

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

УЧЕБНАЯ СИСТЕМА ПРОЕКТИРОВАНИЯ

МИКРОПРОГРАММИРУЕМЫХ ЭВМ.

ПРОГРАММНО-ЛОГИЧЕСКАЯ МОДЕЛЬ

Студент


(подпись, дата)

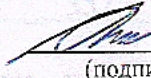
Резунов Вячеслав Дмитриевич
(ФИО полностью)

Руководитель

02.06.22
(подпись, дата)

Н. Н. Коннов
(фамилия, инициалы)

Нормоконтролёр

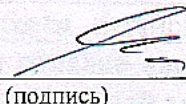

(подпись, дата)

02.06.22

А. С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.22 № 13)

Заведующий кафедрой

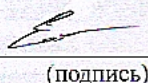

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

07.06.2022 № 2)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

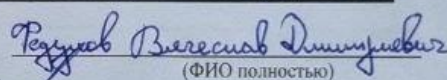
УЧЕБНАЯ СИСТЕМА ПРОЕКТИРОВАНИЯ

МИКРОПРОГРАММИРУЕМЫХ ЭВМ.

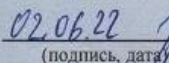
ПРОГРАММНО-ЛОГИЧЕСКАЯ МОДЕЛЬ

Студент


(подпись, дата)


(ФИО полностью)

Руководитель


(подпись, дата)

Н. Н. Коннов
(фамилия, инициалы)

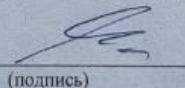
Нормоконтролёр


(подпись, дата)

А. С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от _____ г. № _____)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от

№ _____)

Секретарь ГЭК

(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» *апреля* 20 *22* г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Федулов Вячеслав Дмитриевич гр. 18ВВ1 факультета ФВТ

2. Тема работы Учебная система проектирования микропрограммируемых ЭВМ.
Программно-логическая модель

Тема утверждена приказом ПГУ № 357/о от " 26 " апреля 20 22 г.

3. Руководитель бакалаврской работы профессор, к.т.н. Коннов

4. Задание на работу (назначение разработки, исходные данные и т.п.) Система предназначена для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительной техники с целью выработки компетенций по проектированию и отладке алгоритмов работы микропрограммируемых процессоров и других средств вычислительной техники.

В проекте должен быть разработана программно логическая модель 16-разрядной микропрограммируемой ЭВМ. Включающая: процессор, оперативная память емкостью до 16 Кбайт. Процессор должен обеспечивать:

1. Произвольный формат микрокоманд
2. Формат микрокоманд – системы команд микроЭВМ PDP-11

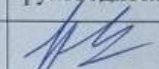
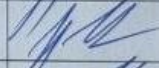
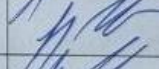
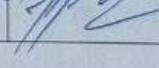
Программно-логическая модель должна быть кроссплатформенной и должна исполняться в среде следующих ОС: Windows7, 10, Linux Альт Образование, Ubuntu.

Язык программирования – Java.

Программно-логическая модель должна функционировать в потактовом и автоматическом режиме с возможностью интерактивного контроля и отладки информации.

5. Перечень подлежащих разработке вопросов анализ темы, аналогов, изучение структуры базовой ЭВМ и системы команд PDP11, разработка архитектуры приложения, выбор языка программирования и технологий на основе которых будет проектироваться система, разработка программно-логической модели, пользовательского интерфейса, тестирование работы системы, изучение требований к организации рабочего места при работе на ПЭВМ.

6. Календарный график выполнения работы

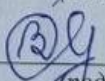
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	10%	26.04.2022-29.04.2022	
2	Разработка архитектуры приложения	10%	30.04.2022-03.05.2022	
3	Выбор средств разработки	5%	04.05.2022-05.05.2022	
4	Разработка ПЛМ	35%	06.05.2022-16.05.2022	
5	Разработка графического интерфейса	10%	17.05.2022-21.05.2022	
6	Тестирование и отладка	15%	22.05.2022-25.05.2022	
7	Оформление выпускной работы	15%	26.05.2022-31.05.2022	

Дата выдачи задания " 26 " апреля 20 22 г.

Руководитель бакалаврской работы

 26.04.22
(подпись, дата)

Задание к исполнению принял студент

 26.04.22
(подпись, дата)

Работу к защите допустить
Декан факультета

 02.06.2022
(подпись, дата)

Реферат

Выпускная квалификационная работа содержит 180 листов, 40 рисунков, 19 таблиц, 13 источников, 7 приложений.

ЭМУЛЯТОР, МИКРОПРОЦЕССОР, МИКРОЭВМ, *JAVA*, K1804, *PDP11*, УЧЕБНЫЙ ТРЕНАЖЕР, УЧЕБНАЯ СИСТЕМА, КРОССПЛАТФОРМЕННЫЙ, *SWING*, ИНТЕРФЕЙС, ПМЛ.

Цель работы – разработка программно-логической модели учебной системы проектирования микропрограммируемых ЭВМ. Система предназначена для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительной техники с целью выработки компетенций по проектированию и отладке алгоритмов работы микропрограммируемых процессоров и других средств вычислительной техники.

В результате проделанной работы была разработана кроссплатформенная модель 16-разрядной микропрограммируемой ЭВМ, включающая: процессор и оперативную память емкостью до 16 Кбайт. Функционирует в потактовом и автоматическом режиме с возможностью интерактивного контроля и отладки информации. Система разработана на языке программирования *Java* с использованием графической библиотеки *Swing*.

					ПГУ 1.09.03.01.41.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Федулов В.Д.			Учебная система проектирования микропрограммируемых ЭВМ. Программно-логическая модель Пояснительная записка		
Провер.		Коннов Н.И.					
Н. Контр.		Бычков А.С.					
Утверд.		Митрохин М.А.					
					Лит.	Лист	Листов
						2	180
					ФВТ, кафедра ВТ, гр. 18ВВ1		

Содержание

Перечень сокращений и обозначений	7
Введение	8
1 Учебная система проектирования микропрограммируемых ЭВМ.	
Программно – логическая модель	10
1.1 Анализ темы	10
1.1.1 Обоснование необходимости разработки	10
1.1.2 Анализ аналогов	11
1.1.3 Постановка задач разработок	12
1.1.4 Структура базовой микроЭВМ	13
1.2 Архитектура системы проектирования	20
1.3 Выбор инструментов разработки	22
1.4 Описание программно-логической модели	24
1.4.1 Класс <i>iTact</i>	25
1.4.2 Класс <i>gr1804</i>	31
1.4.3 Класс <i>Protocol</i>	34
1.4.4 Класс <i>Listing</i>	35
1.4.5 Класс <i>Memory</i>	37
1.4.6 Класс <i>MicroCommandText</i>	41
1.4.7 Класс <i>Decoration</i>	43
1.4.8 Клас <i>Tact</i>	45
1.4.9 Класс <i>GlobalVars</i>	58
1.5 Описание организации пользовательского интерфейса	62
1.6 Тестирование работы системы	68
2 Безопасность жизнедеятельности	74
2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при реализации разрабатываемого проекта . . .	75
2.2 Классификация и категорирование помещения	76

2.3 Санитарно-гигиеническая характеристика помещения, отдела офиса	78
2.3.1 Требования к помещениям для работы с ПЭВМ	78
2.3.2 Организация рабочих мест	79
2.3.3 Микроклимат рабочего помещения	81
2.3.4 Шум и вибрация	82
2.3.5 Ионизирующие и неионизирующие излучения	83
2.3.6 Освещение рабочего места	83
2.4 Электробезопасность	84
2.5 Пожарная безопасность	86
2.6 Расчет общего искусственного освещения помещения, где работают на ПЭВМ	88
Заключение	93
Список использованных источников	94
Приложение А – Листинг программы	96
Приложение Б – <i>UML</i> диаграмма использования	162
Приложение В – <i>UML</i> диаграмма классов	164
Приложение Г – <i>UML</i> диаграмма последовательностей	166
Приложение Д – <i>UML</i> диаграмма деятельности	168
Приложение Е – Презентация	170
Приложение Ж – Свидетельства	179

Перечень сокращений и обозначений

АЛУ – арифметико логическое устройство

К1804 – микропроцессорный комплект серии К1804

МК – микрокоманда

ОС – операционная система

ПО – программное обеспечение

ПЭВМ – персональная электронная вычислительная машина

РК – регистр команд

ЭВМ – электронная вычислительная машина

ЭМПС – эмулятор микропроцессорной системы

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		8

Введение

В настоящее время широкое распространение компьютерных и информационных технологий находят применение в сфере образования. Без компьютерных систем трудно представить процесс обучения современного специалиста в любой области. Использование компьютерных технологий в образовательном процессе высших учебных заведений и условиях дистанционного образования предоставляют обучающимся современные средства, которые способствуют улучшению процессам интеллектуального развития, а также позволяют отточить теоретические знания на практике, нарабатывая профессиональные навыки.

Эмуляторы микропроцессорных систем являются незаменимым инструментом в образовательном процессе [1]. Эмуляция – это использование прикладной программы или устройства, либо их комбинацией, для имитации поведения другой программы или устройства. Эмуляция дает возможность выполнить компьютерную программу на платформе, отличной или идентичной той, для которой она была написана в оригинале. Сам процесс выполнения также называют эмуляцией. Симуляция отличается тем, что лишь воспроизводит поведение программы, при эмуляции целью является точное моделирование состояния имитируемой системы, для работы оригинального машинного кода.

Эмуляторы необходимы для самых различных целей. Например, при разработке новых вычислительных систем, сначала разрабатывается эмулятор, для выполнения программ, написанных для еще не существующих компьютеров. Такой подход позволяет оценить систему команд и разработать базовое программное обеспечение еще до того, как будет реализовано соответствующее оборудование. Так же эмулятор полезен для исследования ситуаций, трудно воспроизводимых с помощью аппаратуры.

Одно из главных преимуществ эмуляторов — это возможность многократного воспроизведения рабочих ситуаций. Если необходимо найти

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

неисправность на определённом участке программы, то данный участок можно повторять снова и снова, пока ошибка не будет обнаружена.

Также, загрузка программы в эмулятор происходит быстрее, чем в реальную аппаратуру, что удобно при разработке и тестировании программного обеспечения.

Полностью программный эмулятор самый безопасный, так как в независимости от того, какой код загрузят в эмулятор, воздействовать на настоящий процессор он все равно не сможет. Так же данный подход позволяет эмулировать произвольные архитектуры процессоров. Например, на оборудовании с 32 – битным процессором можно эмулировать 64 – битный процессор или процессор мобильного телефона.

В данной работе рассматривается разработка кроссплатформенного эмулятора 16 – разрядной микропрограммируемой ЭВМ на базе разрядно – модульного комплекта серии K1804 и системы команд *PDP11*.

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

1 Учебная система проектирования

микропрограммируемых ЭВМ. Программно-логическая модель

1.1 Анализ темы

1.1.1 Обоснование необходимости разработки

На этапе начального освоения вычислительной техники главными требованиями выступают доступность и наглядность для понимания, наличие достаточной детализации рассматриваемых операций, но вместе с тем, и возможность демонстрации работы оборудования в задачах реальной практической сложности [2]. Данные требования противоречивы, потому что принимаются решения, основа которых лежит в желании продемонстрировать все преимущества устройства. При этом серьезный урон наносится методическим возможностям эмулятора, что недопустимо для учебного процесса и зачастую сопровождается отказом от его использования.

В этом смысле значительный интерес представляет секционированный микропроцессорный комплекс 1804, с системой команд *PDP11*. В настоящее время его нельзя назвать современным, но его система команд и структура являются классической, в большей степени повторяет испытанные годами технические решения больших ЭВМ, достаточно гибкая и логически мощная. Это делает понятными базовые системы команд обеспечивая при этом достаточно широкие возможности для демонстрации основных приемов программирования и адресации операндов. Поэтому для вступительного обучения микропроцессорной технике отказываться от создания эмулятора для данного микропроцессорного комплекса было бы неправильно.

Так же принятые правительственные решения о переходе органов власти и бюджетных учреждений, в том числе и ВУЗов, на импортозамещение используемого ПО стимулируют вести работу по переводу учебных программ на отечественное либо свободно распространяемое ПО.

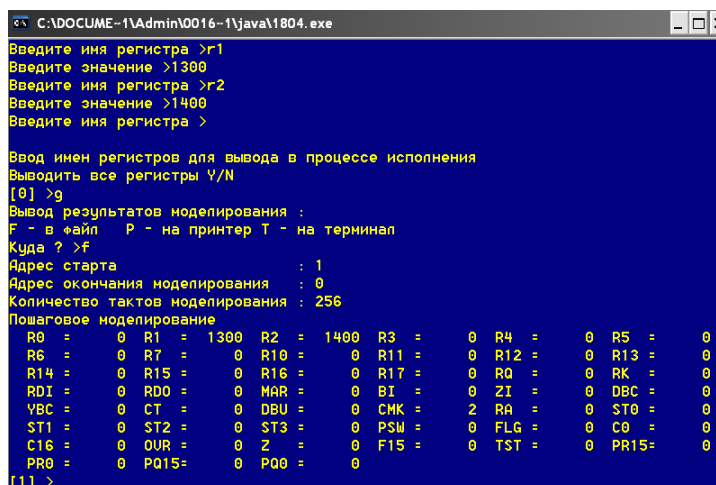
					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

1.1.2 Анализ аналогов

Основное требование при проектировании ЭМПС состоит в том, чтобы создаваемый эмулятор был эффективнее своего аналога, т. е. превосходил его по качеству функционирования, удобству использования и технико-экономической целесообразности.

Таким образом, задача сравнения аналогов проектируемого эмулятора сводится к тому, чтобы выбрать вариант программного решения, имеющий наибольшее количество функциональных возможностей.

На кафедре вычислительной техники ПГУ были разработаны и активно применяются в лабораторном практикуме и курсовом проектировании *DOS* и *Windows* версии программных симуляторов шестнадцатиразрядного процессора с микропрограммным управлением [3, 4, 5]. На рисунке 1.1 представлена *DOS* версия эмулятора.



```

C:\DOCUMENTS\1\ADMIN\0016-1\java1804.exe
Введите имя регистра >r1
Введите значение >1300
Введите имя регистра >r2
Введите значение >1400
Введите имя регистра >

Ввод имен регистров для вывода в процессе исполнения
Выводить все регистры Y/N
[0] >g
Вывод результатов моделирования :
F - в файл P - на принтер T - на терминал
Куда ? >F
Адрес старта : 1
Адрес окончания моделирования : 0
Количество тактов моделирования : 256
Пошаговое моделирование
R0 = 0 R1 = 1300 R2 = 1400 R3 = 0 R4 = 0 R5 = 0
R6 = 0 R7 = 0 R10 = 0 R11 = 0 R12 = 0 R13 = 0
R14 = 0 R15 = 0 R16 = 0 R17 = 0 RQ = 0 RK = 0
RDI = 0 RDO = 0 MAR = 0 BI = 0 ZI = 0 DBC = 0
YBC = 0 CT = 0 DBU = 0 CMK = 2 RA = 0 ST0 = 0
ST1 = 0 ST2 = 0 ST3 = 0 PSW = 0 FLG = 0 C0 = 0
C16 = 0 OUR = 0 Z = 0 F15 = 0 TST = 0 PR15 = 0
PR0 = 0 P015 = 0 PQ0 = 0
[1] >
```

Рисунок 1.1 – *DOS* версия эмулятора

У данной версии имеются следующие недостатки: отсутствие графического интерфейса, отсутствие возможности работы с памятью, запускалась только под ОС – *DOS*.

На рисунке 1.2 представлена *Windows* версия эмулятора. В данной версии эмулятора были исправлены некоторые недостатки предыдущей версии, а именно появился графический интерфейс.

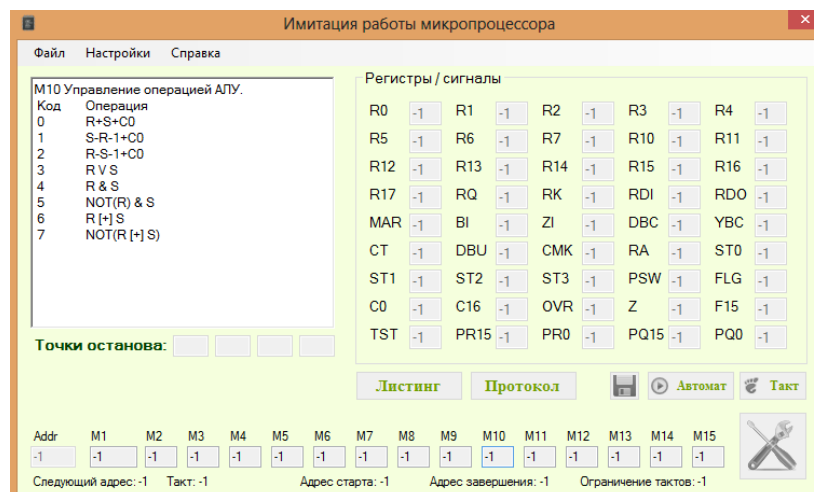


Рисунок 1.2 – Windows версия эмулятора

Однако проблемы в виде отсутствия возможности работы с памятью и привязанность к операционной системе остались. Так же у новой версии появились проблемы с функционалом, который работал некорректно, не было возможности изменить следующую микропрограмму в процессе отладки и программа загружалась только из файла, который был заранее подготовлен с помощью макроса в *MS Visio*.

1.1.3 Постановка задач разработок

В ходе дипломного проекта необходимо разработать программу, предназначенную для использования в учебном процессе для изучения и отработки алгоритмов функционирования процесса с системой команд *PDP11*, в аппаратной среде, построенной на секционированном МПК1804.

Разрабатываемый эмулятор должен выполнять следующие функции:

- чтение и сохранение файлов в формате “.pzu”;
- формирование и вывод листинга программы;
- сохранение листинга в файле формата “.txt”;
- протоколирование программы;
- редактирование программы;
- вывод выполненной и следующей микрокоманды с возможностью редактировать;

- точки останова;
- корректное взаимодействие ядра микропроцессора с памятью;
- редактирование, загрузка и сохранение памяти;
- выбор режима отладки: поэтапная или автоматическая;
- интеграция графического редактора в эмулятор.

Требования к разрабатываемому ПО:

- система должна функционировать на ПЭВМ в операционных средах *Windows* и *Linux*;
- дизайн пользовательского должен адаптироваться под основные разрешения экранов ПК;
- интерфейс должен быть прост и понятен.

1.1.4 Структура базовой микроЭВМ

Согласно [6] базовый состав процессора микроЭВМ входят следующие блоки (Рисунок 1.3): блок обработки данных, интерфейсный блок и блок микропрограммного управления.

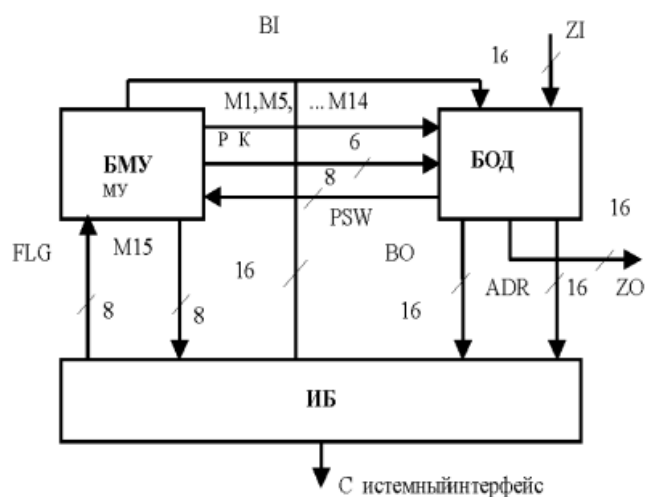


Рисунок 1.3 – Базовая структура микроЭВМ

Блок микропрограммного управления генерирует коды микрокоманд, которые управляют работой блока обработки данных и интерфейсного блока, а также содержит поля $M1...M15$. Блок обработки данных выполняет

обработку информации, содержащейся в полях $M1$, $M5...M14$, вырабатываемых в блоке микропрограммного управления кодов микрокоманд. Слово состояния процессора PSW формируется в результате выполнения операций в блоке обработки данных и поступает в блок микропрограммного управления. Из интерфейсного блока по шине данных BI информация поступает в блоки микропрограммного управления и обработки данных, а с выхода блока обработки данных в интерфейсный блок адрес передается по шине ADR и данные передаются по шине BO . Из блока микропрограммного управления в интерфейсный блок поступает поле $M15$, код которого необходим для управления вводом – выводом, при этом учитываются сигналы FLG , вырабатываемые в интерфейсном блоке.

На рисунке 1.4 приведена функциональная схема блока обработки данных. В состав входят вентили $B1$ и $B2$, обрабатывающий узел, регистр состояния PSW , входной регистр, схема формирования переноса в младший разряд $C0$, дешифратор DC , а также адреса MSA и MSB .

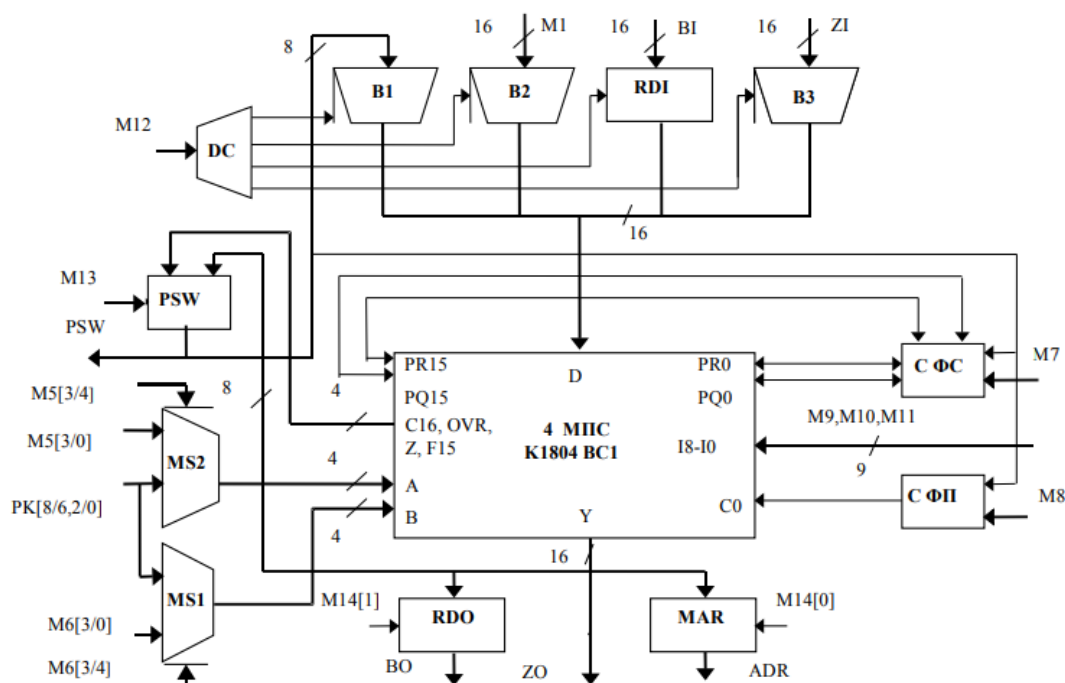


Рисунок 1.4 – Функциональная схема блока обработки данных

Выполнен блок обработки на микросхемах K1804BC1 и выполняет все логические и арифметические операции, предусмотренные в данной микросхеме [7]. Девятиразрядным кодом 18-10, который образуется кодами полей *M9*, *M10* и *M11*, задается выполняемая операция. Во внутреннем регистровом запоминающем устройстве операционного блока кодами *M5* и *M6* задаются адреса регистров *A* и *B*. Однако на входы *A* и *B* могут поступать как коды полей *M5* и *M6*, так и разряды регистра команд – 8...6 или 2...0.

Вход *D* операционного узла принимает информацию с вентилях *B1*, *B2* и *B3*, так же с регистра *RDI*. Управлением подключения к шине *D* осуществляет поле *M12*. Информация с выхода *Y*, в зависимости от кода поля *M14*, может заносится в *RDO* или *MAR*.

Поле *M13* отвечает за занесение информации на *PSW*, а коды полей *M7* и *M8* – соответственно управляют сдвигами и формированием переноса.

Функциональная схема блока микропрограммного управления без аппаратного формирования начального адреса приведена на рисунке 1.5. В состав входят схема формирования сигналов уровней команд, мультиплексор условий, счетчик тактов, регистр микрокоманд, схема управления следующим тактом, регистр команд, постоянное запоминающее устройство и схема управления адресом микрокоманды.

Код выполняемой команды хранится в регистре команд. Выборка сигналов *УР1...УР5*, определяет уровень, к которому относится формат выполняемой команды и выполняется в схеме формирования сигналов уровня команд. Мультиплексор условий выбирает, под действием кодов полей *M2* и *M3*, необходимый сигнал условия из всех поступающих на его вход и формирует проверочный сигнал *TST*, который подается в схему управления адресом микрокоманды. Адрес следующей микрокоманды определяют схема управления адресом микрокоманды совместно с схемой управления следующим адресом в соответствии с полем *M4*, кодом, приходящим на вход *D* и проверяемым условием *TST*.

Блок микропрограммного управления функционирует следующим образом. С выхода постоянно запоминающего устройства коды микрокоманд подаются в регистр микрокоманд, где хранятся один такт. Каждая микрокоманда разбивается на 15 полей. Следующую адрес определяют поля $M1...M4$, а задают микрооперацию поля $M5...M15$. Поле $M4$ определяет тип перехода к следующей микрокоманде, а поля $M2$ и $M3$ осуществляют управление мультиплексором условий. Во время очередного такта выполняются следующие действия: по коду $M2$ и $M3$ определяется значение сигнала TST , а код поля $M4$ – тип перехода; в схеме управления следующим адресом формируются сигналы, которые управляют схемой управления адресом микрокоманды, а на выходе Y формируется адрес следующей микрокоманды; далее код следующей микрокоманды вырабатывается на выходе ПЗУ. Данный код будет занесен в регистр микрокоманд в начале следующего такта.

Эмулируемая микроЭВМ включает 64 основные команды и 8 дополнительных команд: операции с плавающей запятой и расширенной арифметики [8]. Всю систему команд можно разбить на следующие подгруппы: одноадресные и двухадресные команды, выполняющие арифметические и логические операции, команды пересылок, команды вызова и возврата из подпрограммы, команды работы с флагами и прочие команды.

Одноадресные команды определяют операции, которые выполняются над одним операндом, называемым приемником. Формат одноадресных команд представлен на рисунке 1.6. Разряд 15 команды устанавливается в 0, если операция выполняется над словом, и в 1, если операция байтовая.



Рисунок 1.6 – Формат одноадресных команд

Двухадресные команды задают операции, в которых учувствуют два операнда, источник и приемник. Формат двухадресных команд представлен на рисунке 1.7. Команды выполняются как над словом, так и над байтом, в последнем случае у разряда 15 устанавливается 1.



Рисунок 1.7 – Формат двухадресных команд

Команды пересылок выполняют условные и безусловные переходы по адресу, содержащемуся в команде. Формат команды представлен на рисунке 1.8. Особенность формата данных команд является то, что их адресная часть указывает байт смещения, задающий адрес перехода относительно счетчика команд. Смещение задаётся в формате слова и 7 разряд указывает на знак.

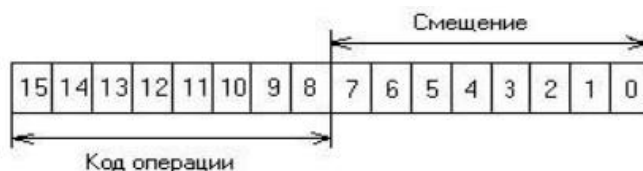


Рисунок 1.8 – Формат команды пересылки

Для избегания многократного кодирования одинаковых участков программ и эффективного использования памяти ЭВМ применяют подпрограммы. Команды обращения к подпрограммам включают операции передачи управления к началу подпрограммы и сохранение адреса возврата. По возвращению из подпрограммы адрес возврата используется для передачи управления участку, который вызывал подпрограмму. Для работы с подпрограммами используют стеки. Стек – это область оперативной памяти,

необходимая для временного хранения данных. При работе с подпрограммами и прерываниями в качестве указателя стеке используется регистр R6.

Команды изменения признаков позволяют программно изменять признаки, которые были установлены в схеме состояния процессора. Формат команд представлен на рисунке 1.9. Разряды с 0 по 3 устанавливают соответствующие признаки, а разряд 4 указывает на значение, которое будет установлено.

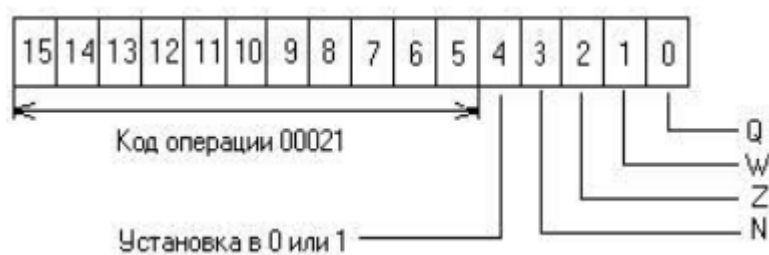


Рисунок 1.9 – Формат команды изменения признаков

Имеется пять основных уровней кодов. Уровень 1 задается битами 14, 13 и 12; уровень 2 – 11 битом кода команды; уровень 3 – полями 15, 10, 9 и 8; уровень 3 – битами 7 и 6; уровень 5 – биты 2, 1 и 0.

Дешифрация начинается с первого уровня и выполняется последовательно. Нулевая комбинация в соответствующих полях является признаком перехода на следующий уровень. Формат команд по уровням представлен на рисунке 1.10.

Первичная дешифрация выполняется в соответствии с рассмотренной системой кодирования.

После завершения первичной дешифрации кода команды выполняется дешифрация способов адресации, выборка операндов и сохранение их в регистрах. Далее выполняется команда и формируются признаки результата.

Выполнение любой команды завершается проверкой состояния системы. Последовательно проверяются сигналы останова (аппаратного и

программного), перекрывания (аппаратные и программные) и разрешения прерывания.

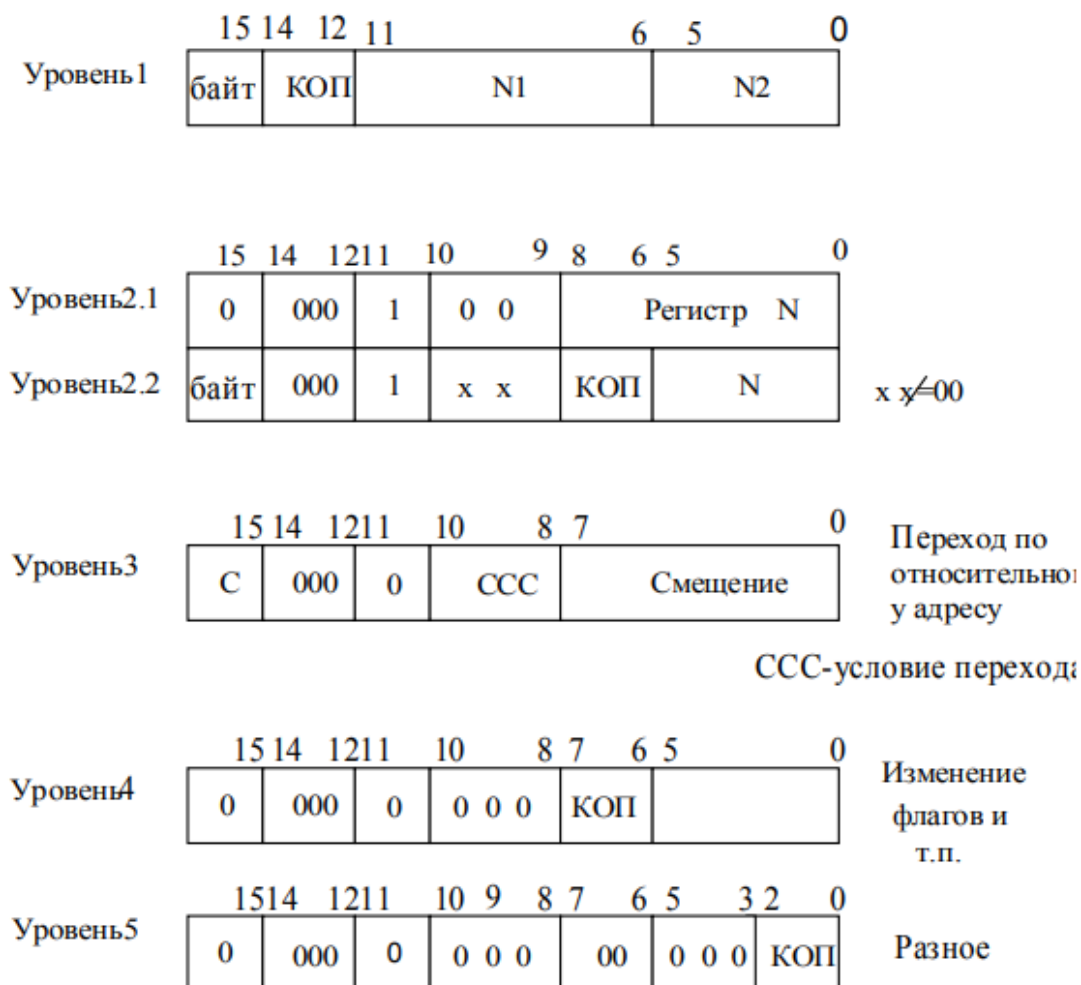


Рисунок 1.10 – Кодировка команд в зависимости от уровня

1.2 Архитектура системы проектирования

Архитектура приложений – это структурный принцип, по которому создано приложение. Каждому архитектурному виду присущи собственные свойства, характеристики и отношения между компонентами. Продуманная технология – это залог работоспособности и удобства будущей программы.

При выборе подходящей архитектуры были рассмотрены монолитная и многослойная архитектуры.

Монолитная архитектура подразумевает наличие общей и единой платформы, где сконцентрированы все компоненты одной программы. Сколько бы ни насчитывалось подобных компонентов, все они унифицированы и при этом управляются из одного места.

Достоинства монолитной архитектуры:

- простая разработка и запуск программы;
- отсутствие сквозных проблем, т.к. все работает в одном приложении;
- улучшенная производительность.

Недостатки монолитной архитектуры:

- большой объём кода;
- сложно модернизируется;
- гибкость отсутствует.

Исходя из недостатков монолитной архитектуры, была выбрана многослойная архитектура.

Эта стратегия подразумевает разделение ответственности в приложении на несколько слоев, которые накладываются друг на друга. Каждый отдельный слой имеет собственное значение.

Многослойная архитектура разделяет программу на следующие слои:

- слой представления — отвечает за интерфейс пользователя;
- слой бизнес-логики — отвечает за функционал и логику приложения и не отвечает за интерфейс;
- слой передачи данных — отвечает за работу с данными и обработку информации.

Схема архитектуры представлена на рисунке 1.11.

После того, как компоненты *UI* получили необходимые данные от пользователя и передали их в бизнес-слой, приложение может использовать эти данные для осуществления бизнес-процесса.

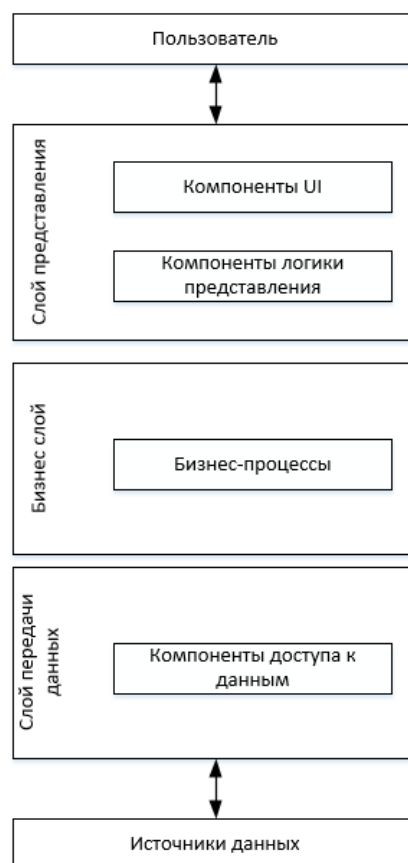


Рисунок 1.11 – Архитектура приложения

Для определения основных сценариев была составлена диаграмма вариантов использования, которая представлена в приложении 1. Она описывает взаимодействия между системой и одним или более действующими лицами.

Исходя из представленной схемы, были выделены основные сценарии: работа с эмулятором и памятью. Остальные сценарии являются второстепенными.

1.3 Выбор инструментов разработки

Для реализации поставленных задач был выбран объектно-ориентированный язык программирования *Java*.

Отличительными особенностями *Java* в сравнении с другими языками программирования общего назначения является :

– платформонезависимость: в отличие от многих других языков, включая *C* и *C++*, *Java*, когда был создан, он не компилировался в платформе конкретной машины, а в независимом от платформы байт-коде [9]. Этот байт код распространяется через интернет и интерпретируется в *Java Virtual Machine (JVM)*, на которой он в настоящее время работает;

– безопасность: методы проверки подлинности основаны на шифровании с открытым ключом;

– архитектурная-нейтральность: компилятор генерирует архитектурно-нейтральные объекты формата файла, что делает скомпилированный код исполняемым на многих процессорах, с наличием системе *Java Runtime*;

– портативность: архитектурно-нейтральный и не имеющий зависимости от реализации аспектов спецификаций — все это делает *Java* портативным. Компилятор в *Java* написан на *ANSI C* с чистой переносимостью, который является подмножеством *POSIX*;

– ООП: в *Java* все является объектом. Дополнение может быть легко расширено, так как он основан на объектной модели;

– интерпретированность: *Java* байт-код переводится на лету в машинные инструкции и нигде не сохраняется. Делая процесс более быстрым и аналитическим, поскольку связывание происходит как дополнительное с небольшого весом процесса;

– высокая производительность: введение *Just-In-Time* компилятора, позволило получить высокую производительность.

Для реализации графического интерфейса был выбран набор графических компонентов - *Swing*. *Swing* в *Java* является частью базового класса *Java*, который является независимым от платформы [10]. Он используется для создания оконных приложений и включает в себя такие компоненты, как кнопка, полоса прокрутки, текстовое поле и т. д. Объединение всех этих компонентов создает графический интерфейс пользователя.

Используемая среда разработки *NetBeans*. Она реализована на *Java* и первоначально была предназначена для разработки *Java*-приложений [11]. Текущая версия дополнительно поддерживает наиболее популярные языки программирования и разметки.

Возможности программирования в *NetBeans*:

- рабочая область среды *IDE* является полностью настраиваемой - существует возможность пользовательской настройки действий, выполняемых с помощью панели, назначения "горячих" клавиш и т.д.;
- редактор может генерировать и вставлять в исходный код стандартные фрагменты кода на *Java* или других языках;
- редактор *NetBeans* делает отступы строк, проверяет соответствие скобок и слов, подсвечивает синтаксис исходного кода;
- браузер классов позволяет просматривать иерархию и структуру любого класса *Java* - отображаются интерфейсы, базовые классы, производные классы и члены классов;
- *NetBeans IDE* является платформой для построения десктоп приложений с функциональным пользовательским интерфейсом, т.к. представляет из себя фреймворк к *Java* библиотеке *Swing*;
- расширенные средства для выполнения контекстно-зависимого поиска по всей среде *IDE*, справочным материалам и всем открытым проектам, и файлам.

1.4 Описание программно-логической модели

В данной программе используется основных восемь классов, каждый из которых отвечает за свою область в программе и имеет свой набор методов. Диаграмма классов приведена в приложении В. В совокупности этих классов программа функционирует согласно заданию.

1.4.1 Класс *iTact*

Класс, реализует окно эмулятора и меню. Это главный класс программы в нем содержится метод `main` – так называемая точка входа в программу. Именно с этого метода начинается выполнение программы. Данный класс расширяет базовый класс *JFrame*. Поля данного класса представлены в таблице 1.2. Диаграмма последовательности предствалена в приложении Г.

Таблица 1.2 – Описание полей класса *iTact*

Тип	Название	Описание
<i>Listing</i>	<i>listingForm</i>	Объект класса <i>Listing</i> , предназначен для вывода листинга программы на экран, с возможностью сохранения в файл
<i>EditPzu</i>	<i>editPzu</i>	Объект класса <i>EditPzu</i> , предназначен для редактирования микропрограммы
<i>Protocol</i>	<i>protocolForm</i>	Объект класса <i>Protocol</i> , предназначен для вывода на экран, протокола отладки микропрограммы, с возможностью сохранения в файл. Используется также в классе <i>gr1804</i> , поэтому указан <i>static</i>
<i>JRadioButton[]</i>	<i>jButtonFlgArray</i>	Массив объектов <i>JRadioButton</i> , предназначен для изменения внешних сигналов эмулятора
<i>JComponent[]</i>	<i>jComponentEnable</i>	Массив объектов <i>JComponent</i> , предназначен для блокировки использования, если не загружена микропрограмма

Тип	Название	Описание
<i>JFormattedTextField[]</i>	<i>JFTFBreakPointArray</i>	Массив объектов <i>JFormattedTextField</i> , предназначен для вывода значений всех регистров эмулятора
<i>gr1804</i>	<i>gr</i>	Объект класса <i>gr1804</i> , предназначен для выполнения микрокоманды, сброс и установка регистров эмулятора микропроцессора

Далее рассмотрим описание методов класса.

Public static void main (String [] args) – это главный метод класса.

Public iTact () – конструктор класса, в котором инициализируются поля, создаются объекты классов, устанавливаются исходные значения регистров эмулятора.

Private void EnableComp (boolean flag) – метод, который устанавливает доступность и недоступность компонентов в зависимости от входного параметра *flag*. Первый цикл отключает или включает элементы управления UI: «Автомат», «Такт» и точки останова. Второй цикл отключает или включает переключатели внешних сигналов. Фрагмент кода:

```
for (JComponent jComponentEnable1 : jComponentEnable) {
    jComponentEnable1.setEnabled(flag);
}
for (JRadioButton jButtonFlgArray1 : jButtonFlgArray) {
    jButtonFlgArray1.setEnabled(flag);
}
```

Private void InitTextField() – метод, выполняющий инициализацию текстовых полей и установку начальных значений. Для передачи текстовых полей микрокоманд в конструктор класса *MicroCommandText* создается 2 связанных списка. В первый список добавляются текстовые поля выполненной микрокоманды, а во второй поля следующей микрокоманды. Далее элементы

связного списка, отвечающие за работу регистров микроЭВМ, связывается с текстовыми полями и устанавливаются значения по умолчанию.

Private void printf() – метод, для вывода значений регистров на экран.

Пример кода:

```
for (int id = 0; id < 45; id++) {  
    model.get(id).setNewValue();  
}
```

В данном цикле выводятся значения регистров в текстовые поля. Далее выполняется вывод выполненной и следующей микрокоманды в соответствующие текстовые поля с помощью функций *PrintCur()* и *PrintNext()* класса *MicrocomandText*.

Для выставления сброса внешних сигналов считывается число из текстового поля *TextFLG*. Далее происходит наложение маски и числа для сброса, соответствующего *UI* компонента. Так как компонентов всего 8, то маска сдвигается 8 раз. Код:

```
int a = 0x01, b = octToDec(TextFLG.getText());  
for (JRadioButton jButtonFlgArray1 : jButtonFlgArray) {  
    // Наложение маски 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,  
    0x80  
    if ((b & a) == 0) {  
        jButtonFlgArray1.setSelected(false);  
    } else {  
        jButtonFlgArray1.setSelected(true);  
    }  
    a = a << 1;    // Сдвиг маски на 1 бит  
}
```

Далее выводится информация о текущем такте.

```
AddrTactLB.setText(Integer.toString(gr.NumTakt));
```

Private void printSetProg () – метод, для вывода адреса старта эмуляции, адреса завершения и ограничения количества тактов. *Integer.toSting()*

необходим для преобразования 10-го числа в 8-е число в текстовом формате.

Фрагмент кода:

```
AddrStartLB.setText(Integer.toString(adrl, 8));  
AddrEndLB.setText(Integer.toString(aend, 8));  
AddrKoltakLB.setText(Integer.toString(KolTakt, 10));
```

Private void refresh() – метод, для обновления данных регистров, перед запуском эмуляции.

В начале копируется значение из текстовых полей, преобразуется в число и заносится в соответствующий элемент списка. Код реализации:

```
for (int id = 0; id < 45; id++) {  
    model.get(id).tvalue =  
    Integer.parseInt(model.get(id).p.getText(), 8);  
}
```

Обновляет следующую микрокоманду после изменений пользователя с помощью функции *RefreshMc()* класса *MicrocomandText*. Далее устанавливаются переключатели внешних сигналов.

Private void modeling() – метод, подготавливающий данные к эмуляции и вызов функции эмуляции объекта *gr*.

Вызывает функции *refresh()*, *GenBreakPoint()*, *Extract()* и *ContSim()* у класса *gr1804*, *printf()*. Если функция *ContSim()* вернула 0, значит конец моделирования, выводится соответствующее сообщение. Если функция *ContSim()* вернула 1-4 значит сработала точка останова, выводится диалоговое окно.

Private void FileOpenActionPerformed(ActionEvent evt) – метод, загружающий микропрограмму. Вызывает функцию чтения файла, занесение в поля, предназначенные для микропрограммы, установка регистров в исходное значение и включение доступности некоторых компонентов управления.

Private void savePzu() – метод, сохраняющий микропрограмму в файл.

Создается окно выбора файлов, по умолчанию открывается каталог запуска программы. Если выбор файла в диалоговом окне прошел успешно, то с помощью функции *getPath()* получаем путь к выбранному файлу. Создается поток вывода для записи данных. Осуществляется проход по двумерному массиву с записью каждого байта в файл. Если команда не пустая, то записывается номер микрокоманды, иначе записывается 0. Затем записывается 4 байта команды. Если выбор файла в диалоговом окне прошел неудачно или файл недоступен, то вызывается исключение.

Private void GenBreakPoint() – метод, сохраняющий значения точек останова из редактируемых полей в массив, вызывается при каждой итерации. *JFTFBreakPointArray* массив точек останова которые будут обрабатываться. Получение текста из текстового поля и преобразование в число. Код:

```
n = octToDec(JFTFBreakPointArray[i].getText());
```

Далее число сравнивается с максимальным размером ОЗУ. Если число меньше, то устанавливаем число, которое было введено, иначе устанавливается 0 и выводится диалоговое окно с указанием ошибки.

Фрагмент кода:

```
if (n < gr.SizeROM) {
    gr.aost[i + 1] = (short) n;
} else {
    str += String.valueOf(i) + " ";
    JFTFBreakPointArray[i].setText("0");
}
```

Private void ReadFile() – метод для чтения данных из файла и сохранения в массив микропрограммы.

Создается входной поток данных, массив байт для чтения команд. Его размер рассчитывается исходя из количества команд и размера команды. В нашем случае максимальное количество команд 4096, а их размер 10 байт. Получается, массив размером 40960 байт. Далее считываем данные из файла в массив и обрабатываем его. В цикле перебираем массив, считываем номер

команды, далее записываем 8 байт команды в двумерный массив, тип – *short*.

Извлечение полей микрокоманды:

```
n = (short) (((int) array1[a + 0] & 0xff) + (((int) array1[a + 1])
<< 8));
pzu[0][n] = (short) (((int) array1[a + 2] & 0xff) + ((int) array1[a
+ 3] << 8));
pzu[1][n] = (short) ((int) (array1[a + 4] & 0xff) + ((int) array1[a
+ 5] << 8));
pzu[2][n] = (short) (((int) array1[a + 6] & 0xff) + ((int) array1[a
+ 7] << 8));
pzu[3][n] = (short) (((int) array1[a + 8] & 0xff) + ((int) array1[a
+ 9] << 8));
```

Private void Crt_Opn(Boolean b) – метод, открытия файла, извлечения первой команды и установка исходных значений.

Вызывает функции класса *gr1804 ResPar()* и *Extract()*, которая извлекает первую микрокоманду. Если *b* истинно, то вызывается функция чтения файла.

Private void setDecor() – метод, установки цветовой темы приложения.

Получаем цвет фона текстового поля. Если полученный цвет светлый, тогда устанавливается темный цвет букв. Если же фон темный, тогда буквы светлые. Фрагмент кода:

```
bcColor1 = model.get(0).p.getForeground();
bcColor2 = Color.RED;
SwingUtilities.updateComponentTreeUI(getRootPane());
if (bcColor1.getBlue() > 128 && bcColor1.getGreen() > 128 &&
bcColor1.getRed() > 128)
{
    bcColor3 = new Color(0, 230, 30);
}
else
{
    bcColor3 = new Color(0, 150, 20);
}
```

1.4.2 Класс *gr1804*

Класс, реализующий эмулятор микропроцессора и взаимодействие основного класса с ядром. Описание полей данного класса представлено в таблице 1.3.

Таблица 1.3 – Описание полей класса *gr1804*

Тип	Название	Описание
<i>Tact</i>	<i>t</i>	Объект класса <i>Tact</i> , предназначен для вызова функции выполнения микрокоманды
<i>short</i>	<i>NumTakt</i>	Предназначен для подсчета выполненных команд
<i>short</i>	<i>SizeROM</i>	Размер памяти микропрограмм

Описание методов данного класса.

Public gr1804 () – конструктор класса, который инициализирует объект *t* класса *Tact* и устанавливает начальные значения для эмулятора микропроцессора.

Public void Extract(short n) – метод, который извлекает микрокоманду из массива микрокоманд. На рисунке 1.12 представлено описание формата микрокоманды.

ФОРМАТ МИКРОКОМАНДЫ		
Поле	Разряды	Значение
M1	63 – 48	Поле констант.
M2	47 – 46	Управление маской и инверсией условия.
M3	45 – 42	Управление мультиплексором условий.
M4	41 – 38	Управление БМУ.
M5	37 – 33	Управление мультиплексором адреса А.
M6	32 – 28	Управление мультиплексором адреса В.
M7	27 – 26	Управление сдвигами.
M8	25 – 24	Управление входом переноса.
M9	23 – 21	Управление источниками операндов АЛУ.
M10	20 – 18	Управление операциями АЛУ.
M11	17 – 15	Управление приемником результата АЛУ.
M12	14 – 12	Управление входной шиной D.
M13	11 – 10	Управление регистром состояния.
M14	9 – 8	Управление выходной шиной Y.
M15	7 – 0	Управление вводом-выводом.

Рисунок 1.12 – Формат микрокоманды

На рисунке 1.13 представлено расположение полей микрокоманд в ПЗУ.

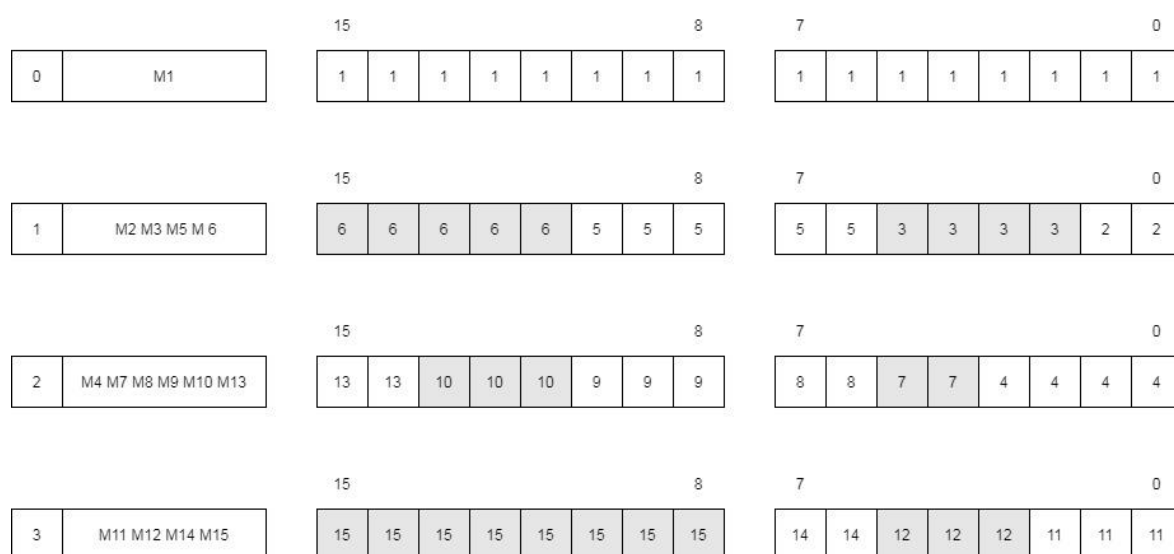


Рисунок 1.13 – Расположение полей МК в ПЗУ

Для того чтобы получить поле микрокоманды необходимо наложить маску и сделать сдвиг. Например, для получения поля *M15* необходимо выполнить конъюнкцию маски $65280_{10} = 1111111100000000_2$ и числа записанного в *pzu[3]*. Далее сдвиг полученного числа на 8 разрядов, с помощью деления на 256 (2 в 8 степени). Код извлечения полей микрокоманды представлен ниже, где *n* – номер микрокоманды:

```

m[15] = (short) ((pzu[3][n] & 65280) / 256);
m[14] = (short) ((pzu[3][n] & 192) / 64);
m[12] = (short) ((pzu[3][n] & 56) / 8);
m[11] = (short) (pzu[3][n] & 7);
m[13] = (short) ((pzu[2][n] & 49152) / 16384);
m[10] = (short) ((pzu[2][n] & 14336) / 2048);
m[9] = (short) ((pzu[2][n] & 1792) / 256);
m[8] = (short) ((pzu[2][n] & 192) / 64);
m[7] = (short) ((pzu[2][n] & 48) / 16);
m[4] = (short) (pzu[2][n] & 15);
m[6] = (short) ((pzu[1][n] & 63488) / 2048);
m[5] = (short) ((pzu[1][n] & 1986) / 64);
m[3] = (short) ((pzu[1][n] & 60) / 4);

```

```

m[2] = (short) (pzu[1][n] & 3);
m[1] = pzu[0][n];
m[0] = n;

```

Public void ResPar() – метод, сброса регистров в исходное значение.

Public int ContSim() – метод, выполняющий эмуляцию микропроцессора.

Сохраняет адрес текущей команды, проверяет его совпадение с адресом завершения или с точками останова, проверяет количество тактов моделирования, выполняет эмуляцию в объекте *t* и возвращает результат моделирования. Диаграмма деятельности представлена в приложении Д.

Перед выполнением эмуляции проверяется ограничение на количество тактов или совпадение адреса текущей команды с адресом завершения микропрограммы. При выполнении условия функция возвращает -1, что означает окончание отладки. Пример кода:

```

int bOst = -1;
if ((KolTakt <= NumTakt) || ((adr2 == aend) && (NumTakt != 0))) {
    return bOst;
}

```

Для последующей проверки совпадения адреса текущей команды с адресом завершения микропрограммы сравнивается переменная *adr1* с *aend*. При совпадении устанавливается переменная *stop* в *true*, что означает остановку отладки, а *st_type* равный *false* означает, что точка останова не была задействована. Иначе проверяется адрес текущей команды с адресами точек останова. При совпадении, переменная *stop* и *st_type* устанавливаются в *true*, что означает срабатывание точки останова и остановку отладки. Так же в переменную *bOst* заносится номер точки останова, который будет возвращен.

Пример кода:

```

stop = false;
if (adr1 == aend) {
    stop = true;
    st_type = false;
} else {

```

```

for (j = 1; j <= naost; j++) {
    if (aost[j] == adr1) {
        stop = true;
        st_type = true;
        bOst = j;}}}

```

После проверок текущего адреса микрокоманды, вызывается функция *tact()* класса *Tact*, увеличивается счетчик команд *NumTakt*, *adr2* копируется в *adr3* для сохранения номера выполненной микрокоманды, вызывается функция *printReg()* у класса *Protocol* для вывода микрокоманды и регистров в файл отчета. Извлечение микрокоманды, ее выполнение и переход к следующей выполняется в цикле, условием выхода которого, является достижение максимального количества тактов, потактовая отладка, срабатывание точек останова или достижение адреса завершения микропрограммы. После выхода из цикла возвращается значение *bOst*.

1.4.3 Класс *Protocol*

Класс, реализующий вывод на экран и сохранение протокола отладки микропрограммы. Поля данного класса описаны в таблице 1.4.

Таблица 1.4 – Описание полей класса *Protocol*

Тип	Название	Описание
<i>int</i>	<i>protocolCount</i>	Счетчик тактов для протокла
<i>JTextArea</i>	<i>protocolArea</i>	Текстовое поле отображения протокола

Данный класс содержит 2 основных метода.

Public void startProt () – метод, предназначенный для вывода начальной информации об отладке: адрес старта и окончания моделирования, количество тактов моделирования, режим работы. Далее выводится информация всех регистров. С помощью функции *format()* у класса *String* получаем строку которая содержит имя регистра и его значение. Под имя регистра выделяется

4 символа, под значение 7 символов. С помощью функции *append()* класса *JTextArea* добавляется полученная строка в текстовое поле. После вывода 6 регистров – переход на новую строку. Пример кода:

```
for (int id = 0, i = 5; id < model.size(); id++, i--) {
    protocolArea.append(String.format("%-4s=%-7o", odel.get(id).name,
        (int) model.get(id).tvalue & 0xffff));
    if (i == 0) {
        protocolArea.append("\n");
        i = 6;}}
```

Public void printReg () – метод, предназначенный для вывода очередной микрокоманды и состояний регистров в текстовое поле. В начале выводиться заголовок, который состоит из полей *M1-M15*, далее выводиться микрокоманда и содержимое регистров. Для решения проблемы преобразования типов, необходимо наложить маску *0xFFFF* для исключения отрицательных чисел. Пример кода:

```
str += String.format("%5o", (int) m[k] & 0xffff);
```

1.4.4 Класс *Listing*

Класс, реализующий вывод на экран и сохранение листинга микропрограммы. Поля данного класса представлены в таблице 1.5.

Таблица 1.5 – Описание полей класса *Listing*

Тип	Название	Описание
<i>JLabel</i>	<i>jLabel</i>	UI компонент для вывода заголовка отчета
<i>JTextArea</i>	<i>listingArea</i>	Текстовое поле отображения листинга

Содержит 3 метода, их описание приведено ниже.

Public void setListing() – метод, устанавливает заголовок окна и выводит все микрокоманды программы. Перед выводом программы текстовое поле очищается. В цикле извлекается микрокоманда, для корректного

преобразования накладывается маска и форматировано выводиться в текстовую область. После вывода всей программы, позиция курсора устанавливается в начало текстовой области. Реализовано скрывание пустых микрокоманд при необходимости.

Public void saveListing () – метод, сохраняющий листинг в текстовый файл. Вызывается диалоговое окно выбора файла, по умолчанию открывается пользовательский каталог и устанавливается фильтр на выбор файлов формата “.txt”:

```
JFileChooser saveListing = new JFileChooser(new
File(System.getProperty ("user.dir")));
FileNameExtensionFilter filter = new
FileNameExtensionFilter("Текст листинга (.txt)", "txt");
saveListing.setFileFilter(filter);
```

После успешного открытия файла, создается класс *BufferedWriter* который записывает текст в поток, предварительно буферизируя записываемые символы, тем самым снижая количество обращений к физическому носителю для записи данных. Сначала записывается заголовок, затем вызывается базовая функция *write()* класса *JTextArea* с помощью которой, содержимое текстовой области записывается в файл. После записи, функцией *close()* файл закрывается.

Public void copyListing () – метод, необходимый для сохранения листинга в буфере обмена. Создается объект буфера обмена. С помощью функции *setContents()* устанавливается текущее содержимое буфера обмена в указанный передаваемый объект, в качестве объекта передаем содержимое *JTextArea* полученного с помощью функции *getText()*. Пример кода:

```
StringSelection stringSelect = new
StringSelection(listingArea.getText());
Clipboard clipboard =
Toolkit.getDefaultToolkit().getSystemClipboard();
clipboard.setContents(stringSelect, null);
```

1.4.5 Класс *Memory*

Класс, реализующий вывод на экран памяти, в виде таблицы, с возможностью ее редактирования, сохранения и загрузки. Поля данного класса представлены в таблице 1.6.

Таблица 1.6 – Описание полей класса *Memory*

Тип	Название	Описание
<i>DefaultTableModel</i>	<i>dtm</i>	Модель таблицы, с помощью которой происходит изменение значений в ячейках, а также их проверку на допустимые значения
<i>TextField</i>	<i>addr</i>	Окно поиска и вывода адреса нажатой ячейки
<i>JTable</i>	<i>memTable</i>	Таблица памяти программы
<i>Color</i>	<i>clrFont, clrBack</i>	Поля, необходимые для хранения цвета шрифтов и фона ячеек таблицы

Имеет 9 методов описанных ниже.

Public MemoryForm() – конструктор, в котором инициализируются компоненты формы, получение модели таблицы и установка шрифтов, а также очищение памяти или установки значений по умолчанию. С помощью функции *getForeground()* и *getBackground()* получаем цвет текста и фона таблицы. Для работы со строками и столбцами таблицы, необходимо с помощью функции *getModel()* получить модель таблицы. Для обработки каждой ячейки отдельно создаем экземпляр класса *CustomTableRenderer* который расширяет *DefaultTableCellRenderer*. Он предназначен для подсвечивания значений которые не равны нулю красным и столбца адреса *Addr* зеленым цветом. Далее с помощью функции *setPrefferedWidth()*

устанавливается предпочтительная ширина ячеек таблицы и с помощью функции *setFont()* задается шрифт текста.

Public void memTableMouseReleased() – метод, предназначенный для установки в поле выбранного адреса, выбранный адрес в таблице памяти. С помощью функции *getSelectedColumn()* и *getSelectedRow()* получает номер колонки и строки у выбранной ячейки. Далее высчитывается адрес данной ячейки по формуле номер строки * 16 + (номер колонки - 1) * 2, полученный результат функцией *toString()* преобразуется в восьмиричную систему счисления, затем в строку и устанавливается в текстовое поле. Пример кода:

```
int a = memTable.getSelectedColumn();
int b = memTable.getSelectedRow();
int c = b * 16 + (a - 1) * 2;
addr.setText(Integer.toString(c, 8));
```

public void addrActionPerformed() – метод осуществляющий поиск ячейки по введенному в текстовое поле адресу.

Получаем введенное число, преобразуем из строки в десятичное число. Далее проверяется адрес, необходимо чтобы он был положительный и не больше максимально возможного. При выполнении условия, проверяется четность полученного числа. Если число не четное, то вычитается единица. Далее путем деления на 16 находится номер строки и чтобы найти номер колонки необходимо остаток от деления на 16 разделить на 2 и прибавить 1. Затем с помощью функции *setSectionInterval()* по найденным строке и столбцу осуществляется переход к ячейке. Фрагмент кода:

```
int adr = Integer.parseInt(addr.getText(), 8);
if (adr >= 0 && adr <= Integer.parseInt(Integer.toString(65535, 8))) {
    if (adr % 2 != 0) {
        adr--;
    }
    int a = adr / 16;
    int b = (adr % 16) / 2 + 1;
```

```
memTable.getSelectionModel().setSelectionInterval(Integer.parseInt(Integer.toString(b, 8)), a);}
```

Public void loadMemory() – метод, предназначенный загрузки памяти из файла. Открывается диалоговое окно выбора файла, при успешном выборе и открытии файла. С помощью функции *setRowCount()* обнуляем таблицу, задавая строк равным 0. Далее создается массив размером с выбранный файл. Данные считываются в массив байт. Далее массив байт преобразуется в массив строк с помощью функции *split* класса *String*. Каждый элемент массива является строкой таблицы, поэтому его необходимо преобразовать еще раз, чтобы получить данные каждой ячейки. Далее с помощью функции *addRow()* добавляется новая строка с полученными значениями в таблицу, цикл повторяется пока не будут прочитаны все строки. Пример кода реализующий описанный алгоритм:

```
String[] row = String.valueOf(open_array).split("\r\n");
for (int idx = 0; idx < row.length; idx++) {
    String[] value = row[idx].split("\\s+");
    if (value.length < 9) {
        dtm.addRow(new Object[]{0, 0, 0, 0, 0, 0, 0, 0, 0});
    } else {
        dtm.addRow(new Object[]{Integer.parseInt(value[0]),
Integer.parseInt(value[1]), Integer.parseInt(value[2]),
Integer.parseInt(value[3]),
Integer.parseInt(value[4]), Integer.parseInt(value[5]),
Integer.parseInt(value[6]),
Integer.parseInt(value[7]),
Integer.parseInt(value[8])}); } }
```

Public void saveMemory() – метод, предназначенный для сохранения памяти в файл. Открывается диалоговое окно выбора файла, при успешном выборе и открытии файла, создается класс *BufferedWriter* который записывает текст в поток. Далее в цикле осуществляется проход по всем строкам таблицы,

Первым записывается адрес, затем в цикле проход по столбцам таблицы текущей строки, с записью каждого элемента в файл. Пример кода:

```
for (int i = 0; i < dtm.getRowCount(); i++) {
    String buffer = String.format("%-4d ", (int) dtm.getValueAt(i,
0));
    for (int j = 1; j < dtm.getColumnCount(); j++) {
        buffer += String.format("%-6d ", (int) dtm.getValueAt(i,
j));    }
    bfw.write(buffer + "\r\n");}
```

public void clearTable() – метод, необходимый для очищения таблицы. Для удаления предыдущей таблицы, вызываем функция *setRowCount()* в которую передаем 0, указывая что наша таблица не имеет строк. Далее создаем объект с нулевыми значениями. После чего в цикле от 0 до максимально возможного адреса в памяти добавляем строки с нулевыми значениями, кроме 0 элемента в котором указывается адрес первого элемента массива данной строки. Пример кода:

```
dtm.setRowCount(0);
Object[] v = new Object[]{0, 0, 0, 0, 0, 0, 0, 0, 0};
for (int idx = 0; Integer.parseInt(Integer.toString(idx, 8)) <=
177760; idx += 16) {
    v[0] = Integer.parseInt(Integer.toString(idx, 8));
    dtm.addRow(v);}
```

Public int octToDec() – метод преобразования числа из восьмеричной в десятичную системы счисления с сохранением типа *int*. Пример реализации, где *a* – входное число, *b* – выходное число:

```
b = Integer.parseInt(Integer.toString(a, 10), 8);
```

Public int getPoMar(int adr) – метод, предназначенный для получения значения из ячейки памяти по указанному адресу. Входной параметр функции, это адрес ячейки таблицы, значение которой необходимо получить и вернуть. Необходимо чтобы адрес был положительный и не больше $2^{16}-1$. При выполнении условия, проверяется четность адреса. Если нечетный, то

вычитается единица. Делением на 16 находится номер строки. Остаток от деления на 16 деленный на 2 и увеличенный на 1 – это номер колонки. Далее получаем значение ячейки, переводим в десятичную и возвращаем. Пример кода, где *adr* – это адрес ячейки памяти:

```
int result = -1;
if (adr >= 0 && adr <= (int) 65535) {
    if (adr % 2 != 0) {
        adr--;    }
    int a = adr / 16;
    int b = (adr % 16) / 2 + 1;
    result = octToDec((int) dtm.getValueAt(a, b));
}
return result;
```

Public void setPoMar(int adr, int value) – метод, предназначенный для записи значения в ячейку памяти по указанному адресу. Входными параметрами функции являются адрес ячейки в памяти и значение которое необходимо установить. Аналогично методу *getPoMar()* находит ячейку и с помощью функции *setValueAt()* устанавливает новое значение в ячейку. Пример кода, где *a* – номер столбца, *b* – номер строки, *value* – новое значение:

```
dtm.setValueAt(decToOct(value), a, b);
```

1.4.6 Класс *MicroCommandText*

Класс, отвечающий за вывод выполненной и следующей микрокоманды.

Поля данного класса описаны в таблице 1.7.

Таблица 1.7 – Поля класса *MicroComandText*

Тип	Название	Описание
<i>ArrayList</i> < <i>JFormattedTextField</i> >	<i>currMC</i>	Связный список, <i>JFormattedTextField</i> который находится в классе <i>iTact</i> . Необходим для вывода выполненной команды

Тип	Название	Описание
<i>ArrayList</i> < <i>JFormattedTextField</i> >	<i>nextMC</i>	Связный список, <i>JFormattedTextField</i> который находится в классе <i>iTact</i> . Необходим для вывода следующей команды

Рассмотрим 6 основных методов данного класса.

MicroCommandText() – конструктор в котором инициализируются СВЯЗНЫЕ СПИСКИ.

Private void setText(boolean flag, int index, int value) – устанавливает текст выполненной или следующей микрокоманды. Выбор зависит от флага, который передают в данную функцию. Так же передается индекс, который указывает на элемент списка, т.е. в какое текстовое поле выводить, и само значение в числовом виде.

Private void Print(boolean a, int n) – необходим для вывода выполненной или следующей микрокоманды с помощью функции *setText()* описанной выше. Организует получение всех полей микрокоманды и поочередную передачу их в функцию *setText()*. Таким образом будут выведены все 15 полей микрокоманды. Пример кода, где входные параметры *a* – это выполненная или следующая микрокоманда(*true / false*), *n* – это номер микрокоманды:

```
setText(a, 15, ((pzu[3][n] & 65280) / 256)); setText(a, 14,
((pzu[3][n] & 192) / 64));
setText(a, 12, ((pzu[3][n] & 56) / 8));
setText(a, 11, (pzu[3][n] & 7));
setText(a, 13, ((pzu[2][n] & 49152) / 16384));
setText(a, 0, (short) n);
setText(a, 10, ((pzu[2][n] & 14336) / 2048));
setText(a, 1, pzu[0][n]);
setText(a, 9, ((pzu[2][n] & 1792) / 256));
setText(a, 8, ((pzu[2][n] & 192) / 64));
setText(a, 7, ((pzu[2][n] & 48) / 16));
setText(a, 4, (pzu[2][n] & 15));
```

```

setText(a, 6, ((pzu[1][n] & 63488) / 2048));
setText(a, 5, ((pzu[1][n] & 1986) / 64));
setText(a, 3, ((pzu[1][n] & 60) / 4));
setText(a, 2, (pzu[1][n] & 3));

```

Public void PrintNext(int n) – метод необходимый для вывода следующей микрокоманды. Вызывает функцию *Print()* и передает номер микрокоманды и параметр *false*.

Public void PrintCurr(int n) – метод необходимый для вывода выполненной микрокоманды. Вызывает функцию *Print()* и передает номер микрокоманды и параметр *true*.

Public void RefreshMc() – метод, отвечающий за обновление следующей микрокоманды перед ее исполнением. Создается целочисленный массив из 16 элементов. Далее получение из текстовых полей строк, преобразование их в числа и запись в целочисленный массив. Затем запись массива в ПЗУ в соответствующую микрокоманду. Входной параметр *n* указывает на номер микрокоманды, которую необходимо обновить.

1.4.7 Класс *Decoration*

Класс, реализует смену темы оформления внешнего цветового интерфейса приложения. Содержит вложенный класс *Тема* поля которого хранят путь подключения темы и ее название. Поля данного класса представлены в таблице 1.8.

Таблица 1.8 – Описание полей класса *Decoration*

Тип	Название	Описание
<i>ArrayList<Тема></i>	<i>tems</i>	Массив доступных тем
<i>int</i>	<i>current</i>	Идентификатор текущей темы
<i>String</i>	<i>pathProper</i>	Содержит путь к файлу настроек
<i>ButtonGroup</i>	<i>bg</i>	Группа кнопок выбора темы

Шесть классов работы с внешним видом, представлено ниже.

Public Decoration() – конструктор необходимый для вывода в меню всех цветовых схем. В начале создан массив строк где каждые 2 элемента отдельная тема. Далее кнопкам, отвечающим за выбор тем устанавливаем название. *I* элемент строкового массива отвечает за адрес подключения темы, следующий за ним элемент содержит название кнопки.

Public void setdecor() – метод, необходимый для определения нажатой в меню кнопки. В цикле перебирается группа кнопок. С помощью функции *isselect()* проверяется нажата кнопка или нет. Если нажата, то вызывается функция *setdecor2()* в которую передается название нажатой кнопки.

Private void setdecor2() – метод устанавливающий тему приложения. Входной параметр функции строковая переменная с именем темы которую необходимо установить. Если строка пустая, то выводится ошибка. С помощью итератора перебирает список всех тем, сравнивая каждое название на совпадение. Если тема найдена, то в переменную *path* заносится путь. Если при проходе тема была не найдена, то выводится ошибка. С помощью функции *setlookandfeel()* устанавливаем тему приложения. Далее сохраняем настройки с помощью функции *saveproperties()*.

Public void getdecor() – метод устанавливающий цветовую тему другим окнам приложения. В переменной *current* храниться номер темы. По этому номеру получаем путь цветовой темы и с помощью *setlookandfeel()* устанавливаем.

Public void saveproperties() – сохранение настроек программы в текстовый файл. С помощью класса *Properties* создается объект, у которого вызывается функция *setProperty()*, создавая ключ и значение цветовой схемы. Далее создается поток вывода и сохраняются пользовательские настройки.

Пример кода:

```
Properties props = new Properties();
props.setProperty("tema", tems.get(current).name);
OutputStream out = new FileOutputStream(new File(pathProper));
props.store(out, "User properties");
```

Public void loadProperties() – загрузка настроек программы из текстового файла. Создается класс пользовательских настроек, с помощью функции *load()* читаются из файла, который был открыт. Функцией *getProperty()* находится название темы. Далее по названию темы определяется кнопка. Затем в зависимости от выбранной кнопки в меню устанавливается тема.

1.4.8 Класс *Tact*

Класс, реализующий ядро микропроцессора. Описание полей данного класса представлено в таблице 1.9.

Таблица 1.9 – Поля класса *Tact*

Тип	Название	Описание
<i>byte</i>	<i>pe_</i>	Разрешение регистра микрокоманд
<i>byte</i>	<i>me_</i>	Разрешение работы программно логической матрицы
<i>byte</i>	<i>cte_</i>	Разрешение счета
<i>byte</i>	<i>ctl_</i>	Разрешение загрузки счетчика
<i>byte</i>	<i>pup</i>	Управляет стеком
<i>byte</i>	<i>fe_</i>	Осуществляет управление стеком
<i>byte</i>	<i>za_</i>	Необходим для установки 0 адреса
<i>byte</i>	<i>s0</i>	0 разряд выбора адреса
<i>byte</i>	<i>s1</i>	1 разряд выбора адреса

Далее будут рассмотрены основные методы.

Private byte rkd () – метод, предназначенный для выделения заданного числа по указанным битам из регистра команд *RK*. Для определения уровня микрокоманды (код операции), способа адресации и адреса регистров первого и второго операндов (адреса источника и приемника). Алгоритм реализован с помощью битовых операций сдвига и наложения маски. Сначала регистр команд сдвигается вправо до нижнего бита, тем самым отсекая правую часть

числа. Затем сдвигается маска, заполненная единицами вправо так чтобы единицы, покрывали те же разряды, из которых необходимо получить значение. После этого происходит наложение маски и сдвинутого регистра команд, что в результате дает значение, которое возвращается функцией.

Фрагмент кода:

```
(byte) ((rk >> lb) & (0xffff >> (15 - hb + lb)));
```

Private int inppsw () – метод, предназначенный для преобразования восьми бит числа типа *int* в массив типа *byte* из восьми элементов. Цикл состоит из восьми итераций, при каждой итерации берется последний бит, записывается в ячейку массива и сдвигается вправо для получения следующего бита. Фрагмент кода представлен ниже:

```
for (int id = 0; id <= 7; id++) {
    rz[id] = (byte) (dd % 2); // rz[i]
    dd = (short) (dd >> 1);}

```

Private short unppsw () – метод, предназначенный для преобразования массива типа *byte* из восьми элементов в число типа *short*. Цикл состоит из восьми итераций, при каждой итерации выполняется сдвиг влево, далее складывается переменная результата с крайним правым битом очередного элемента массива. Код реализации:

```
short rz = 0;
for (int id = 7; id >= 0; id--) {
    rz = (short) (rz << 1);
    rz = (short) (rz + (wd[id] & 1)); }

```

Private void levset () – метод, предназначенный для определения уровня команды. Выделения уровня микрокоманды осуществляется с помощью наложения маски на регистр команд, для проверки соответствующих битов (см. Таблица 1.10).

Private void bmu () – метод, предназначенный для эмуляции блока микропрограммного управления. Сначала выполняется проверка счетчика

тактов *СТ*, если равен нулю, то флаг переполнения счетчика устанавливается в единицу, иначе сбрасывается в ноль. Пример кода:

```
if (ct == 0) {
    ctr = 1; } else {
    ctr = 0;}
```

Таблица 1.10 – Соответствие битов регистра команд и уровня команды

Уровень команды	Биты регистра команд
1	14, 13, 12
2	11
3	15, 10, 9 и 8
4	7 и 6
5	2, 1 и 0

Далее выполняется эмуляция мультиплексора, который реализован с помощью конструкции *switch – case*, где выбор входного сигнала зависит от кода поля *МЗ*. Если значение лежит в диапазоне от 0 до 5, то в сигнал *TST* устанавливается значение соответствующего бита регистра *PSW*. При значении 6 и 7 в сигнал *TST* записывается код переноса счетчика и константы нуля соответственно. Диапазон от 8 до 15 зависит от первого бита поля *М2*. Если он равен 0, тогда в *TST* устанавливается значение соответствующего бита регистра внешних сигналов *FLG*. Иначе устанавливается результат проверки соответствующего уровня команды для диапазона от 8 до 13, а для 14 и 15 значения сигнал *TST* устанавливается в 1, если используется прямая адресация, в противном случае сбрасывается в 0. При варианте равному диапазону от 0 до 3 анализируется первый бит поля *М2*. Если он установлен в единицу, значит значение из поля *М1* копируется в маску, в противном случае заносится 0, что означает отсутствие маски. Пример кода:

```
if (vrnt == 1) {
...
} else if (m[2] >= 2)
```

```
ormask = m[1];
else
    ormask = 0;
```

Нулевой бит поля $M2$ отвечает за инверсию условия. Если он установлен в 1, то необходимо на данном этапе сделать инверсию сигнала TST .

Схема управления следующего адреса в зависимости от входных значений имеет на выходе один из 32 наборов управляющих сигналов для работы других блоков микропрограммного управления. Наборы хранятся в виде табличных значений в массиве, а адрес формируется из четырех битов поля $M4$ и сигнала TST , которые и передается в качестве входного значения. Наложением маски $0x1f$ сбрасывает 5, 6 и 7 биты. Пример кода:

```
adr_by3 = (byte) (((m[4] << 1) | tst) & 0x1f);
d_by3 = by3[adr_by3];
```

В d_by3 будет получен байт информации с управляющими сигналами, их формат представлен на рисунке 1.14. Получение каждого сигнала и запись в соответствующую переменную реализовано с помощью последовательного сдвига вправо и наложения маски $0x01$.

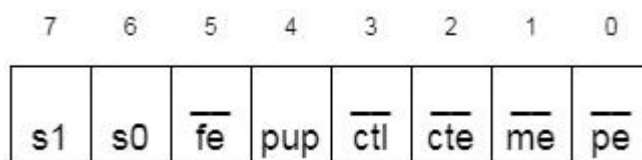


Рисунок 1.14 – Формат хранения управляющих сигналов в 1 байте

Формирование следующего адреса микрокоманды основывается на сигналах $S0$ и $S1$ (см. Таблица 1.11).

Таблица 1.11 – Таблица истинности блока выборки данных

$S1$	$S0$	Адрес микрокоманды передаваемой блоку выборки адреса
0	0	Из счетчика микрокоманд
0	1	Из регистра адреса

<i>S1</i>	<i>S0</i>	Адрес микрокоманды передаваемой блоку выборки адреса
1	0	Из стека
1	1	С прямых входов адреса шины <i>D</i>

Если сигналы *CTL* и *CTE* равны нулю, то задаётся переход к команде с нулевым адресом. Фрагмент кода:

```
za_ = (byte) (ctl_ | cte_);
```

Далее проверяется сигнал разрешения управления стеком *FE*, при нулевом значении разрешается работа со стеком. Значение сигнала *PUP* операцию со стеком. При логической единице выполняется загрузка стека, а при нуле циклический сдвиг. Стек реализован на 5 переменных.

Адрес полученный из блока выборки адреса уволенный на единицу записывается как адрес следующей микрокоманды в регистр *СМК*. Регистр адреса формируется в зависимости от сигналов *PE* и *ME*. Фрагмент кода, где *рна* – старший байт регистра команд, *рп* – младший байт регистра команд:

```
if (pe_ == 0)
    dby = m[1];
else if (me_ == 0)
    dby = рна;
else
    dby = па;
ra = (short) (dby & MaskBy1);
```

Если сигнал разрешения работы счетчика *СТЕ* равен 0, счетчик останавливается. Сигнал загрузки счетчика равный логической 0 разрешает загрузку кода *М1* в счетчик.

Private void zf15 () – метод, предназначенный для формирования флагов *Z* и *F15* по значению результата *F*. Если результат равен 0, то устанавливается флаг *Z* в единицу. Флаг *F15* проверяется наложением на *F* маски 0x8000, проверяя крайний левый бит, если число отрицательное, то устанавливается в единицу. Фрагмент кода, где *f* – результат работы функции:

```
if ((short) f == 0)
```

```

z = 1;
if (((short) f & 0x8000) != 0)
    f15 = 1;

```

Private void msk () – метод, предназначенный для выделения 4-х бит смещенных *DVR* бит.

Private void prvr () – метод, предназначенный для формирования промежуточных переменных *P0..P3, G0..G3, P30, G30*.

Private void add_rs () – метод, предназначенный для выполнения операции сложения (*R+S*). Складываются операнды *R, S* и значение входа переноса *C0*, заносятся в регистр *F*. Код реализации, где где *rrl* и *ssl* – операнды *R* и *S* соответственно, *C0* – флаг переноса, *f* – результат выполнения операции:

```
f = (rrl + ssl + (c0 & 0xff));
```

Далее путем анализа регистра *F* происходит формирование флагов состояния процессора. Наложением маски *0x10000* проверяется 16 бит регистра. Если установлен в 1 бит, значит был перенос и устанавливается флаг *C16* в единицу. Фрагмент кода, где *C16* – флаг переноса, *f* – результат выполнения операции:

```

if ((f & 0x10000) != 0)
    c16 = 1;

```

Далее проверяется переполнение. Для проверки знака числа накладывается маска *0x8000*. Если знак двух операндов одинаковый, а после выполнения операции знак результата изменился, значит произошло переполнение. Фрагмент кода, где *rrl* и *ssl* – операнды *R* и *S* соответственно, *ovr* – флаг переполнения, *f* – результат выполнения операции:

```

ovr = 0;
if ((rrl < 0x8000) && (ssl < 0x8000))
    if (f >= 0x8000)
        ovr = 1;
if ((rrl >= 0x8000) && (ssl >= 0x8000))
    if (f < 0x8000)

```

```
ovr = 1;
```

Далее вызывается функция *zf15*, для формирования флагов *z* и *f15*.

Private void sub_sr () – метод, предназначенный для выполнения операции (*S-R-1+C0*). Операция вычитания реализована с помощью функции сложения *add_rs()*. Вычитание достигается путем инверсии операнда *R*. Фрагмент кода, где *rr* – операнд *R*:

```
rr = (short) (rr ^ 0xFFFF);  
add_rs();
```

Private void or_rs () – метод, предназначенный для выполнения операции (*R or S*). Фрагмент кода, где *f* – результат операции, *rr* и *ss* – операнды *R* и *S* соответственно:

```
f = (short) (rr | ss);
```

Далее в каждой микропроцессорной секции генерируется сигнал переноса и разрешения из которых формируются сигналы переноса *C16* и переполнения *OVR*. Так же вызывается функция формирования флагов *z* и *f15*.

Private void and_rs () – метод, предназначенный для выполнения операции (*R and S*). Фрагмент кода, где *f* – результат операции, *rr* и *ss* – операнды *R* и *S* соответственно:

```
f = (short) (rr & ss);
```

Далее в каждой микропроцессорной секции генерируется сигнал переноса и разрешения из которых формируются сигналы переноса *C16* и переполнения *OVR*. Так же вызывается функция формирования флагов *z* и *f15*.

Private void and_nrs () – метод, предназначенный для выполнения операции (*not(R) and S*). Операция конъюнкции, с инверсией операнда *R*, реализована с помощью функции *and_rs()*, достигается путем инверсии операнда *R*. Фрагмент кода, где *rr* – операнд *R*:

```
rr = (short) ~(rr);  
and_rs();
```

Private void sub_rs () – метод, предназначенный для выполнения операции (*R-S-1+C0*). Операция вычитания реализована с помощью функции

сложения *add_rs()*. Вычитание достигается путем инверсии операнда *S*.

Фрагмент кода, где *ss* – операнд *s*:

```
ss = (short) (ss ^ 0xFFFF);  
add_rs();
```

Private void cr67 () – метод, предназначенный для формирования переноса при операции (*R xor S*) для одной четырех разрядной секции 1804BC1. Фрагмент кода, где *g0...g3* – сигналы генерации переноса, *p0...p3* – сигналы разрешения переноса, *cr* – бит переноса предыдущей секции:

```
return (byte) ~ (g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1  
& p0 & (g0 | ~cr)));
```

Private void ovr67 () – метод, предназначенный для формирования флага *OVR* при операции (*R xor S*) для одной четырех разрядной секции 1804BC1.

Private void xor_rs () – метод, предназначенный для выполнения операции (*R xor S*). Фрагмент кода, где *f* – результат операции, *rr* и *ss* – операнды *R* и *S* соответственно:

```
f = (short) (rr ^ ss);
```

Далее в каждой микропроцессорной секции генерируется сигнал переноса и разрешения из которых формируются сигналы переноса *C16* и переполнения *OVR*. Так же вызывается функция формирования флагов *z* и *f15*.

Private void xor_nrs () – метод, предназначенный для выполнения операции (*not(R xor S)*). Операция сложения по модулю 2, с инверсией операнда *R*, реализована с помощью функции *xor_rs()*, достигается путем инверсии операнда *R*. Фрагмент кода, где *rr* – операнд *R*:

```
rr = (short) ~(rr);  
xor_rs();
```

Private void sr () – метод, предназначенный для сдвига вправо на один разряд с занесением бита, переданного в качестве входного параметра, в старший разряд сдвигаемого слова. Возвращает сдвинутое значение. Фрагмент кода, где *r1* – входное значение операнда, *cr* – значение вдвигаемого бита, *r* – возвращаемый результат:

```

r = ((r1 >> 1) + (0x8000 * (cr & 0x01)));
return r;

```

Private void sl () – метод, предназначенный для сдвига влево на один разряд с занесением бита, переданного в качестве входного параметра, в младший разряд сдвигаемого слова. Возвращает сдвинутое значение. Фрагмент кода, где *r1* – входное значение операнда, *cr* – значение вдвигаемого бита, *r* – возвращаемый результат:

```

r = ((r1<< 1) + (cr & 1));
return r;

```

Private void bod () – метод, предназначенный для эмуляции блока обработки данных. Реализована логика работы двух мультиплексоров, блока управления входной/выходной шиной, схемы формирования сдвигов и переноса, регистра состояния *PSW*, а также микропроцессорная секция.

Схемы формирования адресов регистров *A* и *B* аналогичны, зависят от кода *M5* и *M6*. Входная шина имеет 5 разрядов из которых 2 старших разряда управляющие мультиплексором, реализованном в виде условий. В диапазонах:

- от 0 до 15 – адрес регистра формируется из кода *M5* или *M6*;
- от 16 до 23 – извлекается значения 0, 1, 2 битов регистра команд и заносится в регистр адреса операнда;
- от 24 до 31 – извлекается значения 6, 7, 8 битов регистра команд и заносится в регистр адреса операнда.

Фрагмент кода:

```

if ((m[5] >= 0) && (m[5] <= 15))
    a = m[5];
else if ((m[5] >= 16) && (m[5] <= 23))
    a = rkd((byte) 2, (byte) 0);
else if ((m[5] >= 24) && (m[5] <= 31))
    a = rkd((byte) 8, (byte) 6);

```

Схема формирования переноса реализована с помощью конструкции *switch*, так как зависит от поля *M8* и имеет 4 различных значения представленных в таблице 1.12.

Таблица 1.12 – Таблица зависимости флага переноса *C0* от поля *M8*

Код поля <i>M8</i>	Флаг <i>C0</i>
0	0
1	1
2	Бит 0 регистра <i>PSW</i>
3	Инверсное значение 0 бита регистра <i>PSW</i>

Управление входной шиной осуществляется кодом *M12*. Если код *M12* больше или равен 4, то в регистр *RDI* заносится значение с шины *BI*. Так же значение данной шины заносится в регистр команд, когда поле *M12* равно семи. Вход *D*, при коде *M12* равном:

- 0, 4, 7 принимает значение регистра *RDI*;
- 1 и 5 – значение регистра *ZI*;
- 2 и 6 – подается значение кода *M1*;
- 3 – регистр состояния *PSW*.

Выбор источников операндов АЛУ определяется кодом *M9* и показаны в таблице 1.13, где *R* – первый операнд, *S* – второй операнд, а *A*, *B*, *Q* – адреса регистров. Реализовано с помощью оператора выбора *switch – case*.

Таблица 1.13 – Таблица кодов выбора операндов АЛУ

Код <i>M9</i>	<i>R</i>	<i>S</i>	Код <i>M9</i>	<i>R</i>	<i>S</i>
0	<i>A</i>	<i>Q</i>	4	0	<i>A</i>
1	<i>A</i>	<i>B</i>	5	<i>D</i>	<i>A</i>
2	0	<i>Q</i>	6	<i>D</i>	<i>Q</i>
3	0	<i>B</i>	7	<i>D</i>	0

Поле *M10* определяет арифметико-логическую операцию и приведена в таблице 1.14, где *R* и *S* – операнды, *C0* – бит переноса.

Таблица 1.14 – Управление операцией арифметико – логического устройства

Поле <i>M10</i>	Операция	Поле <i>M10</i>	Операция
0	$R + S + C0$	4	$R \text{ and } S$
1	$S - R - 1 + C0$	5	$\text{not } (R \text{ and } S)$
2	$R - S - 1 + C0$	6	$R \text{ xor } S$
3	$R \text{ or } S$	7	$\text{not } (R \text{ xor } S)$

Результат выполнения арифметико логического устройства записывается в *Y* и формируются флаги четности и знака. Если первый бит поля *M11* равен 0, то выполняет сдвиг вправо, иначе управление сдвига влево. Далее проверяется поле *M7*, значение которого определяет тип сдвига.

При сдвиге вправо и значении кода *M7* равным 0 сдвигается регистр с занесением 0 в старший разряд, младший разряд сдвигается в бит переноса. Так же сдвигается регистр *Q* с занесением в старший разряд нуля. Поле *M7* равное 1, выполняет циклический сдвиг через бит переноса, а регистр *Q* выполняет циклический сдвиг. Код равный 2, выполняет сдвиг регистра вправо с занесением бита переноса в старший разряд, младший разряд регистра заносится в старший разряд регистра *Q*, который так же сдвигается вправо и заносит младший разряд в бит переноса, то есть выполняется циклический сдвиг регистра и регистра *Q* через бит переноса. Код равный 3, выполняет сдвиг, аналогичный коду 2, отличие заключается в занесении при сдвиге в старший разряд регистра результат операции сложение по модулю 2 битов *F15* и *OVR*.

Фрагмент кода:

```
switch (m[7]) {
case 0: pr15 = 0;    pq15 = 0; break;
case 1: pr15 = psw[0];  pq15 = (byte) (pq0 & 1);  break;
```

```

case 2: pr15 = psw[0]; pq15 = (byte) (pr0 & 1); break;
case 3: pr15 = (f15 ^ ovr) & 1; pq15 = (pr0 & 1); break; }

```

При сдвиге влево и значении кода *M7* равным 0 сдвигается регистр с занесением 0 в младший разряд, старший разряд сдвигается в бит переноса. Так же сдвигается регистр *Q* с занесением в младший разряд нуля. Поле *M7* равное 1, выполняет циклический сдвиг через бит переноса, а регистр *Q* выполняет циклический сдвиг. При значении 2 и 3, в младший разряд регистра *Q* заносится 1 и 0 соответственно, старший разряд заносится в младший разряд регистра. Старший разряд регистра сдвигается в бит переноса.

Поле *M11* определяет куда будет записан результат выполнения. Варианты записи результата представлены в таблице 1.15, где *F* – результат работы арифметико логического устройства, *A* – операнд источник, *B* – операнд приемник, $a \rightarrow$ – символ загрузки. Фрагмент кода:

```

switch (m[11]) {
case 0: rq = (short) f; break;
case 2: y = (int) r[a] & 0xffff; r[b] = (short) f; break;
case 3: r[b] = f; break;
case 4: r[b] = f; r[b] = sr(((int) r[b] & 0xffff), pr15);
        rq = sr(((int) rq & 0xffff), pq15); break;
case 5: r[b] = f; r[b] = sr(((int) r[b] & 0xffff), pr15);
        break;
case 6: r[b] = f; r[b] = (short) sl(((int) r[b] & 0xffff), pr0);
        rq = (short) sl(((int) rq & 0xffff), pq0); break;
case 7: r[b] = f; r[b] = sl(((int) r[b] & 0xffff), pr0);
        break; }

```

Таблица 1.15 – Зависимость варианта записи от поля *M11*

Код <i>M11</i>	Загрузка регистра	Загрузка регистра <i>Q</i>	Выход <i>Y</i>
0	–	$F \rightarrow Q$	<i>F</i>
1	–	–	<i>F</i>
2	$F \rightarrow B$	–	<i>A</i>

Код <i>M11</i>	Загрузка регистра	Загрузка регистра <i>Q</i>	Выход <i>Y</i>
3	$F \rightarrow B$	–	<i>F</i>
4	$F / 2 \rightarrow B$	$Q / 2 \rightarrow Q$	<i>F</i>
5	$F / 2 \rightarrow B$	–	<i>F</i>
6	$F * 2 \rightarrow B$	$Q * 2 \rightarrow Q$	<i>F</i>
7	$F * 2 \rightarrow B$	–	<i>F</i>

За формирование регистра состояния отвечает поле *M13*. Реализовано с помощью конструкции выбора switch, где код *M13* означает следующее:

- 0 – сохранение *PSW*;
- 1 – загрузка по признакам микропроцессорной системы K1804BC1 (*c16, ovr, z, f15*);
- 2 – загрузка с учетом сдвига(*c1, ovr, z, f15 xor ovr*);
- 3 – загрузка с шины *Y*.

Поле *M14* управляет занесением *Y* в регистр *RDO* или *MAR*. При коде 1, занесение в регистр *RDO*, а при 2 – *MAR*.

Взаимодействие с ПЗУ осуществляется с помощью поля *M15*. Если значение не равно 0, то необходимо проверить внешние сигналы имитируемого ОЗУ. Активный уровень сигнала обозначает готовность устройства к приему/передаче значения из/в памяти, иначе пропуск, что означает ожидание готовности внешнего устройства к работе. Код *M15* равный 1, означает операцию чтения данных из памяти, а 2 – запись данных в память.

При чтении необходимо дождаться готовности внешнего устройства к отправке данных. С помощью счетчика подсчитывается количество тактов ожидания и при достижении заданного значения продолжается операция чтения. В регистре *MAR* хранится адрес ячейки памяти. В регистр *VI* заносится значение из памяти по адресу из регистра *MAR*. После успешного чтения сбрасывается счетчик ожидания для исключения повторного чтения данных.

При записи необходимо дождаться готовности внешнего устройства к получению данных. Счетчика количества тактов ожидания не требуется, так как у внешнего устройства есть буферный регистр. В регистре *MAR* хранится адрес ячейки памяти, а в регистре *RDO* данные для записи.

Если код поля *M15* равен 0 и есть внешний сигнал, то сбрасывается счетчик тактов ожидания, регистр *BI* обнуляется и внешнему устройству посылается сигнал остановки.

1.4.9 Класс *GlobalVars*

Поля данного класса представлены в таблице 1.16.

Таблица 1.16 – Поля класса *GlobalVars*

Тип	Название	Описание
<i>Memory</i>	<i>memForm</i>	Объект класса <i>Memory</i>
<i>short</i>	<i>pzu</i>	Массив содержащий микрокоманды эмулируемого микропроцессора
<i>short</i>	<i>aend</i>	Адрес окончания моделирования
<i>boolean</i>	<i>step</i>	Остановка после выполненной микрокоманды
<i>boolean</i>	<i>stop</i>	Сигнал полной остановки эмуляции
<i>short</i>	<i>KolTakt</i>	Счетчик выполненных микрокоманд
<i>Short</i>	<i>m</i>	Текущая микрокоманда
<i>Color</i>	<i>bcColor1</i>	Переменная хранящая серый цвет
<i>Color</i>	<i>bcColor2</i>	Переменная хранящая красный цвет
<i>Color</i>	<i>bcColor3</i>	Переменная хранящая зеленый цвет
<i>LinkedList<Record></i>	<i>model</i>	Список <i>UI</i> – компонентов
<i>ListEdit<SerializerEdit></i>	<i>listEdit</i>	Список событий графических объектов в графическом редакторе

Тип	Название	Описание
<i>LinkedList<Block></i>	<i>listEditBlk</i>	Список ссылок на графический элемент - блок
<i>LinkedList<Connector></i>	<i>listEditConn</i>	Список ссылок на графический элемент – соединительная линия
<i>LinkedList<TextBlock></i>	<i>listEditTxt</i>	Список ссылок на графический элемент – текстовое поле

Имеет 5 вложенных классов.

Класс *InputVerifierOct* расширяет базовый класс *InputVerifier* и предназначен для ограничения ввода символов. Переопределений метод *verify()* получает текст из *UI* компонента, сохраняет в строку, с помощью функции *replaceAll()* заменяет символы “8”, “9” и “-” на пустой символ, т.е. удаляет их. Фрагмент кода:

```
String text = ((JTextComponent) input).getText();
text = text.replaceAll("8", "");
text = text.replaceAll("9", "");
text = text.replaceAll("-", "");
```

Далее проверяет размер полученной строки. Максимальное значение 2 байт в восьмеричной системе счисления это 177777. Если длина превышает 6 символов, тогда вызывается функция *substring()*, которая возвращает подстроку данной строки. Подстрока начинается с символа, заданного индексом, и продолжается до конца данной строки или до 6. Полученный текст устанавливаем в *UI* компонент. Код реализации:

```
if (text.length() > 6) {
    text = text.substring(0, 6);
}
((JTextComponent) input).setText(text);
```

Класс *Record* предназначен для хранения значений регистров микропроцессора. Имеет метод *setNewValue()* который устанавливает цвет текста при каждом изменении. Если значение регистра было изменено в

последнем такте, то цвет текста устанавливается зеленым, иначе если значение регистра равно 0, то серый цвет текста, если значение не равно 0, то красный цвет текста. Код:

```
if (octToDecInt(p.getText()) != tvalue) {
    p.setForeground(bcColor3);
} else if (tvalue == 0) {
    p.setForeground(bcColor1);
} else {
    p.setForeground(bcColor2);}
```

Установка значения регистра с наложением маски, для исключения вывода отрицательного значения:

```
p.setText(Integer.toOctalString((int) (tvalue & 0xFFFF)))
```

Поля данного класса представлены в таблице 1.17.

Таблица 1.17 – Поля класса *Record*

Тип	Название	Описание
<i>int</i>	<i>tvalue</i>	Текущее значение регистра
<i>short</i>	<i>stvalue</i>	Начальное значение регистра
<i>boolean</i>	<i>outp</i>	Разрешение на чтение
<i>boolean</i>	<i>inp</i>	Разрешение на запись
<i>string</i>	<i>name</i>	Название регистра
<i>JComponent</i>	<i>p</i>	<i>UI</i> – компонент формы, для вывода значения на экран

Класс *IdPzu* расширяет базовый класс *Object* и предназначен для хранения номера микрокоманды. Содержит переменную *id* целочисленного типа *int*. Данный класс имеет два метода *equals* и *isEmpty*. Первый необходим для сравнения объектов. Фрагмент кода:

```
if (obj == this || ((IdPzu) obj).id == this.id) {
    return true;
}
```

```
return false;
```

Метод *isEmpty* в качестве входного параметра получает микрокоманду, которую проверяет на наличие не пустых полей. Возвращает истину если все поля микрокоманды равны нулю, если хотя бы одно поле не равно нулю возвращает ложь. Фрагмент кода:

```
empty = true;
for (int i = 1; i < 16; i++) {
    if (m[i] != 0) {
        empty = false;
        break;}}
return empty;
```

Класс *ListIdPzu<E>* расширяет базовый класс *LinkedList<E>* и необходим для реализации функции поиска *findRemove()* в связанном списке. Данная функция обходит весь связанный список и проверяет идентификаторы объекты списка на соответствие идентификатора, который был передан в качестве входного аргумента. Пример кода:

```
for (E item : this) {
    if (((IdPzu) item).id == id) {
        this.remove(item);
        break; }}
```

Имеет три перегруженных, вспомогательных метода.

Public static int octodecint() – перегруженная функция, принимает в качестве входных параметров значения типа *int* или *String*. Преобразует значение из восьмеричной в десятичную систему счисления. Возвращает значение в виде целого числа. *Public static String octodecstr()* – выполняет аналогичный функционал, возвращая строку.

Public static int dectooct() - перегруженная функция, принимает в качестве входных параметров значения типа *int* или *String*. Преобразует значение из десятичной в восьмеричную систему счисления. Возвращает значение в виде целого числа. *Public static String dectooctstr()* – выполняет аналогичный функционал, возвращая строку.

Static public int rounding() – метод необходимый для привязки графических объектов к сетке. Перегруженная функция, принимает в качестве входных параметров исходное число типа *int* или *double*, предназначенное для округления, шаг сетки и смещение. Фрагмент кода:

```
int a = n % step;
n -= a;
if (a > 4) {
    if (offset == 0) {
        n += step;
    } else {
        n += offset;
    }
} else {
    n -= offset;
}
return n;
```

Вычисляется в какую сторону необходимо округлить число, в большую или меньшую. Переменная, *a* равна значению в диапазоне от 0 до *step-1*, так как вычисляется остаток от деления, и означает количество пикселей внутри клетки сетки. Далее от исходного числа вычитается найденное количество пикселей, которое не совпадает с сеткой, то есть округление в меньшую сторону. Если количество пикселей в клетке больше или равно половине клетки, то округление идет в большую сторону путем прибавления размера клетки к исходному значению, выровненному по сетке. При необходимости добавляется смещение относительно пересечения сетки.

1.5 Описание организации пользовательского интерфейса

Главное окно программы представлено на рисунке 1.15.

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

Рисунок 1.15 – Главное окно программы

В центре выведены все поля регистров и флагов, для отслеживания текущей информации в них. При необходимости информацию в них можно изменять вручную, после изменения она сохранится.

Слева располагается блок для ручной установки флагов *FLG* и блок добавления точек останова.

Ниже располагаются кнопки редактирования программы, памяти, запуска и отладки.

В самом низу выводится информация о выполненной микрокоманде и микрокоманде, которая будет выполняться следующим тактом.

Окно редактирования программы представлено на рисунке 1.16.

Addr	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	76	1	7	16	0	7	0	0	7	0	2	2	0	0	0
2	1000	1	7	1	0	7	0	0	3	0	1	0	0	2	1
3	7	0	0	16	0	0	0	0	3	0	1	7	0	0	0
4	2	0	0	16	7	7	0	0	5	0	3	2	0	0	0
5	1	3	10	3	0	0	0	0	3	0	1	0	0	0	0
6	50	1	7	1	0	0	0	0	4	0	1	0	0	0	0
7	200	1	7	1	0	0	0	0	4	0	1	0	0	0	0
10	70	1	7	1	0	0	0	0	4	0	0	0	0	0	0
11	120	1	7	1	0	0	0	0	4	0	0	0	0	0	0
12	1	0	10	3	0	0	0	0	0	0	0	0	0	0	0
13	12	0	12	3	0	0	0	0	0	0	0	0	0	0	0
14	300	0	13	1	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 1.16 – Окно редактирования программы

В данном окне вся программа представлена в виде таблицы, при необходимости любое поле, любой МК можно изменить.

Окно памяти микропроцессора представлено на рисунке 1.17.

Addr	0	2	4	6	10	12	14	16
0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	36172
100	10000	10000	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0
140	0	0	0	0	20	0	0	0
160	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0
220	0	0	0	100	0	0	0	0
240	0	0	0	0	0	0	0	0
260	0	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0	0
320	0	0	0	0	0	0	0	0
340	0	0	0	0	0	0	0	0
360	0	0	0	0	0	0	0	0
400	0	0	0	0	0	0	0	0
420	0	0	0	0	0	0	0	0
440	0	0	0	0	0	0	0	0
460	0	0	0	0	0	0	0	0
500	0	0	0	0	0	0	0	0
520	0	0	0	0	0	0	0	0
540	0	0	0	0	0	0	0	0

Рисунок 1.17 – Окно памяти микропроцессора

Вся память представлена в виде таблицы. Для удобства пользователя имеется подсвечивание цветами полей, которые не равны нулю. При наведении на любую ячейку таблицы внизу будет выведен ее адрес, так же если адрес известен заранее введя число в данную ячейку можно произвести поиск.

Внизу расположены кнопки очищения памяти, сохранения и загрузки.

Окно запуска отвечает на настройку начальных параметров и представлено на рисунке 1.18.

Рисунок 1.18 – Окно начальной настройки

Имеет поля для ввода адреса начала и окончания моделирования, задания количества тактов моделирования. Выбор варианта осуществляется с помощью *RadioButton*.

Меню состоит из 4 пунктов:

- файл;
- правка;
- тема;
- справка.

Файл включает в себя функции сохранения и загрузки программы. После выбора одного из них, пользователю предлагается выбрать имя файла и путь к нему.

Пункт отчеты включает в себя окна необходимые для составления отчета по выполненной программе – протокол и листинг.

Окно протоколирования представлено на рисунке 1.19.

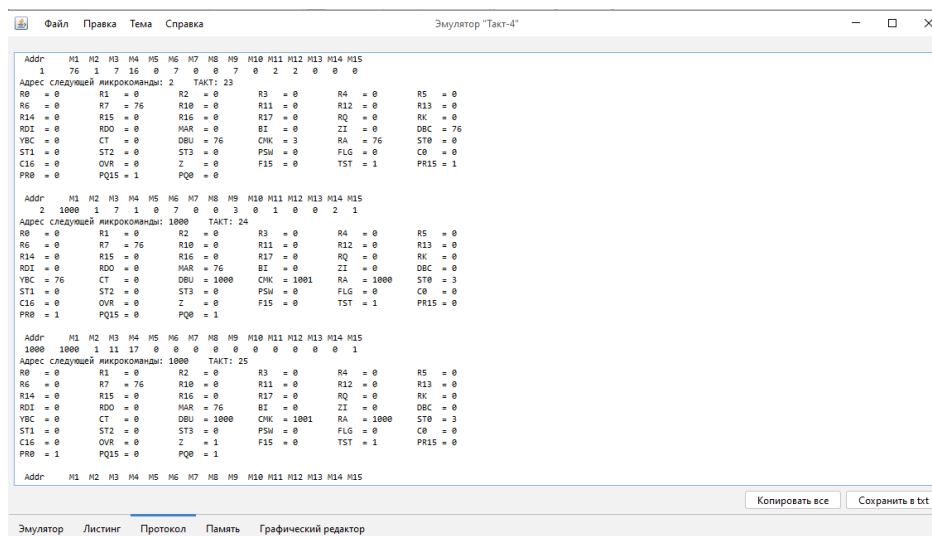


Рисунок 1.19 – Окно протокола

Всю область окна занимает *TextArea* в которую выводится информация о выполненной МК и текущей информации, хранящейся в регистрах.

Окно листинга представлено на рисунке 1.20.

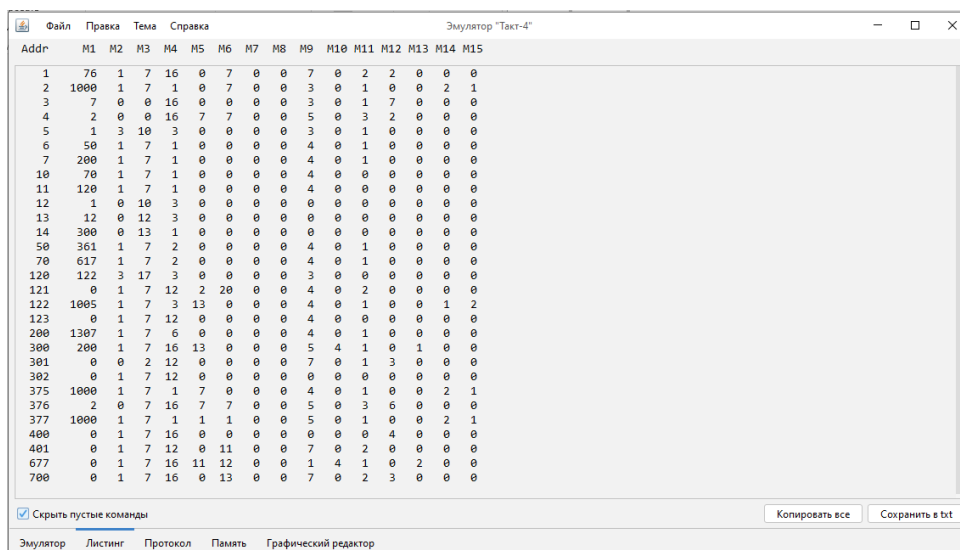


Рисунок 1.20 – Окно листинга

В центре окна располагается *TextArea* в которой выведена каждая МК. Внизу располагаются кнопки копирования и сохранения в текстовом виде. При необходимости пустые команды можно скрыть, поставив галку в поле «Скрыть пустые команды».

В пункте меню Темы на выбор представлено несколько цветовых тем оформления программы (рисунок 1.21). Несколько примеров представлены на рисунках 1.22 – 1.23.

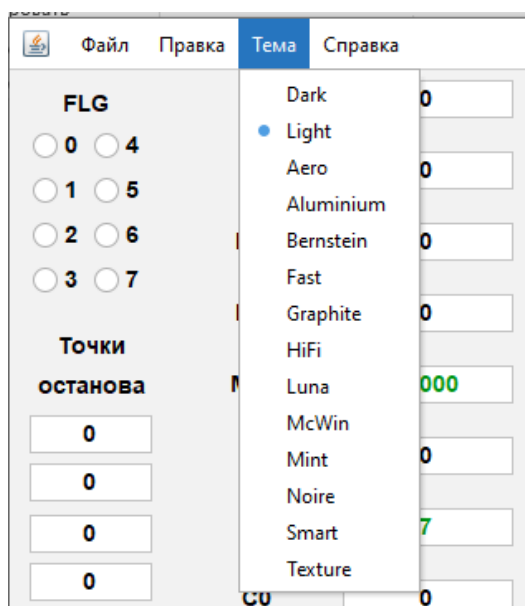


Рисунок 1.21 – Список тем

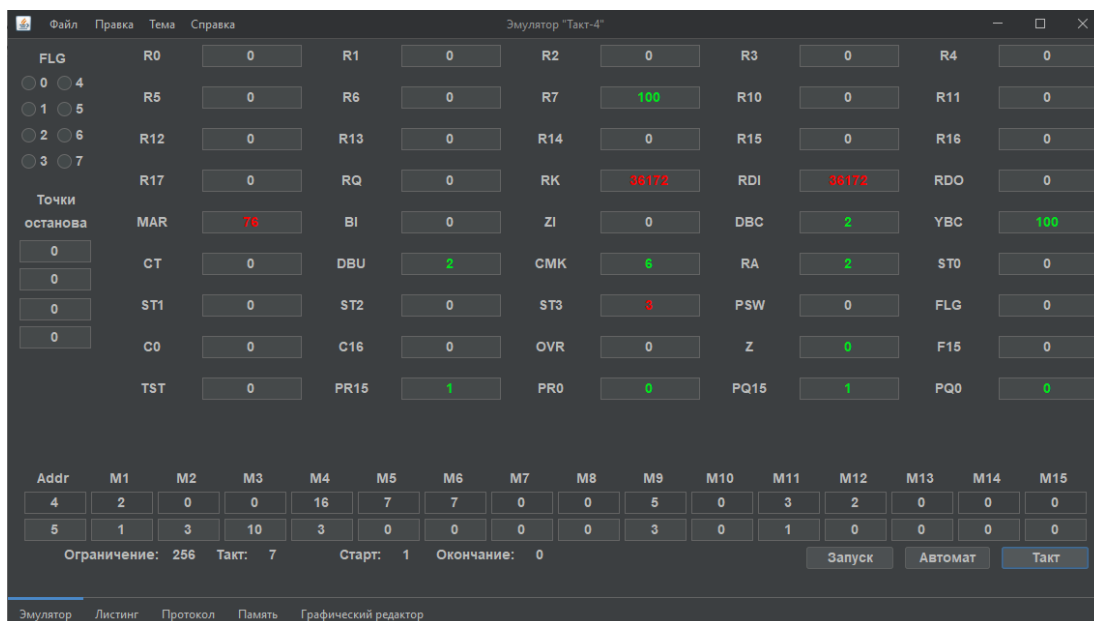


Рисунок 1.22 – Темная тема

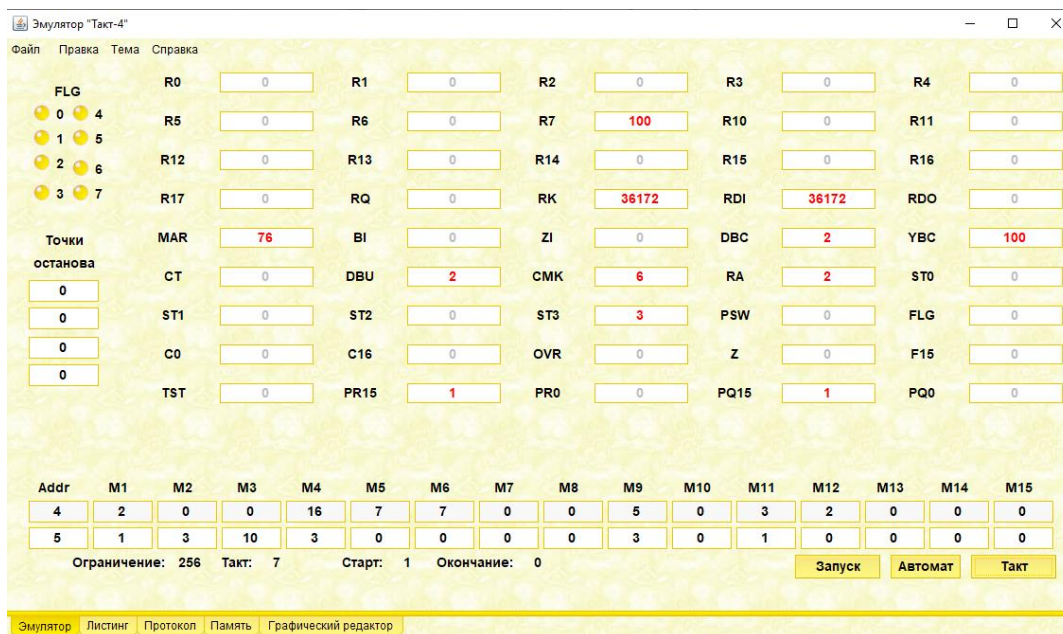


Рисунок 1.23 – Желтая тема

По нажатию на пункт меню «Справка» открывается модальное окно (рисунок 1.24), в котором представлена справочная информация о программе, изображения свидетельств о государственной регистрации программы и контактная информация для связи с разработчиками.

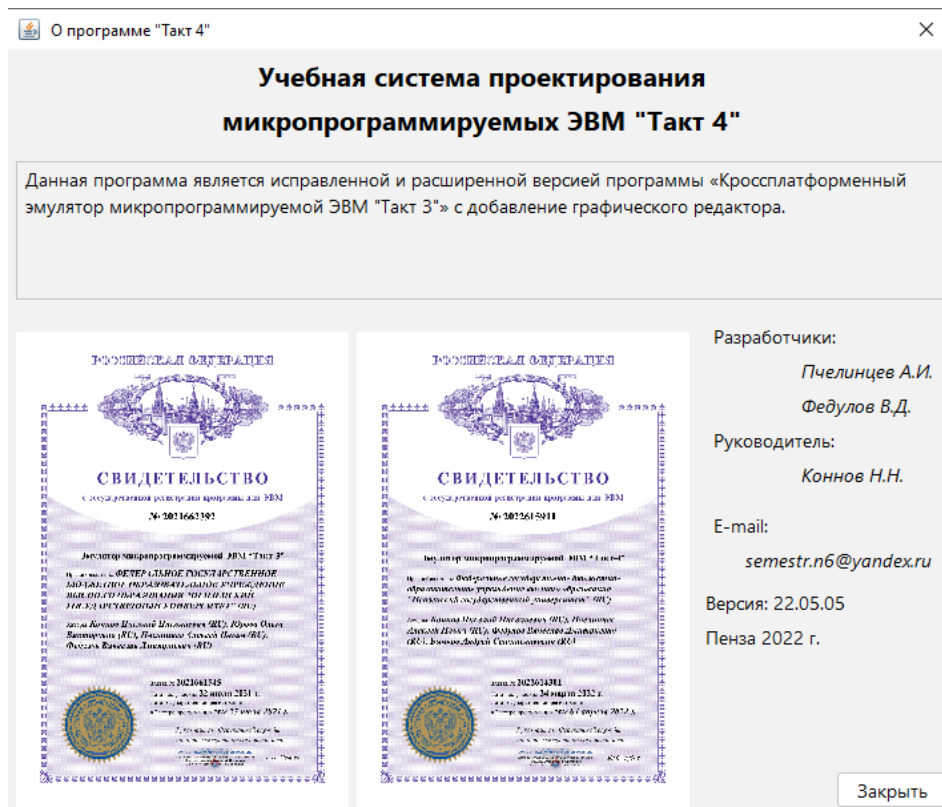


Рисунок 1.24 – Модальное окно «О программе»

1.6 Тестирование работы системы

Для проверки эмулятора на предмет корректной работы, его необходимо сравнить с используемой на данный момент в учебном процессе системой. Сравнение значения регистров, с загрузкой одной и той же программы представлено на рисунках 1.25 – 1.40.

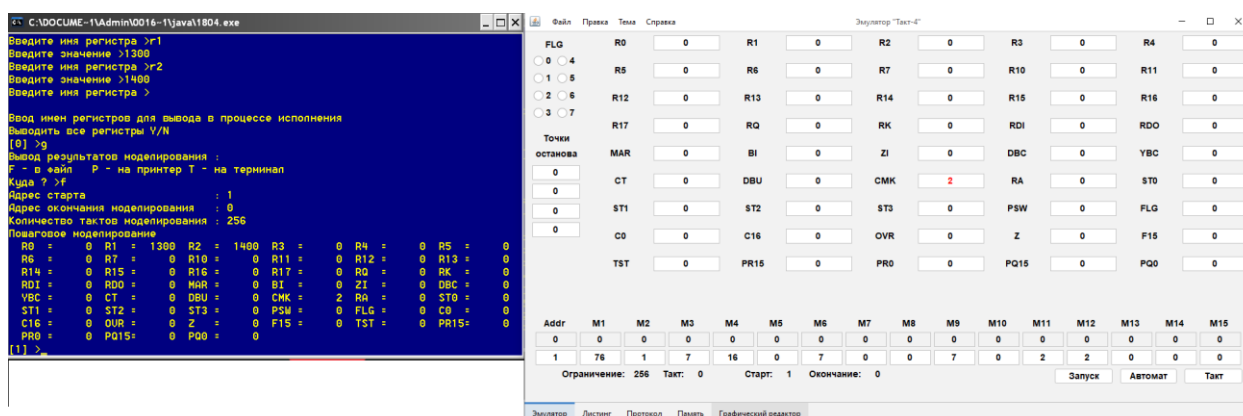
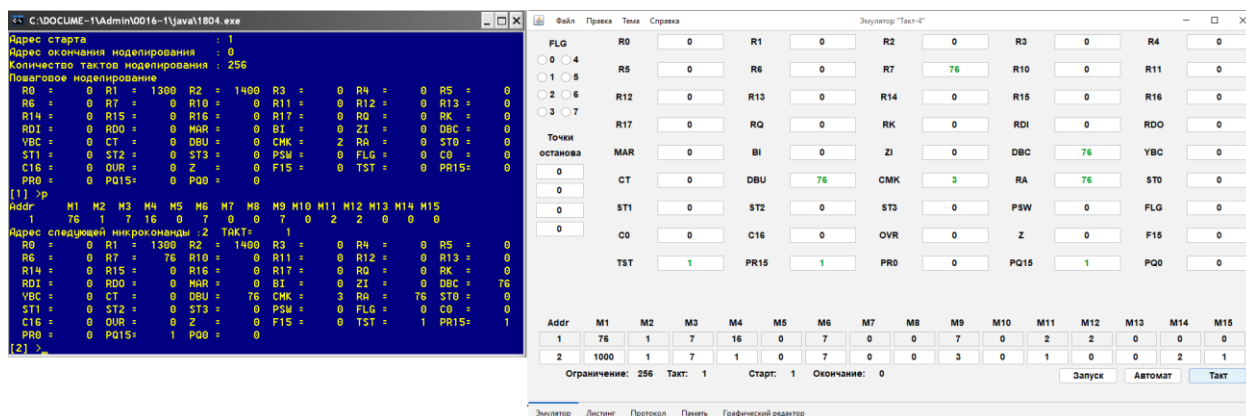
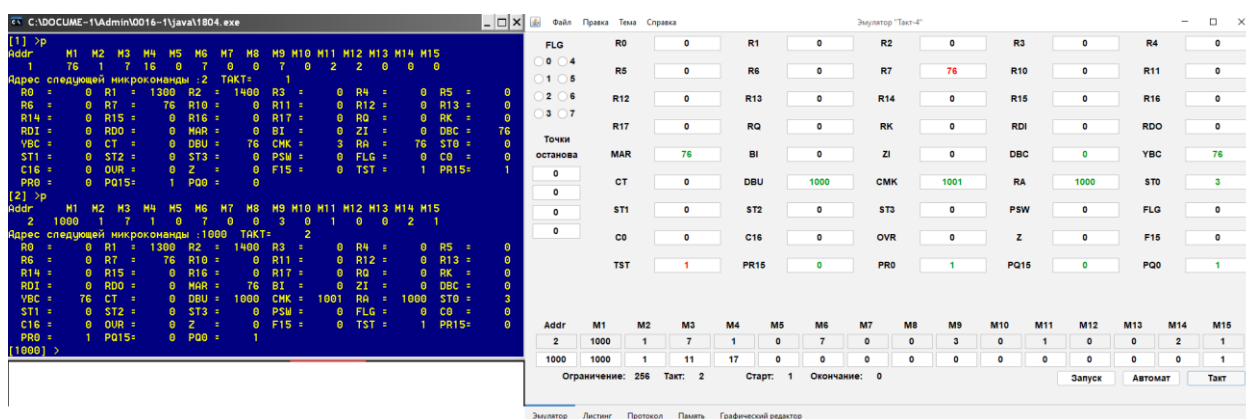
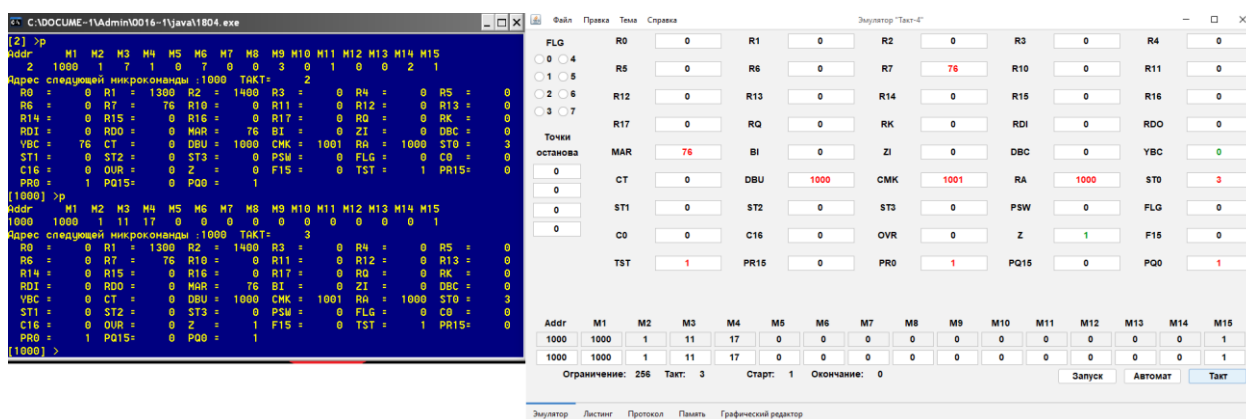


Рисунок 1.25 – Сравнение программ. Такт 0



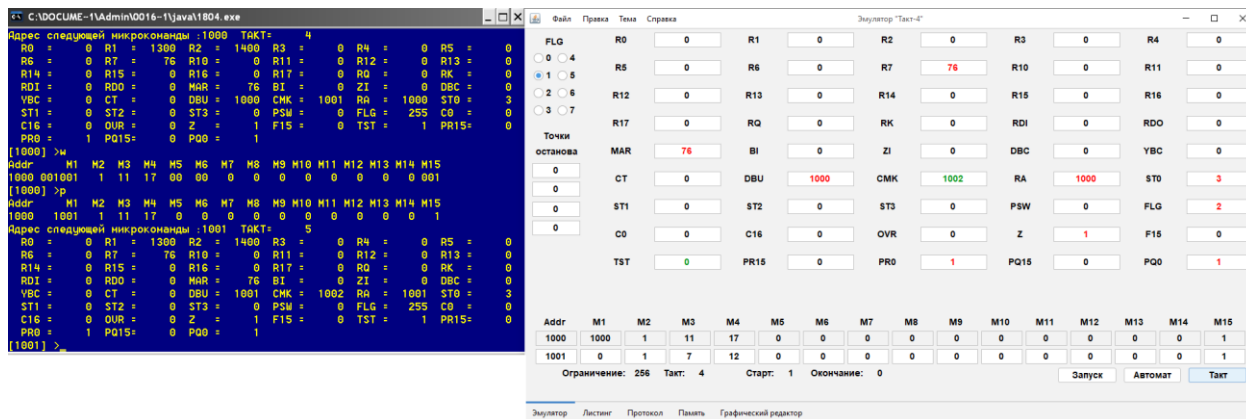


Рисунок 1.29 – Сравнение программ. Такт 4

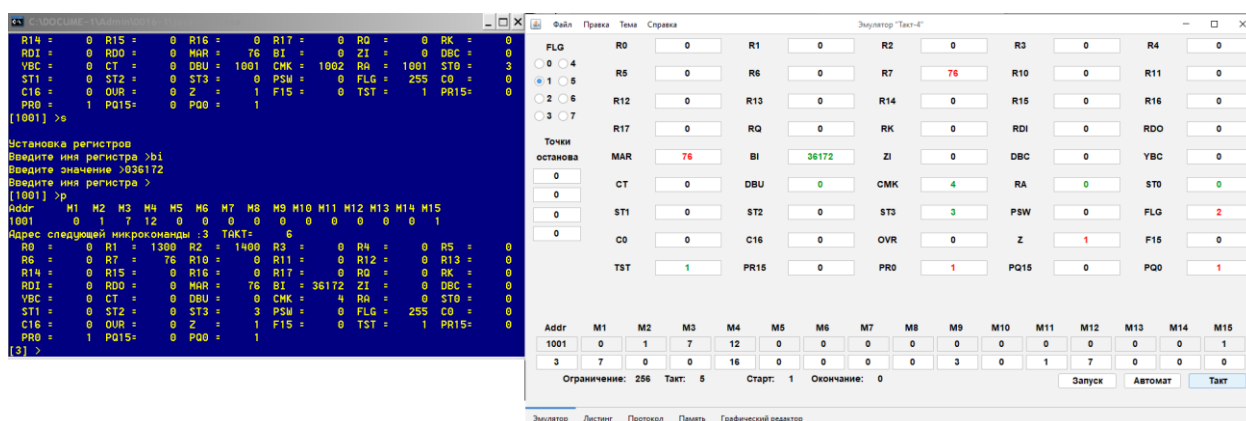


Рисунок 1.30 – Сравнение программ. Такт 5

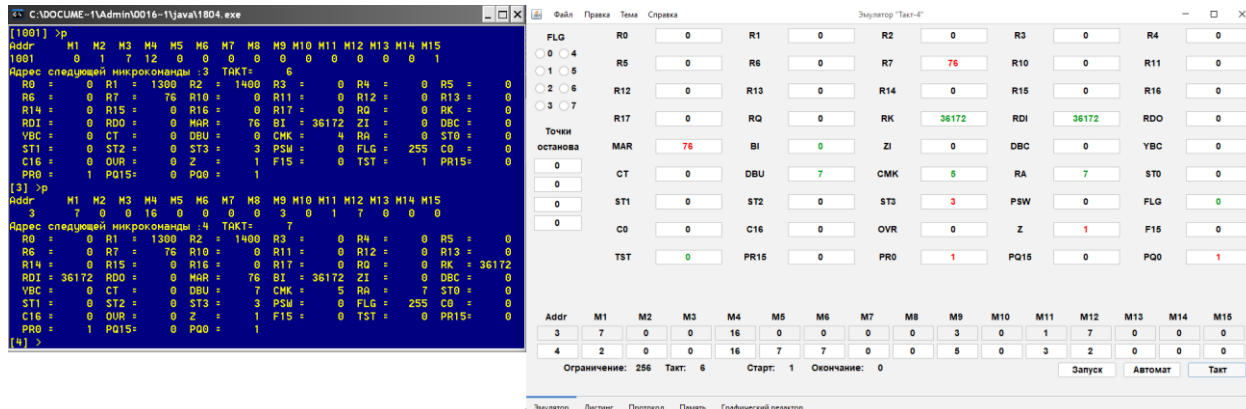


Рисунок 1.31 – Сравнение программ. Такт 6

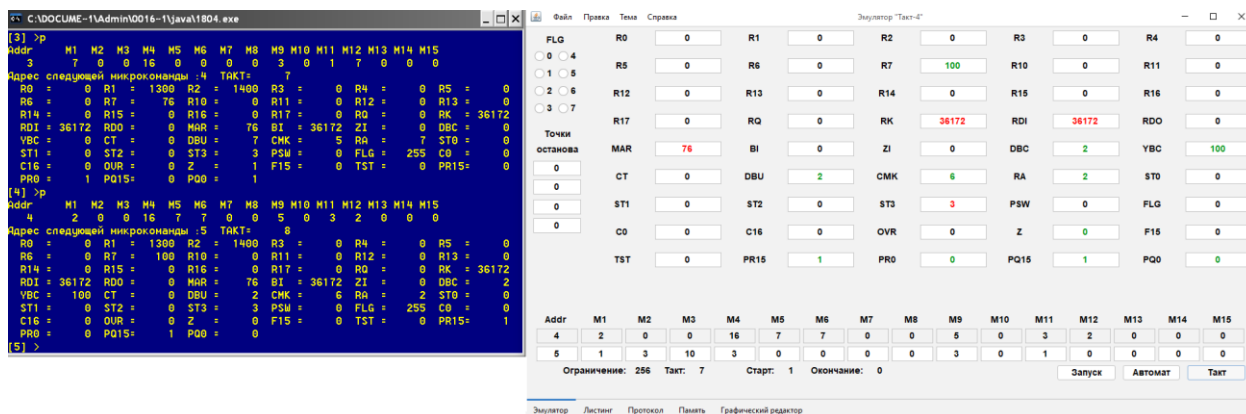


Рисунок 1.32 – Сравнение программ. Такт 7

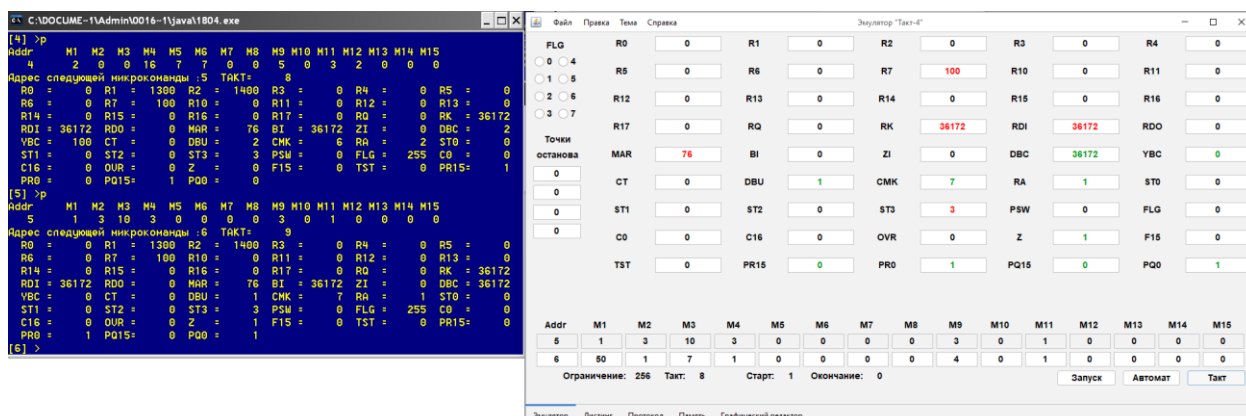


Рисунок 1.33 – Сравнение программ. Такт 8

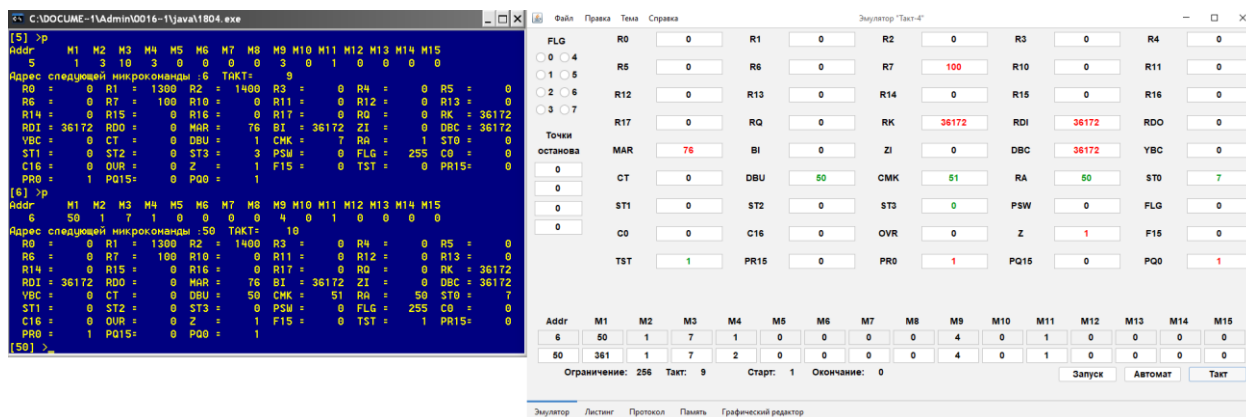


Рисунок 1.34 – Сравнение программ. Такт 9

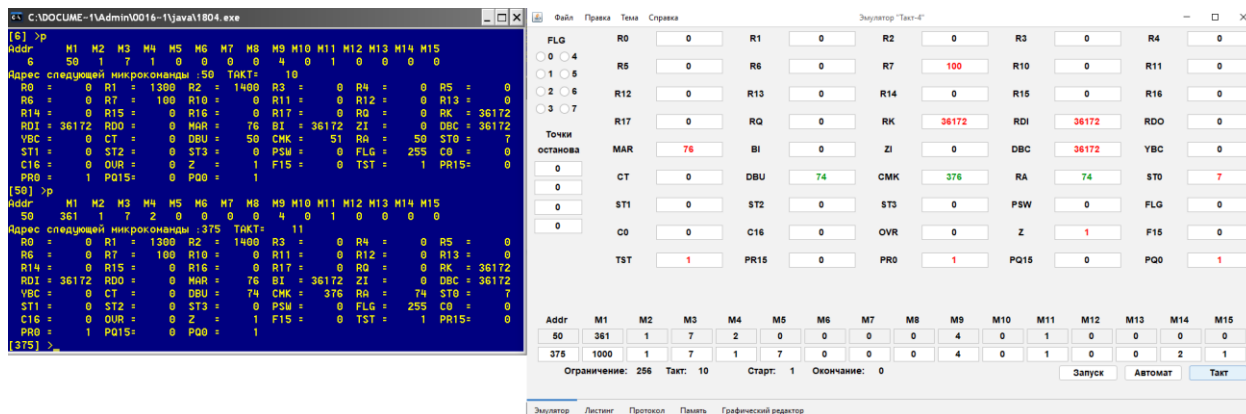


Рисунок 1.35 – Сравнение программ. Такт 10

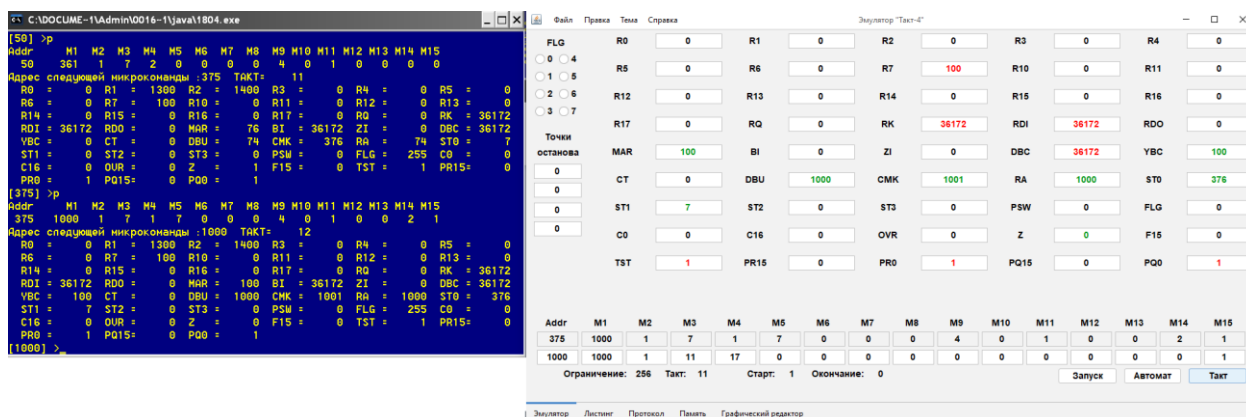


Рисунок 1.36 – Сравнение программ. Такт 11

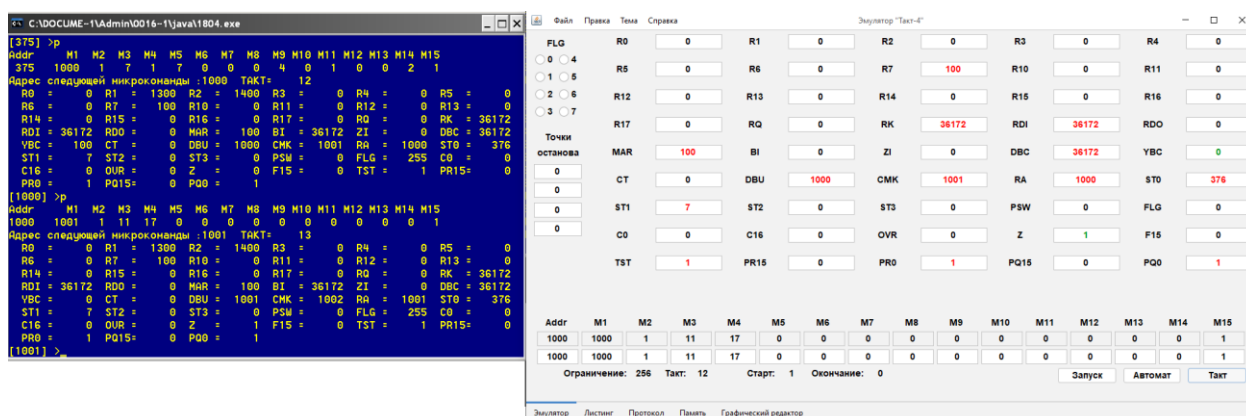


Рисунок 1.37 – Сравнение программ. Такт 12

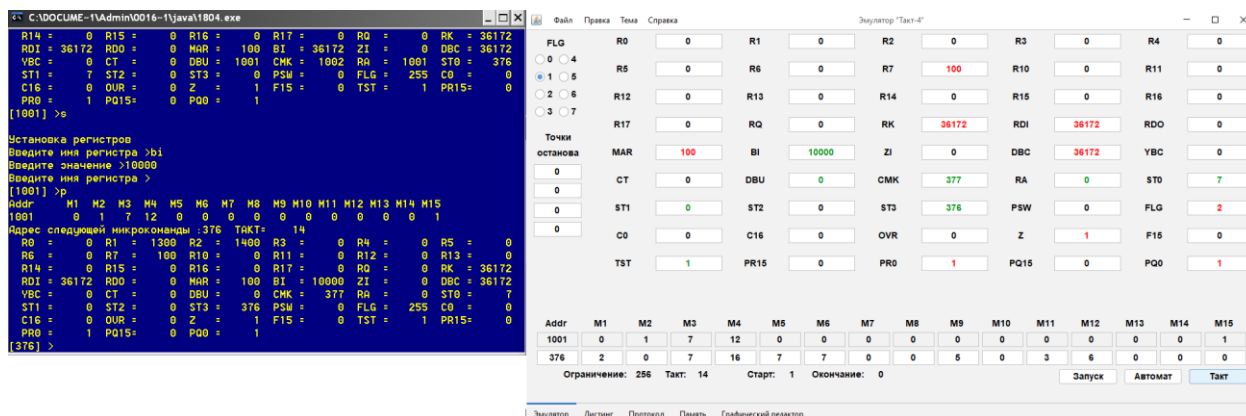


Рисунок 1.38 – Сравнение программ. Такт 13

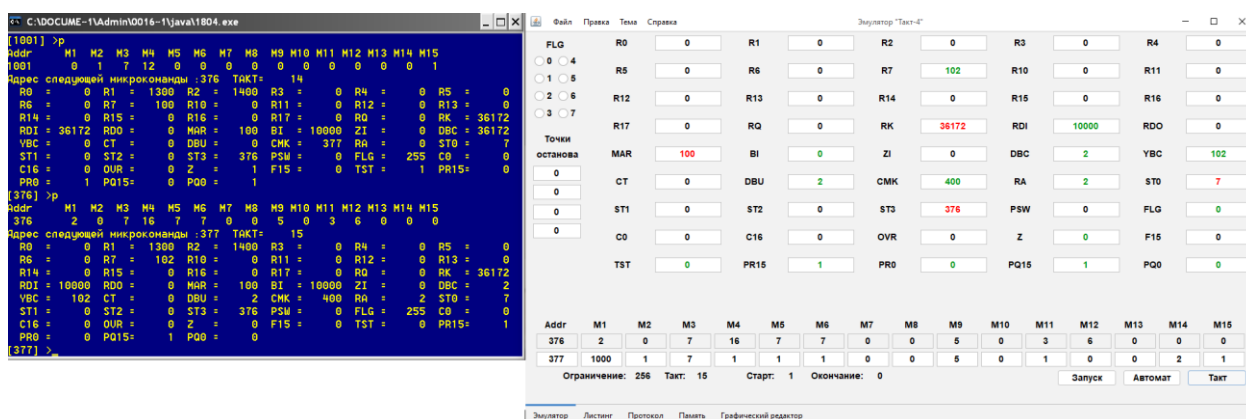


Рисунок 1.39 – Сравнение программ. Такт 14

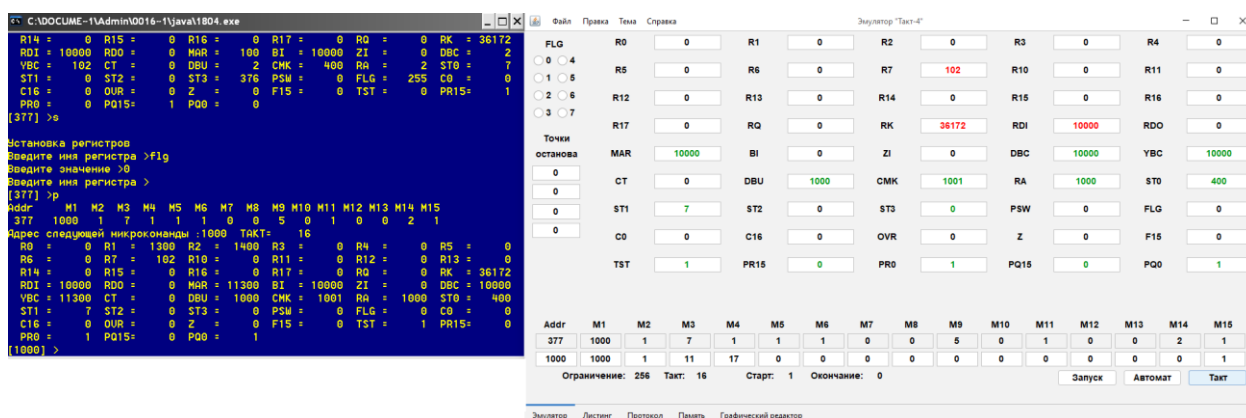


Рисунок 1.40 – Сравнение программ. Такт 15

Из проведённых выше тестов можно заметить, что все регистры совпадают, а значит программа работает корректно.

2 Безопасность жизнедеятельности

Персональная электронная вычислительная машина является источником ряда вредных и опасных факторов производственной среды: электромагнитных полей, статического электричества. Нередко условия труда при работе на ПЭВМ усугубляются повышенными уровнями шума, неудовлетворительными микроклиматическими условиями и недостаточной освещенностью на фоне зрительного и нервно – эмоционального напряжения.

ПЭВМ используются для разработки учебного стимулятора микропрограммируемой ЭВМ, с написанием программного кода, его отладки и компиляции, а также тестированием и оптимизацией. Данные операции характеризуются высокой скоростью переработки информации, высоким темпом работы, низкой потребностью в обмене информацией и сопровождается принятием решений. Работа требует большого умственного и зрительного напряжения, сопровождается локальными мышечными нагрузками.

Особенностью работы на ПЭВМ является постоянное напряжение функций зрительного анализатора, обусловленное необходимостью различения объектов символов, знаков и т.п.

Нервно-эмоциональное напряжение возникает вследствие дефицита времени, большого объема и плотности информации, особенностей диалогового режима общения человека и ПЭВМ, ответственности за безошибочность информации.

Темп работы на ПЭВМ при вводе информации определяется объемом и характером производственного задания, и временем его выполнения. В период выполнения операций ввода данных количество мелких стереотипных движений кистей и пальцев рук за рабочий день может превысить 60 тыс., что в соответствии с гигиенической классификацией труда относится к категории вредных и опасных.

Характеристики помещения отдела офиса приведены в таблице 2.1.

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

Таблица 2.1 – Характеристика помещения отдела офиса

Наименование	Значение
Ширина	4,2 м
Длина	3,8 м
Высота потолка	3,0 м
Площадь помещения	15,96 м ²
Объем помещения	47,88 м ³
Количество рабочих	2 человека
Объем помещения на каждого работающего	23,94 м ³

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при реализации разрабатываемого проекта

Одним из наиболее важных и масштабных направлений в охране трудовой деятельности – это обеспечение максимально комфортных и безопасных для работника факторов при выполнении им функциональных обязанностей.

В процессе реализации разрабатываемого продукта на рабочем месте на программиста могут воздействовать опасные, вызывающие травмы и вредные, вызывающие заболевания производственные факторы. Вредный производственный фактор, в зависимости от интенсивности и продолжительности воздействия, может стать опасным.

Обеспечение безопасной жизнедеятельности программиста в значительной степени зависит от правильной оценки опасных, вредных производственных факторов. Одинаковые по тяжести изменения в организме человека могут быть вызваны различными причинами. Это могут быть какие-либо факторы производственной среды, чрезмерная физическая и умственная

нагрузка, нервно-эмоциональное напряжение, а также разное сочетание этих причин. Рассмотрим подробнее опасные и вредные производственные факторы на рабочем месте программиста:

- повышенный уровень напряжения в электрической цепи, замыкание которой может пройти через тело работающего;
- превышающий уровень шума;
- недостаточная освещенность рабочей зоны;
- повышенная и пониженная температура воздуха;
- чрезмерная запыленность и загазованность воздуха;
- повышенная и пониженная влажность воздуха;
- повышенный уровень ионизирующего излучения;
- повышенный уровень электромагнитных полей;
- повышенный уровень статического электричества;
- опасность поражения электрическим током;
- блеклость экрана дисплея.

Также на программиста влияют опасные и вредные производственные факторы, обладающие свойствами психофизиологического воздействия: умственное перенапряжение, перенапряжение зрительных и слуховых анализаторов, монотонность труда, эмоциональные перегрузки.

Действие указанных неблагоприятных факторов приводит к снижению работоспособности, вызванное развивающимся утомлением.

2.2 Классификация и категорирование помещения

По взрывопожарной и пожарной опасности помещения подразделяются на категории А, Б, В1-В4, Г и Д. Согласно Нормам пожарной безопасности НПБ 105-03, помещения с ЭВМ и ПЭВМ относятся к категории В (пожароопасные), то есть горючие и трудногорючие жидкости, твердые горючие и трудногорючие вещества и материалы, вещества и материалы,

способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть [12].

Пожары классифицируются по виду горючего материала и подразделяются на следующие классы: пожары твердых горючих веществ и материалов (*A*), пожары горючих жидкостей или плавящихся твердых веществ и материалов (*B*), пожары газов (*C*), пожары металлов (*D*), пожары горючих веществ и материалов электроустановок, находящихся под напряжением (*E*), пожары ядерных материалов, радиоактивных отходов и радиоактивных веществ (*F*).

Наиболее вероятные классы пожаров в помещениях с ПЭВМ – «*A*» и «*E*», то есть могут гореть в основном твердые вещества, горение которых сопровождается тлением – класс *A*, или возможны пожары, вызванные возгоранием электрооборудования – класс *E*.

Для защиты от поражения человека электрическим током при устройстве помещений необходимо предусматривать меры обеспечения безопасности. С целью их оптимального выбора разработана классификация помещений. Все помещения делятся по степени поражения людей электрическим током на три класса: без повышенной опасности, с повышенной опасностью, особо опасные.

Помещения без повышенной опасности – это сухие, беспыльные помещения с нормальной температурой воздуха и с изолирующими полами.

Помещения с повышенной опасностью характеризуются наличием хотя бы одного из следующих пяти условий: сырости, когда относительная влажность воздуха длительно превышает 75%; высокой температуры, когда температура воздуха свыше суток превышает +35° С; токопроводящей пыли, когда по условиям производства в помещениях выделяется токопроводящая технологическая пыль в таком количестве, что она оседает на проводах, проникает внутрь машин и т. п.; токопроводящих полов; возможности одновременного прикосновения человека к имеющим

соединение с землей металлоконструкциям зданий, механизмам и т.п., с одной стороны, и к металлическим корпусам электрооборудования – с другой.

Особо опасные помещения характеризуются наличием одного из следующих трех условий: особой сырости, когда относительная влажность воздуха близка к 100%; помещениями с химически активной или органической средой; одновременного наличия двух и более условий.

Из выше перечисленных условий, можно сделать вывод, что помещение где ведется разработка стимулятора относится к категории помещений без повышенной опасности, так как оно сухое, беспыльное, с нормальной температурой воздуха 21 °С и с изолирующими полами.

2.3 Санитарно-гигиеническая характеристика помещения, отдела офиса

Требования к условиям труда программиста, работающего с ПЭВМ и регламентирует СанПиН 2.2.2./2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы»; общие рекомендации по электробезопасности, а также особенности воздействия других факторов даны в «Типовой инструкции по охране труда при работе на персональном компьютере» (ТОИ Р-45-084-01) [13].

2.3.1 Требования к помещениям для работы с ПЭВМ

Помещения для работы на ПЭВМ не разрешается размещать в подвалах. Помещения должны иметь оконные проемы для проникновения естественного света, ориентированные преимущественно на север или северо-запад. Окна, следует оборудовать устройствами для регулирования яркости естественного освещения, такими как шторы, жалюзи.

Объем производственного помещения на одного работающего должен составлять не менее 20 м³, площадь – не менее 6,0 м².

Материалы, применяемые для внутренней отделки помещения, должны иметь гигиенический сертификат, подтверждающий их безопасность.

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

Помещения должны находиться вдали от силовых кабелей, высоковольтных трансформаторов и другого оборудования, способного создавать помехи для офисной техники. В комнатах необходимо организовать защитное заземление, обеспечивающее безопасную работу компьютеров.

Звукоизоляция помещений должна отвечать гигиеническим требованиям и обеспечивать нормируемые параметры шума.

Поверхность пола должна быть ровной, удобной для очистки и влажной уборки, обладать антистатическими свойствами.

2.3.2 Организация рабочих мест

Эргономическими аспектами организации рабочих мест инженера-программиста являются: высота рабочей поверхности, размеры пространства для ног, расстояние от глаз пользователя до экрана, документа, клавиатуры, характеристики рабочего кресла, требования к поверхности рабочего стола, регулируемость элементов рабочего места. Рабочие условия должны обеспечивать комфортную работу, так как правильная организация рабочего места программиста повышает производительность труда. Следуя рекомендациям ГОСТ 12.2.032-78, необходимо организовать рабочее место таким образом, чтобы взаимное расположение всех его элементов соответствовало физическим и психологическим требованиям.

При размещении рабочих мест с ПЭВМ расстояние между рабочими столами с мониторами должно быть не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м.

Рабочие места с ПЭВМ при выполнении творческой работы, требующей значительного умственного напряжения и высокой концентрации внимания, рекомендуется изолировать друг от друга перегородками высотой 1,5 – 2 м.

Экран видеомонитора должен находиться от глаз на расстоянии 600 – 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов.

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		80

Визуальные эргономические параметры ПЭВМ и пределы их изменений, в которых должны быть установлены оптимальные и допустимые диапазоны значений, представлены в таблице 2.2.

Таблица 2.2 – Визуальные эргономические параметры и пределы их изменений

Наименование параметров	Пределы значений параметров	
	Минимальные	Максимальные
Яркость знака (яркость фона), кд/куб. м	35	120
Внешняя освещенность экрана ПК	100	250
Угловой размер знака, угл. мин.	16	60

Конструкция рабочей мебели должна обеспечивать возможность индивидуальной регулировки, соответственно росту программиста, и создавать удобную позу. Часто используемые предметы труда и органы управления должны находиться в оптимальной рабочей зоне.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики: ширину – 800, 1000, 1200 и 1400 мм, глубину – 800 и 1000 мм при нерегулируемой его высоте, равной 725 мм.

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Клавиатуру следует располагать на поверхности стола на расстоянии 100 – 300 мм от края, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности, отделенной от основной столешницы. Опорное приспособление, позволяющее изменить угол наклона

поверхности клавиатуры в пределах от 5 до 15 градусов. Высота среднего ряда клавиш не более 30 мм, с одинаковым ходом для всех клавиш с минимальным сопротивлением нажатию 0,25 Н и максимальным – не более 1,5 Н.

Конструкция рабочего стула (кресла) кресла должна обеспечивать поддержание рациональной рабочей позы при работе на ПЭВМ, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления.

Кресло должно быть снабжено подъемно-поворотным, регулируемым по высоте в пределах 400 – 500 мм, глубине не менее 380 мм и ширине не менее 400 мм. Угол наклона спинки кресла к плоскости сиденья должен изменяться в пределах 90 – 110°. Регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Поверхность сиденья, спинки и других элементов кресла должна быть полумягкой, с нескользящим, слабо электризующимся и воздухопроницаемым покрытием.

2.3.3 Микроклимат рабочего помещения

Микроклимат рабочего помещения – это климат внутренней среды помещения, который объединяет такие параметры, действующие на организм человека, как температура, относительная влажность и скорость движения воздуха (см. Таблица 2.3).

Офис является помещением Iа категории, где выполняются легкие физические работы, поэтому соблюдаются следующие требования: средняя температура воздуха в помещении офиса составляет +22°C, относительная влажность – 47%, атмосферное давление – 750 мм.рт.ст., содержание пыли – не более 10 мг/м воздуха рабочего места, максимальные размеры частиц – 2 мкм, скорость движения воздуха до 0,1м/с.

Для поддержания оптимального микроклимата, независимо от наружных условий: в холодное время года используется водяное отопление, в теплое время года применяется кондиционирование воздуха. Температура

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82

воздуха в помещении офиса не должна опускаться ниже +19°C, и не должна превышать +25°C.

Таблица 2.3 – Оптимальные нормы микроклимата для помещений с ПЭВМ

Период года	Категория работ	Температура воздуха, °C	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	Легкая – 1а	22 – 24	40 – 60	Не более 0,1
	Легкая – 1б	21 – 23	40 – 60	Не более 0,1
Теплый	Легкая – 1а	23 – 25	40 – 60	Не более 0,1
	Легкая – 1б	22 – 24	40 – 60	Не более 0,1

Оптимальные показатели микроклимата обеспечивают состояние нормального функционального и теплового состояния организма человека без напряжения механизма терморегуляции. Они обеспечивают ощущение комфорта и высокую работоспособность сотрудников.

2.3.4 Шум и вибрация

Шум ухудшает условия труда и оказывает вредное действие на организм человека. Вентиляционные установки, кондиционеры и периферийное оборудование для ПЭВМ могут стать источниками шумовых помех на рабочем месте программиста. Длительное воздействие этих шумов отрицательно сказывается на эмоциональном состоянии персонала, они испытывают раздражительность, головокружение, головные боли, снижение памяти, повышенную утомляемость, боли в ушах. Все это снижает работоспособность человека и его производительность, качество и безопасность труда.

Согласно ГОСТ и 12.1.003-76 ССБТ эквивалентный уровень шума не должен превышать 50 дБ(А). Длительное воздействие интенсивного шума выше 80 дБ(А) на слух человека приводит к его частичной или полной потере.

Мое рабочее место в офисе с низким уровнем общего шума. В качестве мер по снижению шума предприняты следующие меры:

- облицовка потолка и стен выполнены звукопоглощающим материалом;
- экранирование рабочего места;
- использование в работе оборудования, производящего минимальный шум.

Вибрация в помещении отсутствует.

2.3.5 Ионизирующие и неионизирующие излучения

Основным источником электромагнитных излучений от мониторов ПЭВМ является трансформатор высокой частоты строчной развертки. Конструкция монитора ПЭВМ должна обеспечивать мощность экспозиционной дозы рентгеновского излучения в любой точке на расстоянии 0,05 м от экрана и корпуса монитора при любых положениях регулирующих устройств и не должна превышать $7,74 \times 10^{-4}$ А/кг, что соответствует эквивалентной дозе, равной 0,1 мбэр/час (100 мкр/час).

Напряженность электромагнитного поля на расстоянии 50 см вокруг ПЭВМ по электрической составляющей должна быть не более:

- в диапазоне частот 5 Гц - 2 кГц - 25 в/м;
- в диапазоне частот 2 - 400 кГц - 2,5 в/м.

Плотность магнитного потока индукции должна быть не более:

- в диапазоне частот 5 Гц - 2 кГц - 250 нТл;
- в диапазоне частот 2 - 400 кГц - 25 нТл.

Поверхностный электростатический потенциал не должен превышать 500 в.

2.3.6 Освещение рабочего места

Правильно спроектированное освещение улучшает условия зрительной работы, снижает утомляемость, способствует повышению производительности труда, благотворно влияет на производственную среду,

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

оказывая положительное психологическое воздействие на персонал, повышает безопасность труда и снижает травматизм.

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах.

Неправильное направление света на рабочем месте может создавать резкие тени, блики, дезориентировать работающего. Все эти причины могут привести к несчастному случаю или профзаболеваниям.

Рабочие столы следует размещать таким образом, чтобы экраны были ориентированы боковой стороной к световым проемам, чтобы естественный свет падал преимущественно слева.

Согласно СанПин 2.2.2/2.4.1340-03 необходимо применить систему комбинированного освещения, так как естественное освещение слабое, на рабочем месте должно применяться и искусственное освещение. Освещение не должно создавать бликов на поверхности экрана ПЭВМ, освещенность не должна быть более 300 лк.

2.4 Электробезопасность

Электробезопасность – это комплекс мер и действий, направленных на минимизацию риска причинения вреда электротоком, а также магнитным и электрическим полями или статическим электричеством.

Офисное оборудование, включая ПЭВМ, относится к электрическим приборам, поэтому работодатель должен обеспечить электробезопасность работников. Во избежание короткого замыкания, а значит, возникновения пожара и получения электротравмы, помещения офиса, где размещаются рабочие места с ПЭВМ, должны быть оборудованы защитным заземлением (занулением) в соответствии с техническими требованиями по эксплуатации СанПиН 2.2.2/2.4.1340-03.

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		85

Помещение офиса для разработки стимулятора относится к классу помещений без повышенной опасности, помещение сухое, с нормальной температурой воздуха, с токонепроводящим полом, без токопроводящей пыли и малым числом заземленных приборов. Сопротивление изоляции для таких помещений должно соответствовать норме $R_{\text{изол}}$ больше 0,5 МОм.

Питание оборудования осуществляется электричеством 220В, 50Гц, однофазная сеть с дополнительным заземляющим проводом. Данное помещение относится к 1а классу помещений без повышенной опасности, а также не имеет наружных электроустановок. Из сказанного выше следует, что защитное заземление выполнять не следует. При необходимости, оно должно составлять $R_{\text{заземления}}$ меньше 4 Ом.

В офисе имеется плоттер, сканер и лазерный принтер. Все эти устройства являются законченными изделиями и удовлетворяют требованиям электробезопасности. Вероятность поражения человека электрическим током при правильном использовании оргтехники практически сведена к нулю.

Электробезопасность в помещении офиса обеспечивается техническими способами и средствами защиты, а также организационными и техническими мероприятиями.

Разберем основные причины поражения человека электрическим током на рабочем месте при разработке симулятора:

- прикосновение к металлическим нетоковедущим частям, таким как корпус, периферия компьютера, которые могут оказаться под напряжением в результате повреждения изоляции;
- при касании незащищенных контактов, в случае разборки ПЭВМ;
- нерегламентированное использование электрических приборов;
- отсутствие инструктажа сотрудников по правилам электробезопасности.

При работе ПЭВМ, на его корпусе накапливается статическое электричество. На расстоянии 5 – 10 см от экрана напряженность

электростатического поля составляет 60 – 280 кВ/м, то есть в 10 раз превышает норму 20 кВ/м. Для уменьшения напряжённости применять применение увлажнители и нейтрализаторы, антистатическое покрытия пола.

2.5 Пожарная безопасность

Пожарная безопасность – это вероятность исключения возможности возникновения и развития пожара и воздействия на людей опасных факторов пожара, а также обеспечивается защита материальных ценностей.

Основными причинами пожара при работе программиста являются: короткое замыкание, плавление изоляции проводов, их оголение, и как следствие, короткое замыкание, которое сопровождается искрением и ведет к перегрузкам элементов электронных схем, которые, перегреваясь, сгорают с искрением, поэтому следует большое внимание оказывать пожарной безопасности.

Помещение, где установлены ПЭВМ, относятся к категории “В” - пожароопасные.

В соответствии с требованиями пожарной безопасности в помещении установлена пожарная сигнализация, в отделе офиса имеются дымовые извещатели в количестве 2 штук. Количество извещателей выбирается исходя из площади помещения, высоты потолков и требований по установке извещателей, для высоты потолка до 3,5 м.

Для тушения пожаров в начальной стадии развития могут быть использованы огнетушители углекислотный ОУ-5 и порошковый ОП-5-01 предназначенный для тушения электроустановок, находящихся под напряжением до 1000 В, применяется для тушения ПЭВМ и оргтехники. При входе в помещение отдела стоит один огнетушитель ОУ-5, из расчета один огнетушитель на 40 – 50 м² площади, но не менее двух в помещении.

Для ликвидации пожара в начальной стадии в коридоре офиса размещается пожарный кран. В отделе, где установлена вычислительная

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		87

техника, недопустимо применять воду и пенные огнетушители, так как в этом случае существует опасность повреждения или полного выхода из строя ПЭВМ и другого оборудования.

Для профилактики возникновения пожара необходимо вовремя выявлять и устранять неисправности, проводить плановый осмотр и своевременно устранять все неисправности электрооборудования, электропроводки, электрических розетках и выключателях. Выполнять своевременный ремонт электроприборов, качественное исправление поломок. В целях уменьшения вероятности возникновения пожара вследствие короткого замыкания необходимо, чтобы электропроводка была скрытой.

Мебель в помещениях не должна препятствовать быстрой эвакуации людей. Расположение электрических кабелей и различных проводов должно исключать их повреждение, поражение работников электрическим током, а также они не должны мешать передвижению по помещению.

В летний период во время грозы возможно попадание молнии вследствие чего возможен пожар. Во избежание этого необходимо установить на крыше здания молниеотвод.

В целях предотвращения пожара рекомендуется проводить с программистами, работающими в офисе, противопожарный инструктаж, на котором ознакомить с правилами противопожарной безопасности, а также обучить использованию первичных средств пожаротушения.

В случае возникновения пожара необходимо отключить электропитание, вызвать по телефону пожарную команду, эвакуировать людей из помещения согласно плану эвакуации, и приступить к ликвидации пожара огнетушителями. При наличии небольшого очага пламени можно воспользоваться подручными средствами с целью прекращения доступа воздуха к объекту возгорания.

2.6 Расчет общего искусственного освещения помещения, где работают на ПЭВМ

Целью расчета является определение числа и расположения светильников, а также числа и мощности ламп, необходимых для обеспечения требуемого нормативного уровня освещенности для работы программиста на ПЭВМ.

Расчет искусственного освещения будет проводиться для помещения площадью $S = 15,96 \text{ м}^2$, длина $a = 3,8 \text{ м}$, ширина $b = 4,2 \text{ м}$, высота $h = 3,0 \text{ м}$.

В соответствии с СанПиНом 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы» освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300 – 500 лк. Для расчета требуемый нормативный уровень освещенности E_n выберем равной 400 лк.

Определяем разряд и подразряд зрительной работы, нормы освещённости на рабочем месте по таблице 4: характеристика зрительной работы – высокой точности, разряд – 3, подразряд – г, комбинированное освещение – 400 лк, общее освещение – 200 лк.

Размещение светильников определяется следующими размерами:

Высота светильников над полом рассчитывается по формуле (2.1):

$$h_{\pi} = H - h_c, \quad (2.1)$$

где, $H = 3 \text{ м}$ высота помещения;

$h_c = 0,25 \text{ м}$ – расстояние светильников от перекрытия.

Расчёт высоты от пола до светильника по формуле (2.1):

$$h_{\pi} = 3 - 0,25 = 2,75 \text{ м}.$$

Расчет высоты от рабочей поверхности до светильника по формуле (2.2):

$$h = h_{\text{п}} - h_{\text{р}}, \quad (2.2)$$

где, $h_{\text{п}}$ – высота от пола до светильника;

$h_{\text{р}}$ – высота расчетной поверхности равна 0,7 м (для помещений, связанных с работой ПЭВМ).

Математический расчет высоты от поверхности до светильников выполняется по формуле (2.2):

$$h = 2,7 - 0,7 = 2,05 \text{ м.}$$

Рассчитываем число светильников N по формуле (2.3):

$$N = \frac{S}{L \cdot M}, \quad (2.3)$$

где, S – площадь помещения;

L – расстояние между центрами светильников;

M – расстояние между параллельными рядами.

Площадь помещения находится по формуле (2.4):

$$S = a \cdot b, \quad (2.4)$$

где, a – длина;

b – ширина.

Расстояние между центрами светильников рассчитывается по формуле (2.5):

$$L = 1,75 \cdot H. \quad (2.5)$$

Выполним расчеты по формулам (2.4) и (2.5):

$$S = 4,2 \cdot 3,8 = 15,96 \text{ м}^2;$$

$$L = 4 \cdot 1,75 = 5,25 \text{ м.}$$

Рассчитаем расстояние между параллельными рядами – M по формуле (2.6):

$$M = 0,6 \cdot H_p, \quad (2.6)$$

где, $H_p = H$.

Исходя из формулы (2.6) принимаем $M = 2$ м. Подставим полученные значения в формулу (2.3):

$$N = \frac{15,96}{5,25 \cdot 2} = 1,5.$$

Таким образом, принимаем $N = 2$ шт.

Определим расчетный световой поток по формуле (2.7):

$$\Phi = \frac{E_n \cdot S \cdot Z \cdot K}{N \cdot \mu}, \quad (2.7)$$

где, Z – коэффициент неравномерности освещения равный 1,1;

K – коэффициент учитывающий загрязнение светильника равный 1,5;

E_n – нормируемая минимальная освещенность равная 300;

μ – коэффициент использования светового потока равный 0,61.

Произведем математический расчет светового потока по формуле (2.7):

$$\Phi = \frac{300 \cdot 15,96 \cdot 1,1 \cdot 1,5}{2 \cdot 0,61} = 6\,475 \text{ лм.}$$

Индекс помещения определяется по формуле (2.8):

$$i = \frac{a \cdot b}{h_p \cdot (a+b)}, \quad (2.8)$$

где, a – ширина помещения;

b – длина помещения;

h_p – высота рабочей поверхности над полом.

Подставляя значение в формулу (2.8) найдем индекс помещения:

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

$$i = \frac{4,2 \cdot 3,8}{0,7 \cdot (4,2 + 3,8)} = 2,85.$$

Зная индекс помещения и коэффициенты отражения, из таблицы коэффициентов использования определяем $n = 0,46$.

Подставляем полученные значения в формулу (2.3) для определения количества светильников:

$$N_{cu} = \frac{400 \cdot 15,96 \cdot 1,1 \cdot 1,5}{6475 \cdot 2 \cdot 0,46} = 1,77.$$

Таким образом, принимаем количество светильников равным 2, каждый из которых укомплектован двумя люминесцентными лампами.

В данном разделе дипломной работы был проведен анализ вредных и опасных производственных факторов, действующих на рабочем месте инженера-программиста. На основании изученной литературы по данной проблеме, были указаны оптимальные размеры рабочего стола и кресла, а также сформулированы предложения по улучшению параметров рабочего места. Соблюдение условий, определяющих оптимальную организацию рабочего места инженера-программиста, позволит сохранить хорошую работоспособность в течение всего рабочего дня, повысит как в количественном, так и в качественном отношении производительность труда программиста, что в свою очередь будет способствовать быстрее разработке и отладке программного продукта.

Так как работа программиста по виду трудовой деятельности относится к группе В – творческая работа в режиме диалога с ПЭВМ, а по напряженности работы ко II категории тяжести (СанПиН 2.2.2.542-96), необходимо сократить время работы за ПЭВМ, делать перерывы, суммарное время которых должно составлять 50 минут при 8-ми часовой смене. Площадь рабочего помещения составляет 7,9 м² и объем помещения на одного работающего составляет 23,94 м³, что соответствует нормам СанПин.

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

Созданные условия должны обеспечивать комфортную работу программиста.

Заключение

В ходе выполнения данной выпускной квалификационной работы была разработана программно-логическая модель 16 – разрядной микропрограммируемой ЭВМ, состоящая из процессора и оперативной памяти емкостью до 16 Кбайт. Согласно заданию, задачи и цели, которые были поставлены на выпускную квалификационную работу, выполнены.

Система протестирована и готова к использованию в учебном процессе для подготовки бакалавров и специалистов в области вычислительной техники.

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		94

Список использованных источников

1. Коннов Н. Кросс-платформенный эмулятор микропрограммируемой ЭВМ / Н.Н. Коннов, А.И. Пчелинцев, В.Д. Федулов., О.В. Юрова //Сборник научных статей по материалам XVIII Международной научно-технической конференции Новые информационные технологии и системы (НИТиС-2021). – Пенза: Изд-во Пенз. гос. технол. ун-тет., 2021.

2. Коннов Н. Н. Учебный кроссплатформенный эмулятор микропрограммируемой ЭВМ / Н.Н. Коннов, А.И. Пчелинцев, В.Д. Федулов, А.С. Бычков // Информатика: проблемы, методы, технологии: сборник материалов XXII международной научно-практической конференции // под редакцией Д. Н. Борисова; Воронеж, Воронежский государственный 1 университет, 10-12 февраля 2022 г. – Воронеж : «ВЭЛБОРН», 2022.

3. Гурин Е. И., Коннов Н. Н., Механов В. Б. Проектирование процессора ЭВМ на секционированных микропроцессорных БИС: Методические указания к курсовому проектированию [Электронный ресурс] / Режим доступа: <http://window.edu.ru/resource/316/66316>.

4. Коннов, Н. Учебный программный эмулятор ЭВМ с микропрограммным управлением / Н. Н. Коннов, С. Е. Филин, П. П. Вирясов, К. О. Барабаш // Университетское образование (МКУО– 2014): сб. ст. XVIII Междунар. науч.-метод. конф., посвящ. 200-летию со дня рождения М. Ю. Лермонтова (г. Пенза, 10–11 апреля 2014 г.) // под ред. А. Д. Гулякова, Р. М. Печерской. – Пенза : Изд-во ПГУ, 2014.

5. Коннов, Н. Программный симулятор ЭВМ с микропрограммным управлением / Н.Н. Коннов, Н.В. Исхаков, Д.А. Прокуров // Актуальные вопросы современной науки: теория и практика научных исследований : сб. науч. ст. II Всероссийской научно-практической конференции (Пенза, 1-15 ноября 2018 г.) – Пенза: Изд-во Пенз. гос. технол. ун-тет., 2018.

6. Коннов, Н. Модернизация учебного программного обеспечения по курсу ЭВМ и периферийные устройства / Н. Н. Коннов, А. П. Любезнов, М.

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95

Ю. Новиков, И. В. Янович // Информационные технологии в науке и образовании. Проблемы и перспективы : сб. науч. ст. III ежегодной межвуз. студ. науч.-практ. конф. – Пенза : Изд-во ПГУ, 2016.

7. Таненбаум Э. Архитектура компьютера / Э. Таненбаум– СПб. : Б Питер, 2018.

8. Гурин Е. И., Коннов Н. Н., Механов В. Б. Проектирование процессора ЭВМ на секционированных микропроцессорных БИС: Методические указания к курсовому проектированию [Электронный ресурс] / Режим доступа: <http://window.edu.ru/resource/316/66316>.

9. Шилдт Г. *Java 8. Полное руководство* / Герберт Шилдт; – М.: «Питер». Вильямс", 2015.

10. Монахов В.В. Язык программирования *Java* и среда *NetBeans*: [учеб. пособие] / В.В. Монахов. – СПб.: БХВ-Петербург, 2011.

11. Гуськова О. И. Объектно ориентированное программирование в *Java*: учебное пособие / О. И. Гуськова. – Москва : МПГУ, 2018.

12. Нормы пожарной безопасности НПБ 105-03 "Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности" / утв. Приказом №314 МЧС РФ от 18 июня 2003 г.

13. СанПиН 2.2.2_2.4.1340-03 гигиенические требования к ПЭВМ и организации работы / Министерство здравоохранения РФ, 2003.

Приложение А

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

Листинг программы

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

Класс *iTact.java*

```
private final Visio visioForm;
    private final Listing listingForm;
    public static Protocol protocolForm;
    private MicroCommandText mct;
    private String pathFileTest;
    private final JRadioButton[] jButtonFlgArray;
    private final JComponent[] jComponentEnable;
    private final JFormattedTextField[] JFTFBreakPointArray;
    boolean tabVisio = false;
    private int sizeArrayStart, sizeReadArray;
    private int flgTemp = 0;
    private final gr1804 gr;
    //=====
    public iTact() throws InterruptedException {

        pathFileTest = "test-14.pzu";

        initComponents();

        jMenuBar1.setPreferredSize(jMenuBar1.getMinimumSize());
        jMenuBar1.setSize(jMenuBar1.getMinimumSize());
        listEdit.setJTextField(jTextFieldtest1);
        InitTextField();
        gr = new gr1804();
        //=====
        jButtonFlgArray = new JRadioButton[]{
            jRadioButtonFlg0,
            jRadioButtonFlg1,
            jRadioButtonFlg2,
            jRadioButtonFlg3,
            jRadioButtonFlg4,
            jRadioButtonFlg5,
            jRadioButtonFlg6,
            jRadioButtonFlg7};
```

```

jComponentEnable = new JComponent[]{
    jButtonAuto,
    jButtonStep,
    TextBP1,
    TextBP3,
    TextBP4,
    TextBP2
};

EnableComp(false);
jRBOOnFlg.setVisible(false);
jButton15.setVisible(false);

//=====
JFTFBreakPointArray = new JFormattedTextField[]{
    TextBP1,
    TextBP2,
    TextBP3,
    TextBP4
};

for (JFormattedTextField JFTFBreakPointArray1 :
JFTFBreakPointArray) {
    JFTFBreakPointArray1.setInputVerifier(new
InputVerifierOct());
}
decor = new Decoration(buttonGroupStyle);
decor.loadProperties();
setDecor();

memForm = new Memory(memTable, addrMem);    // global
listingForm = new Listing(jLabelListing, listingArea,
jCheckBoxVisibleEmptyMK);

protocolForm = new Protocol(protocolArea);

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		99

```

        visioForm = new Visio(jPanelVisioView,
jTabbedPaneVisioList,
            jTextFieldVisioCountBlock,
jTextFieldVisioZoomCurrent,
            jTextFieldVisioCurrentNewBlock, memForm,
            jPopupMenuBlock,
            jPopupMenuTabbed
        );
        printf();
        EnableComp(true);
setExtendedState(MAXIMIZED_BOTH);
    }
private void EnableComp(boolean flag) {
    for (JComponent jComponentEnable1 : jComponentEnable) {
        jComponentEnable1.setEnabled(flag);
    }

    for (JRadioButton jButtonFlgArray1 : jButtonFlgArray) {
        jButtonFlgArray1.setEnabled(flag);
    }
}
private void printf() {

    if (jRBOOnFlg.isSelected()) {
        model.get(34).tvalue = flgTemp; //?
    }

    for (int id = 0; id < 45; id++) {
        model.get(id).setNewValue();
    }

    mct.PrintCurr(gr.adr3);
    mct.PrintNext(adrl);

    // Установка FLG jRadioBut

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		100

```

        int a = 0x01, b = octToDecInt(TextFLG.getText());
        for (JRadioButton jButtonFlgArray1 : jButtonFlgArray) {
            // Наложение маски 0x01, 0x02, 0x04, 0x08, 0x10,
0x20, 0x40, 0x80
            if ((b & a) == 0) {
                jButtonFlgArray1.setSelected(false);
            } else {
                jButtonFlgArray1.setSelected(true);
            }
            a = a << 1;        // Сдвиг маски на 1 бит
        }
        AddrTactLB.setText(Integer.toString(gr.NumTakt));
    }

private void printSetProg() {
    AddrStartLB.setText(Integer.toString(adrl, 8));
    AddrEndLB.setText(Integer.toString(aend, 8));
    AddrKoltakcLB.setText(Integer.toString(KolTakt, 10));
}

private void refresh() {
    for (int id = 0; id < 45; id++) {
        model.get(id).tvalue =
Integer.parseInt(model.get(id).p.getText(), 8);
    }
    mct.RefreshMc(adrl);

    if (jRBOOnFlg.isSelected()) {
        flgTemp = model.get(34).tvalue;
        model.get(34).tvalue = flgTemp | 2;
    }
}

private void modeling() {
    refresh();
    GenBreakPoint();

    gr.Extract(adrl);

```

```

        int result = gr.ContSim();

        printf();

        if (result == 0) {
            OkDialog okDialog = new OkDialog(this, true,
"Окончание моделирования");
        } else if (result != -1) {
            OkDialog okDialog = new OkDialog(this, true, "Точка
останова: " + String.valueOf(result));
        }
    }

    private void
jButtonStepActionPerformed(java.awt.event.ActionEvent evt) {
        step = true;
        modeling();
    }

    private void FileOpenActionPerformed(java.awt.event.ActionEvent
evt) {

        JFileChooser fileopen = new JFileChooser(new
File(System.getProperty("user.dir")));

        if (fileopen.showDialog(null, "Открыть файл") ==
JFileChooser.APPROVE_OPTION) {
            pathFileTest = fileopen.getSelectedFile().getPath();
            Crt_Opn(true);
            printf();
            EnableComp(true);
            visioForm.generate();
        }
    }

    private void FileSaveActionPerformed(java.awt.event.ActionEvent
evt) {

```

```

        savePzu();
    }
    private void savePzu() {

        JFileChooser filesave = new JFileChooser(new
File(System.getProperty("user.dir")));

        if (filesave.showDialog(null, "Сохранить") ==
JFileChooser.APPROVE_OPTION) {
            pathFileTest = filesave.getSelectedFile().getPath();
            try {
                FileOutputStream fos = new
FileOutputStream(pathFileTest);
                for (short idxR = 0; idxR <= 4095; idxR++) {
                    if (pzu[0][idxR] == 0 && pzu[1][idxR] == 0
&& pzu[2][idxR] == 0 && pzu[3][idxR] == 0) {
                        fos.write(toBytes((short) 0));
                    } else {
                        fos.write(toBytes(idxR));
                    }
                    for (short idxC = 0; idxC <= 3; idxC++) {
                        fos.write(toBytes(pzu[idxC][idxR]));
                    }
                }
                fos.close();
            } catch (IOException ex) {

                Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }

        public static byte[] toBytes(short s) {
            return new byte[]{(byte) (s & 0x00FF), (byte) ((s &
0xFF00) >> 8)};
        }
    }

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		103

```

    }

    // Кнопка автомат
    private void
    jButtonAutoActionPerformed(java.awt.event.ActionEvent evt) {
        step = false;
        modeling();
    }

    //test-6 - скрытая - только для отладки
    private void
    jButton15ActionPerformed(java.awt.event.ActionEvent evt) {
        Crt_Opn(true);
        model.get(1).tvalue = octToDecInt(1300);
        model.get(2).tvalue = octToDecInt(1400);
        printf();
    }

    private void GenBreakPoint() {

        int n;
        String str = "";
        for (int i = 0; i < JFTFBreakPointArray.length; i++) {
            n = octToDecInt(JFTFBreakPointArray[i].getText());
            if (n < gr.SizeROM) {
                gr.aost[i + 1] = (short) n;
            } else {
                str += String.valueOf(i) + " ";
                JFTFBreakPointArray[i].setText("0");
            }
        }

        if (!str.equals("")) {
            OkDialog okDialog = new OkDialog(this, true, "Выход
за пределы ПЗУ:\nТочки останова: " + str); // можно
исправить (необязательно)
        }
    }

```

```

        gr.naost = 4;
    }
    void ReadFile() {

        try {
            FileInputStream in = new
FileInputStream(pathFileTest);

            byte[] array1 = new byte[40960]; //
4096*10байт(количество команд на размер команды)

            try {
                sizeReadArray = in.read(array1);
                sizeArrayStart = array1.length;
                idPzuArray.clear();
                for (int a = 0, j = 0, n; a < sizeReadArray && a
< sizeArrayStart; a += 10, j++) {

                    n = (short) (((int) array1[a + 0] & 0xff) +
(((int) array1[a + 1]) << 8));

                    pzu[0][n] = (short) (((int) array1[a + 2] &
0xff) + ((int) array1[a + 3] << 8));
                    pzu[1][n] = (short) ((int) (array1[a + 4] &
0xff) + ((int) array1[a + 5] << 8));
                    pzu[2][n] = (short) (((int) array1[a + 6] &
0xff) + ((int) array1[a + 7] << 8));
                    pzu[3][n] = (short) (((int) array1[a + 8] &
0xff) + ((int) array1[a + 9] << 8));

                    System.out.println("\n 0 - " + pzu[0][n] +
"\n 1 - " + pzu[1][n] + "\n 2 - " + pzu[2][n] + "\n 3 - " +
pzu[3][n]);

                    m[15] = (short) ((pzu[3][n] & 65280) / 256);
                    m[14] = (short) ((pzu[3][n] & 192) / 64);

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		105

```

        m[12] = (short) ((pzu[3][n] & 56) / 8);
        m[11] = (short) (pzu[3][n] & 7);
        m[13] = (short) ((pzu[2][n] & 49152) /
16384);

        m[10] = (short) ((pzu[2][n] & 14336) /
2048);

        m[9] = (short) ((pzu[2][n] & 1792) / 256);
        m[8] = (short) ((pzu[2][n] & 192) / 64);
        m[7] = (short) ((pzu[2][n] & 48) / 16);
        m[4] = (short) (pzu[2][n] & 15);
        m[6] = (short) ((pzu[1][n] & 63488) / 2048);
        m[5] = (short) ((pzu[1][n] & 1986) / 64);
        m[3] = (short) ((pzu[1][n] & 60) / 4);
        m[2] = (short) (pzu[1][n] & 3);
        m[1] = pzu[0][n];
        m[0] = (short) n;

        if (!(n == 0 && a != 0)) {
            idPzuArray.add(new IdPzu((short) n,
(short) a));

            for (int b = 0; b < 16; b++) {
                System.out.print(" " + m[b]);
            }
            System.out.println("");
        }

        int test = 1;

    } catch (IOException ex) {
        Logger.getLogger(iTact.class
            .getName()).log(Level.SEVERE, null, ex);
    }

```

```

        } catch (FileNotFoundException ex) {
            Logger.getLogger(iTact.class
                .getName()).log(Level.SEVERE, null, ex);
        }
        protocolForm.init();
    }

    void Crt_Opn(Boolean b) {
        gr.ResPar();
        if (b) {
            ReadFile();
        }
        adr1 = 1;
        gr.adr3 = 0;
        gr.Extract(adr1);
        printf();
        printSetProg();
    }

    public static void main(String args[]) {

        java.awt.EventQueue.invokeLater(() -> {
            setDefaultCloseOperation(false);
            try {
                new iTact().setVisible(true);
            } catch (InterruptedException ex) {

                Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
                    ex);
            }
        });
    }

    private void setDecor() {
        decor.setDecor();
        if (decor.current == 0 || decor.current == 1) {
            super.setSize(900, 625);
        } else {

```

```

        super.setSize(950, 675);
    }
    bcColor1 = model.get(0).p.getForeground();          // цвет
динамический
    bcColor2 = Color.RED;
    SwingUtilities.updateComponentTreeUI(getRootPane());
    if (bcColor1.getBlue() > 128 && bcColor1.getGreen() >
128 && bcColor1.getRed() > 128) {
        bcColor3 = new Color(0, 230, 30);
    } else {
        bcColor3 = new Color(0, 150, 20);
    }
    printf();
}

private void
settingButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Кнопка "Установить" в настройках перед запуском
    aend = (short)
octToDecInt(Integer.parseInt(endSetting.getText()));
    adr1 = (short)
octToDecInt(Integer.parseInt(startSetting.getText()));
    KolTakt = (short)
Integer.parseInt(countSetting.getText());

    if (vrtButton1.isSelected()) {
        nvr = 1;
    } else {
        nvr = 4;
    }
    protocolForm.init();
    printSetProg();
    Crt_Opn(false);
    startSettingForm.setVisible(false);
}

```

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
    JFileChooser fileopen = new JFileChooser(new
File(System.getProperty("user.dir")));
    FileNameExtensionFilter filter = new
FileNameExtensionFilter(
        "Tact", "tact");
    fileopen.setFileFilter(filter);
    fileopen.setFileSelectionMode(JFileChooser.FILES_ONLY);
    if (fileopen.showDialog(null, "Сохранить файл") ==
JFileChooser.APPROVE_OPTION) {
        String path = fileopen.getSelectedFile().getPath();
        try {
            visioForm.save(path);
            OkDialog okDialog = new OkDialog(this, true,
"Сохранено!");
        } catch (IOException ex) {

Logger.getLogger(iTact.class.getName()).log(Level.SEVERE,
"Ошибка! Сохранения \n Local: {0}\nGlobal: {1}", new
String[]{ex.getLocalizedMessage(), ex.getMessage()});
            OkDialog okDialog = new OkDialog(this, true,
"Ошибка при сохранения данных!\n Local: " +
ex.getLocalizedMessage() + "\nGlobal: " + ex.getMessage());
        }
    }

    private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileopen = new JFileChooser(new
File(System.getProperty("user.dir")));
        FileNameExtensionFilter filter = new
FileNameExtensionFilter(
            "Tact", "tact");

```

```

        fileopen.setFileFilter(filter);
        fileopen.setFileSelectionMode(JFileChooser.FILES_ONLY);
        if (fileopen.showDialog(null, "Открыть файл") ==
JFileChooser.APPROVE_OPTION) {
            String path = fileopen.getSelectedFile().getPath();
            try {
                visioForm.load(path);
                OkDialog okDialog = new OkDialog(this, true,
"Загружено!");
            } catch (IOException ex) {

Logger.getLogger(iTact.class.getName()).log(Level.SEVERE,
"Ошибка! Сохранения \n Local: {0}\nGlobal: {1}", new
String[]{ex.getLocalizedMessage(), ex.getMessage()});
                OkDialog okDialog = new OkDialog(this, true,
"Ошибка при чтении данных!\n Local: " + ex.getLocalizedMessage()
+ "\nGlobal: " + ex.getMessage());
            }
        }
    }
}

```

Класс *Tact.java*

```

import static t3.GlobalVars.*;
public class Tact {
    private short lastCt = 0;
    private byte p30, g30, p0, p1, p2, p3, g0, g1, g2, g3;
    private byte c4, c8, c12;
    private final byte[] psw = new byte[8];
    private final byte[] flg = new byte[8];
    private final byte[] pr = new byte[4];
    private final byte[] ps = new byte[4];

    public short SizeROM, MaskAdr, MaskBy1;

    private byte pe_, me_, cte_, ctl_, pup, fe_, sl0, za_, s0,
sl, ctr;

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		110

```

private int vrnt, ormask, pna, pa, buff, stb, bva;

private final int[] r = new int[16];

private final byte[] by3 = new byte[]{
    (byte) 0xf2, (byte) 0xf2, (byte) 0x3e, (byte) 0xde,
(byte) 0xfd, (byte) 0xfd, (byte) 0x3e,
    (byte) 0xfe, (byte) 0x1e, (byte) 0x16, (byte) 0x5e,
(byte) 0xde, (byte) 0x3f, (byte) 0xff,
    (byte) 0x7e, (byte) 0xfe, (byte) 0xaa, (byte) 0x0e,
(byte) 0xfa, (byte) 0x3e, (byte) 0x2e,
    (byte) 0x8e, (byte) 0x2e, (byte) 0xce, (byte) 0x36,
(byte) 0x36, (byte) 0xae, (byte) 0x0e,
    (byte) 0x3e, (byte) 0x3e, (byte) 0x3e, (byte) 0xfe};

private byte lev1, lev2_1, lev2_2, lev3, lev4, lev5;
private byte adr_by3, d_by3;

private short a, b;

Tact() {
}

public void tact() {
    for (int id = 0; id < 16; id++) {
        r[id] = model.get(id).tvalue;    // get(id + 1) было
    }
    rq = model.get(16).tvalue;
    rk = model.get(17).tvalue;
    rdi = model.get(18).tvalue;
    rdo = model.get(19).tvalue;
    mar = model.get(20).tvalue;
    bi = model.get(21).tvalue;
    zi = model.get(22).tvalue;
    d = model.get(23).tvalue;
}

```

```

y = model.get(24).tvalue;
ct = model.get(25).tvalue;
dby = model.get(26).tvalue;
cmk = model.get(27).tvalue;
ra = model.get(28).tvalue;
st0 = model.get(29).tvalue;
st1 = model.get(30).tvalue;
st2 = model.get(31).tvalue;
st3 = model.get(32).tvalue;
pswd = model.get(33).tvalue;
flgd = model.get(34).tvalue;
c0 = (byte) model.get(35).tvalue;
c16 = (byte) model.get(36).tvalue;
ovr = (byte) model.get(37).tvalue;
z = (byte) model.get(38).tvalue;
f15 = (byte) model.get(39).tvalue;
tst = (byte) model.get(40).tvalue;
pr15 = (byte) model.get(41).tvalue;
pr0 = (byte) model.get(42).tvalue;
pq15 = (byte) model.get(43).tvalue;
pq0 = (byte) model.get(44).tvalue
inppsw(psw, pswd);
    flgd = inppsw(flq, flgd);
    bmu();
    bod();
    pswd = unppsw(psw);
    flgd = unppsw(flq);
for (int id = 0; id < 16; id++) {
    model.get(id).tvalue = r[id];           // get(id +
1) было
    }

    model.get(16).tvalue = rq;
    model.get(17).tvalue = rk;
    model.get(18).tvalue = (short) rdi;

```

```

        model.get(19).tvalue = (short) rdo;
        model.get(20).tvalue = (short) mar;
        model.get(21).tvalue = (short) bi;
        model.get(22).tvalue = zi;
        model.get(23).tvalue = d;
        model.get(24).tvalue = (short) y;
        model.get(25).tvalue = ct;
        model.get(26).tvalue = dbby;
        model.get(27).tvalue = cmk;
        model.get(28).tvalue = ra;
        model.get(29).tvalue = st0;
        model.get(30).tvalue = st1;
        model.get(31).tvalue = st2;
        model.get(32).tvalue = st3;
        model.get(33).tvalue = pswd;
        model.get(34).tvalue = flgd;
        model.get(35).tvalue = c0;
        model.get(36).tvalue = c16;
        model.get(37).tvalue = ovr;
        model.get(38).tvalue = z;
        model.get(39).tvalue = f15;
        model.get(40).tvalue = tst;
        model.get(41).tvalue = pr15;
        model.get(42).tvalue = pr0;
        model.get(43).tvalue = pq15;
        model.get(44).tvalue = pq0;
    }

    private byte rkd(byte hb, byte lb) {
        return (byte) ((rk >> lb) & (0xffff >> (15 - hb + lb)));
    }

    private int inppsw(byte[] rz, int dd) {
        for (int id = 0; id <= 7; id++) {
            rz[id] = (byte) (dd % 2); // rz[i]
            dd = (short) (dd >> 1);
        }
    }

```

```

        return dd;
    }
private short unppsw(byte[] wd) {
    short rz = 0;
    for (int id = 7; id >= 0; id--) {
        rz = (short) (rz << 1);
        rz = (short) (rz + (wd[id] & 1)); //wd[i]
    }
    return rz;
}
private void levset() {
    lev1 = 0;
    lev2_1 = 0;
    lev2_2 = 0;
    lev3 = 0;
    lev4 = 0;
    lev5 = 0;
    if ((rk & 0x7000) != 0) {
        lev1 = 1;
    } else if ((rk & 0x0800) != 0) {
        if ((rk & 0x0600) == 0) {
            lev2_1 = 1;
        } else {
            lev2_2 = 1;
        }
    } else if ((rk & 0x0700) != 0) {
        lev3 = 1;
    } else if ((rk & 0x00C0) != 0) {
        lev4 = 1;
    } else {
        lev5 = 1;
    }
}
private void bmu() {
    if (ct == 0) {

```

```

        ctr = 1;
    } else {
        ctr = 0;
    }
    vrnt = (short) ((nvr >> 2) & 0x01);
    switch (m[3]) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            tst = psw[m[3]]; //psw[m[3]]
            break;
        case 6:
            tst = ctr;
            break;
        case 7:
            tst = 0;
            break;
        case 8:
        case 9:
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
            tst = flg[m[3] - 8];
            break;
    }
    if (vrnt == 1) {
        if ((m[4] == 2) || (m[4] == 6)) {
            ormask = m[1];
        } else {

```

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		115

```

ormask = 0;
}
if ((m[3] >= 8) && (m[2] >= 2)) {
    levset();
    switch (m[3]) {
        case 8:
            tst = lev1;
            break;
        case 9:
            tst = lev2_1;
            break;
        case 10:
            tst = lev2_2;
            break;
        case 11:
            tst = lev3;
            break;
        case 12:
            tst = lev4;
            break;
        case 13:
            tst = lev5;
            break;
        case 14:
            if (rkd((byte) 11, (byte) 9) == 0) {
                tst = 1;
            } else {
                tst = 0;
            }
            break;
        case 15:
            if (rkd((byte) 5, (byte) 3) == 0) {
                tst = 1;
            } else {
                tst = 0;
            }
        }
    }
}

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		116

```

        }
        break;
    }
}
} else if (m[2] >= 2) {
    ormask = m[1];
} else {
    ormask = 0;
}
if ((m[2] & 0x01) == 1) {
    tst = (byte) ~tst;
}
tst = (byte) (tst & 0x01);
adr_by3 = (byte) ((m[4] << 1) | tst) & 0x1f);

d_by3 = by3[adr_by3];
buff = d_by3 & 0xff;
pe_ = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
me_ = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
cte_ = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
ctl_ = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
pup = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
fe_ = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
s10 = (byte) (buff & 0x03);
s0 = (byte) (buff & 0x01);
buff = (short) (buff >> 1);
s1 = (byte) (buff & 0x01);
buff = (short) (buff >> 1);

```

```

        za_ = (byte) (ctl_ | cte_);
        pna = (short) ((m[1] & 0xFF00) + rkd((byte) 15, (byte)
8));

        pa = (short) ((m[1] & 0xFF00) + rkd((byte) 7, (byte)
0));

        if (pe_ == 0) {
            dby = m[1];
        } else if (me_ == 0) {
            dby = pna;
        } else {
            dby = pa;
        }
        dby = (short) (dby & MaskBy1);
        switch (s10) {
            case 0:
                bva = cmk;
                System.out.print("cmk = ");
                break;
            case 1:
                bva = ra;
                System.out.print("ra = ");
                break;
            case 2:
                bva = st0;
                System.out.print("st0 = ");
                break;
            case 3:
                bva = dby;
                System.out.print("dby = ");
                break;
        }
        System.out.print(String.valueOf(decToOct(bva)) + "\n");

        bva = (short) (bva | ormask & MaskBy1);

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		118

```

        if (za_ == 0) {
            bva = 0;
        }
        adr1 = (short) (bva & MaskAdr);
        if (fe_ == 0) {
if (pup == 1) {
            st3 = st2;
            st2 = st1;
            st1 = st0;
            st0 = cmk;
        } else {
            stb = st0;
            st0 = st1;
            st1 = st2;
            st2 = st3;
            st3 = stb;
        }
    }
    cmk = (short) ((bva + 1) & MaskBy1);
    ra = (short) (dby & MaskBy1);
    if (cte_ == 0) {
        ct = (short) (ct - 1);
    }
    if (ctl_ == 0) {
        ct = m[1];
    }
}
private void zf15() {
    z = 0;
    if ((short) f == 0) {
        z = 1;
    }
    f15 = 0;
    if (((short) f & 0x8000) != 0) {
        f15 = 1;
    }
}

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		119

```

    }

}

private void msk(int xx, byte[] x, byte dvr)
{
    xx = (short) (xx >> dvr);
    for (int id = 0; id <= 3; id++) {
        x[id] = (byte) (xx % 2);
        xx = (short) (xx >> 1);
    }
}

private void prvr(byte dvr) {
    msk(rr, pr, dvr);
    msk(ss, ps, dvr);
    p0 = (byte) ((pr[0] | ps[0]) & 1);
    p1 = (byte) ((pr[1] | ps[1]) & 1);
    p2 = (byte) ((pr[2] | ps[2]) & 1);
    p3 = (byte) ((pr[3] | ps[3]) & 1);
    g0 = (byte) (pr[0] & ps[0] & 1);
    g1 = (byte) (pr[1] & ps[1] & 1);
    g2 = (byte) (pr[2] & ps[2] & 1);
    g3 = (byte) (pr[3] & ps[3] & 1);
    p30 = (byte) (~(p3 & p2 & p1 & p0) & 0x01);
    g30 = (byte) (g3 & g2 & g1 & g0 & 0x01);
}

private void add_rs() {
    int rrl, ssl, as;
    rrl = rr & 0xffff;
    ssl = ss & 0xffff;
    as = rrl + ssl + c0;
    f = (rrl + ssl + (c0 & 0xff));
    c16 = 0;
    if ((as & 0x10000) != 0) {
        c16 = 1;
    }
}

```

```

        ovr = 0;
        if ((rrl < 0x8000) && (ssl < 0x8000)) {
            if (f >= 0x8000) {
                ovr = 1;
            }
        }
        if ((rrl >= 0x8000) && (ssl >= 0x8000)) {
            if (f < 0x8000) {
                ovr = 1;
            }
        }
        zf15();
    }

private void sub_sr() {
    rr = (short) (rr ^ 0xFFFF);
    //rr=-rr; не наше
    add_rs();
}

private void or_rs() {
    byte c11 = c0, c2 = 0;
    f = (short) (rr | ss);

    for (int id = 0; id <= 3; id++) {
        prvr((byte) (id * 4));
        c2 = (byte) (p30 ^ c11);
        c11 = c2;
    }
    c16 = (byte) (c2 & 0x01);
    ovr = c16;
    zf15();
}

private void and_rs() {
    byte c11 = c0, c2 = 0;
    f = (short) (rr & ss);

```

```

        for (int id = 0; id <= 3; id++) {
            prvr((byte) (id * 4));
            c2 = (byte) (g30 ^ c11);
            c11 = c2;
        }
        c16 = (byte) (c2 & 0x01);
        ovr = c16;
        zfl5();
    }

private void and_nrs() {
    rr = (short) ~(rr);
    and_rs();
}

private void sub_rs() {
    ss = (short) (ss ^ 0xFFFF);
    add_rs();
}

private byte cr67(byte cr) {
    return (byte) ~(g3
        | (p3 & g2)
        | (p3 & p2 & g1)
        | (p3 & p2 & p1 & p0 & (g0 | ~cr)));
}

private byte ovr67(byte cr) {
    byte pr1, pr2;
    pr1 = (byte) (~p2
        | (~p2 & ~p2)
        | (~p2 & ~p2 & ~p2)
        | (~p2 & ~p2 & ~p2 & cr));
    pr2 = (byte) (~p3
        | (~g3 & ~p2)
        | (~g3 & ~p2 & ~p2)
        | (~g3 & ~p2 & ~p2 & ~p2)
        | (~g3 & ~p2 & ~p2 & ~p2 & cr));
    return (byte) (pr1 ^ pr2);
}

```

```

    }

private void bod() {
    if ((m[5] >= 0) && (m[5] <= 15)) {
        a = m[5];
    } else if ((m[5] >= 16) && (m[5] <= 23)) {
        a = rkd((byte) 2, (byte) 0);
    } else if ((m[5] >= 24) && (m[5] <= 31)) {
        a = rkd((byte) 8, (byte) 6);
    } else {
        System.out.print("M[5] out of range");
    }

    if ((m[6] >= 0) && (m[6] <= 15)) {
        b = m[6];
    } else if ((m[6] >= 16) && (m[6] <= 23)) {
        b = rkd((byte) 2, (byte) 0);
    } else if ((m[6] >= 24) && (m[6] <= 31)) {
        b = rkd((byte) 8, (byte) 6);
    } else {
        System.out.print("M[6] out of range");
    }

    if ((m[8] >= 0) && (m[8] <= 3)) {
        switch (m[8]) {
            case 0:
                c0 = 0;
                break;
            case 1:
                c0 = 1;
                break;
            case 2:
                c0 = psw[0];
                break;
            case 3:
                c0 = (byte) (~(psw[0]) & 0x01);
                break;
        }
    }
}

```

```

        } else {
            System.out.print("M[8] out of range");
        }
    if (m[12] <= 7) {
        switch (m[12]) {
            case 0:
            case 4:
            case 7:
                d = (short) rdi;
                break;
            case 1:
            case 5:
                d = zi;
                break;
            case 2:
            case 6:
                d = m[1];
                break;
            case 3:
                d = pswd;
                break;
        }
        if (m[12] >= 4) {
            rdi = bi;
        }
        if (m[12] == 7) {
            rk = bi;
        }
    } else {
        System.out.print("M[12] out of range");
    }

    if ((m[9] >= 0) && (m[9] <= 7)) {
        switch (m[9]) {
            case 0:
                rr = r[a];

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		124

```

        ss = rq;
        break;
    case 1:
        rr = r[a];
        ss = r[b];
        break;
    case 2:
        rr = 0;
        ss = rq;
        break;
    case 3:
        rr = 0;
        ss = r[b];
        break;
    case 4:
        rr = 0;
        ss = r[a];
        break;
    case 5:
        rr = d;
        ss = r[a];
        break;
    case 6:
        rr = d;
        ss = rq;
        break;
    case 7:
        rr = d;
        ss = 0;
        break;
    }
} else {
    System.out.print("M[9] out of range.");
}

if ((m[10] >= 0) && (m[10] <= 7)) {

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		125

```

switch (m[10]) {
    case 0:
        add_rs();
        break;
    case 1:
        sub_sr();
        break;
    case 2:
        sub_rs();
        break;
    case 3:
        or_rs();
        break;
    case 4:
        and_rs();
        break;
    case 5:
        and_nrs();
        break;
    case 6:
        xor_rs();
        break;
    case 7:
        xor_nrs();
        break;
}
} else {
    System.out.print("M[10] out of range.");
}

if ((m[7] <= 3) && (m[11] <= 7)) {
    y = f;
    if ((m[11] & 0x02) == 0) {
        if (m[11] <= 3) {
            pr0 = 1;
            pq0 = 1;

```

```

    } else {
        pr0 = (byte) (f % 2);
        pq0 = (byte) (rq % 2);
    }
    switch (m[7]) {
        case 0:
            pr15 = 0;
            pq15 = 0;
            break;
        case 1:
            pr15 = psw[0];
            pq15 = (byte) (pq0 & 1);
            break;
        case 2:
            pr15 = psw[0];
            pq15 = (byte) (pr0 & 1);
            break;
        case 3:
            pr15 = (byte) ((f15 ^ ovr) & 1);
            pq15 = (byte) (pr0 & 1);
            break;
    }
} else {
    if (m[11] <= 3) {
        pr15 = 1;
        pq15 = 1;
    } else {
        pr15 = (byte) (f >> 15);
        pq15 = (byte) (rq >> 15);
    }
    switch (m[7]) {
        case 0:
            pr0 = 0;
            pq0 = 0;
            break;

```

```

        case 1:
            pr0 = psw[0];
            pq0 = (byte) (pq15 & 1);
            break;
        case 2:
            pr0 = (byte) (pq15 & 1);
            pq0 = 1;
            break;
        case 3:
            pr0 = (byte) (pq15 & 1);
            pq0 = 0;
            break;
    }
}
if ((m[11] & 0x02) == 0) {
    switch (m[7]) {
        case 0:
        case 1:
            c1 = pr0;
            break;
        case 2:
        case 3:
            c1 = pq0;
            break;
    }
} else {
    c1 = pr15;
}
switch (m[11]) {
    case 0:
        rq = (short) f;
        break;
    case 1:
        break;
    case 2:

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		128

```

        y = (int) r[a] & 0xffff;
        r[b] = (short) f;
        break;
    case 3:
        r[b] = (short) f;
        break;
    case 4:
        r[b] = (short) f;
        r[b] = (short) sr(((int) r[b] & 0xffff),
pr15);

        rq = (short) sr(((int) rq & 0xffff), pq15);
        break;
    case 5:
        r[b] = (short) f;
        r[b] = (short) sr(((int) r[b] & 0xffff),
pr15);

        break;
    case 6:
        r[b] = (short) f;
        r[b] = (short) sl(((int) r[b] & 0xffff),
pr0);

        rq = (short) sl(((int) rq & 0xffff), pq0);
        break;
    case 7:
        r[b] = (short) f;
        r[b] = (short) sl(((int) r[b] & 0xffff),
pr0);

        break;
    }
} else {
    System.out.print("M[7] or M[11] out of range.");
}
if (m[13] <= 3) {
    switch (m[13]) {
        case 0:

```

```

        break;
    case 1:
        psw[0] = c16;
        psw[1] = ovr;
        psw[2] = z;
        psw[3] = f15;
        break;
    case 2:
        psw[0] = (byte) (c1 & 1);
        psw[1] = ovr;
        psw[2] = z;
        psw[3] = (byte) (f15 ^ ovr);
        break;
    case 3:
        inppsw(psw, (short) y);
        break;
    }
} else {
    System.out.print("M[13] out of range.");
}
if (m[14] <= 3) {
    switch (m[14]) {
        case 0:
        case 3:
            break;
        case 1:
            rdo = y;
            break;
        case 2:
            mar = y;
            break;
    }
} else {
    System.out.print("M[14] out of range.");
}

```

```

        if (m[15] == 0 && (flg[1] == 1)) {
            flg[1] = 0;
            bi = 0;
            lastCt = 0;
        } else if ((m[15] == 1) || (m[15] == 2)) && flg[1] ==
1) {

            if (lastCt == 1 || m[15] == 2) {
                switch (m[15]) {
                    case 1:
                        bi = memForm.getPoMar(mar);
                        break;
                    case 2:
                        memForm.setPoMar(mar, rdo);
                        break;
                }
                lastCt = 0;
            } else {
                lastCt = 1;
            }
        }
    }
}

```

Класс *gr1804.java*

```

public class gr1804 {

    private boolean st_type = false;

    private char c = 0;
    private short ast = 0;

    public short[] aost = new short[21];
    public short naost = 0;

    private short adr2 = 0, adrbuf = 0;
    public int adr3 = 0;
}

```

```

public short begpzu = 0, endpzu = 0;

public short NumTakt = 0;

private boolean Modeling = false;

public short SizeROM;

private final Tact t;

gr1804() {
    t = new Tact();

    c = 'T';
    step = true;
    KolTakt = 256;
    naost = 0;
    ast = 0;
    aend = 0;
    t.MaskBy1 = 0x00FF;
    t.MaskAdr = 0x00FF;
    adr1 = 0;
    adr2 = 0;

    for (int id = 0; id < 45; id++) {
        model.get(id).inp = false;
        model.get(id).outp = false;
        model.get(id).tvalue = (int) 0;
        model.get(id).stvalue = 0;
    }

    for (int id = 0; id < 23; id++) {
        model.get(id).inp = true;
    }

    model.get(25).inp = true;

```

```

for (int id = 27; id < 35; id++) {
    model.get(id).inp = true;
}

begpzu = 0;
adr1 = begpzu;
endpzu = 1;
SizeROM = 4096;
if (SizeROM > 256) {
    t.MaskBy1 = 0xFFF;
} else {
    t.MaskBy1 = 0xFF;
}
t.MaskAdr = (short) (SizeROM - 1);
for (int id = 0; id <= 4095; id++) {
    for (int j = 0; j <= 3; j++) {
        pzu[j][id] = 0;
    }
}

}

public void Extract(short n) {
    m[15] = (short) ((pzu[3][n] & 65280) / 256);
    m[14] = (short) ((pzu[3][n] & 192) / 64);
    m[12] = (short) ((pzu[3][n] & 56) / 8);
    m[11] = (short) (pzu[3][n] & 7);
    m[13] = (short) ((pzu[2][n] & 49152) / 16384);
    m[10] = (short) ((pzu[2][n] & 14336) / 2048);
    m[9] = (short) ((pzu[2][n] & 1792) / 256);
    m[8] = (short) ((pzu[2][n] & 192) / 64);
    m[7] = (short) ((pzu[2][n] & 48) / 16);
    m[4] = (short) (pzu[2][n] & 15);
    m[6] = (short) ((pzu[1][n] & 63488) / 2048);
    m[5] = (short) ((pzu[1][n] & 1986) / 64);
}

```

```

        m[3] = (short) ((pzu[1][n] & 60) / 4);
        m[2] = (short) (pzu[1][n] & 3);
        m[1] = pzu[0][n];
        m[0] = n;
    }

    public int ContSim() {
        int bOst = -1;
        short j;
        if ((KolTakt <= NumTakt) || ((adr2 == aend) && (NumTakt
!= 0))) {
            Modeling = true;
            return bOst;
        }
        Modeling = false;
        do {
            adr2 = adr1;

            Extract(adr1);

            stop = false;
            if (adr1 == aend) {
                stop = true;
                st_type = false;
            } else {
                for (j = 1; j <= naost; j++) {
                    if (aost[j] == adr1) {
                        stop = true;
                        st_type = true;
                        bOst = j;
                    }
                }
            }

            t.tact();
            NumTakt++;

```

```

        adrbuf = adr1;
        adr1 = adr2;
        adr3 = adr2;

        adr1 = adrbuf;
            bOst = 0;
        }
    }

    protocolForm.print();
} while (!stop && !step && (NumTakt < KolTakt));
return bOst;
}

public void ResPar() {
    System.out.print("Вывод результатов моделирования
:TECT");

    adr1 = ast;

    NumTakt = 0;
    for (int id = 0; id < 45; id++) {
        model.get(id).tvalue = model.get(id).stvalue;
    }
    model.get(27).tvalue = (short) (ast + 1);

    model.get(27).tvalue = (short) 2;

    System.out.print("");
}
}

```

Класс *Protocol.java*

```
public class Protocol {
```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		135

```

private final JTextArea protocolArea;
private int protocolCount = 1;
private String NameColMK;

Protocol(JTextArea protocolArea1) {
    this.protocolArea = protocolArea1;

    String[] mas = new String[]{
        "Addr", "M1", "M2", "M3", "M4", "M5", "M6", "M7",
        "M8", "M9", "M10", "M11", "M12", "M13", "M14", "M15"
    };
    NameColMK = "";

    for (int k = 0; k < 16; k++) {
        switch (k) {
            case 0:
                NameColMK += String.format(" %4s", mas[k]);
                break;
            case 1:
                NameColMK += String.format("%7s", mas[k]);
                break;
            case 10:
                NameColMK += String.format("%5s", mas[k]);
                break;
            default:
                NameColMK += String.format("%4s", mas[k]);
                break;
        }
    }
}

public void init() {

    protocolArea.selectAll();
    protocolArea.replaceSelection("");
}

```

```

        protocolArea.append("\nВывод результатов моделирования
:" + "\n");
        protocolArea.append("Адрес старта                                : "
+ adr1 + "\n");
        protocolArea.append("Адрес окончания моделирования          : "
+ aend + "\n");
        protocolArea.append("Количество тактов моделирования : "
+ KolTakt + "\n");
        protocolArea.append("Пошаговое моделирование\n");

        for (int id = 0, i = 5; id < model.size(); id++, i--) {
            protocolArea.append(String.format("%-4s = %-7o  ",
model.get(id).name, (int) model.get(id).tvalue & 0xffff));
            if (i == 0) {
                protocolArea.append("\n");
                i = 6;
            }
        }
        protocolArea.append("\n");

    }

    public void print() {

        protocolArea.append("\n" + NameColMK + "\n");

        String str = "";
        for (int k = 0; k < 16; k++) {
            switch (k) {
                case 0:
                    str += String.format("%5o", (int) m[k] &
0xffff);
                    break;
                case 1:

```

```

        str += String.format("%7o", (int) m[k] &
0xffff);

        break;
    default:
        str += String.format("%4o", m[k]);
        break;
    }
}

protocolArea.append(str + "\n");

protocolArea.append("Адрес следующей микрокоманды: " +
Integer.toString(adrl, 8) + "    ТАКТ: " + protocolCount +
"\n");

for (int id = 0, i = 5; id < model.size(); id++, i--) {
    protocolArea.append(String.format("%-4s = %-7o  ",
model.get(id).name, (int) model.get(id).tvalue & 0xffff));
    if (i == 0) {
        protocolArea.append("\n");
        i = 6;
    }
}

protocolArea.append("\n");
protocolCount++;
}

public void save() throws IOException {
    FileNameExtensionFilter filter = new
FileNameExtensionFilter("Текстовый файл (.txt)", "txt");

    JFileChooser saveListing = new JFileChooser(new
File(System.getProperty("user.dir")));
    saveListing.setFileFilter(filter);

```

```

        if (saveListing.showDialog(null, "Сохранить") ==
JFileChooser.APPROVE_OPTION) {

            String path =
saveListing.getSelectedFile().getPath();
            if (!path.contains(".")) {
                path = path + ".txt";
            }

            try (Writer writer = new BufferedWriter(new
OutputStreamWriter(
                new FileOutputStream(path), "utf-8"))) {
                protocolArea.write(writer);
            }
        }

        public void copy() {
            StringSelection stringSelect = new
StringSelection(protocolArea.getText());
            Clipboard clipboard =
Toolkit.getDefaultToolkit().getSystemClipboard();
            clipboard.setContents(stringSelect, null);
            protocolArea.selectAll();
        }
    }
}

```

Класс *MicroCommandText.java*

```

public class MicroCommandText {

    private final ArrayList<JFormattedTextField> currMC;
    private final ArrayList<JFormattedTextField> nextMC;

    //=====

    MicroCommandText(ArrayList currmc, ArrayList nextmc) {

```

```

currMC = currmc;
nextMC = nextmc;

}

//=====

private void setText(boolean flag, int index, int value) {
    if (flag) {

currMC.get(index).setText(Integer.toOctalString((int) (value &
0xFFFF)));
    } else {

nextMC.get(index).setText(Integer.toOctalString((int) (value &
0xFFFF)));
    }
}

private void Print(boolean a, int n) {
    setText(a, 15, ((pzu[3][n] & 65280) / 256));
    setText(a, 14, ((pzu[3][n] & 192) / 64));
    setText(a, 12, ((pzu[3][n] & 56) / 8));
    setText(a, 11, (pzu[3][n] & 7));
    setText(a, 13, ((pzu[2][n] & 49152) / 16384));
    setText(a, 10, ((pzu[2][n] & 14336) / 2048));
    setText(a, 9, ((pzu[2][n] & 1792) / 256));
    setText(a, 8, ((pzu[2][n] & 192) / 64));
    setText(a, 7, ((pzu[2][n] & 48) / 16));
    setText(a, 4, (pzu[2][n] & 15));
    setText(a, 6, ((pzu[1][n] & 63488) / 2048));
    setText(a, 5, ((pzu[1][n] & 1986) / 64));
    setText(a, 3, ((pzu[1][n] & 60) / 4));
    setText(a, 2, (pzu[1][n] & 3));
    setText(a, 1, pzu[0][n]);
    setText(a, 0, (short) n);
}

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		140

```

    }

    public void PrintNext(int n) {
        Print(false, n);
    }

    public void PrintCurr(int n) {
        Print(true, n);
    }

    //=====
    public void RefreshMc(int n) {
        int[] mas = new int[16];
        for (int i = 0; i < 16; i++) {
            mas[i] = (short)
octToDecInt(nextMC.get(i).getText());
        }
        pzu[0][n] = (short) mas[1];
        pzu[1][n] = (short) (mas[6] * 2048 + mas[5] * 64 +
mas[3] * 4 + mas[2]);
        pzu[2][n] = (short) (mas[13] * 16384 + mas[10] * 2048 +
mas[9] * 256 + mas[8] * 64 + mas[7] * 16 + mas[4]);
        pzu[3][n] = (short) (mas[15] * 256 + mas[14] * 64 +
mas[12] * 8 + mas[11]);
    }
}

```

Класс *Memory.java*

```

public class Memory {

    private javax.swing.JTextField addr;
    private javax.swing.JTable memTable;
    private final DefaultTableModel dtm;
    private Color clrFont, clrBack;

    public Memory(JTable memTable, JTextField addr) {

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		141

```

        this.memTable = memTable;
        this.addr = addr;
        clrFont = memTable.getForeground();
        clrBack = memTable.getBackground();

        dtm = (DefaultTableModel) memTable.getModel();

        CustomTableRenderer centerRend = new
Memory.CustomTableRenderer();
        centerRend.setHorizontalAlignment(JLabel.CENTER);

        memTable.setDefaultRenderer(Integer.class, centerRend);

memTable.getColumnModel().getColumn(0).setPreferredWidth(60);

        JTableHeader ht = memTable.getTableHeader();
        ht.setFont(new Font("Arial", 1, 14));

        ClearTable();
        setDefaultValue();

    }

    public void setDefaultValue() {
        Object[] v = new Object[9];
        for (int idx = 0; Integer.parseInt(Integer.toString(idx,
8)) <= 177760; idx += 16) {
            for (int j = 0, k = 0, h; j < 9; j++) {
                h = Integer.parseInt(Integer.toString(idx + (j -
k) * 2, 8));

                if (k == 1) {
                    switch (h) {
                        case 76:

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		142

```

        h = 36172;
        break;
    case 102:
        h = 10000;
        break;
    case 11300:
        h = 2;
        break;
    case 100:
        h = 10000;
        break;
    case 11400:
        h = 11500;
        break;
    case 11500:
        h = 5;
        break;
    default:
        h = 0;
        break;
    }
}
k = 1;
v[j] = h;
}
dtm.addRow(v);
}
}

```

```

public void memTableMouseReleased() {

    int a = memTable.getSelectedColumn();
    int b = memTable.getSelectedRow();
    int c = b * 16 + (a - 1) * 2;
}

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		143

```

        addr.setText(Integer.toString(c, 8));
    }

    public void addrActionPerformed() {

        int adr = Integer.parseInt(addr.getText(), 8);

        if (adr >= 0 && adr <=
Integer.parseInt(Integer.toString(adr, 8))) {
            if (adr % 2 != 0) {
                adr--;
            }
            int a = adr / 16;
            int b = (adr % 16) / 2 + 1;

memTable.getSelectionModel().setSelectionInterval(Integer.parseI
nt(Integer.toString(b, 8)), a);
        }
    }

    // Кнопка "Загрузить"
    public void loadMemory() {
        FileReader myfile = null;
        try {
            String pathFile;
            JFileChooser fileopen = new JFileChooser(new
File(System.getProperty("user.dir")));
            if (fileopen.showDialog(null, "Загрузить") ==
JFileChooser.APPROVE_OPTION) {
                pathFile = fileopen.getSelectedFile().getPath();

                dtm.setRowCount(0);

                File f = new File(pathFile);

```

```

        char[] open_array = new char[(int) f.length()];

        myfile = new FileReader(f);
        myfile.read(open_array);

        String[] row =
String.valueOf(open_array).split("\r\n");
        for (int idx = 0; idx < row.length; idx++) {
            String[] value = row[idx].split("\\s+");
            if (value.length < 9) {
                dtm.addRow(new Object[]{0, 0, 0, 0, 0,
0, 0, 0, 0});
            } else {
                dtm.addRow(new
Object[]{Integer.parseInt(value[0]), Integer.parseInt(value[1]),
Integer.parseInt(value[2]),
                Integer.parseInt(value[3]),
Integer.parseInt(value[4]), Integer.parseInt(value[5]),
Integer.parseInt(value[6]),
                Integer.parseInt(value[7]),
Integer.parseInt(value[8])});
            }
        }

        myfile.close();
    }
} catch (FileNotFoundException ex) {
    Logger.getLogger(Memory.class
        .getName()).log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(Memory.class
        .getName()).log(Level.SEVERE, null, ex);
} finally {
    if (myfile != null) {
        try {

```

```

        myfile.close();

    } catch (IOException ex) {
        Logger.getLogger(Memory.class
            .getName()).log(Level.SEVERE, null,
ex);
    }
}

// Кнопка "Сохранить"
public void saveMemory() {

    BufferedWriter bfw = null;
    try {
        String pathFile;
        JFileChooser fileopen = new JFileChooser(new
File(System.getProperty("user.dir")));
        if (fileopen.showDialog(null, "Сохранить") ==
JFileChooser.APPROVE_OPTION) {
            pathFile = fileopen.getSelectedFile().getPath();

            bfw = new BufferedWriter(new
FileWriter(pathFile));

            for (int i = 0; i < dtm.getRowCount(); i++) {

                String buffer = String.format("%-4d ", (int)
dtm.getValueAt(i, 0));

                for (int j = 1; j < dtm.getColumnCount();
j++) {

                    buffer += String.format("%-6d ", (int)
dtm.getValueAt(i, j));

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		146

```

        }
        bfw.write(buffer + "\r\n");
    }
    bfw.close();
}
} catch (IOException ex) {
    Logger.getLogger(Memory.class
        .getName()).log(Level.SEVERE, null, ex);
} finally {
    if (bfw != null) {
        try {
            bfw.close();

        } catch (IOException ex) {
            Logger.getLogger(Memory.class
                .getName()).log(Level.SEVERE, null,
ex);
        }
    }
}

}

public void ClearTable() {
    dtm.setRowCount(0);

    Object[] v = new Object[]{0, 0, 0, 0, 0, 0, 0, 0, 0};

    for (int idx = 0; Integer.parseInt(Integer.toString(idx,
8)) <= 177760; idx += 16) {
        v[0] = Integer.parseInt(Integer.toString(idx, 8));
        dtm.addRow(v);
    }
}
}

```

```

    public void formComponentShown(java.awt.event.ComponentEvent
    evt) {

        }

    public int octToDec(int a) {
        return Integer.parseInt(Integer.toString(a, 10), 8);
    }

    public int getPoMar(int adr) {

        int result = -1;

        if (adr >= 0 && adr <= (int) 65535) {
            if (adr % 2 != 0) {
                adr--;
            }
            int a = adr / 16;
            int b = (adr % 16) / 2 + 1;

            result = octToDec((int) dtm.getValueAt(a, b));
        }

        return result;
    }

    // Запись в память
    public void setPoMar(int adr, int value) {
        if (adr >= 0 && adr <= (int) 0xffff) {
            if (adr % 2 != 0) {
                adr--;
            }
            int a = adr / 16, b = (adr % 16) / 2 + 1;

            dtm.setValueAt(decToOct(value), a, b);
        }
    }

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		148

```

        } else {
            System.out.println("Неверный адрес в mar (" +
String.valueOf(adr) + ")");
        }
    }

//=====
====

    public class CustomTableRenderer extends
DefaultTableCellRenderer {

        @Override
        public Component getTableCellRendererComponent(JTable
table, Object value,
            boolean isSelected, boolean hasFocus, int row,
int column) {

            Component c =
super.getTableCellRendererComponent(table, value, isSelected,
            hasFocus, row, column);

            if
(table.getColumnName(column).compareToIgnoreCase("Addr") == 0) {
                if (clrFont.getBlue() > 128 &&
clrFont.getGreen() > 128 && clrFont.getRed() > 128) {
                    c.setForeground(new Color(0, 210, 30));
                } else {
                    c.setForeground(new Color(0, 150, 20));
                }
            } else {
                if (((int) value) != 0) {
                    c.setForeground(Color.RED);
                } else {
                    c.setForeground(clrFont);

```

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
						149
Изм.	Лист	№ докум.	Подпись	Дата		

```

        }

    }

    if (row % 2 == 0) {
        c.setBackground(clrBack);
    } else {
        if (clrBack.getRed() > 128 && clrBack.getGreen()
> 128 && clrBack.getBlue() > 128) {
            c.setBackground(new Color(clrBack.getRed() -
10, clrBack.getGreen() - 10, clrBack.getBlue() - 10));
        } else {
            c.setBackground(new Color(clrBack.getRed() +
10, clrBack.getGreen() + 10, clrBack.getBlue() + 10));
        }
    }

    return c;
}

}

public void setDataTable(int[] mas) {

    dtm.setRowCount(0);

    Object[] v = new Object[9];
    for (int idx = 0, i = 0;
Integer.parseInt(Integer.toString(idx, 8)) <= 177760; idx += 16)
    {
        for (int j = 0, k = 0, h; j < 9; j++) {
            h = Integer.parseInt(Integer.toString(idx + (j -
k) * 2, 8));

            if (k == 1) {
                h = mas[i];
            }
        }
    }
}

```

```

        i++;
    }
    k = 1;
    v[j] = h;
}
dtm.addRow(v);
}
}

public int[] getDataTable() {

    int[] mas = new int[8 * 4096 + 2];

    for (int idx = 0, i = 0, row = 0;
Integer.parseInt(Integer.toString(idx, 8)) <= 177760; idx += 16,
row++) {

        for (int j = 1; j < 9; j++, i++) {
            mas[i] = (int) dtm.getValueAt(row, j);
        }
    }
    return mas;
}
}

```

Класс *Listing.java*

```

public class Listing {

    Listing(JLabel jLabel1, JTextArea listingArea1, JCheckBox
jCheckBox) {

        this.jLabel = jLabel1;
        this.listingArea = listingArea1;
        this.jCheckBox = jCheckBox;
    }

    private final javax.swing.JLabel jLabel;
    private final javax.swing.JTextArea listingArea;

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		151

```

private final JCheckBox jCheckBox;

public void setListing() {

    //-----
    // Заголовок сверху
    String[] mas = new String[]{
        "Addr", "M1", "M2", "M3", "M4", "M5", "M6", "M7",
        "M8", "M9", "M10", "M11", "M12", "M13", "M14", "M15"
    };

    String str2 = "";
    for (int k = 0; k < 16; k++) {
        switch (k) {
            case 0:
                str2 += String.format(" %4s", mas[k]);
                break;
            case 1:
                str2 += String.format("%7s", mas[k]);
                break;
            case 10:
                str2 += String.format("%5s", mas[k]);
                break;
            default:
                str2 += String.format("%4s", mas[k]);
                break;
        }
    }
    jLabel.setText(str2);

    //-----
    // 0 - вывод всего
    // 1 - f ==0
    // Текстовое поле с командами
    listingArea.selectAll();
}

```

```

        listingArea.replaceSelection("");
        boolean check = jCheckBox.isSelected();
        for (int j = 0, n = 0;
Integer.parseInt(Integer.toString(j, 8)) <
Integer.parseInt(Integer.toString(idPzuArray.size()+1, 8), 8) &&
n < 4096; n++) {

            m[15] = (short) ((pzu[3][n] & 65280) / 256);
            m[14] = (short) ((pzu[3][n] & 192) / 64);
            m[12] = (short) ((pzu[3][n] & 56) / 8);
            m[11] = (short) (pzu[3][n] & 7);
            m[13] = (short) ((pzu[2][n] & 49152) / 16384);
            m[10] = (short) ((pzu[2][n] & 14336) / 2048);
            m[9] = (short) ((pzu[2][n] & 1792) / 256);
            m[8] = (short) ((pzu[2][n] & 192) / 64);
            m[7] = (short) ((pzu[2][n] & 48) / 16);
            m[4] = (short) (pzu[2][n] & 15);
            m[6] = (short) ((pzu[1][n] & 63488) / 2048);
            m[5] = (short) ((pzu[1][n] & 1986) / 64);
            m[3] = (short) ((pzu[1][n] & 60) / 4);
            m[2] = (short) (pzu[1][n] & 3);
            m[1] = pzu[0][n];
            m[0] = (short) n;

            boolean f = true;
            int idx = idPzuArray.indexOf(new IdPzu((short) n,
(short) 0));
            if (idx != -1) {
                j++;
                f = idPzuArray.get(idx).isEmpty(m);
            }
            if (!check || (check && !f)) {
                String str = "";
                for (int k = 0; k < 16; k++) {
                    switch (k) {

```

```

        case 0:
            str += String.format("%5o", (int)
m[k] & 0xffff);

            break;
        case 1:
            str += String.format("%7o", (int)
m[k] & 0xffff);

            break;
        default:
            str += String.format("%4o", m[k]);
            break;
    }

    }

    listingArea.append(str + "\n");
}

}

    listingArea.setCaretPosition(0);
}

public void saveListing() {

    File writeFile;
    Writer writer = null;
    JFileChooser saveListing = new JFileChooser(new
File(System.getProperty("user.dir")));
    FileNameExtensionFilter filter = new
FileNameExtensionFilter("Текст листинга (.txt)", "txt");
    saveListing.setFileFilter(filter);

    if (saveListing.showDialog(null, "Сохранить") ==
JFileChooser.APPROVE_OPTION) {
        writeFile = new
File(saveListing.getSelectedFile().getPath());
        try {

```

```

        writer = new BufferedWriter(new
OutputStreamWriter(
            new FileOutputStream(writeFile), "utf-
8"));

        writer.write(jLabel.getText() + "\r\n");
        listingArea.write(writer);
    } catch (IOException ex) {
        System.out.println("Ошибка: запись в файл txt
линстинга");
    } finally {
        if (writer != null) {
            try {
                writer.close();
            } catch (IOException ex) {
                System.out.println("Ошибка: закрытие
файла линстинга (txt)");
            }
        }
    }
}

public void copyListing() {

    StringSelection stringSelect = new
StringSelection(listingArea.getText());
    Clipboard clipboard =
Toolkit.getDefaultToolkit().getSystemClipboard();
    clipboard.setContents(stringSelect, null);
    listingArea.selectAll();
}
}

```

Класс *Decoration.java*

```
public class Decoration {
```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		155

```

private final ArrayList<Tema> tems = new ArrayList();
public int current;
private final String pathProper = "prop.txt";

private final ButtonGroup bg;

public Decoration(ButtonGroup bg1) {

    bg = bg1;

    current = 0;

    String[] m1 = new String[]{
        "FlatDarkLaf", "Dark",
        "FlatLightLaf", "Light",
        "com.jtattoo.plaf.aero.AeroLookAndFeel", "Aero",
        "com.jtattoo.plaf.aluminium.AluminiumLookAndFeel",
"Aluminium",
        "com.jtattoo.plaf.bernstein.BernsteinLookAndFeel",
"Bernstein",
        "com.jtattoo.plaf.fast.FastLookAndFeel", "Fast",
        "com.jtattoo.plaf.graphite.GraphiteLookAndFeel",
"Graphite",
        "com.jtattoo.plaf.hifi.HiFiLookAndFeel", "HiFi",
        "com.jtattoo.plaf.luna.LunaLookAndFeel", "Luna",
        "com.jtattoo.plaf.mcwin.McWinLookAndFeel", "McWin",
        "com.jtattoo.plaf.mint.MintLookAndFeel", "Mint",
        "com.jtattoo.plaf.noire.NoireLookAndFeel", "Noire",
        "com.jtattoo.plaf.smart.SmartLookAndFeel", "Smart",
        "com.jtattoo.plaf.texture.TextureLookAndFeel",
"Texture"};

    Enumeration<AbstractButton> buttons = bg.getElements();

```

```

        for (int i = 0; buttons.hasMoreElements() && i <
m1.length; i += 2) {

            tems.add(new Tema(m1[i], m1[i + 1]));
            if (i == current) {
                AbstractButton button = buttons.nextElement();
                button.setSelected(true);
            } else {
                buttons.nextElement().setText(m1[i + 1]);
            }

        }

    }

//=====

public void setDecor() {

    AbstractButton button;

    for (Enumeration<AbstractButton> buttons =
bg.getElements(); buttons.hasMoreElements();) {

        button = buttons.nextElement();

        if (button.isSelected()) {
            bg.clearSelection();
            button.setSelected(true);
            setDecor2(button.getText());
            break;
        }

    }

}

private void setDecor2(String name) {

```

```

        if (name.equals("")) {
            System.out.println("Ошибка в программе: тема не
загрузилась \nName не найден: ");
            return;
        }

        String path = "";

        Iterator<Tema> iter = tems.iterator();
        for (current = 0; iter.hasNext(); current++) {
            Tema t = iter.next();
            if (t.name.equals(name)) {
                path = t.path;
                break;
            }
        }

        if (path.equals("")) {
            System.out.println("Ошибка в программе: тема " +
current + " не загрузилась \nPath не найден: ");
            return;
        }

        switch (current) {
            case 0:
                FlatDarkLaf.install();
                FlatDarkLaf.updateUI();
                break;
            case 1:
                FlatLightLaf.install();
                FlatLightLaf.updateUI();
                break;
            default:
                try {

```

```

        UIManager.setLookAndFeel(path);
    } catch (ClassNotFoundException |
IllegalAccessException | InstantiationException |
UnsupportedLookAndFeelException ex) {

    Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
ex);

        System.out.println("Ошибка в программе: тема " +
current + " не загрузилась \nПуть: " + tems.get(current).path +
"\nНазвание: " + tems.get(current).name);
    }
    break;
}
saveProperties();
}

public void getDecor() {
    switch (current) {
        case 0:
            FlatDarkLaf.install();
            FlatDarkLaf.updateUI();
            break;
        case 1:
            FlatLightLaf.install();
            FlatLightLaf.updateUI();
            break;
        default:
            String path = tems.get(current).path;
            try {
                UIManager.setLookAndFeel(path);
            } catch (ClassNotFoundException |
IllegalAccessException | InstantiationException |
UnsupportedLookAndFeelException ex) {

```

```

Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
ex);

                System.out.println("Ошибка в программе: тема
" + current + " не загрузилась \nПуть: " +
tems.get(current).path + "\nНазвание: " +
tems.get(current).name);
            }
            break;
        }
    }

//=====

public void loadProperties() {
    current = 0;
    try {
        FileInputStream fileInputStream = new
FileInputStream(pathProper);

        Properties prop = new Properties();
        prop.load(fileInputStream);

        String name = prop.getProperty("tema");
        System.out.println("Загружена тема:" + name);

        AbstractButton button;
        boolean f = true;

        for (Enumeration<AbstractButton> buttons =
bg.getElements(); buttons.hasMoreElements();) {

            button = buttons.nextElement();

            if (button.getText().equals(name)) {
                bg.clearSelection();
            }
        }
    } catch (IOException ex) {
        Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

```

        button.setSelected(true);
        f = false;
        break;
    }
}
if (f) {
    System.out.println("Ошибка в программе: тема " +
pathProper + " не обнаружена");
}

    } catch (IOException e) {
        System.out.println("Ошибка в программе: файл " +
pathProper + " не обнаружено");
    }
}

public void saveProperties() {
    try {

        Properties props = new Properties();
        props.setProperty("tema", tems.get(current).name);

        OutputStream out = new FileOutputStream(new
File(pathProper));

        props.store(out, "User properties");

    } catch (FileNotFoundException ex) {

        Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
ex);

    } catch (IOException ex) {

        Logger.getLogger(iTact.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

```

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		161

```

    }

}

//=====

private class Tema {

    String path;
    String name;

    Tema(String path1, String name1) {
        path = path1;
        name = name1;
    }
}
}

```

Приложение Б

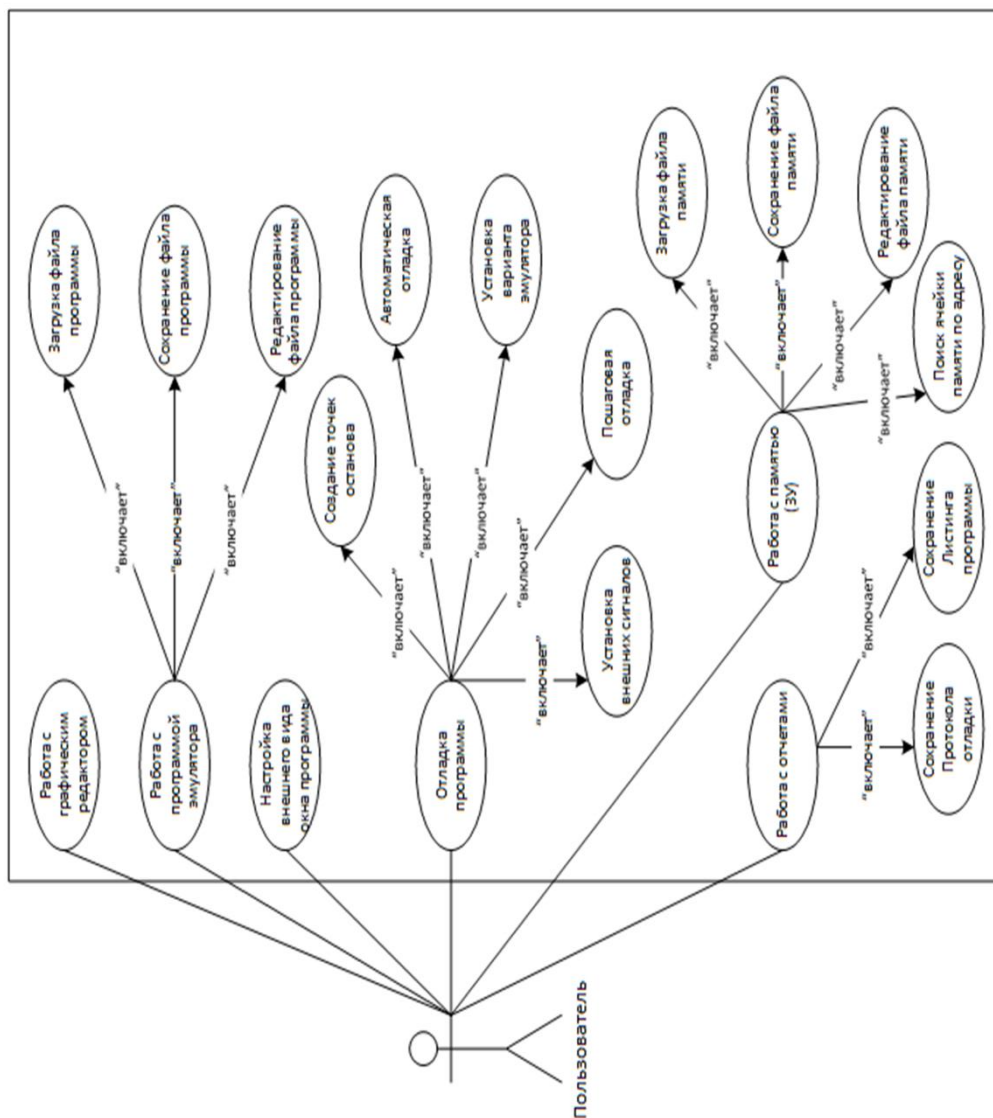
(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

UML – диаграмма прецедентов

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		163



ИТУ 1.09.03.01.41.002 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.41.001 ПЗ

Приложение В

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

UML – диаграмма классов

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		165

Приложение Г

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

UML – диаграмма последовательностей

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		167

Приложение Д

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

UML – диаграмма деятельности

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		169

Приложение Е

(Справочное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

Презентация

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		171



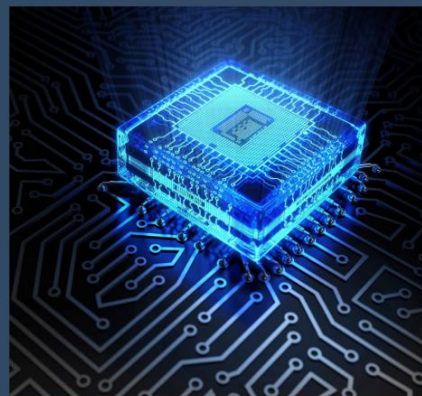
Пензенский
государственный
университет



Кафедра
вычислительной
техники

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Учебная система проектирования микропрограммируемых ЭВМ. Программно-логическая модель



АВТОР РАБОТЫ: СТУДЕНТ ГРУППЫ 18ВВ1 ФЕДУЛОВ В.Д.

НАУЧНЫЙ РУКОВОДИТЕЛЬ: ПРОФЕССОР, К.Т.Н. КОННОВ Н.Н.

Актуальность

Подготовка современного специалиста в области вычислительной техники предполагает изучение архитектуры процессоров на двух уровнях:

- Ассемблерном, с использованием стандартных средств низкоуровневого программирования
- Микропрограммном, с использованием эмуляторов ЭВМ

Использование программных эмуляторов микропрограммируемых ЭВМ в лабораторном практикуме и курсовом проектировании дисциплин, связанных с изучением архитектуры, позволяет студентам разрабатывать различные алгоритмы выполнения арифметических операций с фиксированной и плавающей точкой, выполнять микропрограммную имплементацию шестнадцатиразрядных ЭВМ с различной процессорной архитектурой

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		172

1

Разработан под
Windows платформу

2

Невозможность
редактирования МК
в процессе
выполнения

3

Отсутствие
возможности
работы с ОЗУ

4

Отсутствие
возможности
взаимодействия с
периферийными
устройствами

5

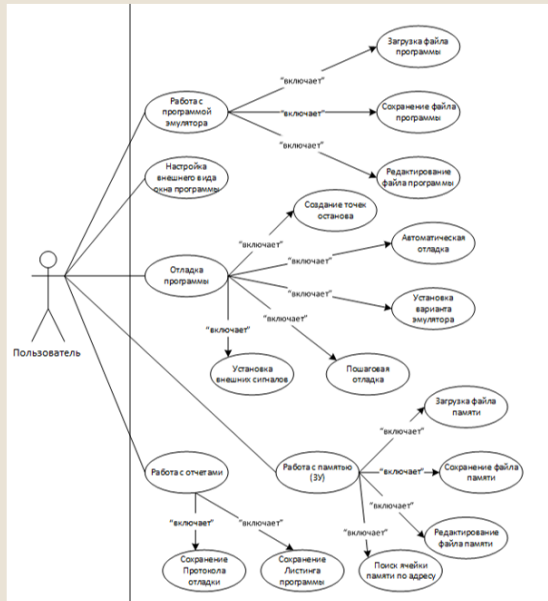
Отсутствие
возможности
обработки
перерываний

Недостатки известных эмуляторов

Цели и задачи

В проекте должна быть разработана программно-логическая модель 16 разрядной микропрограммируемой ЭВМ. Включающая: процессор и оперативную память емкостью до 16 Кбайт. Процессор должен обеспечивать формат команд поддерживающий реализацию системы команд PDP-11. Программно-логическая модель должна быть кроссплатформенной. Программно-логическая модель должна функционировать в потактовом и автоматическом режиме с возможностью интерактивного контроля и отладки информации.

Выбор решения



Кроссплатформенность

Выбор объектно-ориентированного языка программирования Java

Редактирование МК

Доработка интерфейса программы

Работа с ОЗУ

Разработка модуля и алгоритма взаимодействия с ОЗУ

Периферийная система

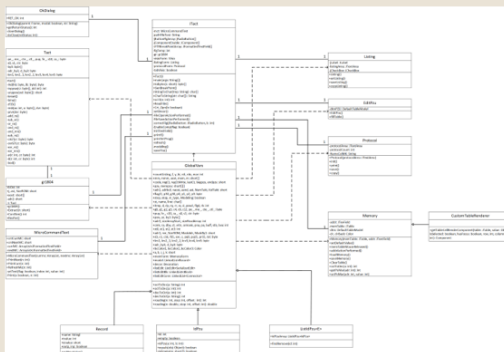
Доработка интерфейса программы

Прерывания

Разработка алгоритма обработки прерывания

Описание программы

Используется восемь основных классов, каждый из которых отвечает за свою область в программе и имеет свой набор методов



gr1804

Взаимодействие с ядром

Tact

Ядро микропроцессора

Decoration

Внешний вид интерфейса

Memory

Редактор памяти

Protocol

Отчет отладки МП

Listing

Листинг МП

iFact

Главный класс реализующий окно эмулятора.

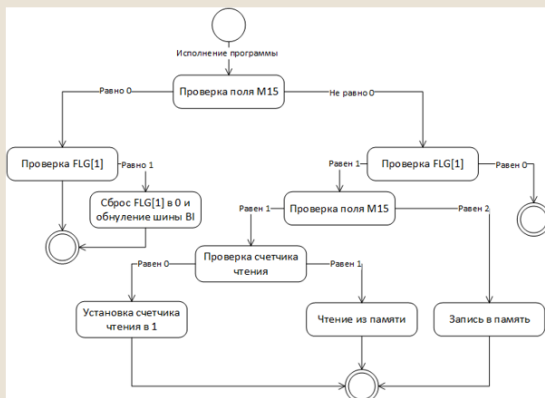
GlobalVars

Хранение и обмен переменными между классами

Алгоритм работы с памятью

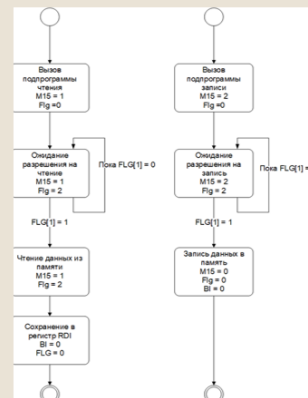
• Алгоритм чтения

Формирование сигнала запроса чтения для ОЗУ. Ожидание сигнала от ОЗУ, разрешающего чтение данных с шины BI. Чтение данных с шины в регистр RDI (Register Data Input)

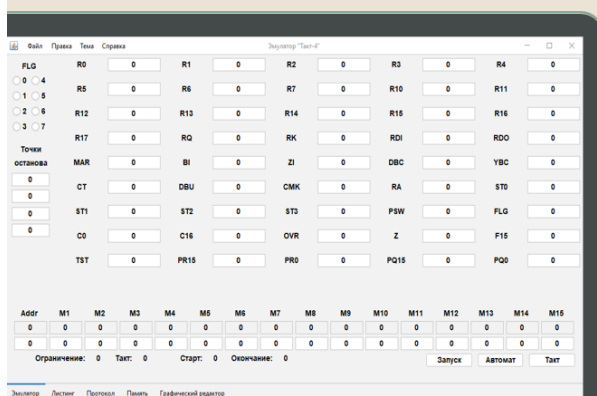


• Алгоритм записи

Формирование сигнала запроса записи для ОЗУ. Ожидание сигнала от ОЗУ, разрешающего запись данных из регистра RDO (Register Data Output).

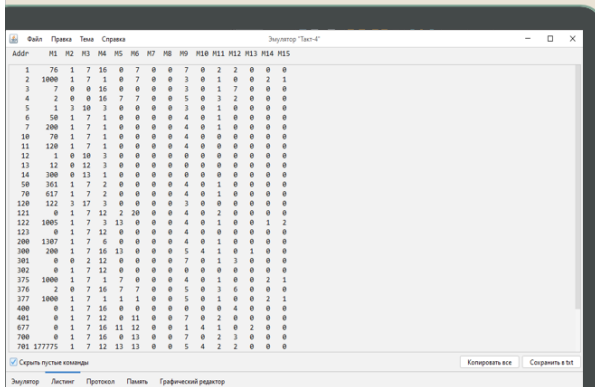


Главное окно



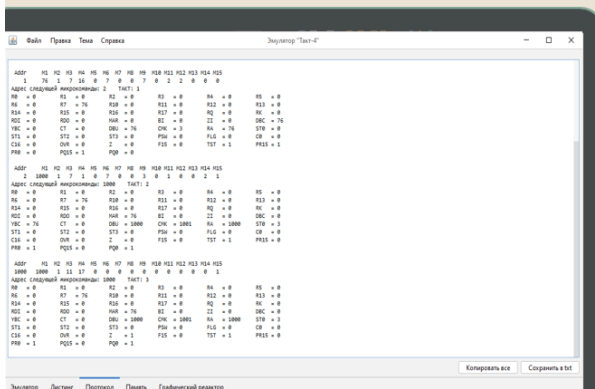
- Порты ввода внешних сигналов
- Точки останова
- Регистры микропроцессора
- Текущая и следующая микрокоманда
- Органы управления ходом отладки

Окно листинга



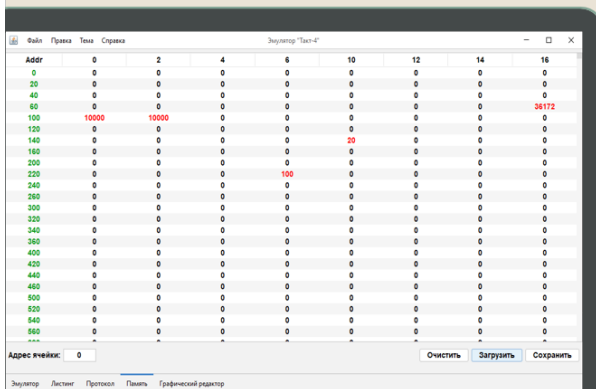
- Таблица из 15 полей
- Возможность скрытия пустых микрокоманд
- Копирование в буфер обмена
- Экспорт в текстовый файл

Окна протокола



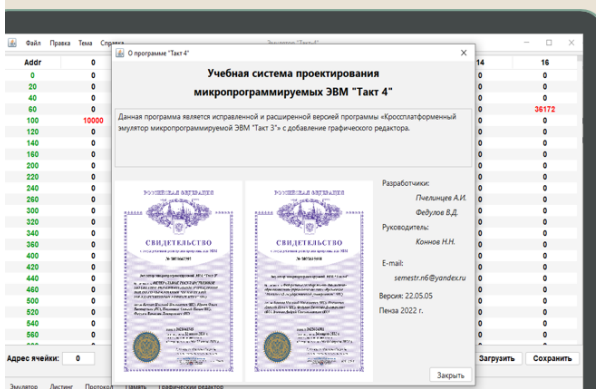
- Копирование в буфер обмена
- Экспорт в текстовый файл
- Формирование протокола в процессе отладки

Окно памяти



- Поиск ячейки памяти
- Очистка таблицы памяти
- Загрузка и сохранение
- Редактирование ячеек памяти

Справочное окно



- Краткое описание программы
- Контакты разработчиков

Характеристики программы

 <p>Windows 7/8/10 Linux Альт Образование/ Ubuntu/Astra</p>	 <p>JRE версия 1.8.0 и выше</p>	 <p>ОЗУ не менее 256 Мбайт</p>	 <p><u>Дисковая</u> <u>память не менее</u> <u>5 Мбайт</u></p>
--	--	--	--

Свидетельства



Заключение

В ходе выполнения данной выпускной квалификационной работы были проведены следующие работы:

- Разработана и протестирована программно-логическая модель
- Разработаны алгоритмы работы с ОЗУ
- Разработанная система не зависит от операционной системы

Система готова к использованию в учебном процессе для подготовки бакалавров и специалистов в области вычислительной техники

Спасибо
за внимание!

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		179

Приложение Ж

(Справочное)

Учебная система проектирования микропрограммируемых ЭВМ.

Программно – логическая модель

Свидетельства

					<i>ПГУ 1.09.03.01.41.001 ПЗ</i>	Лист
						180
Изм.	Лист	№ докум.	Подпись	Дата		

РОССИЙСКАЯ ФЕДЕРАЦИЯ



СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021662392

Эмулятор микропрограммируемой ЭВМ "Такт-3"

Правообладатель: **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ "ПЕНЗЕНСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ" (RU)**

Авторы: **Коннов Николай Николаевич (RU), Юрова Ольга
Викторовна (RU), Пчелинцев Алексей Ильич (RU),
Федулов Вячеслав Дмитриевич (RU)**



Заявка № 2021661545

Дата поступления **22 июля 2021 г.**

Дата государственной регистрации
в Реестре программ для ЭВМ **27 июля 2021 г.**

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 0x02A5CFB800B1ACFB9A40A2F08092E9A118
Владелец **Ивлиев Григорий Петрович**
Действителен с 15.01.2021 по 15.01.2035

Г.П. Ивлиев

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		181

РОССИЙСКАЯ ФЕДЕРАЦИЯ



СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022615911

Эмулятор микропрограммируемой ЭВМ "Такт-4"

Правообладатель: *Федеральное государственное бюджетное образовательное учреждение высшего образования "Пензенский государственный университет" (RU)*

Авторы: *Коннов Николай Николаевич (RU), Пчелинцев Алексей Ильич (RU), Федулов Вячеслав Дмитриевич (RU), Бычков Андрей Станиславович (RU)*



Заявка № 2022614301

Дата поступления 24 марта 2022 г.

Дата государственной регистрации

в Реестре программ для ЭВМ 04 апреля 2022 г.

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 68b80077e14e40f0a94edbd24145d5c7
Владелец **Зубов Юрий Сергеевич**
Действителен с 2.03.2022 по 26.05.2023

Ю.С. Зубов

					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		182



					ПГУ 1.09.03.01.41.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		183