алгоритм загрузки программы в память

Переменная хранения ip;

Переменная хранения индекса записываемой ячейки cell;

Переменная типа записываемой строки elem;

memory[] – массив памяти;

Цикл пока не конец файла;

| Прочитать первое число строки файла в cell;

| Прочитать букву вида строки файла в elem;

| Если elem = i

| | Прочитать число, идущее после буквы в целочисленную ячейку memory[cell];

| Если elem = f

| | Прочитать число, идущее после буквы в вещественную ячейку memory[cell];

| Если elem = c

| | Прочитать команду, идущую после буквы в массив memory[cell];

| Если elem = e

| | Прочитать число, идущее после буквы в ip;

| | Индекс начальной ячейки программы IP = ip;

Конец цикла;

Закрыть файл;

# 1.2.4 Алгоритм работы виртуального процессора

Счетчик команд R[15] = IP;

Цикл пока не команда STOP;

| Выполнить команду, согласно первому числу в ячейке массива memory[R[15]] – код команды;

| Если арифметическая операция, то

| | Вывести на консоль значение регистра, указанного в ячейке memory[R[15]];

| А Если операция сравнения

| | Вывести на консоль значения флагов N и Z;

| R[15]++;

Конец цикла;

# 1.2.5 Алгоритм выполнения команд

Переменная имени файла Name;

Вводится имя файла с исполняемой программой;

Если файл не удалось открыть

| Завершить выполнение программы;

Открыть файл для чтения;

| Вызвать функцию загрузчика;

| Вызвать функцию интерпретатора;

| Удалить memory[];

Конец работы программы;

# 1.2.6 Алгоритмы команд

@ - это операции +/-/\*/:

Арифметика c регистрами - команды типа F0

| R[a] = R[b] @ R[c];

Cравнение регистров - команды типа F0

| Если регистр R[a] = R[c]

| | Z = true;

| Если регистр R[a] < R[c]

| | N = true;

Арифметика c числами - команды типа F1

| R[a] = R[b] @ im;

Cравнение регистров c числами - команды типа F1

| Если регистр R[a] = im

| | Z = true;

| Если регистр R[a] < im

| | N = true;

Команды перехода – команды типа F3

| Если операция BEQ

| | Если Z

| | | R[15]+= disp;

| Если операция BNE

| | Если !Z

| | | R[15]+= disp;

| Если операция BLT

| | Если N

| | | R[15]+= disp;

| Если операция BGE

| | Если !N

| | | R[15]+= disp;

| Если операция BLE

| | Если N || Z

| | | R[15]+= disp;

| Если операция BGT

| | Если !(N || Z)

| | | R[15]+= disp;

| Если операция BR

| | R[15]+= disp;

| Если BSR

| | R[14] = R[15];

| | R[15]+= disp;

| Если RET

| | R[15] = R[14] ;

# 1.3 Входные и выходные данные

В таблице 1.2 представлены входные и выходные данные программы.

Таблица 1.2 – Входные и выходные данные программы

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Имя файла с исполняемой программой | Значение регистров |
|  | Флаги состояния |

# 1.4 Системные требования

Рекомендуемая конфигурация:

1. Intel-совместимый процессор с частотой не менее 1,6 ГГц;
2. не менее 512 МБ ОЗУ;
3. не менее 2 MБ свободного места на диске;
4. дисковод CD-ROM/DVD-ROM.

**2 РАБОЧИЙ ПРОЕКТ**

## 2.1 Общие сведения о работе системы

Программный продукт разработан в среде Code::blocks 13.12 на языке C++. Программа работает под управлением операционной системы Windows XP (x86) Professional (SP3) и более поздними.

## 2.2 Функциональное назначение программного продукта

Разработанный программный продукт предназначен для эмуляции работы процессора. Программа имеет следующие функциональные возможности: Обьем памяти – 65536 байт;

## 2.3 Инсталляция и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера файлы:
   1. core.exe
   2. Тестовые файлы float.txt, int.txt, jmp.txt, loop.txt
2. Запустить core.exe

## 2.4 Общий алгоритм программного продукта

Программный продукт состоит из 5 модулей: «main.cpp», «CPU.h», «CPU.cpp», «Commands.h» и «Commands.cpp».

В таблице 2.1 приведены основные составляющие модуля CPU.h, используемого в программе.

Таблица 2.1 – Составляющие модуля CPU.h

|  |  |  |
| --- | --- | --- |
| **Прототип** | **Содержание** | **Назначение** |
| struct rr | unsigned type : 2;  unsigned op : 4;  unsigned a : 4;  unsigned b : 4;  unsigned : 10;  unsigned c : 4; | Тип команды F0  Код операции  Индекс первого регистра  Индекс второго регистра  Пустые 10 бит ячейки  Индекс третьего регистра |
| struct rn | unsigned type : 2;  unsigned op : 4;  unsigned a : 4;  unsigned b : 4;  unsigned im : 18; | Тип команды F1  Код операции  Индекс первого регистра  Индекс второго регистра  Константа |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| **Прототип** | **Содержание** | **Назначение** |
| struct rd | unsigned type : 2;  unsigned op : 4;  unsigned a : 4;  unsigned b : 4;  unsigned disp : 18; | Тип команды F2  Код операции  Индекс первого регистра  Индекс второго регистра  Смещени |
| struct d | unsigned type : 2;  unsigned op : 4;  unsigned disp : 26; | Тип команды F3  Код операции  Смещение |
| struct gen\_type | unsigned copr : 6; unsigned arg : 26; | Полный код операции  Пустые 26 бит ячейки |
| union type\_com | rr f1;  rn f2;  rd f3;  d f4;  gen\_type f5;  float f;  int I; | Объект структуры rr  Объект структуры rn  Объект структуры rd  Объект структуры d  Объект структуры gen\_type  Ячейка для хранения вещественного числа  Ячейка для хранения целого числа |
| union ROM | float f  int i | Хранение вещественного числа в массиве регистров  Хранение целого числа в массиве регистров |
| class CPU | void interpreter();  void reset();  void set\_IP(int ip); | Метод – интерпретатор. Считывает из памяти команды, выполняет их и выводит на экран состояние регистров,  Инициализирует процессор;  Устанавливает IP |

В таблице 2.2 приведены основные составляющие модуля Command.h, используемого в программе.

Таблица 2.2 – Составляющие модуля Command.h

|  |  |
| --- | --- |
| **Прототип** | **Назначение** |
| virtual void execute(CPU &p1); | Виртуальная функция вызова команды |
| virtual void operator()(CPU &p1); | Вызов функции execute (CPU &p1) |

В таблице 2.2 приведены основные классы модуля Commands.h, используемого в программе.

Таблица 2.2 - Классы модуля Commands.h

|  |  |
| --- | --- |
| **Прототип** | **Назначение** |
| Class MOVI | Присвоение целочисленному регистру R[a] некоторого регистра R[c] |
| Class MOVF | Присвоение вещественному регистру R[a] некоторого регистра R[c] |
| Class ADDI | Сложение двух целых регистров R[b] и R[c], занесение результата в R[a] |
| Class ADDF | Сложение двух дробных регистров R[b] и R[c], результат в R[a] |
| Class SUBI | Разность двух целых регистров R[b] и R[c], занесение результата в R[a] |
| Class SUBF | Разность двух дробных регистров R[b] и R[c], занесение результата в R[a] |
| Class MULI | Умножение двух целых регистров R[b] и R[c], занесение результата в R[a] |
| Class MULF | Умножение двух дробных регистров R[b] и R[c], результат в R[a] |
| Class DIVI | Деление двух целых регистров R[b] и R[c], занесение результата в R[a] |
| Class DIVF | Деление двух дробных регистров R[b] и R[c], занесение результата в R[a] |
| Class CMPI | Сравнение двух целых регистров R[b] и R[c], изменение флагов N и Z |
| Class CMPF | Сравнение двух дробных регистров R[b] и R[c], изменение флагов N и Z |
| Class MOVLI | Присвоение целочисленному регистру R[a] некоторого числа memory[im] |
| Class MOVLF | Присвоение вещественному регистру R[a] некоторого числа memory[im] |
| Class ADDLI | Целочисленное сложение R[b] и memory[im], занесение результата в R[a] |
| Class ADDLF | Вещественное сложение R[b] и memory[im], занесение результата в R[a] |
| Class SUBLI | Целочисленная разность R[b] и memory[im], занесение результата в R[a] |
| Class SUBLF | Вещественная разность R[b] и memory[im], занесение результата в R[a] |
| Class MULLI | Целочисленное произведение R[b] и memory[im], результат в R[a] |
| Class MULLF | Вещественное произведение R[b] и memory[im], результат в R[a] |
| Class DIVLI | Целочисленное деление R[b] и memory[im], занесение результата в R[a] |
| Class DIVLF | Вещественное деление R[b] и memory[im], занесение результата в R[a] |
| Class CMPLI | Целочисленное сравнение R[b] и memory[im], изменение флагов N и Z |
| Class CMPLF | Вешественное сравнение R[b] и memory[im], изменение флагов N и Z |
| Class BEQ | Переход на ячейку с индексом disp, если Z |
| Class BNE | Переход на ячейку с индексом disp, если !Z |
| Class BLT | Переход на ячейку с индексом disp, если N |
| Class BGE | Переход на ячейку с индексом disp, если !N |
| Class BLE | Переход на ячейку с индексом disp, если Z или N |
| Class BGT | Переход на ячейку с индексом disp, если !(Z или N) |
| Class BR | Переход на ячейку с индексом disp |
| Class BSR | Вызов подпрограммы с началом в ячейке с индексом disp |

Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Class RET | Возврат на исходную ячейку после вызова подпрогаммы |
| Class STOP | Конец работы программы |

## 2.5 Разработанные меню и интерфейсы

После запуска программы на выполнение появляется окно в котором необходимо ввести имя файла с текстом программы (рис. 2.2).

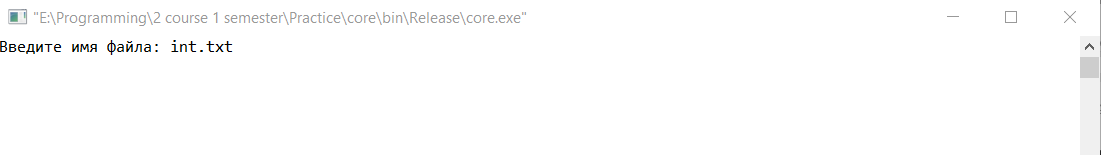


Рисунок 2.2 – Страница с запросом имени файла

После ввода имени файла будет произведено выполнение программы, находящейся в введенном файле (рис. 2.3).



Рисунок 2.3 – Выполнение программы

## 2.6 Сообщения системы

В таблице 2.3 приведены сообщения системы.

Таблица 2.3 – Сообщения системы

|  |  |  |
| --- | --- | --- |
| **№ п\п** | **Сообщение** | **Причина возникновения** |
| 1 | Не удалось открыть файл | Отсутствие введенного файла в каталоге с core.exe или неправильно введенное имя |

# 3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

1. Запустить программу на выполнение. Появится начальное окно с запросом имени исполняемого файла (см. рис. 2.1).
2. Ввести имя файла int.txt
3. Убедиться, что программа выдала результат, показанный на рисунке 3.1



Рисунок 3.1 – Правильная работа программы, для первого алгоритма

1. Ввести имя файла float.txt
2. Убедиться, что программа выдала результат, показанный на рисунке 3.2



Рисунок 3.2 - Правильная работа программы, для второго алгоритма

1. Ввести имя файла jmp.txt
2. Убедиться, что программа выдала результат, показанный на рисунке 3.3



Рисунок 3.3 - Правильная работа программы, для третьего алгоритма

1. Ввести имя файла loop.txt
2. Убедиться, что программа выдала результат, показанный на рисунке 3.4



Рисунок 3.4 - Правильная работа программы, для четвертого алгоритма

# ЗАКЛЮЧЕНИЕ

В результате учебной практики была разработана виртуальная машина RISC процессоров.

Программа отвечает поставленным требованиям и может быть использована для эмулирования работы RISC процессоров.

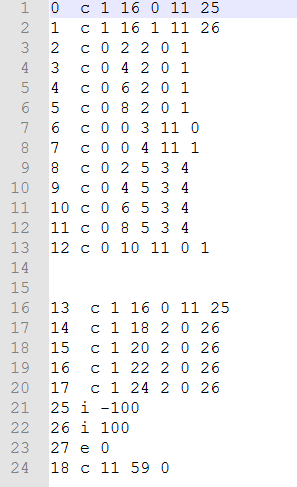
# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лаптев В. В. С++ Объектно-ориентированное программирование: Учебное пособие. — СПб.: Питер, 2008. —464 с: ил.
2. Вирт Н. Построение компиляторов. М.: ДМК Пресс, 2010. - 192 с.
3. Готтшлинг П. Современный С++ для программистов, инженеров и ученых. – М.: ООО “И.Д. Вильямс”, 2016 г. – 512 с.

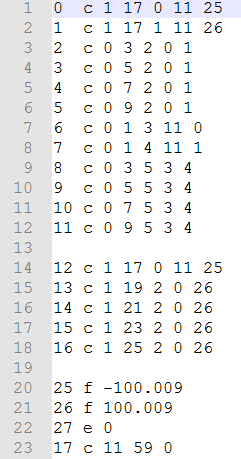
# ПРИЛОЖЕНИЕ

**Тестовые программы**

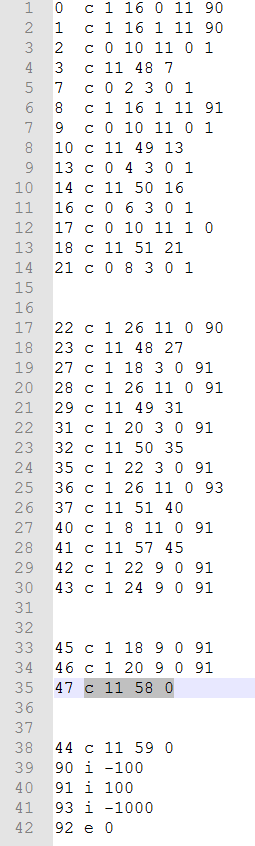
Файл int.txt. Проверка целочисленной арифметики.



Файл float.txt. Проверка вещественной арифметики.



Файл jmp.txt. Проверка команд перехода.



Файл loop.txt. Проверка циклов.

