|  |  |  |
| --- | --- | --- |
|  | ***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 VM03»**  по дисциплине «Архитектура вычислительных систем, операционные системы» | | |
| Допущен к защите  «\_\_\_» \_\_\_\_\_\_\_\_202\_\_г.  Руководитель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Оценка, полученная на защите «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_» | | Проект выполнен  обучающимся группы ДИПРб-21 Лиджигоряевым В.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Руководитель  доцент Белов С.В. |
| Члены комиссии:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Белов С.В.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Куркурин Н.Д.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Морозов А.В. | |  |

Астрахань – 2021

**ФЕДЕРАЛЬНОЕ АГЕНТСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ ПО РЫБОЛОВСТВУ**

**АСТРАХАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ** | Кафедра «Автоматизированные системы  обработки информации и управления» |
| Заведующий кафедрой  д.т.н., профессор  Т.В. Хоменко\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202 \_\_ г. |

**ЗАДАНИЕ**

**на выполнение Курсового проекта**

Обучающийся ***Лиджигоряев Владимир Алексеевич***

Группа ***ДИПРб-21***

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

Тема ***«Реализация виртуальной машины VM03»***

Дата получения задания «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г  
Руководитель ***доцент***\_\_\_\_\_\_\_\_\_ ***Лаптев В.В.*** «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.

должность, степень, званиеподписьФИО

Срок представления обучающимся КП на кафедру «22»  декабря  2021 г.

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ***Лиджигоряев В.А.*** «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.

подпись ФИО

**Задачи**

Реализовать двухадресную виртуальную машину с двумя адресными регистрами.

PSW = IP + Flags, 16 + 16 = 32 бит.

*Память* – байтовая; размер адреса – 16 бит.

2 адресных регистра – 16 бит.

*Типы данных*:

Целые знаковые – 4 байта

Дробные – 4 байта

*Структура команды*: 3 байта (24 бита), 4 байта (32 бита); результат по первому адресу.

32 бита: КОП – 6 бит, bb – 2 бита, смещение1 – 12 бит, смещение2 – 12 бит

bb = 01 – первый – регистр, второй = смещение + регистр

bb = 10 – первый = смещение + регистр, второй – регистр

bb = 11 – оба = смещение + регистр

Загрузка адресных регистров:

24 бита: КОП – 6 бит, bb – 2 бита, адрес (константа) – 16 бит

Пересылки

Арифметика (целая беззнаковая) в адресном регистре, сохранение адресного регистра

Арифметика дробная, арифметика целая — в памяти

*Переходы*:

Безусловный – 1 слово = 16 бит (использование bb отличается от стандартного):

КОП – 6 битов, bb – 2 бита, смещение (со знаком) – 8 бит

bb = 00 – относительный по смещению: IP = IP + смещение (не далее, как

на 127-128 слов)

bb = 10 – прямой по первому регистру: IP = a1 + смещение

bb = 01 – прямой по второму регистру: IP = a2 + смещение

bb = 11 – запрещено (резерв)!

Условный – аналогично с проверкой флагов

Вызов подпрограммы – 32 бит:

обычная форма команды с использованием bb;

первый адрес – переход, второй – запоминание адреса возврата

Возврат – безусловный прямой переход по второму регистру: a2 + 0

**Список рекомендуемой литературы**

1. Аблязов Р.З. Программирование на ассемблере на платформе x86-x64. – М.: ДМК Пресс, 2011. – 304 с.: ил.
2. Таненбаум Э., Остин Т. Архитектура компьютера 4-е изд. –СПб.: Питер, 2013. – 698 с.: ил.
3. Акимова Е.В. Вычислительная техника : учебное пособие / Е.В.Акимова. – Санкт-Петербург: Лань, 2020. – 68 с.: ил. – (Учебники для вузов. Специальная литература).
4. Белугина С.В. Архитектура компьютерных систем. Курс лекций: учебное пособие/ С.В. Белугина. – Санкт Петербург : Лань, 2020. – 160 с.: ил. – (Учебники для вузов. Специальная литература).

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ** | К заданию на курсовой проектпо дисциплине  «Архитектура вычислительных систем, операционные системы» |
| Заведующий кафедрой  д.т.н., профессор  Т.В. Хоменко\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202 \_\_ г. |

**КАЛЕНДАРНЫЙ ГРАФИК**

Курсового проектирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Разделы, темы и их содержание, графический материал | Дата сдачи | Объем, % |
| 1 | Выбор темы | 06.09.2021 | 1 |
| 2 | Разработка модели, проектирование системы   * *введение,* * *технический проект,* * *программа и методика испытаний, в том числе – данные для тестирования* * *литература* | 01.10.2021 | 25 |
| 3 | Программная реализация системы   * *работающая программа,* * *рабочий проект* * *скорректированное техническое задание (при необходимости)* | 30.10.2021 | 40 |
| 4 | Тестирование и отладка системы, эксперименты   * *работающая программа с внесёнными изменениями,* * *окончательные тексты всех разделов* | 25.11.2021 | 50 |
| 5 | Компоновка текста  Подготовка презентации и доклада   * *пояснительная записка* * *презентация* * *электронный носитель с текстом пояснительной записки, исходным кодом проекта, презентацией и готовым программным продуктом* | 02.12.2021 | 59 |
| 6 | Защита курсового проекта | 25.12.2021-29.12.2021 | 60-100 |

С графиком ознакомлен «21» сентября 2021 г.

Лиджигоряев В.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, 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 Белов С.В.

подпись, ученая степень, звание, фамилия, инициалы

**СОДЕРЖАНИЕ**

Введение 6

1 Технический проект 7

1 Технический проект 7

1.1 Анализ предметной области 7

1.1.1 Процессор 7

1.1.2 Память 7

1.1.3 Регистр флагов 8

1.1.4 Система команд 8

1.2 Технология обработки информации 11

1.2.1 Формат данных 11

1.2.2 Инфологическая модель 12

1.2.3 Алгоритм загрузчика 12

1.2.4 Алгоритм загрузки процессором следующей команды 13

1.2.5 Алгоритм работы процессора 13

1.2.6 Алгоритм выставления флагов 14

1.2.7 Алгоритм выполнения арифметических команд 14

1.3 Входные и выходные данные 14

1.4 Системные требования 14

2 Рабочий проект 16

2.1 Общие сведения о работе системы 16

2.2 Функциональное назначение программного продукта 16

2.3 Инсталляция и выполнение программного продукта 16

2.4 Описание программы 16

2.5 Разработанные меню и интерфейсы 23

2.6 Сообщения системы 26

3 Программа и методика испытаний 27

3.1 Проверка работоспособности запуска программ с целочисленной арифметикой 27

3.2 Проверка работоспособности запуска программ с дробной арифметикой 27

3.3 Проверка работоспособности запуска программ с подпрограммами 27

Заключение 28

Список использованных источников 29

Приложение 1 Коды тестовых программ 30

Приложение 2 Диаграмма классов 31

# ВВЕДЕНИЕ

Проектирование, создание, тестирование процессора или операционной системы довольно сложные, трудоемкие, а главное довольно затратные материально и по времени.

Действительно, для создания процессора нужно создать его физическое воплощение, а перед этим происходит проектирование, это занимает труд десятков, если не сотен инженеров, и занимает месяцы подготовки, процессоры состоят из огромного множества маленьких транзисторов, которые нужно соединять между собой.

Создание операционной системы занимает длительное время разработки и тестирования, тестирование ОС требует проверку программного продукта на разных физических компьютерах с разной архитектурой, на тестовых стендах, иногда могут быть проблемы с тем, что тестировщик может зафиксировать дефект на своей машине, которой программисту на своей машине не удается повторить. После того, как программист ставит статус проблемы как решенный, тестировщик зовет его к своей машине и наглядно демонстрирует ошибку. В этой ситуации доходит до того, что программисту приходится установить на машину тестировщика множество и «отлавливать» дефект, полностью останавливая работу тестировщика. Таким образом получаются серьезные потери времени команды.

Все эти проблемы позволяет решить виртуальные машины. На их основе можно спроектировать целый процессор, создать его систему команд, протестировать его работу, увидеть некоторые проблемы, если они возникнут, еще на этапе проектирования и решить их на этом этапе. С помощью виртуальных машин также можно создавать неограниченное количество пользовательских конфигураций на своей одной физической машине и протестировать новые операционные системы, в том числе их бета-версии. С помощью этого компаниям, создающим какой-либо программные продукт или процессор, можно избежать серьезных материальных и временных затрат.

Целью создания программного продукта является изучение основ архитектуры вычислительных систем и получение понимания работы виртуальной машины.

Назначение программного продукта – интерпретация пользовательских программ и их выполнение.

# 1 ТЕХНИЧЕСКИЙ ПРОЕКТ

## 1.1 Анализ предметной области

### 1.1.1 Процессор

Процессор – это важнейшая часть любой компьютерной системы, именно на нём и происходит почти вся вычислительная деятельность [1].

Основная задача процессора – выполнение инструкций, которые находятся в памяти компьютера. Всё время после включения компьютера и до самого выключения процессор выполняет инструкции. В зависимости от режима, в котором находится процессор, инструкции выполняются по-разному.

Инструкция или машинная команда – некоторый код, фактически обычное число, которое находится в памяти и обозначает некоторое действие [1]. Инструкции бывают разные и разного размера – они могут занимать от одного до нескольких байт. По сути, они дают указание процессору, что ему надо делать. Для того чтобы процессор знал, откуда из памяти брать команды для выполнения, он использует специальный регистр инструкции. Выполняя очередную инструкцию, процессор обновляет этот указатель, так чтобы он указывал на следующую команду в памяти.

### 1.1.2 Память

Память (системная память) – одна из главных составляющих любой системы. Память бывает двух видов:

* + - 1. Физическая
      2. Линейная

Процессор также имеет свою память двух видов: кэш-память и регистровую. Кэш-память – это такая же системная память, просто находящаяся внутри кристалла процессора. Кэш-память является копией некоторой области памяти основной памяти, к которой процессор часто обращается.

Регистровую память представляют регистры. Регистр – это устройство, которое хранит в себе некоторую информацию, т.е. некоторое значение. Разрядность значения определяет разрядность регистра. Одни регистры могут хранить только определенную память, другие – любую. Те регистры, которые могут хранить любую информацию, называются регистрами общего назначения, другие являются специальные регистрами, которые влияют на работу процессора. Регистры бывают разной разрядности, но большинство 8-, 16-, 32- и 64 – разрядные.

В VM03 процессор имеет два 16-разрядных регистра.

Оперативная память состоит из ячеек, каждая из которых может хранить некоторую порцию информации. Каждая ячейка имеет свой номер, который называется адресом. Если память содержит n ячеек, то они будут иметь адреса от 0 до n-1. Все ячейки памяти содержат одинаковое число битов. Если ячейка состоит из k битов, то она может содержать любую из комбинаций.

В VM03 каждая ячейка состоит из 8 битов (1 байта), размер адреса – 16 бит.

### 1.1.3 Регистр флагов

Регистр флагов – регистр управления или PSW (Program State Word – слово состояния программы). Этот регистр содержит различные биты, которые нужны центральному процессору. Самые важные биты – это коды условия. Они устанавливаются после каждой выполненной команды и отражают состояние ее результата.

Основные биты кода условий:

N – устанавливается, если результат был отрицательным;

Z – устанавливается, если результат был равен 0;

V или O – устанавливается, если результат вызвал переполнение;

C – устанавливается, если результат вызвал выход переноса самого левого бита;

A – устанавливается, если произошел выход переноса бита 3 (служебный перенос)

P – устанавливается, если результат четный.

Коды условий важны, так как они используются при сравнениях и условных переходах.

В VM03 представлены следующие биты кода условий:

NF – устанавливается, если результат был отрицательным;

ZF – устанавливается, если результат был равен 0;

OF - устанавливается, если результат вызвал переполнение;

### 1.1.4 Система команд

Команда состоит из кода операции и некоторой дополнительной информации, например, откуда поступают операнды и куда должны отправляться результаты. Процесс определения, где находятся операнды (то есть их адреса), называется адресацией. Размер команд может быть разный, команды могут содержать разное количество адресов или вообще не содержать адреса.

В VM03 представлены 3-байтные и 4-байтные команды. Каждая команда имеет 6-битный код операции, 3-байтная команда содержит 1 адрес, 4-байтная – 2 адреса. Адресами 4-байтной команды являются смещения относительно адреса в соответствующем регистре, адрес 3-байтной команды представляет константу, итоговый адрес операнда. Также в структуре команды существует 2-бита (bb), которые определяют где находятся операнды, в некоторых командах эти 2 бита используются нестандартно.

Команды VM03:

stop = 0, - Остановка выполнения программы (32 бит, безадресная команда)

move = 1, - Пересылка данных в памяти (2-ух операндная, 32 бит), bb должет быть равен 11

Целочиленная арифметика

iAdd = 2, - Сложение целых чисел (2-ух операндная, 32 бит)

iSub = 3, - Вычитание целых чисел (2-ух операндная, 32 бит)

iMul = 4, - Умножение целых чисел (2-ух операндная, 32 бит)

iDiv = 5, - Деление целых чисел (2-ух операндная, 32 бит)

iMod = 6, - Остаток от деления (2-ух операндная, 32 бит)

iInc = 7, - Инкремент (одно-операндная, 32 бит)

iDec = 8, - Декремент (одно-операндная, 32 бит)

iAnd = 9, - Логическое И (2-ух операндная, 32 бит)

iOr = 10, - Логическое ИЛИ (2-ух операндная, 32 бит)

iNot = 11, - Логическое НЕ (одно-операндная, 32 бит)

iXor = 12, - Исключающее или (одно-операндная, 32 бит)

Вещественная арифметика

rAdd = 13, - Сложение вещественных чисел (2-ух операндная, 32 бит)

rSub = 14, - Вычитание вещественных чисел (2-ух операндная, 32 бит)

rMul = 15, - Умножение вещественных чисел (2-ух операндная, 32 бит)

rDiv = 16, - Деление вещественных чисел (2-ух операндная, 32 бит)

Целочисленная беззнаковая арифметика (с регистрами)

bb используется в качестве выбора регистра, c которым будем работать. Если задан первый бит, то значение в первом регистре увеличивается на операнд команды (константу), если задан второй бит, то значение во втором регистре увеличивается на операнд команды.

rgAdd = 17, - Прибавление значения к регистру (одно-операндная, 24 бит)

rgSub = 18, - Вычитание значения из регистра (одно-операндная, 24 бит)

Работа с подпрограммой

call = 19, - Вызов подпрограммы (2-ух операндная, 32 бит), первый адрес – адрес перехода, второй – запоминание адреса возврата

ret = 20, - Возврат из подпрограммы (2-ух операндная, 32 бит), происходит безусловный переход к адресу в памяти, который равен значению в регистре

Сравнения чисел

icmp = 21, - Сравнение целых чисел (2-ух операндная, 32 бит)

rcmp = 22, - Сравнение дробных чисел (2-ух операндная, 32 бит)

Загрузка, сохранение регистров

bb используется в качестве выбора регистра, в который загружаются (куда сохраняются) данные из памяти (в память)

Если bb == 01, то загружаем (сохраняем) в (из) первый регистр, если bb == 10, то загружаем (сохраняем) во (из) второй регистр.

save = 30, - Сохранение значения адресного регистра в памяти (одно-операндная, 24 бит)

load = 31, - Загрузка адресных регистров из памяти (одно-операндная, 24 бит)

Команды ввода/вывода

bb используется в качестве выбора типа вводимого (выводимого) числа. Если bb == 01, то записываем (выводим) int32, если bb == 10, то записываем (выводим) real32

input = 32, - Ввод чисел (одно-операндная, 24 бит)

output = 33, - Вывод чисел (одно-операндная, 24 бит)

Переходы

Команды переходов имеют следующий формат:

Безусловный – 1 слово = 16 бит (использование bb отличается от стандартного):

КОП – 6 битов, bb – 2 бита, смещение (со знаком) – 8 бит

bb = 00 – относительный по смещению: IP = IP + смещение (не далее, как на 127-128 слов)

bb = 10 – прямой по первому регистру: IP = a1 + смещение

bb = 01 – прямой по второму регистру: IP = a2 + смещение

bb = 11 – запрещено (резерв)

Условный происходит аналогично, только с проверкой флагов.

jmp = 34, - Прямой безусловный переход (одно-операндная, 24 бит)

je = 35, - Переход, если операнды равны (24 бит)

jne = 36, - Переход, если операнды не равны (24 бит)

jg = 37, - Переход, если первый операнд больше второго (24 бит)

jge = 38, - Переход, если первый операнд больше или равен второму (24 бит)

jl = 39, - Переход, если первый операнд меньше второго (24 бит)

jle = 40, - Переход, если первый операнд меньше или равен второму (24 бит)

## 1.2 Технология обработки информации

Анализ предметной области показал, что программа рассчитана на одного пользователя. На рис. 1.1. показана диаграмма вариантов использования.



Рисунок 1.1 – Диаграмма вариантов использования

Вариант использования «Запустить виртуальную машину» включает обязательную функцию «Ввести путь к файлу с программным кодом»

### 1.2.1 Форматы данных

В программном продукте память будет представлена в виде одномерного массива состоящего из N беззнаковых целых 8-битных чисел, где N = . 24-битная команда представляет структуру, состоящую из кода операции (6-битное беззнаковое целое число), bb (2-ух битное беззнаковое целое число), адрес 16-битное беззнаковое целое число). 32-битная команда представляет структуру, состоящую из кода операции (6-битное беззнаковое целое число), bb (2-ух битное беззнаковое целое число), первое смещение (12-битное беззнаковое целое число), второе смещение (12-битное беззнаковое целое число).

В текстовом файле с программным кодом должны использоваться следующие теги:

* тег a, операнды: (адрес) – загрузка адреса стартовой ячейки памяти, с которой начинаем записывать программу в память, и установка текущего адреса для загрузки на эту ячейку;
* тег i, операнды: (число) – загрузка целого числа в память по адресу, равному текущему адресу для загрузки;
* тег r, операнды: (число) – аналогичная загрузка числа по текущему адресу для загрузки, только число – дробое;
* тег c, операнды: (Код операции) (bb) (адрес или первое смещение) (второе смещение) – загрузка команды в память, первым операндом является адрес, если команда – 24-битная, иначе там должно быть первое смещение, второе смещение должно быть записано, если команда является 32-битное двух-операндной.
* тег e, операнды: (адрес) – загрузка указателя на текущую выполняемую команда на переданный адрес.

Если операндов у тегов несколько, то они записываются через пробел.

### 1.2.2 Инфологическая модель

В ходе анализа предметной области были разработаны диаграммы основных классов программы и классов, представляющих команды машины (см. Приложение 2).

### 1.2.3 Алгоритм загрузчика

Дано: fName – строка, именем текстового файла с программным кодом.

Требуется: загрузить все данные и команды из файла в память.

Решение:

Присвоить целочисленной переменной address 0.

Присвоить символьной переменной sym значение ‘ ’.

Открыть файл с именем fName на чтение.

Пока sym не равно ‘e’, повторять

| Считать очередную строку из файла в строку str.

| Разделить str на подстроки по символу пробела и добавить эти подстроки в

| одномерный массив строк is.

| Присвоить sym значение первого символа первого элемента is.

| Если sym = a, то

| | Записать в address значение из is[1].

| Иначе если sym = i, то

| | Считать целое число из is[1] в последующие 4 ячейки памяти, начиная с

| | ячейки, адрес которой равен is[2].

| | Увеличить address на 4.

| Иначе если sym = r, то

| | Считать дробное число из is[1] в последующие 4 ячейки памяти, начиная с

| | ячейки, адрес которой равен is[2].

| | Увеличить address на 4.

| Иначе если sym = c, то

| | Если число, представленное в строке is[1] < 30, то

| | | Считать 32-битную команду, КОП которой равен is[1], bb в is[2], а два

| | | смещения в is[3] и is[4] соответственно.

| | | Загрузить эту команду в память, начиная с ячейки, адрес которой равен

| | | address.

| | | Увеличить address на 4.

| | Иначе

| | | Считать 24-битную команду, КОП которой равен is[1], bb в is[2], а адрес

| | | находится в is[3].

| | | Загрузить эту команду в память, начиная с ячейки, адрес которой равен

| | | address.

| | | Увеличить address на 3.

| | конец ветвления

| Иначе если sym = e, то

| | Создать экземпляр структуры 32-битной команды, все поля которой равны 0.

| | Загрузить команду в память, начиная с ячейки, равной address.

| | Увеличить address на 4.

| конец ветвления

Конец алгоритма.

### 1.2.4 Алгоритм загрузки процессором следующей команды

Дано: память (memory), указатель на текущую выполняемую команду (psw), cmd (поле процессора, см. Приложение 1).

Требуется: загрузить из памяти следующую команду и присвоить ее cmd.

Решение:

Присвоить 8-битной беззнаковой целочисленной переменной code\_bb значение memory[psw].

Записать в поле cmd.cmd32.code первые шесть бит code\_bb.

Записать в поле cmd.cmd32.bb последние два бита code\_bb.

Конец алгоритма.

### 1.2.5 Алгоритм работы процессора

Дано: код операции текущей команды (КОП), указатель на текущую команду (IP).

Требуется: выполнять текущую и загружать следующую команду.

Решение:

Вызвать вспомогательный алгоритм 1.2.4.

Пока КОП не равно коду команды остановки процессора, повторять

| Выполнить текущую команду.

| Если КОП < 30, то

| | Увеличить IP на 4.

| Иначе

| | Увеличить IP на 3.

| конец ветвления

| Вызвать вспомогательный алгоритм 1.2.4.

конец цикла

Конец алгоритма.

### 1.2.6 Алгоритм выставления флагов

Дано: результат арифметической операции.

Требуется: выставить флаги NF, ZF, OF.

Решение:

Если результат операции == 0, ZF = 1, иначе 0.

Если результат операции < 0, ZF = 1, иначе 0.

Если значение результата операции выходит за пределы, то OF = 1, иначе 0.

Конец алгоритма.

### 1.2.7 Алгоритм выполнения арифметических команд

Дано: proc – ссылка на объект типа Processor.

Требуется: записать результат операции по первому адресу.

Решение:

Присвоить беззнаковым 16-битным целочисленным переменным os1, os2 значения смещений у текущей команды в proc.

Присвоить беззнаковой 16-битной целочисленной переменной address1 значение, равное сумме значения в первом регистре proc и os1.

Присвоить беззнаковой 16-битной целочисленной переменной address2 значение, равное сумме значений во втором регистре proc и os2.

Вычислить результат операции c операндами, имеющими адреса address1 и address2 соответственно, записать его на место первого операнда в памяти.

Установить флаги PSW по результату команды.

Конец алгоритма.

## 1.3 Входные и выходные данные

Входные данные: строка: путь файлу с программным кодом.

Выходные данные: результат выполнения программы.

## 1.4 Системные требования

Рекомендуемая конфигурация:

* Intel-совместимый процессор с частотой не менее 1,6 ГГц;
* не менее 2 Гб ОЗУ;
* не менее 20 MБ свободного места на диске;
* дисковод CD-ROM/DVD-ROM.

Операционная система: Windows 10.

# 2 РАБОЧИЙ ПРОЕКТ

## 2.1 Общие сведения о работе системы

Программный продукт разработан в интегрированной среде Visual Studio Community 2019 (версия 16.11.2) на языке C++ 17 стандарта. Программа работает под управлением операционной системы Windows 10.

## 2.2 Функциональное назначение программного продукта

Разработанный программный продукт предназначен для выполнения программ для виртуальной машины VM03. Программа имеет следующие функциональные возможности:

* предоставление пользователю ввести путь к текстовому файлу с программным кодом;
* считывает текстовый файл с программным кодом и загружает программу в память;
* вывод результата выполнения программы.

## 2.3 Инсталляция и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера папку “VM03”, содержащую исполняемый

файл vm03.exe и файлы: IMath\_program.txt, IO\_program.txt, Jumps\_program.txt,

Move\_program.txt, RMath\_program.txt и Subprogram\_program.txt.

1. Запустить на выполнение файл vm03.exe.

## 2.4 Описание программы

В программе используются следующие объявления:

1. byte\_t означает 8-битное беззнаковое целое число;
2. address\_t означает 16-битное беззнаковое целое число.
3. union cmd\_t

{

cmd32 cmd32;

cmd24 cmd24;

};

В таблице 2.1 приведено описание класса Exceptions (Исключения).

Таблица 2.1 – Описание класса Exceptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| text | строка | private | текст исключения |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| explicit base\_exception(std::string ex\_text) | public | инициализация поля text значением ex\_text |
| const char\* what() const noexcept final | public | возвращает текст исключения |

В таблице 2.2 приведено описание класса Memory (память).

Таблица 2.2 – Описание класса Memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| mem | byte\_t\* | | private | | массив ячеек памяти |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| Memory() | | public | | выделяет память под массив mem на число, равное максимальному значению address\_t | |
| byte\_t& operator[](const address\_t& addr) | | public | | возвращает значение ячейки памяти по адресу addr | |

Классы invalid\_path и invalid\_command\_format наследуются от Exceptions. Своих полей классы не имеют, из методов: конструкторы, которые вызывают конструктор базового класса.

В таблице 2.3 приведено описание абстрактного класса Command (Команда)

Таблица 2.3 – Описание класса Command

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| virtual void operator()(Processor& processor) = 0 | public | Оператор выполнения команды |

Классы Command24 и Command32 наследуются от класса Command.

В таблице 2.4 приведено описание абстрактного класса Command24 (24-битная команда)

Таблица 2.4 – Описание класса Command24

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | | **Квалификатор доступа** | | **Назначение** | |
| size | статическая константа byte\_t | | | private | | Определяет размер команды, равна 3 | |
| **Метод** | | **Квалификатор доступа** | | | **Назначение** | | |
| static address\_t GetAddress(Processor& proc) | | public | | | Возвращает значение операнда текущей команды из памяти | | |
| **Наследники** | | | | | | |
| Input | | | Ввод данных | | | |
| Output | | | Вывод данных | | | |
| Jump | | | Переходы | | | |
| Load | | | Загрузка регистров | | | |
| Save | | | Сохранение регистров | | | |

В таблице 2.5 приведено описание абстрактного класса Command32 (32-битная команда)

Таблица 2.5 – Описание класса Command32

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| size | статическая константа byte\_t | | private | | Определяет размер команды, равна 4 |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| static void GetOs1Os2(Processor& proc, address\_t& os1, address\_t& os2) | | public | | Помещает в os1 и os2 значения операндов текущей команды процессора из памяти | |

Продолжение таблицы 2.5

|  |  |
| --- | --- |
| **Наследники** | |
| Call | Вызов подпрограммы |
| Ret | Возврат из подпрограммы |
| IntMath | Целочисленная арифметика |
| RealMath | Дробная |
| Move | Перемещение данных |
| RegsMath | Регистровая арифметика |

В таблице 2.6 приведено описание класса IntMath

Таблица 2.6 – Описание класса IntMath

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| exec | std::function<int64\_t(int64\_t, int64\_t)> | | private | | Функциональный объект, выполняет переданною ему операцию |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| static void set\_flags(Processor& proc, int64\_t num) | | private | | Установка флагов по результату операции | |
| void operator()(Processor& proc) | | public | | Перегрузка оператора функции | |

В таблице 2.7 приведено описание класса RealMath

Таблица 2.7 – Описание класса RealMath

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| exec | std::function<double(double, double)> | | private | | Функциональный объект, выполняет переданною ему операцию |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| static void set\_flags(Processor& proc, double num) | | private | | Установка флагов по результату операции | |
| void operator()(Processor& proc) | | public | | Перегрузка оператора функции | |

В таблице 2.8 приведено описание класса RegsMath

Таблица 2.8 – Описание класса RegsMath

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| exec | std::function<address\_t(address\_t, address\_t)> | | private | | Функциональный объект, выполняет переданною ему операцию |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| void operator()(Processor& proc) | | public | | Перегрузка оператора функции | |

В таблице 2.9 приведено описание класса Jump

Таблица 2.9 – Описание класса Jump

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| exec | std::function<address\_t(address\_t, address\_t)> | | private | | Функциональный объект, выполняет переданною ему операцию |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| void operator()(Processor& proc) | | public | | Перегрузка оператора функции | |
| void go\_to(Processor& proc) | | public | | Безусловный переход | |

В таблице 2.10 приведено описание структуры PSW

Таблица 2.10 – Описание структуры PSW

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| NF : 1 | byte\_t | Флаг отрицания |
| ZF : 1 | byte\_t | Флаг отрицания |
| OF : 1 | byte\_t | Флаг отрицания |
| reserved : 12 | address\_t | Зарезервированная часть |
| IP | address | Указатель на текущую команду |

Продолжение таблицы 2.10

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| PSW() | public | Устанавливает все флаги в 0 |
| void set\_ZF(double num) | public | Устанавливает флаг ZF, если num = 0 |
| void set\_NF(double num) | public | Устанавливает флаг NF, если num < 0 |
| void set\_ZF(int64\_t num) | public | Устанавливает флаг ZF, если num = 0 |
| void set\_NF(int64\_t num) | public | Устанавливает флаг NF, если num < 0 |
| void set\_OF(int64\_t num) | public | Устанавливает флаг OF, если num < минимального значения int32\_t или num > максимального значения int32\_t |
| void set\_OF(double num) | public | Устанавливает флаг OF, если num < минимального значения float или num > максимального значения float |
| void reset\_flags() | public | Устанавливает все флаги в 0 |

В таблице 2.11 приведено описание структуры cmd24

Таблица 2.11 – Описание структуры cmd24

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| code | byte\_t | Код операции |
| bb | byte\_t | Формат команды |
| address | address\_t | Адрес |

В таблице 2.12 приведено описание структуры cmd32

Таблица 2.12 – Описание структуры cmd32

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| code | byte\_t | Код операции |
| bb | byte\_t | Формат команды |
| os1 | address\_t : 12 | Первое смещение |

Продолжение таблицы 2.12

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| os2 | address\_t : 12 | Второе смещение |

В таблице 2.13 приведено описание класса Processor

Таблица 2.13 – Описание класса Processor

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| psw | PSW | private | Регистр флагов |
| startIp | address\_t | private | Начальный IP, задаваемый тегом а |
| cmd | cmd\_t | private | Текущая выполняемая команда |
| commands | class Command\*[41] | private | массив указателей на абстрактный класс Command |
| IP | address | private | Указатель на текущую команду |
| Operations | enum | private | Перечисление всех команд с их кодами |
| memory | Memory | public | Память |
| regs | address\_t [2] | public | Регистры |

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| Processor() | public | Инициализирует массив commands |
| cmd\_t GetCurrentCmd() | public | Возвращает cmd |
| void SetIP(address\_t IP) | public | Присваивает psw.IP значение IP |
| PSW GetPSW() | public | Возвращает psw |
| void reset() | public | Устанавливает флаги в 0, присваивает psw.IP startIP |

Продолжение таблицы 2.13

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| void run() | public | Запускает процессор |

## 2.5 Разработанные меню и интерфейсы

При запуске программы через командную строку и передачи IMath.txt – имени текстового файла, содержащего код тестовой программы (см. приложение 1), как параметра командной строки (рис.2.1).



Рисунок 2.1 – Запущенная программа с кодом из IMath.txt

Программа из IMath.txt запрашивает у пользователя операнды или операнды и выводит результат операции (операции из целочисленной арифметики)

При вводе 100 и 200, как двух операндов, выводится результат операции сложения этих чисел (рис.2.2).

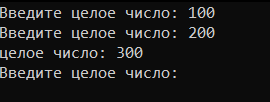


Рисунок 2.2 – Вывод результата сложения

При вводе 100 и 150, как двух операндов для следующей операции, выводится результат операции вычитания этих чисел (рис.2.3).

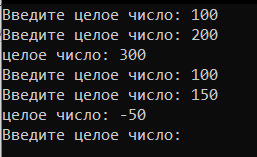


Рисунок 2.3 – Вывод результата сложения

При вводе 5 и 5, как двух операндов для следующей операции, выводится результат операции умножения этих чисел (рис.2.4).

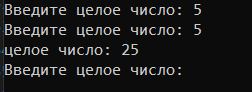


Рисунок 2.4 – Вывод результата умножения

При вводе 15 и 5, как двух операндов для следующей операции, выводится результат операции деления этих чисел (рис.2.5).

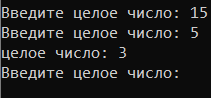


Рисунок 2.5 – Вывод результата деления

При вводе 7 и 5, как двух операндов для следующей операции, выводится результат операции деления этих чисел (рис.2.6).

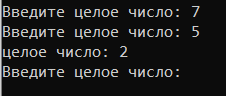


Рисунок 2.6 – Вывод результата остатка от деления

При вводе 1, как операнда для следующей операции, выводится результат операции инкремента этого числа (рис.2.7).

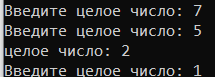


Рисунок 2.7 – Вывод результата инкремента

При вводе 4, как операнда для следующей операции, выводится результат операции декремента этого числа (рис.2.8).

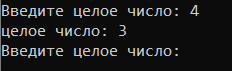


Рисунок 2.8 – Вывод результата декремента

При вводе 1 и 4, как операндов для операции логического умножения, 1 и 4, как операндов для операции логического сложения, 1, как операнда для логического отрицания и 1 и 1, как операндов для исключающего ИЛИ, выводится следующий результат (рис.2.9).

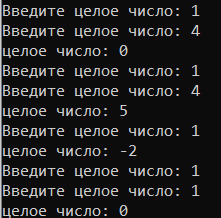


Рисунок 2.9 – Вывод результата операций

При запуске программы из RMath.txt – имени текстового файла, содержащего код тестовой программы (см. приложение 1), предлагаются ввести операнды для операций сложения, вычитания, умножения и деления.

При вводе 10.5 и 6.5, как операндов для сложения, 50.2 и 48.3, как операндов для вычитания, 0.5 и 8, как операндов для умножения, 1 и 3, как операндов для деления, выводится следующий результат (рис.2.10).

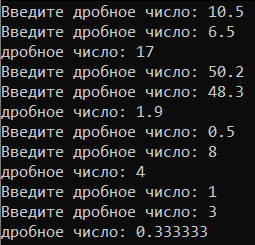


Рисунок 2.10 – Вывод результата операций

При запуске программы из Subprogram.txt – имени текстового файла, содержащего код тестовой программы (см. приложение 1), предлагаются ввести число, которое в подпрограмме будет увеличено на единицу и выведено на экран. Подпрограмма вызывается из основной программы.

При вводе 10, выводится 11 (рис. 2.11).



Рисунок 2.11 – Вывод результата выполнения подпрограммы

## 2.6 Сообщения системы

В таблице 2.9 приведены сообщения системы.

Таблица 2.9 – Сообщения системы

|  |  |  |
| --- | --- | --- |
| **№ п\п** | **Сообщение** | **Причина возникновения,  способ устранения** |
| 1 | Введите целое число: | Предлагается ввести целое число для записи в память |
| 2 | Введите дробное число: | Предлагается ввести дробное число для записи в память |
| 3 | целое число: (число) | Вывод числа из память на экран |
| 4 | дробное число: (число) | Вывод числа из память на экран |
| 5 | Задан неверный формат команды | Загруженная программа содержит команду, которая записана неправильно |
| 6 | Задан неверный путь к файлу с программой | Файла с именем, переданным в качестве параметра командной строки, не найден. |

В случае появления других сообщений следует обратиться к разработчику.

**3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ**

## Текстовые файлы IMath.txt, RMath.txt, Subprogram.txt содержат коды тестовых программ, содержимое файлов показано в Приложении 1.

## 3.1 Проверка работоспособности запуска программ с целочисленной арифметикой

1. Запустить программу с “IMath.txt” в качестве параметра командной строки (см. рис. 2.1).
2. Ввести те же значения, что вводятся на рис. 2.2 – 2.9, убедиться вывод результатов операций, такой же как на рис. 2.2 – 2.9 (см. рис.2.3).
3. Запустить программу повторно с “IMath.txt”, сложить 0 и 0, убедиться, что результатом является 0.
4. Вычесть из -10 -5, убедиться, что выводится -5.
5. Умножить 0 и 4, убедиться, что выводится 0.
6. Разделить 6 на 7, убедиться, что выводится 0.

## 3.2 Проверка работоспособности запуска программ с дробной арифметикой

1. Запустить программу с “RMath.txt” в качестве параметра командной строки и убедиться, что предлагается ввести дробное число.
2. Ввести те же значений, что вводятся на рис. 2.10, убедиться вывод результатов операций, такой же как на рис. 2.10.
3. Запустить программу повторно с “RMath.txt”, сложить 0.01 и 0.09, убедиться, что результатом является 0.1.
4. Вычесть из -7.5 -8.5, убедиться, что выводится 1.
5. Умножить 1.5 и 4, убедиться, что выводится 6.
6. Разделить 0 на 7, убедиться, что выводится 0.

## 3.3 Проверка работоспособности запуска программ с подпрограммами

1. Запустить программу на “Subprogram.txt” в качестве параметра командной строки и убедиться, что предлагается ввести целое число.
2. Ввести 2, убедиться, что выводится 3.
3. Запустить программу повторно с “Subprogram.txt”, ввести -1, убедиться, что результатом выводится 0.
4. Запустить программу повторно с “Subprogram.txt”, ввести 0, убедиться, что результатом выводится 1.

# ЗАКЛЮЧЕНИЕ

В результате курсового проектирования разработана программа-интерпретатор для виртуальной машины VM03. Для данной машины можно писать программы, которые она будет интерпретировать и выполнять. Командами, используемыми для написания программ для VM03, могут быть команды, описанные в системе команд. Также возможно расширение списка команд с минимальными изменениями кода самой машины VM03.

Программа отвечает поставленным требованиям и может быть эксплуатироваться для выполнения пользовательских программ.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аблязов Р.З. Программирование на ассемблере на платформе x86-x64. – М.: ДМК Пресс, 2011. – 304 с.: ил.
2. Таненбаум Э., Остин Т. Архитектура компьютера 4-е изд. –СПб.: Питер, 2013. – 698 с.: ил.
3. Акимова Е.В. Вычислительная техника : учебное пособие / Е.В.Акимова. – Санкт-Петербург: Лань, 2020. – 68 с.: ил. – (Учебники для вузов. Специальная литература).
4. Белугина С.В. Архитектура компьютерных систем. Курс лекций: учебное пособие/ С.В. Белугина. – Санкт Петербург : Лань, 2020. – 160 с.: ил. – (Учебники для вузов. Специальная литература).
5. Лаптев В.В. С++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия “Учебное пособие”).
6. Прата С. Язык программирования С++. Лекции и упражнения, 6-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 2012. – 1248 с. : ил.

# ПРИЛОЖЕНИЕ 1

# Коды тестовых программ

1. Программа с целочисленной арифметикой (IMath.txt):

a 0

i 0 ; 0 - 3

c 31 01 0 ; Загрузка 0 в первый регистр

c 31 10 0 ; Загрузка 0 в первый регистр

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 2 11 1000 1004 ; Сложение чисел из 1000 и 1004 ячеек и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 3 11 1000 1004 ; Вычитание чисел из 1000 и 1004 ячеек и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 4 11 1000 1004 ; Умножение чисел из 1000 и 1004 ячеек и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 5 11 1000 1004 ; Деление чисел из 1000 и 1004 ячеек и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 6 11 1000 1004 ; Остаток от деления чисел из 1000 и 1004 ячеек

; и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 7 11 1000 1004 ; Инкремент числа из 1000 ячейки и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 8 11 1000 1004 ; Декремент чисел из 1000 ячейки и сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 9 11 1000 1004 ; Логическое И чисел из 1000 и 1004 ячейки и

; сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 10 11 1000 1004 ; Логическое ИЛИ чисел из 1000 и 1004 ячейки и

; сохранение в 1000 - 1003 (95 - 98)

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 11 11 1000 1004 ; Логическое НЕ чисел из 1000 и 1004 ячейки и

; сохранение в 1000 - 1003 (95 - 98)

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

c 32 01 1000 ; Запись целого числа в ячейки 1000 - 1003

c 32 01 1004 ; Запись целого числа в ячейки 1004 - 1007

c 12 11 1000 1004 ; Исключающее ИЛИ чисел из 1000 и 1004 ячейки и

; сохранение в 1000 - 1003

c 33 01 1000 ; Вывод целого числа из ячейки 1000 - 1003

e 4

1. Программа с дробной арифметикой (RMath.txt):

a 0

i 0 ; 0 - 3

c 31 01 0 ; Загрузка 0 в первый регистр

c 31 10 0 ; Загрузка 0 в первый регистр

c 32 10 1000 ; Запись дробного числа в ячейки 1000 - 1003

c 32 10 1004 ; Запись дробного числа в ячейки 1004 - 1007

c 13 11 1000 1004 ; Сложение чисел из 1000 и 1004 ячеек и сохранение в 1000 - 1003

c 33 10 1000 ; Вывод дробного числа из ячейки 1000 - 1003

c 32 10 1000 ; Запись дробного числа в ячейки 1000 - 1003

c 32 10 1004 ; Запись дробного числа в ячейки 1004 - 1007

c 14 11 1000 1004 ; Вычитание чисел из 1000 и 1004 ячеек и сохранение в

; 1000 - 1003

c 33 10 1000 ; Вывод дробного числа из ячейки 1000 - 1003

c 32 10 1000 ; Запись дробного числа в ячейки 1000 - 1003

c 32 10 1004 ; Запись дробного числа в ячейки 1004 - 1007

c 15 11 1000 1004 ; Умножение чисел из 1000 и 1004 ячеек и сохранение в

; 1000 - 1003

c 33 10 1000 ; Вывод дробного числа из ячейки 1000 - 1003

c 32 10 1000 ; Запись дробного числа в ячейки 1000 - 1003

c 32 10 1004 ; Запись дробного числа в ячейки 1004 - 1007

c 16 11 1000 1004 ; Деление чисел из 1000 и 1004 ячеек и сохранение в

; 1000 - 1003

c 33 10 1000 ; Вывод дробного числа из ячейки 1000 - 1003

e 4

1. Программа с вызовом подпрограммы (Subprogram.txt):

; Программа с подпрограммой

a 0

i 0 ; (0 - 3)

; Начало подпрограммы

c 7 11 0 0

c 33 01 0

c 20 00

; Конец подпрограммы

; Основная программа

c 32 01 0

c 19 11 4 17

e 15

# ПРИЛОЖЕНИЕ 2

# Диаграмма классов

На рисунке П1 приведена диаграмма основных классов.

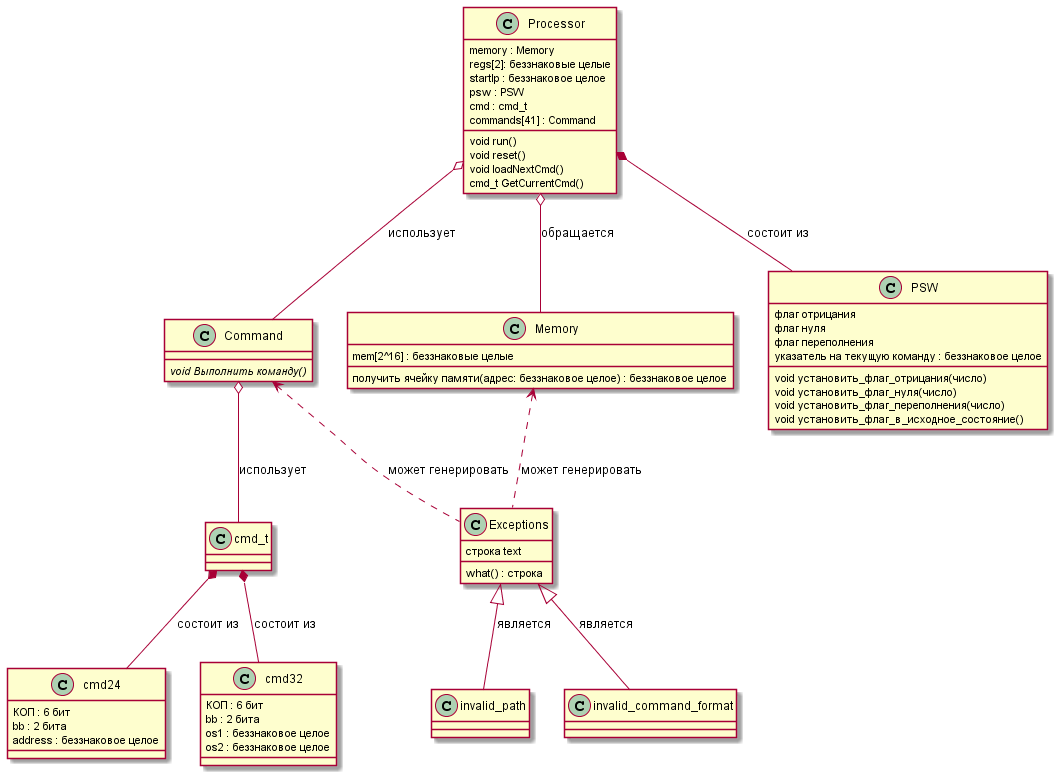


Рисунок П1 – диаграмма основных классов

# Диаграмма классов, представляющих команды

На рисунке П2 приведена диаграмма классов, представляющих команды

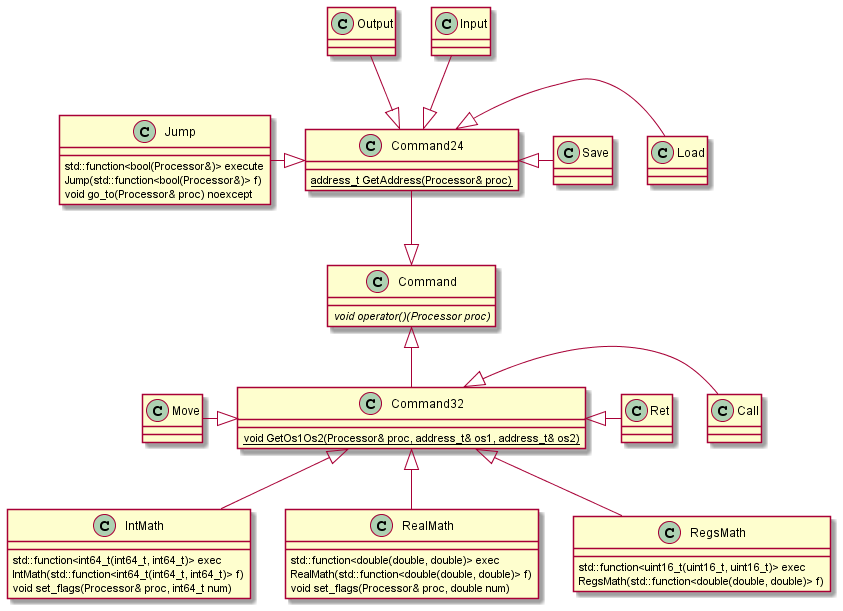


Рисунок П2 – диаграмма классов, представляющих команды