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 VM12

по дисциплине «Архитектура вычислительных систем, операционные системы»

Допущен к защите
«__» _____20__г.
Руководитель
_____

Проект выполнен
обучающимся группы ДИПРб-21
Иргалиевым А.А.
_____

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 ***Иргалиев Амин Альбертович***

Группа ***ДИПРб-21***

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

Тема ***Реализация виртуальной машины VM12***

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 _____ ***Иргалиев А.А.*** «_____»_____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.

Иргалиев А.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 Лаптев В. В.
подпись, ученая степень, звание, фамилия, инициалы

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Виртуальные машины представляют собой эмуляцию устройств на другом устройстве, позволяют запускать виртуальный компьютер (как обычную программу) с нужной операционной системой на вашем компьютере с той же или отличающейся ОС. Например, имея на своем компьютере Windows, вы можете запустить Linux или другую версию Windows в виртуальной машине и работать с ними как с обычным компьютером. Это идеальная среда для тестирования других операционных систем, включая бета-выпуски, оценки данных, зараженных вирусом, создания резервных копий операционных систем и запуска программного обеспечения и приложений на операционных системах, для которых они изначально не предназначены. Идея виртуальной машины лежит в основе целого ряда операционных систем, в частности, IBM VM/CMS (и её советского клона СВМ) и DEC VAX/VMS.

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); моделирования информационных систем с клиент – серверной архитектурой на одной ЭВМ; упрощения управления кластерами — виртуальные машины могут просто мигрировать с одной физической машины на другую во время работы; тестирования и отладки системного программного обеспечения; проверки программ на содержание вредоносного ПО.

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 и интерпретация

Зачастую виртуальная машина эмулирует работу реального компьютера. На виртуальную машину, так же как и на реальный компьютер можно инсталлировать операционную систему, у виртуальной машины так же есть BIOS, оперативная память, жесткий диск (выделенное место на жёстком диске реального компьютера), могут эмулироваться периферийные устройства. На одном компьютере может функционировать несколько виртуальных машин. Виртуальная машина исполняет некоторый машинно-независимый код (например, байт-код, шитый код, p-код) или машинный код реального процессора.

1.1.2 Архитектура виртуальной машины

Размер адреса = 16 бит. Размер памяти соответственно равно $2^{16} = 65536$.

Стек: 32 слова по 32 бита, указатель на вершину стека SP – в PSW. Реализован с помощью динамического массива.

Структура PSW (Program State Word) занимает 32 бита. Он состоит из флагов, занимающих 11 бит, 16-битного IP (instruction pointer) – регистра, содержащего адрес следующей команды, подлежащей исполнению и указателя на вершину стека - 5 бит.

Структура Flags включает в себя флаг Bool – флаг сравнений, overflow – флаг для переполнения стека, underflow – флаг, для переполнения памяти и reserved –биты для дополнительных флагов.

Типы данных: целые знаковые, беззнаковые и дробные – 32 бита (табл. 1.1).

В памяти хранятся целые, беззнаковые и дробные типы данных с размером 32 бита. Типы данных, соответствующие заданной архитектуре компьютера подробно описаны в таблице 1.1.

Таблица 1.1 — 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

Продолжение таблицы 1.1

Название	Размер (в битах)	Описание
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

На рисунке 1.1. проиллюстрирован формат команд**.**

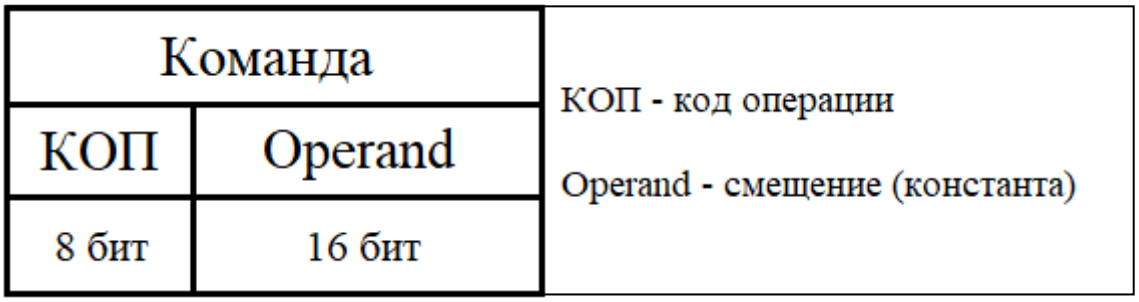


Рисунок 1.1 – Формат команд

Структура команды занимает 24 бита. Она состоит из кода операции, занимающего 8 бит и смещения (константа), которое занимает 16 бит.

Реализованный режим адресации: прямая адресация – адрес операнда содержится в поле команды address.

Реализованные команды можно разделить на 6 групп:

1. Команда stop.
2. Команды пересылки данных.
3. Команды целой арифметики.
4. Команды дробной арифметики.
5. Команды ввода и вывода.
6. Команды перехода.

Результаты команд помещаются либо в стек, либо в память.

В таблице 1.2 описаны команды арифметики.

Таблица 1.2 — Описание команд арифметики

Тип команды	**Коды операции**	**Описание**
Целочисленная арифметика	1	Сложение память + стек. Результат в стеке
	2	Сложение память + стек. Результат в памяти
	3	Вычитание память – стек. Результат в стеке
	4	Вычитание память – стек. Результат в памяти
	5	Деление память / стек. Результат в стеке
	6	Деление память / стек. Результат в памяти
	7	Умножение память * стек. Результат в стеке
	8	Умножение память * стек. Результат в памяти
Дробная арифметика	9	Сложение память + стек. Результат в стеке
	10	Сложение память + стек. Результат в памяти
	11	Вычитание память - стек. Результат в стеке
	12	Вычитание память - стек. Результат в памяти
	13	Деление память / стек. Результат в стеке
	14	Деление память / стек. Результат в памяти
	15	Умножение память * стек. Результат в стеке
	16	Умножение память * стек. Результат в памяти

В таблице 13 описаны команды переходов.

Таблица 1.3 — Описание команд переходов

Тип команды	**Коды операции**	**Описание**
Переход к подпрограмме	30	Вызов подпрограммы.
Возврат из подпрограммы	31	Косвенный прямой переход по адресу в стеке (команда устанавливает адрес для IP из вершины стека).
Прямой переход	29	В команде в явной форме указывается метка, на которую нужно перейти
Косвенный прямой	34	Переход осуществляется по адресу, который содержится в ячейке памяти
Условный переход	28	Переход осуществляется по адресу смещения, если последняя логическая операция удачная

В таблице 1.4 описаны команд сравнений.

Таблица 1.4 — Описание команд сравнений

Тип команды	**Коды операции**	**Описание**
Сравнение. Целый элемент в памяти больше элемента в стеке	22	Проверка на то, что целый элемент в памяти больше целого элемента в стеке
Сравнение. Целый элемент в памяти меньше элемента в стеке	21	Проверка на то, что целый элемент в памяти меньше целого элемента в стеке
Сравнение. Вещественный элемент в памяти больше элемента в стеке	24	Проверка на то, что вещественный элемент в памяти больше вещественного элемента в стеке
Сравнение. Вещественный элемент в памяти меньше элемента в стеке	23	Проверка на то, что вещественный элемент в памяти меньше вещественного элемента в стеке
Сравнение на равенство	25	Если значение в памяти равно значению в стеке, то флагу Bool присваивается 1, иначе 0

В таблице 1.5 описаны команды ввода – вывода.

Таблица 1.5 — Описание команд ввода – вывода

Тип команды	**Коды операции**	**Описание**
Ввод целого знакового числа	17	Введенное значение помещается в память по адресу операнда (address)
Ввод вещественного числа	19	Вывод числа из памяти по адресу (address)
Вывод целого знакового числа	18	Введенное значение помещается в память по адресу операнда (address)
Вывод вещественного числа	20	Вывод числа из памяти по адресу (address)

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).

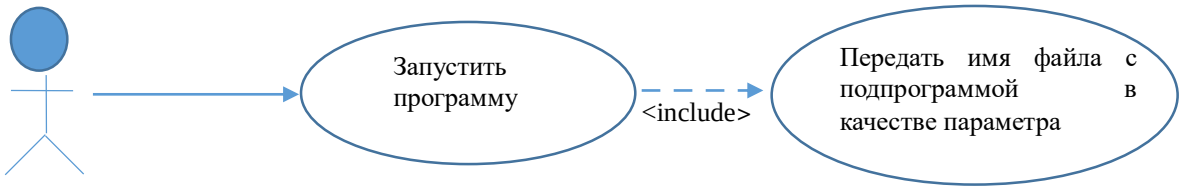


Рисунок 1.2 – Диаграмма вариантов использования

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

1.2.1 Формат исполняемого файла

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

Команды имеют следующий вид: **c cop address** (Например, c 1 1000).

c – объявление команды,

cop – код операции,

address – это адрес, используемый в качестве операнда.

Помимо объявления команд в файле могут быть следующие директивы:

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

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

f Fval – загрузить в память число с плавающей точкой Fval,

p IP – установить адрес IP (адрес первой исполняемой команды) на address,

s stop - Загрузить команду stop (для остановки программы).

В файле могут находиться комментарии, описывающие работы подпрограммы с помощью тега «/».

1.2.2 Диаграмма классов

Абстрактный класс Command определяет интерфейс для переопределения производными классами. Производные классы – классы арифметических действий, логических операций, переходов, пересылок и класс стоп (П.3).

1.2.3 Алгоритм работы загрузчика

Дано: Процессор, строка с названием файла

Открыть текстовый исполняемый файл с названием ***VM***

uint16_t **StartAddress** = 0

Если файл открыт, то

| пока не конец файла

| | получить строку **Do** из файла

| | обрезать строку до тега «/»

| | Если строка **Do** не пуста, то

| | | char **Make** = **Do**[0]

| | | Вырезать из строки **Do** первый символ и пробел

| | | Switch (Make)

| | | | case ‘c’: Вызвать алгоритм **1.2.8**

| | | | case ‘f’: Вызвать алгоритм **1.2.9**

| | | | case ‘i’: Вызвать алгоритм **1.2.10**

| | | | case ‘p’: Считать из строки **Do** значение **IP**

| | | | case ‘a’: Считать из строки **Do** значение **StartAddress**

| | | | case ‘s’: Вызвать команду стоп, чтобы остановить работу программы

1.2.4 Алгоритм работы процессора

Делать

| Переменной **CmdReg** структуры CommandStruct присвоить команду по адресу команды в памяти

| Выполнить команду

| **IP** += 3

Пока не загружена команда **stop**

Пояснение **IP** += 3: данная строка означает то, что после выполнения команды IP меняется на 3 байта (Команда – 24 бита), то есть устанавливается IP следующей команды.

1.2.5 Алгоритм арифметической команды – результат в памяти

DataType **result**

result.val присвоить операнд из памяти (арифметическое действие) операнд из стека по адресу **SP**

добавить число в память по адресу **Operand (**Алгоритм **1.2.3**)

1.2.6 Алгоритм арифметической команды – результат в стеке

DataType **result**

result.val присвоить операнд из памяти (арифметическое действие) операнд из стека по адресу **SP**

добавить число в стек по адресу **SP**

1.2.7 Алгоритм пересылки числа из памяти в стек

Если в стеке меньше 4 байт свободно, то

| Присвоить флагу переполнения стека = 1

Иначе

| Присвоить флагу переполнения стека = 0

| **SP** -= 4

| Добавить число из памяти по адресу **Operand** в стек по адресу **SP**

1.2.8 Алгоритм пересылки числа из стека в память

Если **SP** > 65535, то

| Присвоить флагу переполнения памяти = 1

Иначе

| Присвоить флагу переполнения памяти = 0

| Добавить число из стека по адресу **SP** в память по адресу **Operand** (Алгоритм 1.2.3)

| **SP** += 4

1.2.9 Алгоритм условного перехода

Если флаг **Bool** = 1, то

| выполняется прямой переход, т.е. **IP** = **Operand** – 3, чтобы в цикле работы процессора

| | при переходе на следующую команду вызывалась нужная

1.2.10 Алгоритм перехода к подпрограмме

Если в стеке меньше 4 байт свободно, то

| Присвоить флагу переполнения стека = 1

Иначе

| Присвоить флагу переполнения стека = 0

| **SP** -= 2

| Добавить адрес **IP** в стек по адресу **SP**

| выполняется прямой переход, т.е. **IP** = **Operand** – 3

1.2.11 Алгоритм возврата из подпрограммы

Если SP > 65535, то

| Присвоить флагу переполнения памяти = 1

Иначе

| Присвоить флагу переполнения памяти = 0

| Присвоить **IP =** адрес, лежащий в стеке по адресу **SP, - 3**

| **SP** += 2

1.2.12 Алгоритм сравнения

Если значение в памяти по адресу operand равно значению в стеке по адресу SP

| Присвоить флагу Bool = 1

Иначе

| Присвоить флагу Bool = 0

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 7. Язык – C++.

2 РАБОЧИЙ ПРОЕКТ

2.1 Общие сведения о работе системы

Программный продукт разработан в интегрированной среде CodeBlocks (версия 17.12) на языке C++ (стандарт C++ 14 и более поздние), с использованием компилятора MinGW GCC (версия 8.2.0-3) (Visual Studio 2017). Программа работает под управлением операционной системы Windows 7 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:

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

Программа имеет следующие функциональные ограничения:

- можно передавать только одно название файла в качестве параметра.

2.3 Инсталляция и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера папку VM, содержащую сам интерпретатор и исполняемые файлы (Arifm.txt, Transition.txt, EqualAndIfgoto.txt, FirstFloatGreater.txt, InSubProgram.txt и Transmission.txt).
2. Убедиться, что в папке с программой находится файл KursVM.exe.
3. Запустить файл KursVM.exe из командной строки.
4. Для проверки работоспособности программы, передать имя одного из вышеперечисленных файлов.
5. Сменить шрифт в настройках консольного окна на Lucida Console для отображения кириллицы и увеличить размер шрифта при необходимости (рекомендуется 16 или 18 пунктов).

Все файлы, лежащие в папке «VM», представлены в таблице 2.1

Таблица 2.1 – Файлы из папки «VM»

Название	**Описание**
«KursVM»	Основной исполняющий файл программы
Arifm.txt	Файл с примером подпрограммы
Transition.txt	Файл с примером подпрограммы
EqualAndIfgoto.txt	Файл с примером подпрограммы
FirstFloatGreater.txt	Файл с примером подпрограммы
InSubProgram.txt	Файл с примером подпрограммы
Transmission.txt	Файл с примером подпрограммы

2.4 Описание программы

В таблице 2.2 представлено описание класса Command

Таблица 2.2 – Описание класса Command

Поле	**Тип**	**Назначение**
Cpu	CPU&	Процессор
Метод		**Описание**
Command(CPU& Cpu)		Конструктор класса – задает текущий процессор
virtual void operator()(CPU& Cpu)		Выполнение команд
~Command() {}		Деструктор класса – высвобождение памяти

В таблице 2.3 представлено описание структуры Flags

Таблица 2.3 – Описание структуры Flags

Поле	**Тип**	**Назначение**
Bool	uint8_t	флаг для сравнений и переходов
overflow	uint8_t	флаг, для переполнения стека
underflow	uint8_t	флаг, для переполнения памяти
reserved	uint16_t	биты для дополнительных флагов

В таблице 2.4 представлено описание структуры ProgramStateWord

Таблица 2.4 – Описание структуры ProgramStateWord

Поле	**Тип**	**Назначение**
IP	Address(uint16_t)	адрес команды в памяти
SP	Address(uint16_t)	вершина стека
flags	Flags	флаги программы, описанные в таблице 2.2

В таблице 2.5 представлено описание структуры CommandStruct

Таблица 2.5 – Описание структуры CommandStruct

Поле	**Тип**	**Назначение**
cop	uint8_t	код операции (номер команды)
Operand	uint16_t	адрес операнда в памяти

В таблице 2.6 представлено описание класса CPU

Таблица 2.6 – Описание класса CPU

Поле	**Тип**	**Назначение**
Stack	uint8_t*	стек из 128 элементов
CmdReg	CommandStruct	структура команды (коп, адрес операнда), описанная в таблицу 2.4
flags	Flags	флаги программы, описанные в таблице 2.2
PSW	ProgramStateWord	структура, описанная в таблицу 2.3
Метод		**Описание**
CPU()		конструктор класса – создает и обнуляет память
~CPU()		деструктор класса – высвобождение памяти
void CPU_run() noexcept		запуск процессора
uint8_t get_cop() const noexcept		получить код операции команды
Address get_Operand() const noexcept		получить адрес операнда в памяти
void LoadCommand() noexcept		загрузка команд по адресу команды в памяти
void push_stack (const CommandStruct& cmd, const Address& address) noexcept		Добавление команды в стек
void push_stack(const DataType& num, const Address& address) noexcept		Добавление операнда в стек
void push_stack(const Address& Val, const Address& To) noexcept		Добавление адреса в стек
DataType GetNumOfStack(const Address& address) const noexcept		Считывание операнда из стека
CommandStruct GetCmdOfStack(const Address& address) const noexcept		Считывание команды из стека
Address GetAddrOfStack(const Address& address) const noexcept		Считывание адреса из стека

В таблице 2.7 представлено описание объединения DataType

Таблица 2.7 – Описание объединения DataType

Поле	**Тип**	**Назначение**
Ival	int32_t	целое знаковое/ беззнаковое число
Fval	float	вещественное число
UI8Type[4];	uint8_t	массив байт числа

В таблице 2.8 представлено описание объединения LocalAddress

Таблица 2.8 – Описание объединения LocalAddress

Поле	**Тип**	**Назначение**
val	uint16_t	адрес
UI8Type[2];	uint8_t	массив байт адреса

В таблице 2.9 представлено описание объединения LocalCommandStruct

Таблица 2.9 – Описание объединения LocalCommandStruct

Поле	**Тип**	**Назначение**
cmd	CommandStruct	структура команды (коп, адрес операнда), описанная в таблицу 2.4
UI8Type[3];	uint8_t	массив байт команды

В таблице 2.10 представлено описание класса Memory

Таблица 2.10 – Описание класса Memory

Поле	**Тип**	**Назначение**
memory	uint8_t*	Память из 65538 элементов
Метод		**Описание**
Memory()		конструктор класса – создает и обнуляет память
~Memory()		деструктор класса – высвобождение памяти
void push_memory(const CommandStruct& cmd, const Address& address) noexcept		Добавление команды в память
void push_memory(const DataType& num, const Address& address) noexcept		Добавление операнда в память
void push_memory(const Address& Val, const Address& To) noexcept		Добавление адреса в память

Продолжение таблицы 2.10

Метод	Описание
DataType GetNum(const Address& address) const noexcept	Считывание операнда из памяти
CommandStruct GetCmd (const Address& address) const noexcept	Считывание команды из памяти
Address GetAddr(const Address& address) const noexcept	Считывание адреса из памяти

2.5 Разработанные меню и интерфейсы

Разработанная программа является консольной. Она анализирует предоставленный ей файл, загружает инструкции в оперативную память и выполняет их из неё.

На рисунке 2.1 показан вывод работы программы при успешном запуске.

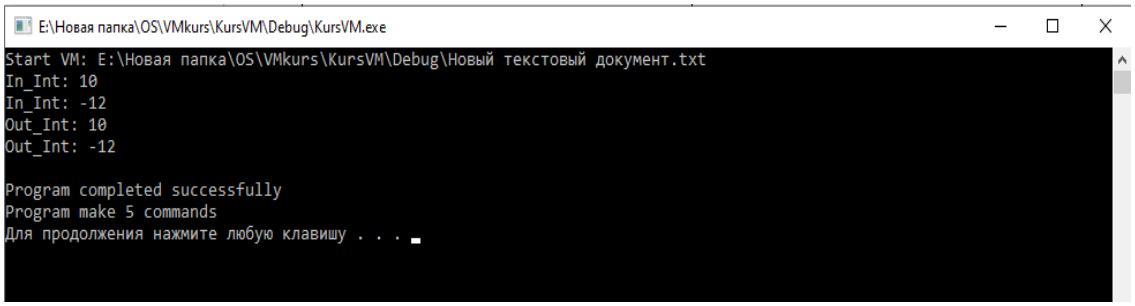


Рисунок 2.1 – Успешный запуск программы

2.6 Сообщения системы

В таблице 2.11 приведены сообщения системы.

Таблица 2.11 – Сообщения системы

№ п\п	Сообщение	Причина возникновения, способ устранения
1	File with this name not found	Файл с заданным именем не найден
2	Program completed successfully	Программа успешно завершена

В случае появления других сообщений следует обратиться к разработчику.

3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

1. Запустить командную строку, перейти в директорию с программой, ввести KursVM.exe и имя файла с подпрограммой как параметр командной строки.
1. Через пробел ввести имя несуществующего файла и запустить программу.
2. Убедиться, что было выведено сообщение 1 (табл. 2.10).
3. Запустить программу через командную строку и указать имя существующего файла.
4. Убедиться, что после работы программы было выведено сообщение 2 (табл. 2.10).

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования был разработан интерпретатор одноадресной стековой виртуальной машины.

Программа отвечает поставленным требованиям и может быть использована для выполнения программ данной архитектуры компьютера.

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. Григорьев В. Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 1. Программная архитектура. – М., ГРАНАЛ, 1993 – с.346, ил. 87.

ПРИЛОЖЕНИЕ 1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на выполнение курсового проекта

по дисциплине «Архитектура вычислительных систем, операционные системы»

Направление 090304 – Программная инженерия

Исполнитель: обучающийся гр. ДИПРб21 **Иргалиев А.А.**

Тема: Реализация виртуальной машины VM12»

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 «Архитектура вычислительных систем, операционные системы».
- Распоряжение по кафедре АСОИУ №____ от «___»_____ 201__г.

4 Плановые сроки выполнения – осенний семестр 2018/19 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:

- предоставление пользователю результатов работы программы;
- выполнение программы из исходного файла;
- **интерфейс программы:** текст русский, шрифт кириллический, заголовки, термины и другая важная информация выделены цветом.

Система имеет функциональные ограничения: можно передавать только одно название файла в качестве параметра.

5.2 Требования к эксплуатационным характеристикам

Программа не должна аварийно завершаться при любых действиях пользователя

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

5.3 Требования к программному обеспечению:

Средства разработки: интегрированная среда Code::Blocks 17.12, язык C++ (стандарт C++ 14 и более поздние) (Visual Studio 2017).

Операционная система: Windows 7 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 20 МБ свободного места на диске;
- Дисковод CD-ROM/DVD-ROM.

5.5 Требования к реализации: требовалось разработать стековую виртуальную машину, один операнд в памяти, другой в стеке, которая имеет следующую архитектуру:

PSW = IP + SP + Flags, 16 + 5 + 11 = 32 бита

Стек: 32 слова по 32 бита, указатель стека SP – в PSW

Память: байтовая, размер адреса – 1 бит

Типы данных:

- Целые знаковые, беззнаковые – 4 байта
- Дробные – 4 байта

Структура команды: 3 байта

- 24 бит: КОП – 8 бит, смещение (константа) – 16 бит
- Результат – в стеке или в памяти

Пересылки/загрузки/сохранения

Арифметика целая

Арифметика дробная

Переходы:

- Безусловный: прямой, относительный, косвенный прямой
- Условный – то самое, только с проверкой флагов
- К подпрограмме – адрес возврата в стеке
- Возврат из подпрограммы – косвенный прямой переход по адресу в стеке

6 Стадии и этапы разработки

6.1 Эскизный проект (ЭП)

- Анализ предметной области.
- Подготовка проектной документации.

6.1 Технический проект (ТП)

- Разработка структур и форм представления данных.
- Разработка структуры программного комплекса.
- Подготовка пояснительной записки.

6.2 Рабочий проект (РП)

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

6.3 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

ПРИМЕРЫ ПРОГРАММ

Пример программы с арифметикой (исполняемый файл Arifm.txt)

a 100 /установка начального адреса 100

i 120 /ввод константы - целое число 120 по адресу 100 в память

i 100 /ввод константы - целое число 100 по адресу 104 в память

c 26 104 /пересылка числа в стек, лежащего по адресу 104 в памяти

c 2 100 /сложение чисел по адресу 100 в памяти + по адресу SP (сумма остается в памяти)

c 18 100 /вывод суммы по адресу 100

s

p 108 /установка IP

/Выходные данные: Out_Int: 220

Пример программы со сравнением(равенство) и условным переходом (исполняемый файл EqualAndIfgoto.txt)

a 100 /Начальный адрес 100

i 30 /целочисленная константа(100 адрес)

i 30 /целочисленная константа(104 адрес)

f 55.5 /вещественная константа108 адрес

c 26 104 /пересылка элемента с памяти в стек

c 25 100 /Если число в стеке и в памяти по адресу 100 равны, то логический флаг = 1, иначе 0

c 28 2000 /условный переход в адрес 2000, если логический флаг = 1

c 18 100 /если переход не состоялся вывод целого числа из памяти по адресу 100

s /команда стоп

a 2000 /адрес 2000, куда выполнен переход

c 20 108 /вывод элемента по адресу 108

s /команда стоп

p 112 /установка IP

/Выходные данные: Out_Float: 55.5

Пример программы со сравнением(значение в памяти больше значения в стеке) (исполняемый файл FirstFloatGreater.txt)

a 1000 /начальный адрес 1000

f 333.333 /вещественная константа(1000 адрес)

f 111.111 /вещественная константа(1004 адрес)

c 26 1004 /пересылка в стек значения с адресом 1004 в памяти

c 24 1000 /если значение в памяти по адресу 1000 больше значения в стеке, то логический флаг Bool = 1

c 28 500 /если Bool = 1 совершить переход по адресу 500

s /если Bool = 0 просто завершить программу

a 500 /адрес перехода

c 20 1000 /вывести вещественное число по адресу в памяти 1000

s /команда стоп, завершение программы

p 1008

/Выходные даные: Out_Float: 333.333

Пример программы с переходом к подпрограмме и возврата из неё (исполняемый файл InSubProgram.txt)

a 1000 /начальный адрес

i 200 /целочисленная константа(адрес 1000)

i 100 /целочисленная константа(адрес 1004)

c 26 1004 /пересылка значения в памяти по адресу 1004 в стек

c 30 500 /переход к подпрограмме по адресу 500

c 2 1000 /сумма: значение в памяти + значение в стеке

c 18 1000 /вывод суммы

s

a 500 /адрес подпрограммы

c 33 1008 /пересылка адреса со стека в память по адресу 1008, для того, чтобы совершать арифметические действия в подпрограмме

c 6 1000 /деление: значение в памяти / значение в стеке

c 18 1000 /вывод деления

c 32 1008 /пересылка с памяти по адресу в 1008 обратно в стек

c 31 0 /возврат из подпрограмы

p 1008 /установка IP

/Выходные данные: Out_Int: 2, Out_Int: 102

Пример программы с прямым переходом (исполняемый файл Transition.txt)

a 300 /начальный адрес

f 123.321 /вещественная константа(адрес 300)

i 474 /целочисленная константа(адрес 304)

c 17 500 /команда ввода целого числа в память по адресу 500

c 26 500 /пересылка числа по адресу 500 в стек

c 29 1000 /прямой переход по адресу 1000

c 18 304 /вывод целочисленной константы по адресу 304 (не состоится)

c 2 304 /сумма: значение по адресу 304 в памяти + значение в стеке

c 18 304 /вывод суммы (не состоится)

a 1000 /адрес перехода

c 20 300 /вывод вещественной константы из памяти по адресу 300 (единственное что состоится)

s

p 308

/Выходные данные: Out_Float: 123.321

Пример программы с пересылками

(исполняемый файл Transmission.txt)

a 0 /начальный адрес 0

i 34 /целочисленная константа(адрес 0)

f 12.21 /вещественная константа(адрес 4)

c 26 0 /пересылка числа из памяти по адресу 0 в стек

c 27 1004 /пересылка числа из стека в память по адресу 1004

c 18 1004 /вывод целого числа из памяти по адресу 1004

c 26 4 /пересылка числа из памяти по адресу 4 в стек

c 27 500 /пересылка числа из стека в память по адресу 500

c 20 500/вывод вещественного числа из памяти по адресу 500

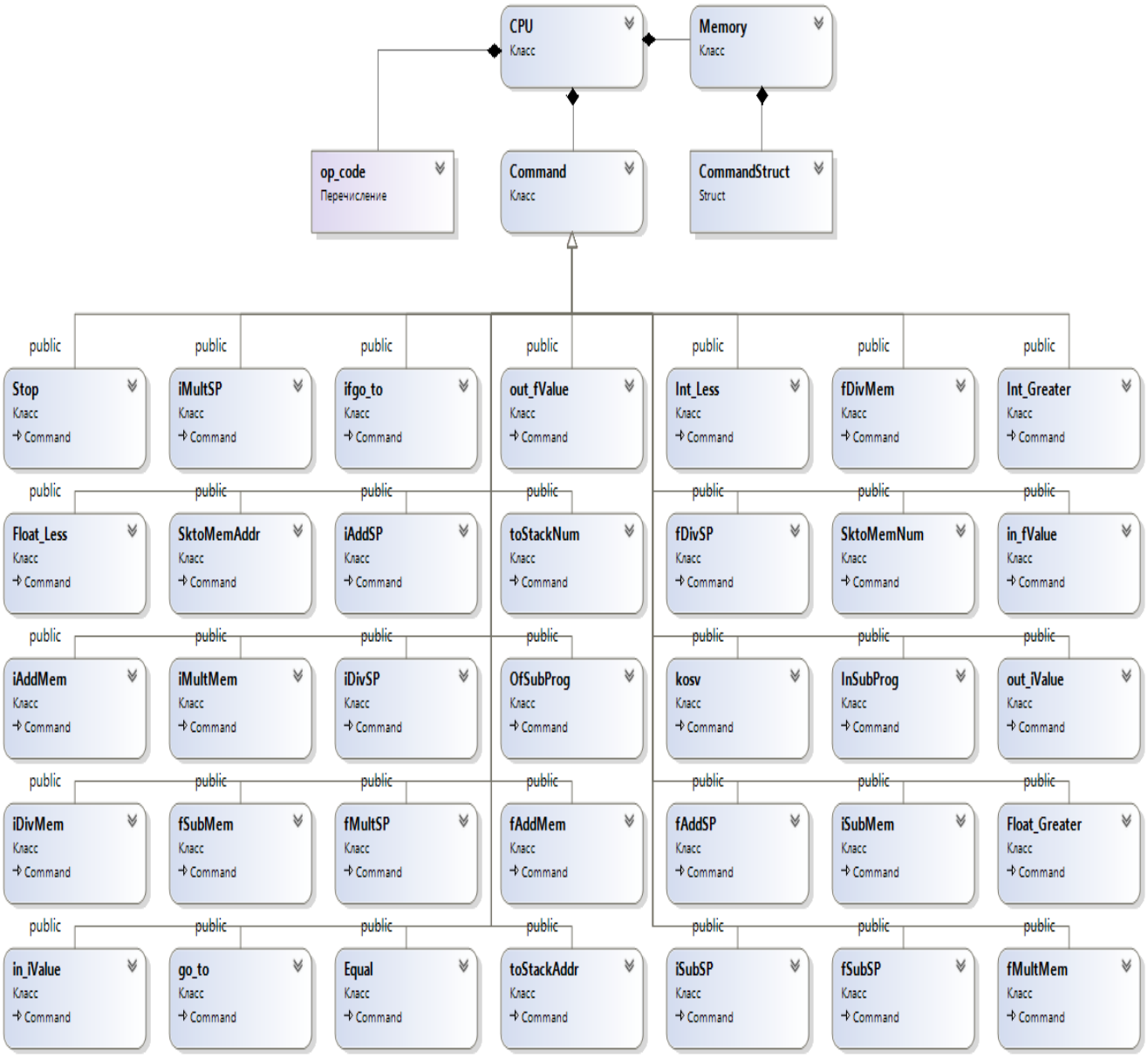
s

p 8

/Выходные данные: Out_Int: 34, Out_Float: 12.21

ПРИЛОЖЕНИЕ 3

ДИАГРАММА КЛАССОВ



П.3 - диаграмма классов