Федеральное агентство по рыболовству
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Астраханский государственный технический университет»
Система менеджмента качества в области образования, воспитания, науки и инноваций сертифицирована DQS по международному стандарту ISO 9001:2015

Институт информационных технологий и коммуникаций
Направление подготовки 09.03.04 Программная инженерия
Профиль «Разработка программно-информационных систем»
Кафедра «Автоматизированные системы обработки информации и управления»

КУРСОВОЙ ПРОЕКТ

Реализация виртуальной машины VM10

по дисциплине «Архитектура вычислительных систем, операционные системы»

Допущен к защите
«__» _____20__г.
Руководитель
_____

Проект выполнен
обучающимся группы ДИПРб-21
Катуниным С.С.
_____

Оценка, полученная на защите
«_____»

Руководитель
доцент Лаптев В.В.

Члены комиссии:

_____ Лаптев В.В.

_____

_____

Астрахань – 2019

ФЕДЕРАЛЬНОЕ АГЕНТСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ ПО РЫБОЛОВСТВУ

АСТРАХАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УТВЕРЖДАЮ
Заведующий кафедрой
к.т.н., доцент
С.В. Белов_____
«____»_____201 __ г.

Кафедра
«Автоматизированные системы
обработки информации и управления»

ЗАДАНИЕ

на выполнение курсового проекта

Обучающийся ***Катунин Сергей Сергеевич***

Группа ***ДИПРб-21***

Дисциплина ***Архитектура вычислительных систем, операционные системы***

Тема ***Реализация виртуальной машины VM10***

Дата получения задания «_____»_____201___г.

Срок представления обучающимся КП на кафедру «_____»_____201___г.

Руководитель ***доцент***_____ ***Лаптев В.В.*** «_____»_____201___г.

должность, степень, звание подпись ФИО

Обучающийся _____ ***Катунин С.С.*** «_____»_____201___г.

подпись ФИО

Задачи

Разработка программного продукта, который

- загружает исходный код исполняемой программы из текстового файла в память
- интерпретирует и исполняет команды, которые были загружены в память

Список рекомендуемой литературы

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. – СПб.: Питер, 2013.- 816 с.: ил.

2. Assembler / В. Юров – СПб.: Питер, 2002. – 624 с.: ил.

3. Лаптев В. В. C++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия «Учебное пособие»).

УТВЕРЖДАЮ
Заведующий кафедрой

к.т.н., доцент

С.В. Белов_____

«____»_____20 ___г.

К заданию на курсовой проект
по дисциплине
«Архитектура вычислительных систем,
операционные системы»

КАЛЕНДАРНЫЙ ГРАФИК
курсового проектирования

№ п/п	Разделы, темы и их содержание, графический материал	Дата сдачи	Объем, %
1	Выбор темы	15.09.2019	1
2	Техническое задание	30.09.2019	3
3	Разработка модели, проектирование системы ▪ *введение,* ▪ *технический проект,* ▪ *программа и методика испытаний,* ▪ *литература*	31.10.2019	25
4	Программная реализация системы ▪ *работающая программа,* ▪ *рабочий проект* ▪ *скорректированное техническое задание (при необходимости)*	30.11.2019	40
5	Тестирование и отладка системы, эксперименты ▪ *работающая программа с внесёнными изменениями,* ▪ *окончательные тексты всех разделов*	15.12.2019	50
6	Компоновка текста Подготовка презентации и доклада ▪ *пояснительная записка* ▪ *презентация* ▪ *электронный носитель с текстом пояснительной записки, исходным кодом проекта, презентацией и готовым программным продуктом*	20.12.2019	59
7	Защита курсового проекта	27.12.2019	60-100

С графиком ознакомлен «____» _____ 20 ___г.

Катунин С.С. _____, обучающийся группы ДИПРб-21

(фамилия, инициалы, подпись)

График курсового проектирования выполнен
без отклонений / с незначительными отклонениями / со значительными отклонениями

нужное подчеркнуть

Руководитель курсового проекта_____ доцент Лаптев В. В.
подпись, ученая степень, звание, фамилия, инициалы

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Виртуальная машина представляет собой эмуляцию компьютерной системы. Виртуальные машины могут поддерживать отдельные процессы или систему в целом — в зависимости от уровня абстракции, на котором происходит виртуализация. Виртуализация стала важным инструментом разработки компьютерных систем, а виртуальные машины используются в самых разных областях — от операционных систем до языков программирования и архитектуры процессоров. Освобождая разработчиков и пользователей от ресурсных ограничений и недостатков интерфейса, виртуальные машины снижают уязвимость системы, повышают интероперабельность программного обеспечения и эксплуатационную гибкость аппаратной платформы. Ярчайшими примерами использования виртуальных машин в программировании являются интерпретаторы языков Java, Python, C# (как и вся архитектура .Net), а также фреймворк LLVM для создания компиляторов и сопутствующих утилит (например, компилятор Clang для языка C++ и не только).

Концепция виртуальной машины как совокупности ресурсов, которые симулируют поведение реальной машины, появилась в Кембридже в конце 1960-х годов в виде расширения концепции виртуальной памяти манчестерской вычислительной машины Atlas.

Виртуальные машины могут использоваться для защиты информации и ограничения возможностей программ (песочница); исследования производительности ПО или новой компьютерной архитектуры; эмуляции различных архитектур (например, эмуляторы игровых приставок); изолированный запуск и исследование вредоносного кода, который не затрагивает основную систему; упрощения управления кластерами — виртуальные машины могут просто мигрировать с одной физической машины на другую во время работы; тестирования и отладки системного программного обеспечения; создания резервных копий операционных систем и запуска программного обеспечения и приложений на операционных системах, для которых они изначально не предназначены, и т. д.

Целью курсового проекта является разработка и реализация интерпретатора виртуальной машины с заданной архитектурой.

Назначение программы – исполнение программ, написанных в соответствии с системой команд виртуальной машины.

1 ТЕХНИЧЕСКИЙ ПРОЕКТ

1.1 Анализ предметной области

1.1.1 Виртуальная машина

Виртуальная машина – программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы и исполняющая программы для этой платформы на host-платформе (host — хост-платформа, платформа-хозяин). Виртуальный компьютер может иметь помимо процессора свою виртуальную оперативную память, жесткий диск и другие периферийные устройства Виртуальная машина исполняет некоторый машинно-независимый код (например, байт-код, шитый код, p-код) или машинный код реального процессора.

1.1.2 Память

Память состоит из слов по 32 бита. Размер адреса = 16 бит. Размер памяти соответственно равно 2^{16} = 65536.

1.1.3 Процессор

Процессор - главная часть компьютера, исполняющая машинные инструкции (код программ). Процессор в данной архитектуре включает в себя: регистр PSW (Program State Word), блок команд, 16 регистров общего назначения и память.

Регистр PSW занимает 32 бита и состоит из флагов, занимающих 16 бит и 16-битного IP (instruction pointer) – регистра, содержащего адрес следующей команды, подлежащей исполнению. Регистр состояния программы описан в таблице 1.1.

Таблица 1.1 — Описание регистра PSW

Название	**Размер (в битах)**	**Описание**
IP	16	Содержит адрес следующей выполняемой команды в оперативной памяти
FLAGS	16	Регистр флагов-результатов, используемых в командах переходов

Регистр флагов в свою очередь имеет собственную структуру, описанную в таблице 1.2.

Таблица 1.2 — Описание регистра флагов

№ Бита	Обозначение	Название	Описание
0	ZF	Zero Flag	Флаг нуля (принимает значение 1, если число равно нулю, 0 — в противном случае)
1	SF	Sign Flag	Флаг знака (принимает значение 1, если число отрицательное, 0 — в противном случае)
2...15	reserved	Reserved	Резервные биты

1.1.4 Типы данных

В памяти хранятся целые, беззнаковые и дробные типы данных с размером 32 бита. Типы данных, соответствующие заданной архитектуре компьютера подробно описаны в таблице 1.3.

Таблица 1.3 — Описание реализованных типов данных

Название	Размер (в битах)	Описание
Целые знаковые	32	Двоичное значение со знаком. Знак в этом двоичном числе содержится в 31 бите. Ноль в этом бите соответствует положительному числу, а единица — отрицательному. Отрицательные числа представляются в дополнительном коде. Числовой диапазон для этого типа данных следующий: Числовой диапазон для этого типа следующий: от -2^{31} до $+2^{31}-1$. Используется для целочисленных арифметических вычислений
Целые беззнаковые	32	Двоичное значение без знака. Числовой диапазон для этого типа следующий: от 0 до $2^{32}-1$. Используется для целочисленных арифметических вычислений, а так же в операциях связанных с памятью и адресами
Дробные	32	Формат представления чисел с плавающей точкой соответствует стандарту IEEE 754. Этот тип данных используется для дробных арифметических вычислений

1.1.5 Формат команд

Команды делятся на короткие и длинные. На рисунке 1.1. проиллюстрирован формат команд.

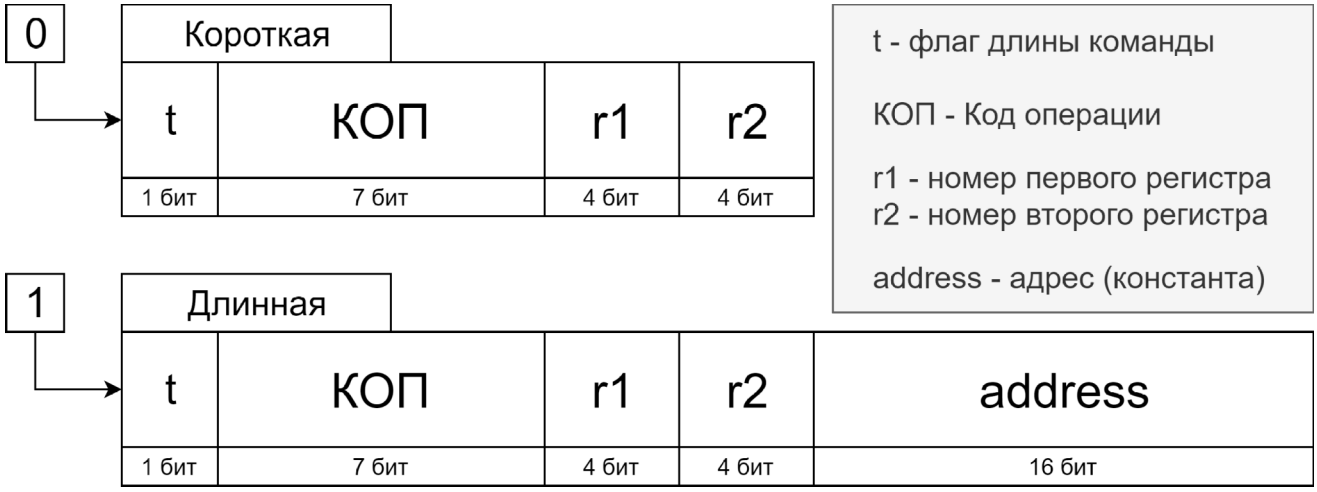

Рисунок 1.1 – Формат команд

1.1.6 Режимы адресации операндов

Реализованные режимы адресаций:

- Непосредственная адресация – операнд (целое со знаком или беззнаковое число) находится в поле команды address.
- Регистровая адресация – операнд находится в одном из регистров.
- Косвенно-регистровая адресация – адрес операнда находится в одном из регистров.
- Прямая адресация – адрес операнда содержится в поле команды address.
- Базовая адресация – адрес операнда равен сумме содержимого первого регистра (в некоторых операциях) и смещения (16-разрядного) в поле команды address.

1.1.7 Описание команд

Реализованные команды можно разделить на 6 групп:

1. Особые команды (например пустая команда-заглушка nop).
2. Команды пересылки данных.
3. Команды целой арифметики.
4. Команды дробной арифметики.
5. Команды ввода и вывода.
6. Команды перехода.

Реализована как двухадресная, так и трехадресная арифметика. Результат помещается в регистр r2.

В таблице 1.4 описаны особые команды, такие как команда остановки, которая не указывается явно программистом в программе, и команда заглушка, необходимая для записи одной 16-битной команды в коде программы.

Таблица 1.4 — Описание особых команд

t	КОП	Название	Размер (в битах)	Описание
1	0	Стоп	32	Команда остановки
0	1	Не операция	16	Пустая команда-заглушка

В таблице 1.5 приведены все команды пересылок данных, которые обеспечивают обмен данных между регистрами, загрузку и сохранения данных в память.

Таблица 1.5 — Описание команд пересылки данных

t	КОП	Название	Размер (в битах)	Описание
0	2	Обмен	16	Обменять содержимое регистров
0	3	Переместить	16	Пересылка из регистра [r1] в регистр [r2]
1	4	Загрузить	32	Пересылка память — регистр [r2]. Значение в регистре [r1] является режимом загрузки. При [r1] = 0 – пересылка: значение из памяти в регистр [r2]; при [r1] = 1 – непосредственная загрузка целочисленного значения со знаком (в этом режиме выполняется расширение знака) вместо адреса в регистр [r2]; при [r1] = 2 – непосредственная загрузка константы-адреса (беззнаковое число) в регистр [r2]
0	5	Сохранить	16	Пересылка регистр [r1] – память. Короткий вариант команды. Число из [r1] сохраняется по адресу, лежащему в регистре [r2]
1	5	Сохранить	32	Пересылка регистр [r1] – память. Длинный вариант команды. Число из [r1] сохраняется по адресу в команде (поле address), а регистр [r2] не используется

Продолжение таблицы 1.5

t	КОП	Название	Размер (в битах)	Описание
0	6	Косвенный доступ	16	Косвенный доступ к данным по адресу в регистре [r1]. Короткий вариант команды. Полученное значение сохраняется в регистр [r2]
1	6	Косвенный доступ	32	Косвенный доступ к данным по адресу в регистре [r1]. Длинный вариант команды. Полученное значение сохраняется по адресу в команде (поле address), а регистр [r2] не используется

В таблице 1.6 описаны все команды арифметики и команда ввода-вывода, с различными режимами работы (для ввода / вывода целых, дробных и беззнаковых чисел).

Таблица 1.6 — Описание команд арифметики и команды ввода-вывода.

t	КОП	Название	Размер (в битах)	Описание
0	7 / 11	Целое / дробное сложение	16	iAdd – целое сложение, fAdd – дробное. Короткий вариант команды (двухадресная арифметика). Суммируются значения из регистров [r1] + [r2], а результат помещается в регистр [r2]
1	7 / 11	Целое / дробное сложение	32	iAdd – целое сложение, fAdd – дробное. Длинный вариант команды (трехадресная арифметика). Регистр [r1] является базовым (в нем хранится адрес), а адрес операнда (числа, значения) составляется из суммы адреса базового регистра и смещения в команде (поле address), второй же операнд суммы находится в [r2]. Результат помещается в [r2]

Продолжение таблицы 1.6

t	КОП	Название	Размер (в битах)	Описание
0 / 1	8 / 12	Целое / дробное вычитание	16 / 32	Целое и дробное вычитание. Двухадресная и трехадресная арифметика для остальных арифметических операций работает по схожему принципу со сложением.
0 / 1	9 / 13	Целое / дробное умножение	16 / 32	Целое и дробное умножение
0 / 1	10 / 14	Целое / дробное деление	16 / 32	Целое и дробное деление
0	15	Ввод/вывод	16	Операции ввода и вывода. Значение в регистре [r2] команды является режимом ввода и вывода. Существуют следующие режимы: 0 – ввод целого числа, 1 — ввод беззнакового, 2 – ввод дробного, 3 – вывод целого, 4 – вывод беззнакового, 5 – вывод дробного числа. Короткий вариант команды. Происходит ввод-вывод в / из регистра [r1].
1	15	Ввод/вывод	32	Длинный вариант команды ввода-вывода. Ввод-вывод осуществляется по адресу в команде (поле address), а регистр [r1] не используется.

В таблице 1.7 описаны все команды переходов, с различными режимами работы (прямой переход, относительный переход вперед или назад).

Таблица 1.7 — Описание команд переходов.

t	КОП	Название	Размер (в битах)	Описание
0	40	Перейти	16	Команда безусловного перехода. Короткий вариант команды. Осуществляется косвенный переход (по адресу, указанному в регистре [r2]).

Продолжение таблицы 1.7

t	КОП	Название	Размер (в битах)	Описание
1	40	Перейти	32	Команда безусловного перехода. Длинный вариант команды. Значение в регистре [r1] команды является режимом перехода. При [r1] = 0 – прямой переход, при [r1] = 1 – относительный переход вперед, при [r1] = 2 – относительный переход назад
0 / 1	41	Перейти если zf	16 / 32	Команда условного перехода (в случае если флаг ZF – нулевого результата = 1). Режимы переходов в этой команде перехода и в последующих (кроме команды ret) реализованы так же, как и в команде jmp. Флаг ZF = 1, когда число равно нулю
0 / 1	42	Перейти если не zf	16 / 32	Команда условного перехода (в случае если флаг ZF – нулевого результата = 0). Флаг ZF = 0, когда число не равно нулю
0 / 1	43	Перейти если sf	16 / 32	Команда условного перехода (в случае если флаг SF – знак результата = 1). Флаг SF = 1, когда число отрицательно
0 / 1	44	Перейти если не sf	16 / 32	Команда условного перехода (в случае если флаг SF – знак результата = 0). Флаг SF = 0, когда число положительно или равно нулю
0 / 1	45	Вызов подпрограммы	16 / 32	Вызов подпрограммы (запись текущего адреса — адреса выхода в регистр [r15] и безусловный переход по нужному адресу самой подпрограммы)
0 / 1	46	Возврат из подпрограммы	16 / 32	Возврат из подпрограммы (команда устанавливает адрес для IP из регистра [r15]). Короткий и длинный варианты команды не отличаются действием

1.2 Технология обработки информации

Анализ предметной области показал, что программа рассчитана на одного пользователя. Пользователь имеет возможность запустить программу на выполнение, передав в качестве параметра командной строки (консоли). В ходе анализа предметной области была построена диаграмма вариантов использования (рис. 1.2).

Рисунок 1.2 – Диаграмма вариантов использования

Вариант использования «Запустить программу на выполнение» включает обязательную функцию «Передать имя файла в качестве параметра».

1.2.1 Диаграмма классов

В ходе анализа предметной области была разработана диаграмма классов. Она представлена в *Приложении 2.*

Главным классом данной виртуальной машины является CPU (процессор), он включает в себя:

- массив из 128 указателей на класс Command (команда);
- булеву переменную, обозначающую, был ли осуществлен переход;
- объект класса cmd (текущая выполняемая команда);
- объект класса PSW – регистр PSW;
- объект класса Memory – память
- 16 объектов класса data_t – регистры общего назначения.

Методы:

- получить бит t из команды;
- получить номер 1-ого или 2-ого регистра из команды;
- получить адрес из команды;
- загрузить команду по адресу в IP;
- обнулить счётчик IP и флаги;
- запустить процессор.

Класс Memory представляет собой динамический массив, состоящий из 65536 элементов типа uint32_t (слово).

Класс Memory имеет следующие методы:

- загрузить слово в память по адресу;
- получить слово из памяти по адресу.

Класс Command является абстрактным базовым классом для всех команд, в нём объявлен виртуальный метод — функтор (оператор круглых скобок), ниже перечислены классы-наследники (команды), которые переопределяют этот метод:

- команда заглушка (nop);
- команда загрузки данных в регистр из памяти (load);
- команда обмена данных между регистрами (exchange);
- команда косвенного доступа к данным (dereference_ptr);
- команда ввода-вывода (io);
- команда сохранения данных в память (save);
- команда перемещения данных между регистрами (move);
- команды целой и дробной арифметики (iMath, fMath), наследуемые от них команды:
  - сложения (iAdd и fAdd);
  - вычитания (iSub и fSub);
  - умножения (iMul и fMul);
  - деления (iDiv и fDiv).
- команда перехода (Jump) и наследуемые от него команды:
  - безусловный переход (jmp);
  - условные переходы с проверкой флага нулевого результата (jzf и jnzf);
  - условные переходы с проверкой флага знака результата (jsf и jnsf);
  - команда вызова подпрограммы (call);
  - команда возврата из подпрограммы (ret);

1.2.2 Формат исполняемого файла

В ходе создания виртуальной машины был разработан формат исполняемого файла. Исполняемый файл — это текстовый файл, содержащий команды, которые процессор должен выполнить. Согласно формату, на каждой строке файла может находится только одна команда.

Команды имеют следующий вид: **c** t cop r1 r2 address.

c – объявление команды,

t – бит длины команды, 1 – в случае длинной команды и 0 – в случае короткой,

cop – код операции команды,

r1, r2 – номера регистров (от 0 до 15),

address – указывается в длинной команде и отсутствует в короткой, это адрес, используемый в качестве операнда или смещения, в зависимости от кода операции.

Запись происходит, как правило, во второй операнд или третий (например, при записи в память). Помимо объявления команд в файле могут быть следующие директивы:

a address – установить текущий адрес, используемый для записи в память на address

i value – загрузить в память целочисленное значение value по текущему адресу, используемому для записи в память

u value – загрузить в память беззнаковое число value

f value – загрузить в память число с плавающей точкой value

e address – загрузить в память команду завершения работы программы и установить адрес IP (адрес первой исполняемой команды) на address

1.2.3 Алгоритм работы загрузчика

Дано: ***filename –*** название файла.

открыть текстовый исполняемый файл с названием ***filename***

если файл не удалось открыть, то вернуть ***ложь***

uint16_t ***address*** = 0

char ***symbol*** = ‘ ‘

пока ***symbol*** != ‘**e**’

| получить строку s из файла

| прочитать ***symbol*** из строки

| если ***symbol*** = ‘***a***’ то прочитать ***address*** из строки

| если ***symbol*** = ‘***i***’ или ‘***u***’ то

| | прочитать беззнаковое число из строки

| | записать число в память по адресу ***address***

| | увеличить ***address*** на 1

| конец ветвления

| если ***symbol*** = ‘***f***’ то

| | прочитать число с плавающей точкой из строки

| | записать число с плавающей точкой в память по адресу ***address***

| | увеличить ***address*** на 1

| конец ветвления

| если ***symbol*** = ‘***c***’ то

| | прочитать бит длины знака ***t***

| | если ***t*** = 0 то

| | | прочитать короткую команду: прочитать ***кол***, ***r1***, ***r2***

| | | прочитать следующую строку ***s***

| | | прочитать вторую короткую команду

| | иначе
| | | прочитать длинную команду: прочитать ***кол***, ***r1***, ***r2***, ***адрес в команде***
| | конец ветвления
| | записать длинную команду (или две коротких) в память по адресу ***address***
| | увеличить ***address*** на 1
| конец ветвления
| если ***symbol*** = ‘***e***’ то
| | записать команду остановки в память по адресу ***address***
| | прочитать ***IP***
| | установить ***IP*** для процессора
| конец ветвления
конец цикла
закрыть файл ***filename***
вернуть ***истину***

1.2.4 Алгоритм работы процессора

Дано: ***cmd*** – регистр в процессоре, который хранит текущую выполняемую длинную команду (или две коротких), ***was_transition*** – переменная-флаг, обозначающая была ли осуществлена команда перехода.

загрузить слово по адресу из регистра ***IP*** и представить слово как команду ***cmd***
прочитать ***КОП*** у ***cmd***
пока ***КОП*** != “стоп“
| выполнить команду с кодом операции ***КОП***
| если команда в текущем слове короткая то выполнить вторую команду из слова
| если ***was_transition*** = ***ложь*** то увеличить ***IP*** на 1
| иначе ***was_transition*** = ***ложь***
| конец ветвления
| загрузить следующее слово и представить его как команду ***cmd***
конец цикла

1.2.5 Алгоритм работы команды exchange (обмен между регистрами)

получить номера регистров ***r1*** и ***r2***
обменять содержимое регистров ***r1*** и ***r2***

1.2.6 Алгоритм работы команды move (пересылка регистр-регистр)

получить номера регистров ***r1*** и ***r2***
поместить в регистр ***r2*** содержимое регистра ***r1***

1.2.7 Алгоритм работы команды load (пересылка память-регистр)

прочитать номер регистра ***r1*** из команды как режим загрузки

прочитать номер регистра ***r2*** и число ***address*** из команды

если ***r1*** = 0 то поместить в ***r2*** число по адресу ***address*** из памяти

а если ***r1*** = 1 то поместить в ***r2*** число ***address,*** представив его как число со знаком

а если ***r1*** = 2 то поместить в ***r2*** число ***address,*** представив его как число без знака

конец ветвления

1.2.8 Алгоритм работы команды save (пересылка регистр-память)

получить бит длины команды ***t***

получить номера регистров ***r1*** и ***r2***

если ***t*** = 0 то получить адрес ***address*** из регистра ***r2***

иначе получить адрес ***address*** из команды

конец ветвления

сохранить содержимое регистра ***r1*** по адресу ***address***

1.2.9 Алгоритм работы команды dereference_ptr (косвенная доступ по адресу)

получить бит длины команды ***t***

получить номера регистров ***r1*** и ***r2,*** где ***r1*** – регистр с адресом

получить число (значение) ***data*** из памяти по адресу из ***r1***

если ***t*** = 0 то сохранить значение ***data*** в регистр ***r2***

иначе сохранить значение ***data*** в память по адресу ***address*** из команды

конец ветвления

1.2.10 Алгоритм установки флагов для команд целой и дробной арифметики

получить номер второго регистра ***r2***

если содержимое ***r2*** = 0, то установить флаг ***ZF*** = 1

иначе установить флаг ***ZF*** = 0

конец ветвления

если содержимое ***r2*** < 0, то установить флаг ***SF*** = 1

иначе установить флаг ***SF*** = 0

конец ветвления

1.2.11 Алгоритм команд целой и дробной арифметики

получить бит длины команды ***t***

получить номера регистров ***r1*** и ***r2***

если ***t*** = 0 то

| выполнить арифметическую операцию между содержимым регистров ***r1*** и ***r2***

| поместить результат в регистр ***r2***

иначе

| получить число из регистра ***r1*** и представить его как базовый адрес ***base***

| получить число ***offset*** из поля адреса команды

| uint16_t ***address*** = ***base*** + ***offset***

| получить значение ***value*** по адресу ***address*** из памяти

| выполнить арифметическую операцию между ***value*** и содержимым регистра ***r2***

конец ветвления

вызвать алгоритм установки флагов для команд целой и дробной арифметики 1.2.9

1.2.12 Алгоритм команды ввода и вывода

получить бит длины команды ***t***

получить номера регистров ***r1*** и ***r2,*** где ***r2 –*** режим ввода и вывода

получить адрес ***address,*** находящийся в команде

если ***r2*** = 0 то

| если ***t*** = 0 то осуществить ввод числа со знаком с клавиатуры в регистр ***r1***

| иначе ввод числа со знаком и сохранение числа в память по адресу ***address***

| конец ветвления

а если ***r2*** = 1 то

| если ***t*** = 0 то осуществить ввод числа без знака с клавиатуры в регистр ***r1***

| иначе ввод числа без знака и сохранение числа в память по адресу ***address***

| конец ветвления

а если ***r2*** = 2 то

| если ***t*** = 0 то осуществить ввод дробного числа с клавиатуры в регистр ***r1***

| иначе ввод дробного числа и сохранение числа в память по адресу ***address***

| конец ветвления

а если ***r2*** = 3 то

| если ***t*** = 0 то осуществить вывод числа со знаком на экран

| иначе вывод числа со знаком по адресу ***address*** из памяти

| конец ветвления

а если ***r2*** = 4 то

| если ***t*** = 0 то осуществить вывод числа без знака на экран

| иначе вывод числа без знака по адресу ***address*** из памяти

| конец ветвления

а если ***r2*** = 5 то

| если ***t*** = 0 то осуществить вывод дробного числа на экран

| иначе вывод дробного числа по адресу ***address*** из памяти

| конец ветвления

конец ветвления

1.2.13 Алгоритм команды безусловного перехода

Дано: ***was_transition*** – переменная-флаг, обозначающая была ли осуществлена команда перехода.

получить бит длины команды ***t***

если ***t*** = 0 то

| получить номер регистра ***r2***

| получить адрес ***address*** из регистра ***r2***

иначе

| получить номер регистра ***r1*** и представить его как режим перехода

| получить адрес ***cmd_address*** из поля адреса команды

| если ***r1*** = 0 то ***address = cmd_address***

| а если ***r1*** = 1 то ***address*** = ***IP*** + ***cmd_address***

| а если ***r1*** = 2 то ***address*** = ***IP – cmd_address***

конец ветвления

IP = ***address***

установить флаг ***was_transition*** на ***истина***

1.2.14 Алгоритм команд условных переходов

проверить флаг (в зависимости от команды: флаг ZF или SF, проверяется на 0 или на 1)

если результат проверки положителен то

| вызвать алгоритм команды безусловного перехода 1.2.12

конец ветвления

1.2.15 Алгоритм команды вызова подпрограммы

сохранить адрес ***IP*** в регистре №15

вызвать алгоритм команды безусловного перехода 1.2.12

1.2.16 Алгоритм команды возврата из подпрограммы

установить для IP адрес, который хранится в регистре №15

1.2.17 Основной алгоритм

Дано: ***argv —*** количество параметров командой строки, ***filename*** – второй параметр командой строки — название исполняемого файла.

если ***argv*** > 1

| вызвать алгоритм загрузчика 1.2.2

| если файл был загружен то
| | вывести сообщение о том, что файл filename был загружен успешно
| | вызвать алгоритм работы процессора 1.2.3
| конец ветвления
иначе вывести сообщение о том, что программа была запущена без параметров командой строки
конец ветвления

1.3 Входные и выходные данные

Входные данные: имя исполняемого файла.

Выходные данные: сообщения системы и вывод.

1.4 Системные требования

Рекомендуемая конфигурация:

- Intel-совместимый процессор с частотой не менее 1,6 ГГц;
- не менее 512 МБ ОЗУ;
- не менее 5 МБ свободного места на диске;
- дисковод CD-ROM/DVD-ROM

Операционная система: Windows XP (x86) с пакетом обновления 3 (SP3) или более поздние. Среда разработки – интегрированная среда Qt Creator (версия 4.11.0), язык C++ (стандарт C++ 11 и более поздние).

2 РАБОЧИЙ ПРОЕКТ

2.1 Общие сведения о работе системы

Программный продукт разработан в интегрированной среде Qt Creator (версия 4.11.0) на языке C++ (стандарт C++ 11 и более поздние), с использованием компилятора MinGW GCC (версия 7.3.0). 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 выполнения программ, написанных в соответствии с системой команд заданной архитектуры виртуальной машины. Программа имеет следующие функциональные возможности:

- выполнение программы из исходного текстового файла;
- предоставление пользователю результатов работы программы.

Программа имеет следующие функциональные ограничения:

- можно передавать только одно названия файла в качестве параметра;
- программа не выдает и не обрабатывает исключения при возникновении ошибок.

2.3 Инсталляция и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера папку EVM_coursework, содержащую интерпретатор EVM_coursework.exe, набор исходных текстов исполняемых программ (test_call_ret.txt, test_conditional_goto_sf.txt, test_conditional_goto_zf.txt, test_dereference_ptr.txt, test_float_arif.txt, test_float_arif_trehadress.txt, test_goto.txt, test_int_arif.txt, test_int_arif_trehadress.txt, test_io.txt, test_perecilki.txt, test_uint_arif.txt), и папку sourse с исходными текстами программы.
2. Запустить файл EVM_coursework.exe из командой строки.
3. Для проверки работоспособности программы, передать имя одного из перечисленных в пункте 1 файлов.

2.4 Описание программы

Программа состоит из 28 классов – 25 из них реализуют команды виртуальной машины, 1 класс – для описания процессора, 1 класс – класс „память“, 1 класс – абстрактный класс „команда“.

В таблице 2.1 приведено описание вложенной структуры для 16-битной команды cmd16_struct объединения cmd (заголовочный файл types.h).

Таблица 2.1 – Описание структуры cmd16_struct

Поле	Тип	Назначение
t	uint8_t	Бит, определяющий длину команды
cop	uint8_t	Код операции
r1	uint8_t	Номер первого операнда-регистра
r2	uint8_t	Номер второго регистра

В таблице 2.2 приведено описание вложенной структуры для 32-битной команды cmd32_struct объединения cmd (заголовочный файл types.h).

Таблица 2.2 – Описание структуры cmd32_struct

Поле	Тип	Назначение
t	uint8_t	Бит, определяющий длину команды
cop	uint8_t	Код операции
r1	uint8_t	Номер первого операнда-регистра
r2	uint8_t	Номер второго регистра
address	uint16_t	Адрес, константа, смещение

В таблице 2.3 приведено описание объединения cmd для команды в одном слове (либо одна 32-битная длинная, либо две коротких 16-битных команды) (заголовочный файл types.h).

Таблица 2.3 – Описание объединения cmd

Поле	Тип	Назначение
c16[2]	uint8_t	Массив из двух коротких команд
c32	uint8_t	Длинная команда

В таблице 2.4 приведено описание объединения data_t, используемого в качестве 32-битного типа данных для регистров общего назначения (заголовочный файл types.h).

Таблица 2.4 – Описание объединения data_t

Поле	Тип	Назначение
i	int32_t	Число со знаком
u	uint32_t	Беззнаковое число
f	float	Число с плавающей точкой

В таблице 2.5 приведено описание объединения word, используемого в качестве 32-битного слова, загружаемого в память (заголовочный файл types.h).

Таблица 2.5 – Описание объединения word

Поле	Тип	Назначение
data	data_t	Данные в одном слове
c	cmd	Длинная команда (или две коротких) в одном слове

В таблице 2.6 приведено описание класса Memory, используемое в качестве поля класса CPU (композиция) (заголовочный файл memory.h).

Таблица 2.6 – Описание класса Memory

Поле	Тип	Назначение
memory	uint32_t*	Динамический массив для памяти
Метод		**Описание**
Memory()		Конструктор для класса, который выделяет память для 2^{16} элементов массива
~Memory()		Деструктор, который освобождает выделенную память
void push (word word, uint16_t address)		Загрузить слово в память по адресу
word get_word (uint16_t address) const		Получить слово из памяти по адресу

В таблице 2.7 приведено описание перечисления COP, в котором перечислены все команды процессора (заголовочный файл cpu.h).

Таблица 2.7 – Описание перечисления COP

Название	Номер	Назначение
stop	0	Команда остановки работы процессора
nop	1	Пустая команда-заглушка
exchange	2	Обмен содержимого регистров
move	3	Пересылка регистр-регистр
load	4	Пересылка память-регистр
save	5	Пересылка регистр-память
dereference_ptr	6	Косвенный доступ к данным по адресу
iAdd	7	Целое сложение
iSub	8	Целое вычитание
iMul	9	Целое умножение
iDiv	10	Целое деление
fAdd	11	Дробное сложение
fSub	12	Дробное вычитание
fMul	13	Дробное умножение
fDiv	14	Дробное деление
io	15	Команды ввода-вывода
jmp	40	Прямой или косвенный безусловный переход

Продолжение таблицы 2.7

Название	Номер	Назначение
jzf	41	Условный переход, если флаг ZF = 1
jnzf	42	Условный переход, если флаг ZF = 0
jsf	43	Условный переход, если флаг SF = 1
jnsf	44	Условный переход, если флаг SF = 0
call	45	Вызов подпрограммы
ret	46	Возврат из подпрограммы

В таблице 2.8 приведено описание вложенной структуры FLAGS для структуры PSW процессора (заголовочный файл cpu.h).

Таблица 2.8 – Описание структуры FLAGS

Поле	Тип	Назначение
ZF	uint16_t	Флаг нуля результата
SF	uint16_t	Флаг знака результата
reserved	uint16_t	В резерве

В таблице 2.9 приведено описание вложенной структуры PSW для одноименного регистра процессора (заголовочный файл cpu.h).

Таблица 2.9 – Описание структуры PSW

Поле	Тип	Назначение
IP	uint16_t	Адрес IP
FLAGS	FLAGS	Флаги

В таблице 2.10 приведено описание класса процессора CPU (заголовочный файл memory.h).

Таблица 2.10 – Описание класса CPU

Поле	Тип	Назначение
cmd	cmd	Текущая выполняемая длинная команда (или две коротких)
command[128]	Command*	Массив указателей на команды
was_transition	bool	Флаг — была ли выполнена команда перехода
PSW	PSW	Регистр PSW = IP + FLAGS
ram	Memory	Память, состоящая из слов
r[16]	data_t	Регистры общего назначения
Метод		**Описание**
CPU()		Конструктор для класса, который выделяет память для указателей команд
~CPU()		Деструктор, который освобождает выделенную память
uint8_t get_cmd_t() const		Получить бит длины текущей команды
uint8_t get_cmd_r1() const		Получить номер первого регистра
uint8_t get_cmd_r2() const		Получить номер второго регистра

Продолжение таблицы 2.10

uint16_t get_cmd_address() const	Получить адрес-константу (в случае 32-битной команды)
void loadCommand()	Загрузить команду: получить из памяти по адресу IP слово и интерпретировать его как команду
void reset()	Сбросить состояние процессора (обнулить флаги и IP)
void run ()	Начать работу процессора

В таблице 2.11 приведено описание абстрактного класса команды Command (заголовочный файл command.h).

Таблица 2.11 – Описание класса Command

Поле	**Тип**	**Назначение**
-	**-**	**-**
Метод		**Описание**
virtual void operator() (CPU& cpu) = 0		Виртуальная функция - функтор
virtual ~Command()		Виртуальный деструктор базового класса
void set_was_transition		Установить флаг was_transition (установить значение флага на истину)

В таблице 2.12 приведено описание класса команд целой арифметики iMath (наследник класса Command) (заголовочный файл command.h).

Таблица 2.12 – Описание класса iMath

Поле	**Тип**	**Назначение**
-	**-**	**-**
Метод		**Описание**
void set_flags (CPU& cpu)		Установить флаги в соответствии с результатом
virtual int32_t calculate (int32_t left, int32_t right)		Соответствующая арифметическая операция (сложение, вычитание, умножение или деление)
virtual void operator() (CPU& cpu)		Переопределенная функция - функтор
virtual ~Command()		Виртуальный деструктор базового класса
void set_was_transition		Установить флаг was_transition (установить значение флага на истину)

Наследники класса iMath (iAdd, iSub, iMul, iDiv) только переопределяют виртуальный метод calculate и больше ничем не отличаются.

В таблице 2.13 приведено описание класса команд дробной арифметики fMath (наследник класса Command) (заголовочный файл command.h).

Таблица 2.13 – Описание класса fMath

Поле	**Тип**	**Назначение**
-	**-**	**-**
Метод		**Описание**
void set_flags (CPU& cpu)		Установить флаги в соответствии с результатом
virtual float calculate (float left, float right)		Соответствующая арифметическая операция (сложение, вычитание, умножение или деление)
virtual void operator() (CPU& cpu)		Переопределенная функция - функтор
virtual ~Command()		Виртуальный деструктор базового класса
void set_was_transition		Установить флаг was_transition (установить значение флага на истину)

Наследники класса fMath (fAdd, fSub, fMul, fDiv) только переопределяют виртуальный метод calculate и больше ничем не отличаются.

В таблице 2.14 приведено описание класса команд переходов Jump (наследник класса Command) (заголовочный файл command.h).

Таблица 2.14 – Описание класса Jump

Поле	**Тип**	**Назначение**
-	**-**	**-**
Метод		**Описание**
virtual void call_go_to (CPU& cpu) = 0		Виртуальная функция вызова операции go_to
void go_to (CPU& cpu)		Операция go_to (безусловный переход)
virtual void operator() (CPU& cpu)		Переопределенная функция - функтор

Наследники класса Jump (jmp, jzf, jnzf, jsf, jnsf, call, ret) только перегружают метод call_go_to. Оставшиеся классы-команды (наследники класса Command): dereference_ptr, exchange, io, load, move, nop и save только перегружают операцию функтор и больше не имеют никаких уникальных методов.

2.5 Разработанные меню и интерфейсы

После запуска программы на выполнение с передачей названия файла в качестве параметра, загрузчик поместит код в память, интерпретатор проанализирует код и выполнит команды. После успешного выполнения программы в консольном окне будет выведено название файла-параметра и результаты выполнения команд, описанных в исполняемом текстовом файле.

На рисунке 2.1 представлен успешный запуск программы.

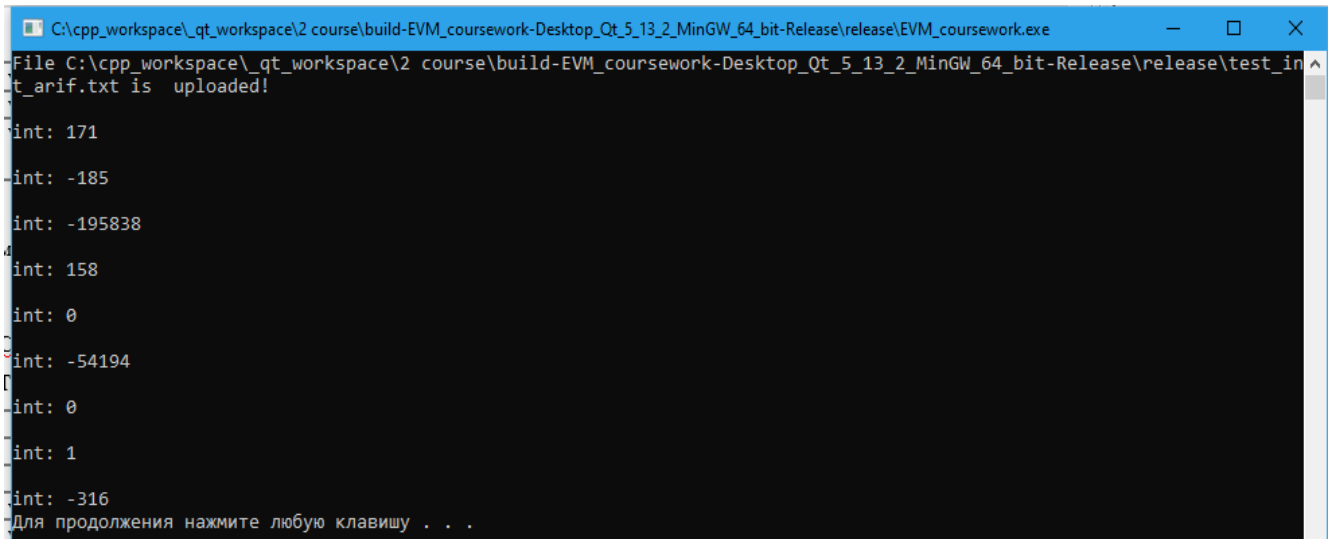

Рисунок 2.1 – Успешный запуск программы

Если же названия в качестве параметра программе передано не было, на экран будет выведено соответствующее сообщение (рис. 2.2).

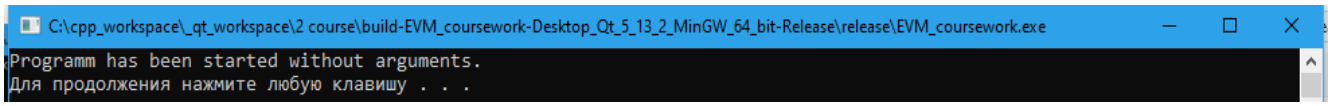

Рисунок 2.2 – Запуск программы без параметров

Если файл не удалось загрузить, на экран будет выведено соответствующее сообщение (рис. 2.3).

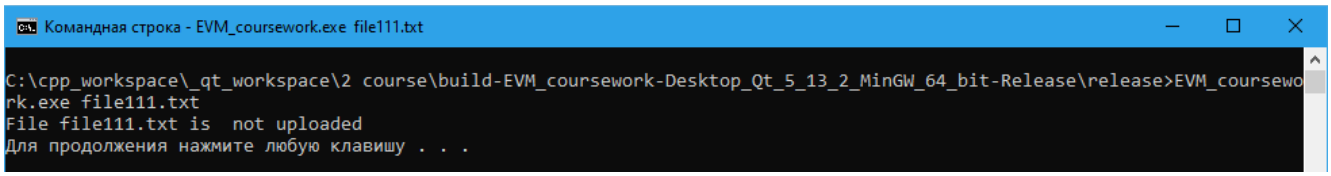

Рисунок 2.3 – Программе не удалось загрузить файл

2.6 Сообщения системы

В таблице 2.15 приведены сообщения системы.

Таблица 2.15 – Сообщения системы

№ п\п	Сообщение	Причина возникновения, способ устранения
1	File %filename% is uploaded!	Файл успешно был прочитан и загружен программой
2	Program has been started without arguments.	Программе не был передан параметр с названием исполняемого текстового файла
3	File %filename% is not uploaded	Файл не удалось загрузить

В случае появления других сообщений следует обратиться к разработчику.

3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

1. Запустить командную строку, перейти в директорию с программой, ввести EVM_coursework.exe и имя исполняемого текстового файла как параметр командой строки.
2. Убедиться, что было выведено сообщение 1 (см. табл. 2.15 и рис. 2.1).
3. Запустить программу снова и указать имя несуществующего файла.
4. Убедиться, что было выведено сообщение 3 (см. табл. 2.15 и рис. 2.3).
5. Запустить программу без агрументов.
6. Убедиться, что было выведено сообщение 2 (см. табл. 2.15 и рис. 2.2).
7. Повторять пункт 1, поочередно передавая в качестве параметра текстовые файлы (test_call_ret, test_conditional_goto_sf, test_conditional_goto_zf, test_dereference_ptr, test_float_arif, test_float_arif_trehadress, test_goto, test_int_arif, test_int_arif_trehadress, test_io, test_perecilki, test_uint_arif) и проверяя результаты выполнения, с целью проверки корректной работы всех команд процессора до тех пор, пока не закончатся все тестовые файлы.

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования был реализован интерпретатор смешанной виртуальной машины с двумя операндами в регистрах и одним в памяти как консольное приложение в объектно-ориентированном стиле, а также загрузчик для виртуальной машины, который загружает содержимое исполняемых текстовых файлов в память, имена которых передаются программе в качестве параметра командой строки при вызове.

Программа отвечает поставленным требованиям и может быть использована для выполнения программ заданной архитектуры компьютера с соответствующим набором команд.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. – СПб.: Питер, 2013.- 816 с.: ил.
2. Assembler / В. Юров – СПб.: Питер, 2002. – 624 с.: ил.
3. Лаптев В. В. C++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия «Учебное пособие»).
4. Саттер Г., Александреску А. Стандарты программирования на C++.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005.— 224 с. :ил.— Парал. тит. англ.
5. Джосаттис, М. Стандартная библиотека C++: справочное руководство, 2-е изд.: Пер. С англ. - М. : ООО „И.Д. Вильямс“, 2014. – 1136 с.: ил. – Парал. тит. англ.

ПРИЛОЖЕНИЕ 1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на выполнение курсового проекта

по дисциплине «Архитектура вычислительных систем, операционные системы»

Направление 090304 – Программная инженерия

Исполнитель: обучающийся гр. ДИПРб21 **Катунин С.С.**

Тема: Реализация виртуальной машины VM10

1 Назначение, цели и задачи разработки

Цель разработки – разработка и реализация интерпретатора виртуальной машины с заданной архитектурой.

Назначение разработки – исполнение программ, написанных в соответствии с системой команд виртуальной машины.

Основные задачи, решаемые разработчиком в процессе выполнения курсового проекта:

- анализ предметной области;
- разработка программного продукта в соответствии с требованиями;
- документирование проекта в соответствии с установленными требованиями.

2 Характер разработки: прикладная квалификационная работа.

3 Основания для разработки

- Учебный план направления 09.03.04 «Программная инженерия» 2018 года набора.
- Рабочая программа дисциплины Программирование и информатика».
- Распоряжение по кафедре АСОИУ №____ от «___»_____ 201_г.

4 Плановые сроки выполнения – осенний семестр 2019/20 учебного года:

Начало «15» сентября 2019 г.

Окончание «27» декабря 2019 г.

5 Требования к проектируемой системе

5.1 Требования к функциональным характеристикам

Проектируемая система представляет собой консольное приложение и должна обеспечивать выполнение следующих основных функций:

- загрузка исходного кода исполняемой программы из файла в память:
  - загрузчик программы для виртуальной машины – независимая функция, вызываемая в функции main() интерпретатора;
  - код программы для виртуальной машины должен быть записан в текстовом файле;
  - загрузчик должен быть способен загружать код программы по любому адресу памяти;

- главная программа должна получать имя файла как параметр командной строки, и передавать загрузчику как параметр при вызове.

- интерпретация и исполнение команд, которые были загружены в память:
  - процессор – отдельный класс, разделение на интерфейс и реализацию;
  - выполнение команд – вызов функтора;
  - память — поле в классе процессор;
  - память – отдельный независимый класс; сама память – динамический массив;
  - команды процессора – иерархия классов-функторов; базовый класс – абстрактный класс Command с перегруженной операцией operator().
- предоставление пользователю результатов работы программы.
- **Интерфейс программы:** текст английский, шрифт латинский.

Система имеет функциональные ограничения:

- можно передавать только одно названия файла в качестве параметра;
- программа не выдает и не обрабатывает исключения при возникновении ошибок.

5.2 Требования к эксплуатационным характеристикам

Программа не должна аварийно завершаться при выполнении правильной корректной исполняемой программы

Время реакции программы на действия пользователя не должно превышать 10 секунд.

5.3 Требования к программному обеспечению:

Средства разработки: интегрированная среда Qt Creator 4.11.0 язык C++ (стандарт C++ 11 и более поздние).

Операционная система: Windows XP (x86) с пакетом обновления 3 (SP3) или более поздние.

5.4 Требования к аппаратному обеспечению:

Рекомендуемая конфигурация:

- Intel-совместимый процессор с частотой не менее 1,6 ГГц;
- не менее 512 МБ ОЗУ;
- не менее 5 МБ свободного места на диске;
- Дисковод CD-ROM/DVD-ROM.

6 Стадии и этапы разработки

6.1 Эскизный проект (ЭП)

- Анализ предметной области.
- Подготовка проектной документации.

6.2 Технический проект (ТП)

- Разработка структур и форм представления данных.

- Разработка структуры программного комплекса.
- Подготовка пояснительной записки.

6.3 Рабочий проект (РП)

- Программная реализация.
- Тестирование и отладка программы.
- Подготовка программной и эксплуатационной документации.

6.4 Эксплуатация (Э)

Описание и анализ результатов проведенного исследования.

7 Требования к документированию проекта

К защите курсового проекта должны быть представлены следующие документы:

- Пояснительная записка к курсовому проекту:
- Презентация доклада.
- Программа, презентация и пояснительная записка к курсовому проекту на оптическом носителе.

Требования к структуре документов определены соответствующими стандартами ЕСПД.

Требования к оформлению определены соответствующими методическими указаниями.

8 Порядок контроля и приемки

Контроль выполнения курсового проекта проводится руководителем поэтапно в соответствии с утвержденным графиком выполнения проекта.

На завершающем этапе руководитель осуществляет нормоконтроль представленной исполнителем документации и принимает решение о допуске (недопуске) проекта к защите.

Защита курсового проекта проводится комиссией в составе не менее двух человек, включая руководителя проекта.

В процессе защиты проекта исполнитель представляет документацию, делает краткое сообщение по теме разработки и демонстрирует ее программную реализацию.

При выставлении оценки учитывается:

- степень соответствия представленной разработки требованиям технического задания;
- качество программной реализации, документации и доклада по теме проекта;
- соблюдение исполнителем графика выполнения курсового проекта.

9 Литература

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. – СПб.: Питер, 2013.- 816 с.: ил.
2. Assembler / В. Юров – СПб.: Питер, 2002. – 624 с.: ил.
3. Лаптев В. В. C++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия «Учебное пособие»).

ПРИЛОЖЕНИЕ 2

Диаграмма классов

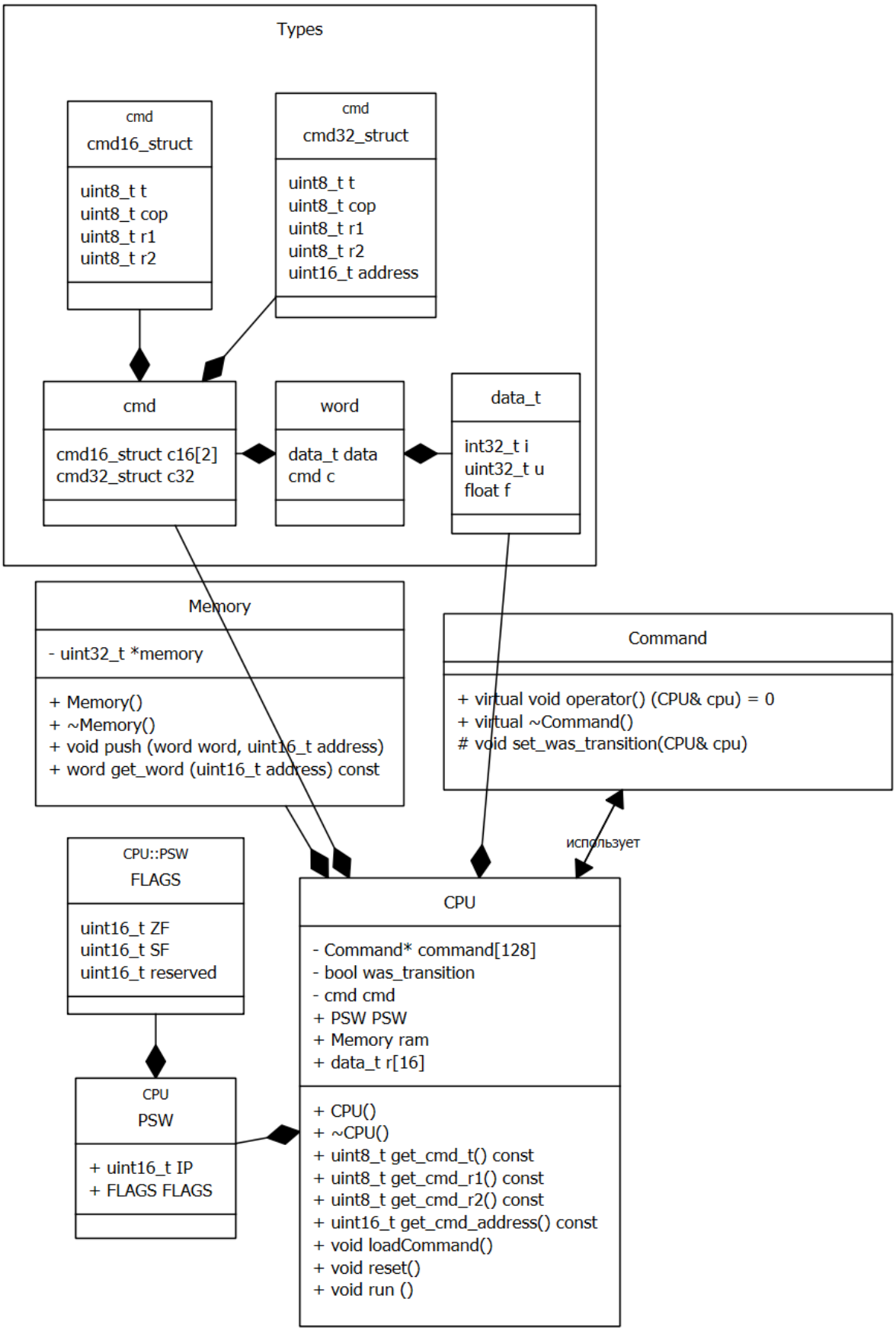

Рисунок П.2.1 — Диаграмма классов (1 часть)

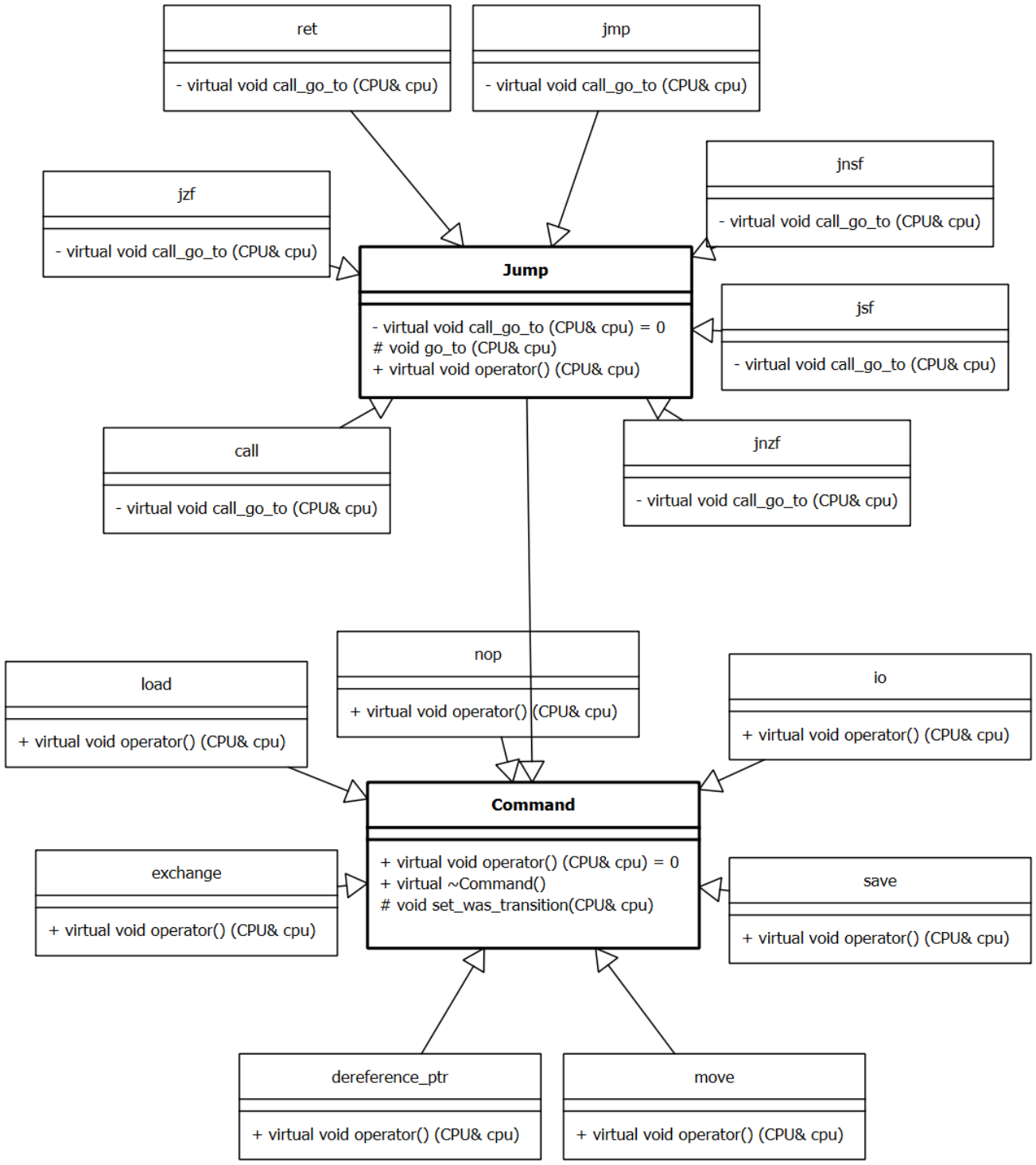

Рисунок П.2.2 — Диаграмма классов (2 часть)

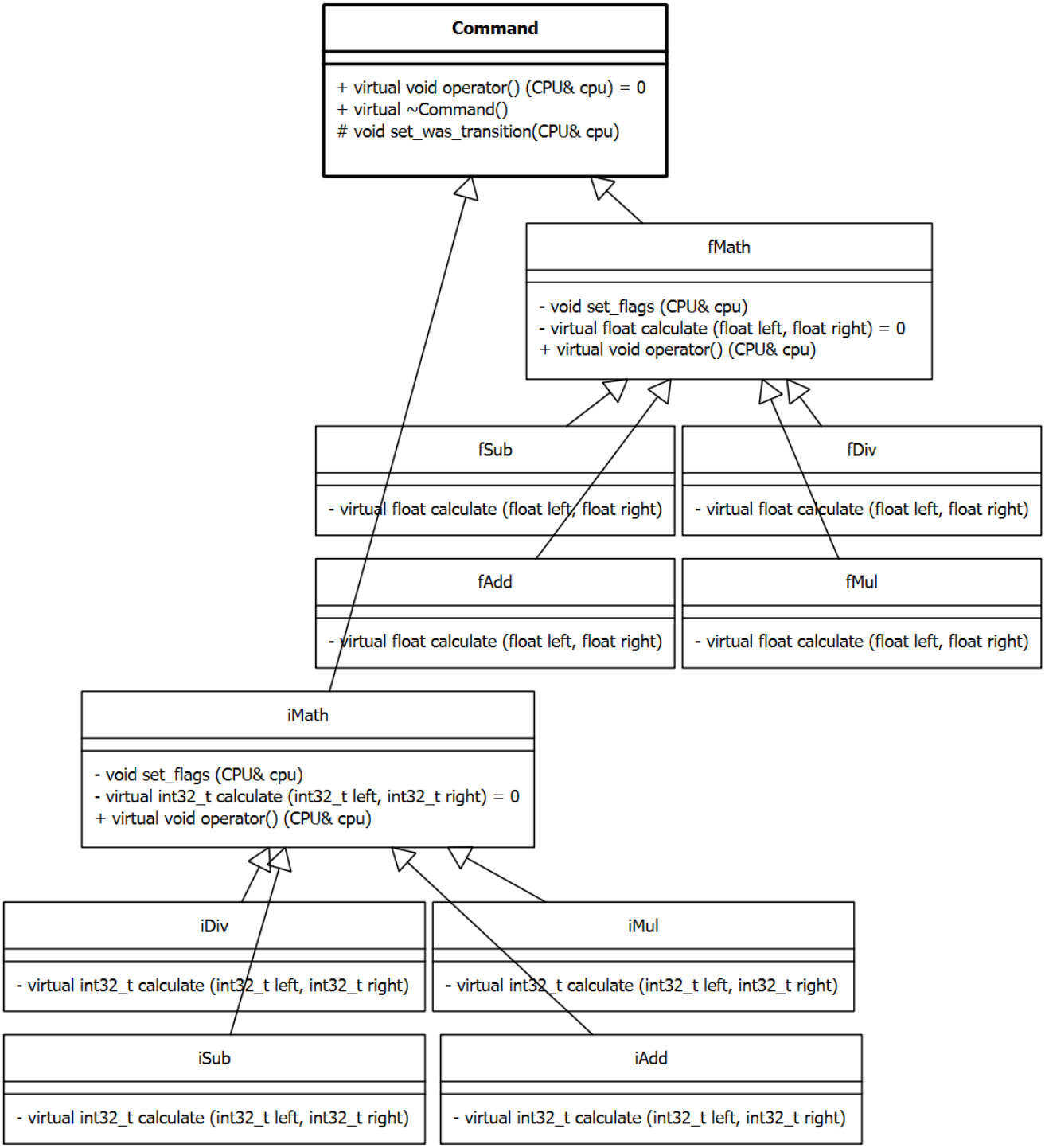

Рисунок П.2.3 — Диаграмма классов (3 часть)

ПРИЛОЖЕНИЕ 3

Примеры исполняемых файлов

Программа test_call_ret

(проверка вызова подпрограммы)

```
a 1000                  ; устанавливаю адрес на 1000
i 158                   ; 1000
i 13                    ; 1001
i 158                   ; 1002
i 195495                ; 1003
i 777                   ; 1004
c 1 4 2 5 7000          ; загрузка адреса 7000 в регистр [5]
c 1 45 0 0 7000         ; прямой переход - вызов подпрограммы по адресу 7000
c 1 15 0 3 1004         ; вывод числа 777 (ожидаемый результат: 777)
a 7000
c 1 4 0 1 1000          ; загрузка числа 158 в [1] регистр
c 1 4 0 2 1001          ; загрузка числа 13 в [1] регистр
c 0 8 1 2               ; вычитание [1] - [2]
c 0 1 0 0
c 1 5 2 0 2500
c 1 15 0 3 2500         ; (ожидаемый результат: 145)
c 1 46 0 0 0
e 1005
```

Программа test_conditional_goto_sf

(проверка условного перехода с проверкой знака результата)

```
a 1000                  ; устанавливаю адрес на 1000
i 158                   ; 1000
i 13                    ; 1001
i 158                   ; 1002
i 195495                ; 1003
i 777                   ; 1004
c 1 4 0 1 1000          ; число 158 -> [1] регистр
c 1 4 0 2 1001          ; число 13 -> [2] регистр
c 1 4 0 3 1002          ; число 158 -> [3] регистр
c 0 8 1 2               ; разность [1] - [2] > 0
c 0 1 0 0
c 1 43 0 0 1100         ; если результат вычитания < 0, то переход к 1100
c 1 44 0 0 1200         ;  если  результат  вычитания  >  0,  то  переход  к  1200
(ожидаемый результат: 13)
c 1 15 0 3 1004         ; вывод числа 777 (ожидаемый результат: 777)
c 0 8 2 3               ; разность [2] - [3] < 0
```

```
c 0 1 0 0
c 1 43 0 0 1300          ; если результат вычитания < 0, то переход к 1300
(ожидаемый результат: 195495)
c 1 44 0 0 1400          ; если результат вычитания > 0, то переход к 1400
c 1 15 0 3 1004          ; вывод числа 777 (ожидаемый результат: 777)
a 1100
c 1 15 0 3 1000          ; вывод числа 158
c 1 40 0 0 1011
a 1200
c 1 15 0 3 1001          ; вывод числа 13
c 1 40 0 0 1011
a 1300
c 1 15 0 3 1003          ; вывод числа 195495
c 1 40 0 0 1015
a 1400
c 1 15 0 3 1000          ; вывод числа 158
c 1 40 0 0 1015
e 1005
```

Программа test_conditional_goto_zf

(проверка условного перехода с проверкой нулевого результата)

```
a 1000                   ; устанавливаю адрес на 1000
i 158                    ; 1000
i 13                     ; 1001
i 158                    ; 1002
i 195495                 ; 1003
i 777                    ; 1004
c 1 4 0 1 1000           ; число 158 -> [1] регистр
c 1 4 0 2 1001           ; число 13 -> [2] регистр
c 1 4 0 3 1002           ; число 158 -> [3] регистр
c 0 8 1 2                ; разность [1] - [2] != 0
c 0 1 0 0
c 1 41 0 0 1100          ; если результат вычитания = 0, то переход к 1100
c 1 42 0 0 1200          ; если результат вычитания != 0, то переход к 1200
(ожидаемый результат: 13)
c 1 15 0 3 1004          ; вывод числа 777 (ожидаемый результат: 777)
c 0 8 1 3                ; разность [1] - [3] == 0
c 0 1 0 0
c 1 41 0 0 1300          ; если результат вычитания = 0, то переход к 1300
(ожидаемый результат: 195495)
c 1 42 0 0 1400          ; если результат вычитания != 0, то переход к 1400
c 1 15 0 3 1004          ; вывод числа 777 (ожидаемый результат: 777)
a 1100
```

```
c 1 15 0 3 1000         ; вывод числа 158
c 1 40 0 0 1011
a 1200
c 1 15 0 3 1001         ; вывод числа 13
c 1 40 0 0 1011
a 1300
c 1 15 0 3 1003         ; вывод числа 195495
c 1 40 0 0 1015
a 1400
c 1 15 0 3 1000         ; вывод числа 158
c 1 40 0 0 1015
e 1005
```

Программа test_dereference_ptr

(проверка косвенного доступа к данным по адресу, лежащему в регистре)

```
a 1000                  ; устанавливаю адрес на 1000
i 158                   ; 1000
i 13                    ; 1001
i 55                    ; 1002
i 195495                ; 1003
i 0                     ; 1004
c 1 4 0 3 1000          ; число 158 -> [3] регистр
c 1 4 0 8 1001          ; число 13 -> [8] регистр
c 1 4 2 10 1002         ; адрес 1002 -> [10] регистр
c 1 15 0 3 1002         ; вывожу число по адресу 1002 - (ожидаемый результат: 55)
c 0 15 10 4             ;  вывожу  содержимое  регистра  10  -  число  1002  (адрес)
(ожидаемый результат: 1002)
c 0 1 0 0
c 1 5 3 0 1002    ; сохраняю по адресу 1002 содержимое [3] регистра
c 1 15 0 3 1002   ; вывожу число по адресу 1002 (ожидаемый результат: 158)
c 0 6 10 14       ; разыменовываю [10] регистр (сохраняю число по адресу, лежащему
в [10] регистре в регистр [14])
c 0 15 14 3       ; вывожу содержимое [14] регистра (ожидаемый результат: 158)
c 1 5 8 0 1002    ; сохраняю по адресу 1002 содержимое [8] регистра - число 13
c 1 6 10 0 7077   ; разыменовываю [10] регистр в память в ячейку 7077
c 1 15 0 3 7077   ; вывожу число по адресу 7077 (ожидаемый результат: 13)
e 1005
```

Программа test_float_arif

(проверка дробной арифметики)

```
a 1000            ; устанавливаю адрес на 1000
f 158.57          ; 1000
f 13.145          ; 1001
f -343.5          ; 1002
```

```
f 195495.089      ; 1003
f 0               ; 1004
c 1 4 0 3 1000    ; число 158.57 -> [3] регистр
c 1 4 0 8 1001    ; число 13.145 -> [8] регистр
c 1 4 0 4 1002    ; число -343.5 -> [4] регистр
c 1 4 0 5 1003    ; число 195495.089 -> [5] регистр
c 1 4 0 6 1004    ; число 0 -> [6] регистр
c 0 11 3 8        ; [3] + [8], результат -> [8] (ожидаемый результат: 171.715)
c 0 11 4 3        ; [4] + [3], результат -> [3] (ожидаемый результат: -185.93)
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 5 3 0 2001    ; сохраняю результат из [3] регистра -> в память в ячейку 2001
c 1 4 0 3 1001    ; число 13.145 -> [3] регистр
c 0 12 4 5        ; [4] - [5], результат -> [8] (ожидаемый результат: -195838.589)
c 0 12 8 3        ; [8] - [3], результат -> [3] (ожидаемый результат: 158.57)
c 1 5 5 0 2002    ; сохраняю результат из [5] регистра -> в память в ячейку 2002
c 1 5 3 0 2003    ; сохраняю результат из [3] регистра -> в память в ячейку 2003
c 0 13 5 6        ; [5] * [6], результат -> [6] (ожидаемый результат: 0)
c 0 13 3 4        ; [3] * [4], результат -> [4] (ожидаемый результат: -54468.795)
c 1 5 6 0 17884   ; сохраняю результат из [5] регистра -> в память в ячейку 17884
c 1 5 4 0 34341   ; сохраняю результат из [3] регистра -> в память в ячейку 34341
c 0 14 6 5        ; [6] / [5], результат -> [5] (ожидаемый результат: 0)
c 0 14 8 3        ; [8] / [3], результат -> [3] (ожидаемый результат: 1.082897)
c 0 14 4 8        ; [4] / [8], результат -> [8] (ожидаемый результат: -317.204641)
c 0 1 0 0         ; команда заглушка (nop)
c 1 5 5 0 2004    ; сохраняю результат из [5] регистра -> в память в ячейку 2004
c 1 5 3 0 2005    ; сохраняю результат из [3] регистра -> в память в ячейку 2005
c 1 5 8 0 2006    ; сохраняю результат из [8] регистра -> в память в ячейку 2006
c 1 15 0 5 2000   ; вывод сохраненных значений
c 1 15 0 5 2001
c 1 15 0 5 2002
c 1 15 0 5 2003
c 1 15 0 5 17884
c 1 15 0 5 34341
c 1 15 0 5 2004
c 1 15 0 5 2005
c 1 15 0 5 2006
e 1005
```

Программа test_float_arif_trehadress

(проверка трехадресной дробной арифметики)

```
a 1000            ; устанавливаю адрес на 1000
f 158             ; 1000
f 13.8            ; 1001
```

```
f 55              ; 1002
f 195495          ; 1003
a 7377
f 777.241         ; 7377
a 1004
f 0               ; 1004
c 1 4 2 3 1000    ; адрес 1000 -> [3] регистр
c 1 4 0 8 1001    ; число 13.8 -> [8] регистр
c 1 11 3 8 6377   ; значение по адресу: ([3]   + 6377) + [8] -> [8] (ожидаемый
результат: 791.041)
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 15 0 5 2000   ; вывод сохраненных значений
e 1005
```

Программа test_goto

(проверка безусловного перехода)

```
a 1000            ; устанавливаю адрес на 1000
i 158             ; 1000
i 13              ; 1001
i 55              ; 1002
i 195495          ; 1003
c 1 4 0 3 1000    ; число 158 -> [3] регистр
c 1 4 0 8 1001    ; число 13 -> [8] регистр
c 1 4 1 4 -343    ; число -343 -> [4] регистр (непосредственная адресация)
c 1 4 0 5 1003    ; число 195495 -> [5] регистр
c 0 2 3 8         ; обмениваю значения в [3] и [8] регистрах
c 0 3 5 4         ; перемещаю значения из [5] регистра в регистр [4]
c 1 4 2 13 2500   ; загружаю адрес 2500 в регистр [13]
c 1 5 3 0 2000    ; сохраняю результат из [3] регистра в память в ячейку 2000
c 1 5 8 0 2001    ; сохраняю результат из [8] регистра в память в ячейку 2001
c 0 5 4 13        ; сохраняю результат из [4] регистра в память по адресу из [13]
регистра
c 0 1 0 0
c 1 4 2 10 1016   ; загружаю адрес 1016 в регистр [10]
c 0 40 0 10       ; косвенный безусловный переход на адрес 1016 (1016 из регистра
[10])
c 0 1 0 0
c 1 15 0 3 2000   ; вывод сохраненных значений по адресу 2000
c 1 40 0 0 1018   ; прямой безусловный переход по адресу 1018
c 1 15 0 3 2001   ; вывод сохраненных значений по адресу 2001
c 1 15 0 3 2500   ; вывод сохраненных значений по адресу 2500 (ожидаемый
результат: 195495)
```

```
c 1 40 1 0 11     ; относительный безусловный переход на 11 вперед (сейчас ip:
1019 -> станет: 1030)
c 1 15 0 3 2005   ; вывод целого числа по адресу 2005 (сейчас ip: 1020)
a 1030            ; 1030
c 1 15 0 0 2005   ; ввод целого числа и сохранение числа по адресу 2005 (сейчас
ip: 1030)
c 1 40 2 0 11     ; относительный безусловный переход на 11 назад (сейчас ip: 1031
-> станет: 1020)
e 1004
```

Программа test_int_arif

(проверка целой арифметики)

```
a 1000            ; устанавливаю адрес на 1000
i 158             ; 1000
i 13              ; 1001
i 55              ; 1002
i 195495          ; 1003
i 0               ; 1004
c 1 4 0 3 1000    ; число 158 -> [3] регистр
c 1 4 0 8 1001    ; число 13 -> [8] регистр
c 1 4 1 4 -343    ; число -343 -> [4] регистр (непосредственная адресация)
c 1 4 0 5 1003    ; число 195495 -> [5] регистр
c 1 4 0 6 1004    ; число 0 -> [6] регистр
c 0 7 3 8         ; [3] + [8], результат -> [8] (ожидаемый результат: 171)
c 0 7 4 3         ; [4] + [3], результат -> [3] (ожидаемый результат: -185)
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 5 3 0 2001    ; сохраняю результат из [3] регистра -> в память в ячейку 2001
c 1 4 0 3 1001    ; число 13 -> [3] регистр
c 0 8 4 5         ; [4] - [5], результат -> [8] (ожидаемый результат: -195838)
c 0 8 8 3         ; [8] - [3], результат -> [3] (ожидаемый результат: 158)
c 1 5 5 0 2002    ; сохраняю результат из [5] регистра -> в память в ячейку 2002
c 1 5 3 0 2003    ; сохраняю результат из [3] регистра -> в память в ячейку 2003
c 0 9 5 6         ; [5] * [6], результат -> [6] (ожидаемый результат: 0)
c 0 9 3 4         ; [3] * [4], результат -> [4] (ожидаемый результат: -54194)
c 1 5 6 0 17884   ; сохраняю результат из [5] регистра -> в память в ячейку 17884
c 1 5 4 0 34341   ; сохраняю результат из [3] регистра -> в память в ячейку 34341
c 0 10 6 5        ; [6] / [5], результат -> [5] (ожидаемый результат: 0)
c 0 10 8 3        ; [8] / [3], результат -> [3] (ожидаемый результат: 1)
c 0 10 4 8        ; [4] / [8], результат -> [8] (ожидаемый результат: -316)
c 0 1 0 0         ; команда заглушка (nop)
c 1 5 5 0 2004    ; сохраняю результат из [5] регистра -> в память в ячейку 2004
c 1 5 3 0 2005    ; сохраняю результат из [3] регистра -> в память в ячейку 2005
```

```
c 1 5 8 0 2006    ; сохраняю результат из [8] регистра -> в память в ячейку 2006
c 1 15 0 3 2000   ; вывод сохраненных значений
c 1 15 0 3 2001
c 1 15 0 3 2002
c 1 15 0 3 2003
c 1 15 0 3 17884
c 1 15 0 3 34341
c 1 15 0 3 2004
c 1 15 0 3 2005
c 1 15 0 3 2006
e 1005
```

Программа test_int_arif_trehadress

(проверка трехадресной целой арифметики)

```
a 1000            ; устанавливаю адрес на 1000
i 158             ; 1000
i 13              ; 1001
i 55              ; 1002
i 195495          ; 1003
a 7377
i 777             ; 7377
a 1004
i 0               ; 1004
c 1 4 2 3 1000    ; адрес 1000 -> [3] регистр
c 1 4 0 8 1001    ; число 13 -> [8] регистр
c 1 7 3 8 6377    ; значение по адресу: ([3]   + 6377) + [8] -> [8] (ожидаемый
результат: 790)
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 15 0 3 2000   ; вывод сохраненных значений
e 1005
```

Программа test_io

(проверка ввода-вывода)

```
a 1000            ; устанавливаю адрес на 1000
u 158             ; 1000
i -13             ; 1001
f 55.5            ; 1002
i 195495          ; 1003
a 7377
i 714             ; 7377
a 1004
i 0               ; 1004
c 1 4 2 3 1000    ; адрес 1000 -> [3] регистр
c 1 4 0 8 1001    ; число -13 -> [8] регистр
```

```
c 1 7 3 8 6377    ; значение по адресу: ([3]  + 6377) + [8] -> [8]
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 15 0 3 2000   ; вывожу целое значение по адресу 2000 (ожидаемый результат:
701)
c 0 15 8 3        ; вывожу целое значение регистра [8] (ожидаемый результат: 701)
c 0 1 0 0         ; команда заглушка (nop)
c 1 4 0 2 1000    ; беззнаковое число 158 -> [2] регистр
c 1 4 0 3 1001    ; целое число -13 -> [3] регистр
c 1 4 0 4 1002    ; дробное число 55.5 -> [4] регистр
c 1 15 0 4 1000   ; вывожу беззнаковое число по адресу 1000 (ожидаемый результат:
158)
c 1 15 0 3 1001   ; вывожу целое число по адресу 1001 (ожидаемый результат: -13)
c 1 15 0 5 1002   ; вывожу дробное число по адресу 1002 (ожидаемый результат:
55.5)
c 0 15 2 4        ; вывожу из регистров (ожидаемый результат: 158)
c 0 15 3 3        ; (ожидаемый результат: -13)
c 0 15 4 5        ; (ожидаемый результат: 55.5)
c 0 1 0 0         ; заглушка (nop)
c 1 15 0 0 2000   ; ввод целого числа и сохранение значения по адресу 2000
c 1 15 0 1 2001   ; ввод беззнакового числа и сохранение значения по адресу 2001
c 1 15 0 2 2002   ; ввод дробного числа и сохранение значения по адресу 2002
c 0 15 10 0       ; ввод целого числа и сохранение значения в регистр [10]
c 0 15 10 3       ; вывод числа из регистра [10]
c 1 15 0 3 2000   ; вывод по адресам
c 1 15 0 4 2001
c 1 15 0 5 2002
e 1005
```

Программа test_perecilki

(проверка команд пересылок save, load, move, exchange)

```
a 1000            ; устанавливаю адрес на 1000
i 158             ; 1000
i 13              ; 1001
i 55              ; 1002
i 195495          ; 1003
c 1 4 0 3 1000    ; число 158 -> [3] регистр
c 1 4 0 8 1001    ; число 13 -> [8] регистр
c 1 4 1 4 -343    ; число -343 -> [4] регистр (непосредственная адресация)
c 1 4 0 5 1003    ; число 195495 -> [5] регистр
c 0 2 3 8         ; обмениваю значения в [3] и [8] регистрах
c 0 3 5 4         ; перемещаю значения из [5] регистра в регистр [4]
c 1 4 2 13 2500   ; загружаю адрес 2500 в регистр [13]
c 1 5 3 0 2000    ; сохраняю результат из [3] регистра в память в ячейку 2000
```

```
c 1 5 8 0 2001    ; сохраняю результат из [8] регистра в память в ячейку 2001
c 0 5 4 13        ; сохраняю результат из [4] регистра в память по адресу из [13]
регистра
c 0 1 0 0
c 1 15 0 3 2000   ;  вывод  сохраненных  значений  по  адресу  2000  (ожидаемый
результат: 13)
c 1 15 0 3 2001   ;  вывод  сохраненных  значений  по  адресу  2001  (ожидаемый
результат: 158)
c 1 15 0 3 2500   ;  вывод  сохраненных  значений  по  адресу  2500  (ожидаемый
результат: 195495)
e 1004
```

Программа test_uint_arif

(проверка беззнаковой целой арифметики)

```
a 1000            ; устанавливаю адрес на 1000
u 158             ; 1000
u 13              ; 1001
u 55              ; 1002
u 195495          ; 1003
u 0               ; 1004
c 1 4 0 3 1000    ; число 158 -> [3] регистр
c 1 4 0 8 1001    ; число 13 -> [8] регистр
c 1 4 2 4 11156   ; число 11156 -> [4] регистр (непосредственная адресация)
c 1 4 0 5 1003    ; число 195495 -> [5] регистр
c 1 4 0 6 1004    ; число 0 -> [6] регистр
c 0 7 3 8         ; [3] + [8], результат -> [8] (ожидаемый результат: 171)
c 0 7 4 3         ; [4] + [3], результат -> [3] (ожидаемый результат: 11314)
c 1 5 8 0 2000    ; сохраняю результат из [8] регистра -> в память в ячейку 2000
c 1 5 3 0 2001    ; сохраняю результат из [3] регистра -> в память в ячейку 2001
c 1 4 0 3 1001    ; число 13 -> [3] регистр
c 0 8 4 5         ; [4] - [5], результат -> [8] (ожидаемый результат: 4294782957)
c 0 8 8 3         ; [8] - [3], результат -> [3] (ожидаемый результат: 158)
c 1 5 5 0 2002    ; сохраняю результат из [5] регистра -> в память в ячейку 2002
c 1 5 3 0 2003    ; сохраняю результат из [3] регистра -> в память в ячейку 2003
c 0 9 5 6         ; [5] * [6], результат -> [6] (ожидаемый результат: 0)
c 0 9 3 4         ; [3] * [4], результат -> [4] (ожидаемый результат: 1762648)
c 1 5 6 0 17884   ; сохраняю результат из [5] регистра -> в память в ячейку 17884
c 1 5 4 0 34341   ; сохраняю результат из [3] регистра -> в память в ячейку 34341
c 0 10 6 5        ; [6] / [5], результат -> [5] (ожидаемый результат: 0)
c 0 10 8 3        ; [8] / [3], результат -> [3] (ожидаемый результат: 1)
c 0 10 4 8        ; [4] / [8], результат -> [8] (ожидаемый результат: 10307)
c 0 1 0 0         ; команда заглушка (nop)
```

```
c 1 5 5 0 2004    ; сохраняю результат из [5] регистра -> в память в ячейку 2004
c 1 5 3 0 2005    ; сохраняю результат из [3] регистра -> в память в ячейку 2005
c 1 5 8 0 2006    ; сохраняю результат из [8] регистра -> в память в ячейку 2006
c 1 15 0 4 2000   ; вывод сохраненных значений
c 1 15 0 4 2001
c 1 15 0 4 2002
c 1 15 0 4 2003
c 1 15 0 4 17884
c 1 15 0 4 34341
c 1 15 0 4 2004
c 1 15 0 4 2005
c 1 15 0 4 2006
e 1005
```