

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. Графический

редактор

Студент

Тер

(подпись, дата)

Тренингов Алексей Ильич

(ФИО полностью)

Руководитель

02.06.22

(подпись, дата)

Н. Н. Коннов

(фамилия, инициалы)

Нормоконтролёр

02.06.22

(подпись, дата)

А. С. Бычков

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

М.А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

07.06.2022 г. № 3)

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. Графический**

редактор

Студент

Григорьев

(подпись, дата)

Григорьев Алексей Ильич

(ФИО полностью)

Руководитель

02.06.22

(подпись, дата)

Н. Н. Коннов

(фамилия, инициалы)

Нормоконтролёр

02.06.22

(подпись, дата)

А. С. Бычков

(фамилия, инициалы)

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

Заведующий кафедрой

М.А. Митрохин

(подпись)

М.А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от

_____ № _____)

Секретарь ГЭК

(подпись)

Е.И. Гурин

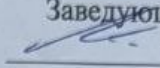
(фамилия, инициалы)

Пенза 2022

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« » 20 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Пчелинцев Алексей Ильич гр. 18BB1 факультета ФВТ

2. Тема работы Учебная система проектирования микропрограммируемых ЭВМ.
Графический редактор

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

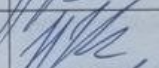
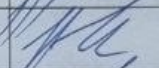
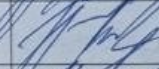
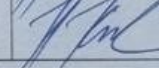
3. Руководитель бакалаврской работы профессор, к.т.н. Коннов Н. Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.) Необходимо разработать графический редактор учебной системы проектирования микропрограммируемых ЭВМ. Редактор предназначен для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительной техники с целью выработки компетенций по проектированию алгоритмов работы микропрограммируемых процессоров и других средств вычислительной техники. Функционал редактора: добавление, редактирование, удаление и перемещение блоков микрокоманд, соединительных линий и текстовых полей; перемещение пользователя по рабочей области, масштабирование, работа с несколькими листами; экспорт схемы в pdf и png; сохранение графических объектов с программой и памятью.
Редактор должен быть кроссплатформенным и должен исполняться в среде следующих ОС: Windows7, 10, Linux Альт Образование, Ubuntu.

Язык программирования – Java.

5. Перечень подлежащих разработке вопросов анализ темы, аналогов, разработка архитектуры приложения, выбор языка программирования и технологий на основе которых будет проектироваться система, разработка графического редактора, тестирование работы системы, изучение эргономических требований к рабочему месту и обеспечение электробезопасности.

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 графического редактора	45%	06.05.2022-21.05.2022	
5	Тестирование и отладка	15%	2.05.2022-25.05.2022	
6	Оформление выпускной работы	15%	26.05.2022-31.05.2022	

Дата выдачи задания " 26 " апреля 20 22 г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить
Декан факультета _____

(подпись, дата)

02.06.2022

РЕФЕРАТ

Выпускная квалификационная работа содержит 200 листов, 74 рисунков, 6 таблиц, 8 источников, 8 приложений.

ЭМУЛЯТОР, МИКРОПРОЦЕССОР, МИКРОЭВМ, JAVA, K1804, PDP11, УЧЕБНЫЙ ТРЕНАЖЕР, КРОССПЛАТФОРМЕННЫЙ, SWING, ИНТЕРФЕЙС, ГРАФИЧЕСКИЙ РЕДАКТОР.

Цель работы – разработка графического редактора учебной системы проектирования микропрограммируемых ЭВМ. Редактор предназначен для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительной техники с целью выработки компетенций по проектированию алгоритмов работы микропрограммируемых процессоров и других средств вычислительной техники.

В результате проделанной работы был разработан кроссплатформенный графический модуль для 16-разрядной микропрограммируемой ЭВМ. Редактор позволяет создавать программу с помощью графических объектов: блок микрокоманды, соединительная линия, текстовое поле. Система разработана на языке программирования *Java* с использованием графической библиотеки *Swing*.

					ПГУ 1.09.03.01.32.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Учебная система микропрограммируемых ЭВМ. Графический редактор. Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Пчелинцев. А.И.					2	203	
Провер.		Коннов Н.Н.							
Н. Контр		Бычков А.С.							
Утверд.		Митрохин М.А.							
					ФВТ, каф. ВТ гр. 18ВВ1				

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Учебная система проектирование микропрограммируемых ЭВМ.	
Графический редактор.....	7
1.1 Анализ темы.....	7
1.1.1 Обоснование необходимости разработки.....	7
1.1.2 Анализ аналогов.....	7
1.1.3 Постановка задачи разработок.....	8
1.2 Архитектура системы.....	9
1.3 Выбор инструментов разработки.....	11
1.4 Описание графического редактора.....	11
1.4.1 Класс <i>Anketa</i>	11
1.4.2 Класс <i>Block</i>	16
1.4.3 Класс <i>JLayeredPaneTest</i>	16
1.4.4 Классы <i>Line</i> и <i>Rect</i>	17
1.4.5 Класс <i>Circle</i>	18
1.4.6 Класс <i>Connector</i>	18
1.4.7 Класс <i>SelectPanel</i>	28
1.4.8 Класс <i>TextBlock</i>	29
1.4.9 Класс <i>Visio</i>	36
1.4.10 Класс <i>ExportPDF</i>	48
1.5 Описание способа организации пользовательского интерфейса.	49
1.6 Тестирование программы.....	55
1.6.1 Создание программы.....	56

1.6.2	Изменение БМК.....	58
1.6.3	Сохранение и загрузка программы.....	62
1.6.4	Перемещение между листами.....	67
1.6.5	Выделение и удаление программы графических объектов	71
1.6.6	Экспорт схемы программы.....	73
1.6.7	Отмена и возврат действий с объектами.....	75
1.6.8	Импорт программ и генерация схемы.....	77
1.6.9	Перемещение и масштабирование рабочего листа.....	79
1.6.10	Создание и удаление рабочего листа.....	81
2	Безопасность жизнедеятельности.....	85
2.1	Эргономические требования к рабочему месту.....	85
2.2	Обеспечение электробезопасности.....	90
	Заключение.....	92
	Список используемых источников.....	93
	Приложение А.....	94
	Приложение Б.....	180
	Приложение В.....	182
	Приложение Г.....	184
	Приложение Д.....	186
	Приложение Е.....	188
	Приложение Ж.....	194
	Приложение З.....	198

Перечень сокращений и обозначений

БМК – блок микрокоманды

К1804 – микропроцессорный комплект серии К1804

ЛЕ – логическая единица

ЛКМ – левая кнопка мыши

ЛН – логический ноль

МК – микрокоманда

ОЗУ – оперативно запоминающее устройство

ОС – операционная система

ПКМ – правая кнопка мыши

ПЭВМ – персональная электронная вычислительная машина

СЛ – соединительная линия

ТБ – текстовый блок (поле)

ЭВМ – электронная вычислительная машина

ЭМПС – эмулятор микропроцессорной системы

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						н 7
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В настоящее время получило широкое распространение использование микропроцессоров в качестве встроенных элементов систем автоматического управления, в том числе как управляющих блоков периферийных узлов вычислительных комплексов. Функции, возлагаемые в такой системе на микропроцессор, полностью определяются программой, хранимой в его памяти.

Поскольку программа работы встроенного микропроцессора изменяется сравнительно редко, к параметрам этой программы и к надежности ее функционирования предъявляются особенно жесткие требования. В связи с этим являются обязательными тщательная отладка и всестороннее тестирование программы работы микропроцессора.

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						н 8
Изм.	Лист	№ докум.	Подпись	Дата		

1 Учебная система проектирование

микропрограммируемых ЭВМ. Графический редактор

1.1 Анализ темы

1.1.1 Обоснование необходимости разработки

Микропроцессоры имеют, как правило, ограниченные возможности ввода-вывода программ и обрабатываемых данных, что затрудняет выполнение работ по отладке программ. Такие работы также затруднены или вообще невозможны при разработке новых микропроцессорных систем. Всё это стимулирует использование универсальных ЭВМ для отладки и тестирования программ микропрограммируемых ЭВМ и микропроцессоров, т.е. использование методики эмуляции программ [1].

Принятые правительственные решения о переходе органов власти и бюджетных учреждений, в т.ч. и ВУЗов, на импортозамещение используемого ПО заставляет вести работу по переводу учебных программ на отечественное либо свободно распространяемое ПО.

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 данный момент существует единственный известный аналог, реализованный в виде макроса для *Microsoft Visio 2003*. Возможности макроса:

- добавление, редактирование, перемещения и удаление блока микрокоманды;

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						н 9
Изм.	Лист	№ докум.	Подпись	Дата		

- добавление, изменение, перемещения и удаление соединительных линий;
- добавление маски, текстового поля, штампа рабочей области;
- проверка вводимых значений;
- экспорт схемы в формат программы для эмулятора;
- графическая реализация окна для редактирования микропрограммы.

Недостатки данного макроса:

- отсутствие возможности изменить микропрограмму во время отладки;
- отсутствие кроссплатформенности, работает только в устаревшем, на сегодняшний день, Microsoft Visio 2003;
- для изменения программы, необходимо заново выполнять экспорт схемы.

При разработке программы необходимо учесть все возможности существующего на данный момент ПО и устранить недостатки. Для повышения конкурентоспособности программного продукта.

1.1.3 Постановка задачи разработок

В ходе выпускной квалификационной работы необходимо разработать программу, предназначенную для использования в учебном процессе для изучения и отработки алгоритмов функционирования процесса с системой команд *PDP11*, для написания и редактирования программ, в аппаратной среде, построенной на секционированном МПК1804. Реализовать следующий функционал редактора:

- добавление, изменение, перемещение и удаление графических объектов – БМК, СЛ и ТБ;
- несколько рабочих листов;
- перемещение по рабочему листу и его масштабирование;
- сохранение схемы программы в файл, экспорт в файлы в форматах *PDF* и *PNG*;

Система должна функционировать на ПЭВМ в следующих операционных средах: *Windows 7,8,10; Linux АльтОбразование, Ubuntu.*

1.2 Архитектура системы

Исходя из поставленной задачи можно определить, что пользователь один, бизнес-цель – это эмулятор микропрограммируемой ЭВМ, а система – ПЭВМ с указанными требованиями. Потребителями архитектуры являются разработчики и тестировщики [2]. Необходимо определить основные сценарии с помощью диаграммы прецедентов (*usecase*), представленной в приложении Д.

Исходя из диаграммы, можно выделить один основной сценарий – работа с графическим редактором. Второстепенными сценариями являются: работа с эмулятором; настройка внешнего вида; работа с отчетами; работа с памятью.

Общее представление приложения:

- тип приложения – классическое *desktop*-приложение. Данные приложения позволяют создавать практически любые по сложности решения. Основным минусом является сложность обновления приложений у конечных пользователей;
- ограничений развертывания – среды ОС *Windows 7/8/10, LinuxUbuntu/Astra/Альт Образование/Manjaro*;
- архитектурный стиль проектирования – многослойная архитектура. Функциональные области приложения разделяются на многослойные группы. Можно выделить такие слои как: слой представления; бизнес-слой; слой доступа к данным;
- подходящая технология – насыщенное клиентское приложение для выполнения преимущественно на клиентских ПК. Для разработки приложений с насыщенными *UI*, развертываемыми и выполняемыми на клиенте, можно использовать технологию *Swing*.

Для оценки времени и объема работ был разработан прототип дизайна приложения (см. Рисунок 1 и Рисунок 2), в котором отражен необходимый функционал. Технические риски отсутствуют.

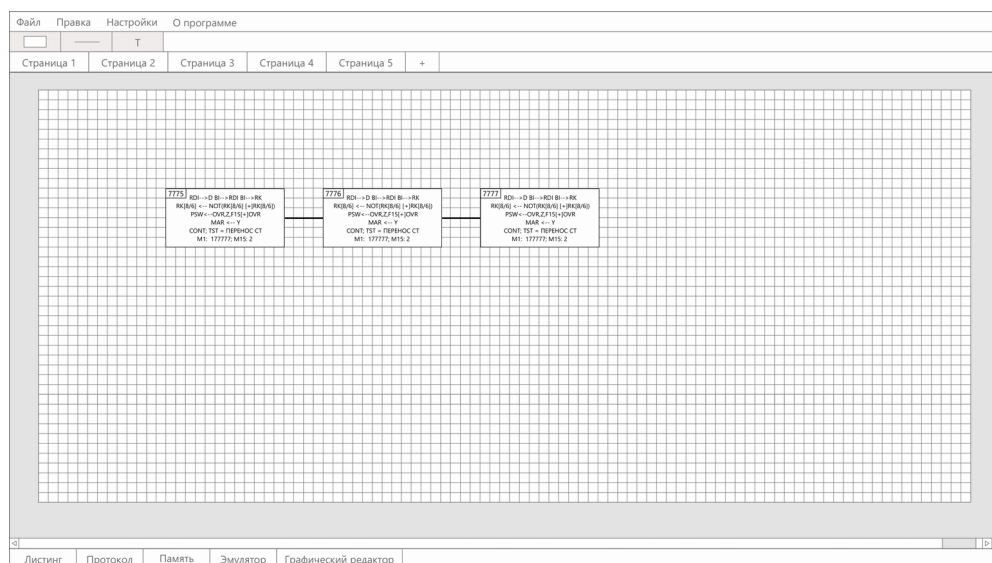


Рисунок 1 – Прототип дизайна приложения, рабочее пространство

Рисунок 2 – Прототип дизайна приложения, окно микрокоманды

На основании прототипа будет разрабатываться полноценный дизайн приложения и его верстка, а также накладываются ограничения на некоторые средства разработки и языки программирования, что позволяет точнее выбрать их.

1.3 Выбор инструментов разработки

Для реализации поставленных задач были рассмотрены следующие технологии, язык программирования и среда разработки:

- *NetBeansIDE* – свободная интегрированная среда разработки приложений (*IDE*) на языках программирования *Java*. конструктор графического интерфейса. Подключение *Swing* и других инструментов для построения внешнего вида программы максимально оптимизирует дизайнерскую часть работы [3]. Общим недостатком из-за *JDK/SDK* будет существенное потребление ресурсов устройства;

- *Java* – это объектно-ориентированный язык программирования. Отличительными особенностями *Java* является: платформонезависимость, архитектурная-нейтральность, интерпретируемость, высокая производительность, динамичность и адаптируемость;

- *Swing* – библиотека для создания графического интерфейса для программ на языке *Java*[4]. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д.

1.4 Описание графического редактора

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

1.4.1 Класс *Anketa*

Класс, реализующий вывод микрокоманды, с возможностью ее редактирования и сохранения. Методы этого класса реализуют работу компонентов интерфейса. Класс расширяет базовый класс *JDialog*.

Поля класса:

- *m* – поля микрокоманды;
- *mBuffer* – ссылка на массив родительского класса, необходим для передачи выходных параметров;

- *flagM7* – исключает повторное добавление элементов в текстовое поле *M7*;

- *mask* – флаг отвечающий за наложение маски;

- *jTextArea* – массив текстовых полей.

Методы класса:

Public Anketa(short[] m1) – конструктор класса, в качестве входных параметров принимает микрокоманду *M1*. Инициализируются графические компоненты окна или формы. Сохраняется ссылка на массив родительского класса, копируется исходный массив в временный для дальнейшего изменения. Далее *UI* компоненты добавляются в список. Вывод исходной микрокоманды в текстовые поля. С 8 по 15 поле устанавливается получение числа микрокоманды и установка выделения на соответствующий пункт выпадающего списка. Пример кода:

```
jComboBoxM8.setSelectedIndex(m[8]);
```

...

```
jComboBoxM15.setSelectedIndex(m[15]);
```

Для заполнения полей *M1–M7* требуются дополнительные алгоритмы обработки, поэтому вызываются функции, которые будут описаны далее. Далее центруется окно, относительно родительского окна.

Private void printM() – метод реализующий печать полученной МК каждого поля отдельно в *jTextField*. Пример кода:

```
for (int i = 0; i < 16; i++) {  
    jTextArea[i].setText(Integer.toString(m[i], 8)); }  
Private void jButtonSaveActionPerformed() – метод,
```

срабатывающий по нажатию на кнопку «Сохранить». Предназначен для считывания и записи всех полей микрокоманды в поле родительского класса микрокоманды. Т.е. сформировать и передать выходное значение. Поле *M1*, *M5* и *M6* необходимо обновить, т.к. они не сохраняются в процессе их изменения. Пример кода:

```
Long l = Long.parseLong(jTextFieldM1.getText(), 8);
```

```

setTextFieldM1((short) l.shortValue());
getComboBoxM56();
System.arraycopy(m, 0, mBuffer, 0, 16);

```

Private void jButtonCancelActionPerformed() – метод, который обрабатывает нажатие на кнопку «Отмена». Данный метод не сохраняет измененную микрокоманду и вызывает функцию *dispose()* закрывающую окно.

Private void jTextFieldM1ActionPerformed() – метод, который обрабатывает текст внутри компонента *UI* после завершения редактирования. Получение строки, преобразование в число и установка его в поле M1.

Private void jButton4ItemStateChanged() – метод, необходимый для обновления полей после инвертирования маски. Сначала изменяется маска, затем изменяются поля M2 и M3, т.к. они зависят от маски:

```

mask = !mask;
int sel = jComboBoxM3.getSelectedIndex();
setComboBoxM2((short) 0, false);
if (jComboBoxM2.getSelectedIndex() >= 0) {
    setComboBoxM3((short) jComboBoxM2.getSelectedIndex(),
    (short) sel, true);
}

```

Private void jComboBoxM2ItemStateChanged() – метод, который вызывается при нажатии на пункт выпадающего списка для поля M2. Получает индексы выбранных полей M2 и M3. Далее устанавливается поле M2, а поле M3 корректируется. Т.к. при разных значениях поля M2 необходимо выдавать разные списки поля M3.

Private void jComboBoxM3ItemStateChanged() – метод, вызываемый, при нажатии на пункт выпадающего списка для поля M3. Получает индекс выбранного пункта и устанавливает в поле M3. Значение поля M3 не влияет на другие поля.

```

int sel = jComboBoxM3.getSelectedIndex();
m[2] = (short) jComboBoxM2.getSelectedIndex();
if (m[2] >= 0 && sel >= 0 && jComboBoxM3.getItemCount() > 15) {

```

```
setComboBoxM3((short) m[2], (short) sel, true);
```

PrivatevoidjComboBoxM11ItemStateChanged() – метод, вызывающийся, при нажатии на пункт выпадающего списка для поля M11. Получает индекс выбранного поля M11 и устанавливает его. Поле M7 корректируется. Т.к. при разных значениях поля M11 необходимо выдавать разные списки поля M7. Примеркода:

```
m[11] = (short) jComboBoxM11.getSelectedIndex();
setComboBoxM7((short) 0, (short)
jComboBoxM11.getSelectedIndex(), false);
printM();
```

PrivatevoidjTable4MousePressed() – метод, который вызывается при нажатии кнопкой мыши на таблицу. Получается индекс строки и колонки выбранной ячейки. Далее проверка на ограничение ввода, запрещено изменять нулевой столбец у всех строк и первый столбец 2 и 3 строки. Получается значение выбранной ячейки, инвертируется и устанавливается обратно:

```
if (col > 1 || col == 1 && row == 0) {
int value = (int) jTable4.getModel().getValueAt(row, col);
jTable4.getModel().setValueAt((value + 1) % 2, row,
col);
```

Затем проходится по выбранной строке пересчитывая значение поля M6:

```
m[6] = 0;
for (int i = 0; i < 5; i++) {
int j = (int) jTable4.getModel().getValueAt(row, i);
m[6] += (short) j << (4 - i);
printM();
}
```

PrivatevoidjTableM5MousePressed() – метод функционирует аналогично предыдущей функции, с тем отличием, что работает с полем M5 и другой таблицей.

PrivatevoidsetTextFieldM1() – метод, необходимый для установки числа в поле M1, а так же вывода его в двоичной системе счисления и установки в текстовое поле микрокоманды. Для вывода в двоичной системе счисления используется класс *DecimalFormat* в котором можно задать формат вывода текста. Так же используется класс *Number*, который предназначен для преобразования обычного числа в число большой разрядности. Кодреализации:

```
DecimalFormat df = new DecimalFormat("####,####,####,####");
String s1 = Integer.toBinaryString(value);
Number n = df.parse(s1);
jLabelM1.setText(df.format(n.longValue()));
jTextFieldM1.setText(Integer.toString(value, 8));
m[1] = value;
printM();
```

PrivatevoidsetComboBoxM2() – метод устанавливающий список поля M2 и выбранный элемент списка. Устанавливаемый набор полей зависит от флага маски. Переменная *flag* указывает на необходимость установки нового значения поля M2 и выбор пункта списка.

PrivatevoidsetComboBoxM3() – метод устанавливающий список поля M3 и выбранный элемент списка. Входные параметры функции: *m2* - значение поля M2, *m3* - значение поля M3, *flag* - устанавливает *value* в *combobox* или нет. Устанавливаемый набор полей зависит от флага маски и установленного значения в поле M2. Переменная *flag* указывает на необходимость установки нового значения поля M3 и выбор пункта списка.

PrivatevoidsetComboBoxM56() – метод, необходимый для установи полей M5 и M6. Переменная *flag* используется для выборки поля в которое будет занесено значение. Далее в зависимости от входного значения поля M5 или M6 устанавливаются значения на компонентах *UI*.

Private void getComboBoxM56() – метод, необходимый для получения полей M5 или M6. Получение индекса выбранной строки. В цикле

считывается значение каждой ячейки данной строки и полученное число сдвигается влево, после чего записывается в поле M5. Аналогичные операции происходят с полем M6. Пример кода, получения поля M5:

```
int index = 0;
if (jrb2.isSelected())
    index = 1;
else if (jrb3.isSelected())
    index = 2;
m[5] = 0;
for (int i = 0; i < 5; i++) {
    int j = (int) jTableM5.getModel().getValueAt(index, i);
    m[5] += (short) j << (4 - i);
    printM();
}
```

Private void setComboBoxM7() – метод необходимый для установки списка поля M7. Поля списка зависят от значения в поле M11. Если при конъюнкции поля M11 с маской 2_{10} получился ноль, то сдвиг вправо, иначе сдвиг влево. Для исключения повторного вывода полей пунктов списка введена переменная типа *Boolean flagM7*. Если входная переменная *flag* равна истине, тогда устанавливается новое значение в поле M7.

Private void jTableM4ItemStateChanged() – метод необходимый для получения и установки нового значения в поля микрокоманды. С помощью функции *getSelectedIndex()* получает номер выбранного пункта, затем преобразуется в тип short и записывается в поле микрокоманды. Аналогично работают методы для записи полей M7-M15, кроме M11.

1.4.2 Класс *Block*

Класс, реализующий отображение графического элемента и его функционал.

Private void printMy() – метод, реализующий вывод координат и текста в блоке.

Public void setMyLocation() – метод, реализующий масштабирование и размещение блока в рабочем пространстве.

Private void setTextBlock() – метод, предназначен для формирования текста в блок.

1.4.3 Класс *JLayeredPaneTest*

Класс, реализующий рабочую область графического редактора.

Protected void paintComponent() – переопределенный метод, предназначенный для прорисовки фоновой сетки.

Public boolean setLocationBlock() – метод, предназначенный для корректного расположения блока в рабочем пространстве, коррекция координат при масштабировании.

Public void setSizeBlock() – метод, необходимый для установки размера блока при масштабировании.

Public void addBlock() – метод, предназначенный для добавления блока в рабочее пространство.

Public void addZoom() – метод, необходим для увеличения масштаба.

Public void subZoom() – метод, необходим для уменьшения масштаба.

1.4.4 Классы *Line* и *Rect*

Public class Line – класс, предназначенный для хранения координат линии сетки. Конструктор класса принимает две координаты крайних точек и цвет линии. Доступные поля класса: координаты крайних точек, цвет линии и толщина, которая по умолчанию устанавливается 1 пиксель.

Public class Rect extends Rectangle – класс, отвечающий за хранение координат квадратов и их размеры. Расширяет базовый класс *Rectangle*. Конструктор класса принимает те же аргументы, что и для класса, который он расширяет, и дополнительный параметр, цвет прямоугольника.

Содержит два доступных поля:

- *color* – объекта класса *Color*, используется для восстановления цвета прямоугольника;
- *flag* – переменная целочисленного типа *int*, обозначает выделение квадрата, при нажатии на него или перекрытием другой графической

фигурой. В таблице X приведены возможные значения переменной и описание.

Таблица 1 – Значения переменной состояния прямоугольника

Значение	Описание
–1	Объект не выделен. Устанавливается по умолчанию в конструкторе и при сбросе других значений
–2	Устанавливается при нажатии на прямоугольник левой клавишей мыши.
–3	Устанавливается при нажатии на прямоугольник правой клавишей мыши.
0...4096	Устанавливается если прямоугольник перекрывается другой графической фигурой, соответствующего номера.

1.4.5 Класс Circle

Circle – доступный класс, предназначенный для отображения указателя редактора соединительных линий. Расширяет базовый класс *Janel* и переопределяет его метод *paintComponent()*, в котором устанавливается цвет указателя и с помощью инструментов рисования создается круг по заданным координатам и с указанным диаметром. Фрагмент кода, где *Color* – класс цветовой палитры, *g* – объект класса *Graphics*:

```
@Override
protected void paintComponent(Graphics g) {
    g.setColor(Color.PINK);
    g.fillOval(0, 0, 10, 10);
}
```

1.4.6 Класс Connector

Connector – доступный класс, реализующий отображение графического элемента «соединительная линия» (СЛ) и его функционал. Расширяет базовый класс *JComponent*, переопределяя его методы, описанные ниже.

Поля класса *Connector*:

Название	Тип	Описание
<i>line</i>	<i>Line2D. Double</i>	Класс, имеющий две точки и методы работы с ними. Основная СЛ.
<i>temp</i>	<i>Line2D. Double</i>	Дополнительная СЛ, предназначена для перетаскивания и перемещения.
<i>depthDefault</i>	<i>float</i>	постоянная толщина СЛ (4 пикселя), относительно которой вычисляется переменная толщина
<i>depth</i>	<i>float</i>	переменная толщина СЛ, зависит от масштаба, и устанавливает толщину линии перед отображением
<i>isSelect</i>	<i>boolean</i>	Переменная, показывающая состояние выделения линии. Логическая единица – значит линия выделена, иначе логический ноль.
<i>isNotSelect</i>	<i>boolean</i>	Переменная для фиксации выделения линии при однократном нажатии. Логическая единица – значит линия выделена нажатием и не сброшена. Логический ноль – линия не выделена или выделение сброшено.
<i>circle</i>	<i>short</i>	показатель выбранной вершины СЛ
<i>zoomOld</i>	<i>double</i>	Значение текущего масштаба, предназначено для восстановления исходного масштаба с последующим вычислением нового.
<i>radius</i>	<i>double</i>	Значение, для привязки курсора мыши к

		вершине линии. Если указатель мыши попадает в круг заданным радиусом, то происходит выделение вершины.
<i>history</i>	<i>SerializeConn</i>	Объекта класса, для фиксации последнего изменения СЛ.

Ниже приведены доступные методы класса *Connector*.

Connector – конструктор класса, принимающий крайние точки СЛ и родительский объект, в качестве входных параметров. Сбрасывается выделение СЛ и добавляются слушатели событий мыши для этого объекта.

paintComponentLine – метод, предназначенный для перерисовки СЛ, принимает ссылку на объект класса *Graphics* и не возвращает никаких значений. Если линия выделена или наведена на вершину СЛ, то выполняется алгоритм отображения круга у вершины, возможные значения переменной выделения приведены в Таблица 2.

Таблица 2 – Значения переменной выделения вершины

Значение	Описание
0	Указатель мыши не наведен ни на одну вершину
1	Указатель мыши наведен на первую вершину
2	Указатель мыши наведен на вторую вершину
3	Выделение СЛ, отображение двух выделенных вершин

getPreferredSize – переопределенный метод базового класса *JComponent*, предназначенный для получения предпочтительного размера объекта СЛ, входных параметров не принимает, а возвращаемое значение объект класса *Dimension*. Размер объекта данного класса вычисляется как сумма

координаты верхнего левого угла с длиной и шириной СЛ по каждой оси соответственно.

Contains—переопределенный метод базового класса *JComponent*, предназначенный для определения попадания точки в СЛ, в качестве входных параметров координаты точки, а возвращаемое значение логического типа: ЛЕ, если расстояние от точки меньше установленной дистанции по умолчанию, и ЛН, если расстояние больше, чем установленная дистанция. Используется для удобства выбора линии и ее вершин, а также для дополнительных проверок. Фрагмент кода, где *line* – это СЛ, *x* и *y* – координаты точки, *depthDefault* – постоянная толщина СЛ:

```
double distance = line.ptSegDist((double) x, (double) y);  
return distance < depthDefault;
```

ContainsEquals—метод, предназначенный для определения типа попадания точки в СЛ, в качестве входных параметров координаты точки, а возвращаемое значение целочисленного типа *short*: 1 – если расстояние между точкой и первой вершиной линии меньше установленной дистанции; 2 – если расстояние между точкой и второй вершиной линии меньше установленной дистанции; 3 – последний вариант, когда точка не попадает в вершины, что означает выделение всей линии. Для определения расстояния используется функция *distance* у класса *Point*, которая вычисляет и возвращает наименьшее расстояние между двумя точками. Фрагмент кода, где *line* – это СЛ, *depthDefault* – постоянная толщина СЛ:

```
if (line.getP1().distance(x, y) < depthDefault) {  
    return 1;  
} else if (line.getP2().distance(x, y) < depthDefault) {  
    return 2;  
} else {  
    return 3;  
}
```

setSelect—метод, предназначенный для установки или снятия выделения и изменением толщины СЛ, в качестве входных параметров принимает

значение логического типа, от которого устанавливается или снимается выделение, возвращаемого значение нет. Фрагмент кода, где *flag* – логическое значение для установки или сброса, *depthDefault* – постоянная толщина СЛ, *depth* – переменная толщина СЛ:

```
if (flag) {
    depth = depthDefault * 2.0f;
} else {
    depth = depthDefault;
}
isSelect = flag;
```

containsArea – метод, предназначенный для проверки СЛ, на попадание ее в прямоугольник выделения. Четыре входных параметра целочисленного типа:

- *x* – координата *X* левого верхнего угла прямоугольника выделения;
- *y* – координата *Y* левого верхнего угла прямоугольника выделения;
- *w* – ширина прямоугольника выделения;
- *h* – высота прямоугольника выделения.

Так как у линии две вершины, поэтому сначала определяется какая вершина левее и выше. Далее относительно нее вычисляется попадание. Фрагмент кода, где первое условие, если первая вершина выше и левее, иначе второе условие, где вторая вершина выше и левее, *line* – это СЛ, *x1* и *y1* – координаты первой вершины, *x2* и *y2* – координаты второй вершины:

```
if (line.x1 < line.x2 || line.y1 < line.y2) {
...
} else {
...
}
```

Если левая верхняя вершина больше левой верхней координаты прямоугольника выделения и правая нижняя координата меньше чем правая нижняя координата прямоугольника выделения, то значит линия находится внутри прямоугольника выделения и ее необходимо выделить. Фрагмент

кода вычисления попадания линии в прямоугольник выделения в первом условии, описанным выше, где x и y – координаты верхнего угла прямоугольника выделения, h и w – высота и ширина прямоугольника выделения соответственно, $line$ – это СЛ, $x1$ и $y1$ – координаты первой вершины, $x2$ и $y2$ – координаты второй вершины, $isSelect$ – флаг выделения СЛ:

```
if (x < line.x1
    && y < line.y1
    && w > line.x2
    && h > line.y2) {
    if (!isSelect) {setSelect(true);}
} else {setSelect(false);}
```

Для второго условия аналогичный алгоритм, только переставлены первая и вторая вершина местами.

setLocationLine –метод, предназначенный для установки позиции СЛ, при масштабировании или перемещении. Тип входного параметра – число с плавающей точкой, означающее коэффициент масштабирования относительно исходного размера. Возвращаемого значение нет. При вычислении позиции изменяются:

- координаты крайних точек СЛ;
- толщина СЛ;
- радиус круга выделения вершин.

Фрагмент кода, где $line$ – это СЛ, $x1$ и $y1$ – координаты первой вершины, $x2$ и $y2$ – координаты второй вершины, $zoomOld$ – текущий коэффициент масштабирования, $zoom$ – новый коэффициент масштабирования, $depthDefault$ – текущая толщина СЛ, $radius$ – радиус круга выделения:

```
line.x1 = line.x1 / zoomOld * zoom;
line.y1 = line.y1 / zoomOld * zoom;
line.x2 = line.x2 / zoomOld * zoom;
line.y2 = line.y2 / zoomOld * zoom;
```

```
depthDefault = (float) (depthDefault / zoomOld * zoom);
radius = (radius / zoomOld * zoom);
zoomOld = zoom;
```

mousePressed – переопределенный метод базового класса *MouseAdapter*, предназначенный для обработки событий нажатия кнопкой мыши на объект СЛ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение не имеет. Переопределение метода необходимо для обработки нажатия левой клавиши мыши, при котором выполняется:

- сохранение стартовой позиции, относительно которой вычисляется перетаскивание;
- проверка попадания указателя мыши в круг;
- установка постоянного и временного выделения СЛ;
- проверка нажатия клавиши *CTRL*, и если не нажата, то вызов функции сброса выделения у остальных объектов;
- перерисовка СЛ;
- сброс круга выделения, при нажатии на другие клавиши мыши.

Фрагмент кода, где *evt* – объект события *MouseEvent*, поле *BUTTON1* – код левой клавиши мыши, *temp* – объект класса *Point*, *circle* – код выделения вершин СЛ, *source* – родитель текущего класса:

```
if (evt.getButton() == MouseEvent.BUTTON1) {
    temp.setLocation(evt.getX(), evt.getY());
    circle = containsEquals(evt.getX(), evt.getY());
    setSelect(true); isNotSelect = true;
    source.getParent().getParent().repaint();

} else {
    circle = 0;
}
```

mouseReleased – переопределенный метод класса *MouseAdapter*, предназначенный для обработки событий отжатия кнопкой мыши с объекта СЛ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение

не имеет. Переопределение метода необходимо для обработки отжатия левой клавиши мыши, при котором выполняется:

- сброс стартовой позиции, относительно которой вычисляется перетаскивание;
- округление координат крайних точек линии, при их перемещении;
- сброс временного выделения;
- перерисовка СЛ;
- сброс круга выделения;
- добавление в список истории событий, события перемещения СЛ, с помощью функции *addStep*.

Если перемещалась первая вершина, то *circle* равен 1, если только вторая вершина, то *circle* равен 2, если перемещалась вся линия, т.е. обе вершины, то *circle* равен 3. Округление происходит, в функции *rounding* класса *GlobalVars*. Фрагмент кода, где *evt* – объект события *MouseEvent*, поле *BUTTON1* – код левой клавиши мыши, *temp* – объект класса *Point*, *circle* – код выделения вершин СЛ, *source* – родитель текущего класса, *isNotSelect* – переменная для временного выделения СЛ, *line* – СЛ, *x1* и *y1* – координаты первой вершины, *x2y2* – координаты второй вершины:

```
isNotSelect = false;
if (evt.getButton() == MouseEvent.BUTTON1) {
    if (circle == 1 || circle == 3) {
        line.x1 = rounding(line.getP1().getX(), 10, 0);
        line.y1 = rounding(line.getP1().getY(), 10, 0);
    }
    if (circle == 2 || circle == 3) {
        line.x2 = rounding(line.getP2().getX(), 10, 0);
        line.y2 = rounding(line.getP2().getY(), 10, 0);
    }
    temp.setLocation(0, 0);
    JComponent source = (JComponent) evt.getSource();
    source.getParent().getParent().repaint();
}
```

```
circle = 0;
addStep(2);
```

mouseDragged – переопределенный метод класса *MouseAdapter*, предназначенный для обработки событий перемещения с зажатой кнопкой мыши по объекту СЛ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение не имеет. Переопределение метода необходимо для реализации эффекта перетаскивания объекта левой клавиши мыши. Три варианта перетаскивания:

- при *circle* равном 1, только первой вершине устанавливается координаты указателя мыши;
- при *circle* равном 2, только второй вершине устанавливается координаты указателя мыши;
- при *circle* равном 3, обе крайние вершины перемещаются на разницу между курсором мыши и стартовой точкой перемещения, которая каждый раз устанавливается как позиция курсора, но уже после перемещения, что позволяет вычислить разницу между новой и предыдущей координатой мыши.

После изменения координат, вызывается перерисовка СЛ через родителя, для исключения наложения изображения СЛ. Фрагмент кода, где *evt* – объект события *MouseEvent*, *temp* – предыдущая позиция курсора мыши, *circle* – код выделения вершин СЛ, *source* – родитель текущего класса, *line* – это СЛ, *x1* и *y1* – координаты первой вершины, *x2* и *y2* – координаты второй вершины, *offsetX* и *offsetY* – координаты разницы по осям соответственно:

```
if (circle != 0) {
    switch (circle) {
        case 1: line.x1 = evt.getX(); line.y1 = evt.getY(); break;
        case 2: line.x2 = evt.getX(); line.y2 = evt.getY(); break;
        default:
            int offsetX = (int) temp.x - evt.getX();
            int offsetY = (int) temp.y - evt.getY();
            line.x1 -= offsetX; line.y1 -= offsetY;
```

```

        line.x2 -= offsetX; line.y2 -= offsetY;
        temp.x = evt.getX();
        temp.y = evt.getY();break; } }
source.getParent().getParent().repaint();

```

mouseMoved–переопределенный метод базового класса *MouseAdapter*, предназначенный для обработки событий наведения указателя мыши на объект СЛ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение не имеет. Переопределение метода необходимо для реализации эффекта подсвечивания вершин СЛ при наведении на них указателя мыши. Для подсвечивания, устанавливается значение в переменную *circle*, с помощью функции *containsEquals*, и вызывается перерисовка объекта СЛ через его родителя. Фрагмент кода, где *evt* – объект события *MouseEvent*:

```

circle = containsEquals(evt.getX(), evt.getY());
((JComponent) evt.getSource()).getParent().repaint();

```

addStep–метод, предназначенный для добавления события изменения СЛ, в качестве входных параметров принимает значение целочисленного типа, которое означает тип изменения, возвращаемого значение нет. Создается объект класса *SerializeConn*, в конструктор которого передается указатель на текущий объект, тип изменения объекта, и указатель на родителя. Фрагмент кода, где *listEdit* – список изменений графических объектов, *a* – это тип изменения, *vp* – указатель на класс *JLayeredPaneTest*:

```

listEdit.add(history);
history = new SerializeConn(this, a, vp);
listEdit.restore(history);

```

paintComponent –переопределенный метод базового класса *JComponent*, предназначенный для отображения объекта. В качестве входных параметров принимает объект класса *Graphics*, методами которого отображается и прорисовывается объект. Возвращаемого значение нет. Переопределение необходимо для отображения пользовательского объекта. Фон объекта устанавливается прозрачным и поверх рисуется линия по координатам. Чтобы нарисовать линию, создается объект класса *Graphics2D*,

устанавливается черный цвет линии, с помощью класса *Color* и его полям *BLACK*. Далее устанавливается толщина пера рисования, для этого создается объект класса *BasicStroke*, в конструктор которого передается толщина пера, тип линии и дополнительный параметр по настройке линии. Фрагмент кода, где *g* – объект класса *Graphics*, *line* – это объект класса *Line2D.double*, т.е. СЛ данного класса:

```
super.paintComponent(g);
Graphics2D g2 = (Graphics2D) g;
g2.setColor(Color.BLACK);    g2.setStroke(new BasicStroke(depth,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g2.draw(line);
```

1.4.7 Класс *SelectPanel*

SelectPanel – доступный класс, реализующий отображение графического элемента «Прямоугольник выделения» (ПВ) и его функционал. Расширяет базовый класс *JPanel*, не переопределяя его методы. Два поля данного класса:

- *startX* – переменная целочисленного типа *int*, равна начальной точки прямоугольника по оси *X*, относительно которой вычисляется размер прямоугольника и его положение.
- *startY* – переменная целочисленного типа *int*, равна начальной точки прямоугольника по оси *Y*, относительно которой вычисляется размер прямоугольника и его положение.

Ниже приведены доступные методы класса *SelectPanel*.

SelectPanel – конструктор класса, без входных параметров, устанавливающий полупрозрачный фон с цветом объекту *JPanel*. Фрагмент кода, где конструктор класса *Color* принимает 4 аргумента цветовой палитры *RGBA*:

```
setBackground(newColor(0, 100, 220, 100));
```

setStart – метод класса, устанавливающий координаты начальной точки угла и сбрасывает размер прямоугольника выделения. Метод принимает

объект класса *Point*, в качестве входного параметра, который и содержит координаты для прямоугольника выделения. Возвращаемых аргументов нет. Фрагмент кода, где *point* – это объекта класса *Point*, содержит координаты точки:

```
setLocation(point);
setSize(0, 0);
startX = point.x;
startY = point.y;
```

setEnd– метод класса, устанавливающий размеры и координаты прямоугольника выделения. Метод принимает координаты *x* и *y* перемещаемого угла, в качестве входных параметров. Возвращаемых аргументов нет. Если координаты фиксированной точки, совпадают с координатами перемещаемой точки прямоугольника выделения, то перерисовки фигуры не происходит. Фрагмент кода, где при совпадении всех координат выполнение программы в условии пропускается:

```
if (startX != x || startY != y) {
    ...
}
```

Если перемещаемый угол прямоугольника правее точки старта, значит простое увеличение размера прямоугольника. Левая верхняя вершина сохраняет координату *x*. Иначе, если левее, необходимо переместить прямоугольник в координату *x* и задать новую ширину. Фрагмент кода, где *width* – новая ширина прямоугольника выделения, *startX* – стартовая координата по оси *X*, *x* – координата по оси *X* перемещаемого угла прямоугольника выделения:

```
int width;
if (startX < x) {
    width = x - startX;
    x = startX;
} else { width = startX - x; }
```

Аналогично вычисляется высота и координата y . Далее устанавливаются вычисленные значения координат и размеров прямоугольник и вызов метода перерисовки объекта, фрагмент кода:

```
setBounds(x, y, width, height);
repaint();
```

1.4.8 Класс *TextBlock*

TextBlock – доступный класс, реализующий отображение графического элемента «Текстовый блок» (ТБ) и его функционал. Расширяет базовый класс *JTextArea*, переопределяя его методы, которые описаны ниже. Поля класса *TextBlock* приведены в Таблице 3.

Таблица 3 – Переменные класса *TextBlock*

Название	Тип	Описание
<i>flagDragged</i>	<i>boolean</i>	разрешает перемещение ТБ при перетаскивании курсора мыши. ЛЕ – разрешено, ЛН – запрещено.
<i>delta</i>	<i>Point</i>	Объект класса <i>Point</i> , используется при перетаскивании и хранящие разницу между установленными и новыми координатами объекта.
<i>stepVP</i>	<i>int</i>	содержит шаг сетки листа.
<i>isSelect</i>	<i>boolean</i>	показывает состояние выделения ТБ. ЛЕ – объект выделен, ЛН – объект не выделен.
<i>isNotSelect</i>	<i>boolean</i>	фиксирует выделение ТБ при однократном нажатии. ЛЕ – объект выделен нажатием и не сброшен. ЛН – объект не выделен или выделение сброшено.
<i>zoomOld</i>	<i>double</i>	Значение текущего масштаба, предназначено для восстановления исходного масштаба с

		последующим вычислением нового.
<i>history</i>	<i>SerializeText</i> <i>t</i>	Объекта класса, для фиксации последнего изменения ТБ.

Ниже приведены методы класса *TextBlock*.

TextBlock– конструктор класса, принимающий крайние координаты левого верхнего угла ТБ, исходный текст, шрифт и шаг сетки, в качестве входных параметров. Устанавливается текст, шрифт, координаты левого верхнего угла, минимальная ширина и высота, прозрачный цвет фона, сбрасывается выделение ТБ и для этого объекта. Фрагмент кода, где

```
setText(text);
setFont(new Font(font.getFamily(), font.getStyle(), 18));
setSize(getPreferredSize());
setLocation(x, y);
delta = new Point(0, 0);
setOpaque(false);
setBackground(new Color(255, 255, 255));
setVisible(true);
```

Так же добавляются слушатели мыши для реализации функционала:

- перемещение, с привязкой к пересечению сетки;
- масштабирования ТБ, изменение размера, координат, и размера шрифта текста;
- добавление новых и восстановление;
- изменения, выделение, удаление и добавление текста;
- удаление одного или нескольких ТБ;
- выделение одного или нескольких ТБ.

Фрагмент кода добавление слушателей событий нажатия клавиш и мыши:

```
addKeyListener(this);
addMouseListener(this);
addMouseMotionListener(this);
```

containsArea –метод, предназначенный для проверки СЛ, на попадание ее в прямоугольник выделения. Входные параметры метода: x и y – координаты левого верхнего угла прямоугольника выделения; w и h – ширина и высота прямоугольника выделения. Линия содержит две вершины, поэтому определяется вершина, которая левее и выше. Относительно этой координаты и вычисляется попадание в область. Фрагмент кода, где *line* – это СЛ, x и y – координаты левого верхнего угла прямоугольника выделения, *isSelect* – переменная, устанавливающая (*true*) или сбрасывающая (*false*) выделение ТБ:

```
if (x < getX() && y < getY()
    && w > getX() + getWidth()
    && h > getY() + getHeight()) {
    isSelect = true;
} else {
    isSelect = false;
}
```

setLocationTextBlock –метод, предназначенный для установки позиции ТБ, при масштабировании или перемещении. Тип входного параметра – число с плавающей точкой, означающее коэффициент масштабирования относительно исходного размера, и шаг сетки листа, для привязки и исключения произвольного смещения ТБ из-за погрешности вычисления коэффициента с плавающей точкой. Возвращаемого значения нет. При вычислении позиции изменяются:

- координата левого верхнего угла ТБ;
- размер шрифта текста ТБ;
- ширина и длина, которая устанавливается минимально допустимой, относительно текста в ТБ.

Фрагмент кода, где *step* – шаг сетки рабочей области, X и Y – новые координаты левого угла ТБ, *zoomOld* – текущий коэффициент

масштабирования, *zoom* – новый коэффициент масштабирования, *font* – текущая шрифт текста в ТБ:

```
int X=(getX() / zoomOld * zoom), Y=(getY() / zoomOld * zoom);
setLocation(rounding(X, step, 0),rounding(Y, step, 0));
Font font = getFont();
setFont(font.deriveFont((font.getSize() / zoomOld * zoom)));
setSize(getPreferredSize());
zoomOld = zoom;
```

keyTyped, *keyPressed*, *keyReleased* –
переопределенные методы базового класса *KeyListener*,
предназначенный для коррекции размера ТБ при редактировании текста.

Входные аргументы не используются, а возвращаемых значений у этих методов нет. Корректировка размера ТБ при редактировании текста осуществляется путем установки после каждого нажатия или отжатия кнопок клавиатуры нового размера ТБ, вычисляемого, как минимальный размер для данного текста с прибавлением небольшого смещения. Все три метода работают аналогично по фрагменту кода, приведенному ниже:

```
setSize(getPreferredSize().width + 50, getPreferredSize().height  
+ 50);
```

mousePressed – переопределенный метод базового класса *MouseAdapter*, предназначенный для обработки событий нажатия кнопкой мыши на объект ТБ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение не имеет. Переопределение метода необходимо для обработки нажатия левой клавиши мыши, при котором выполняется:

- установка постоянного и временного выделения ТБ;
- перерисовка ТБ;
- установка красного цвета тексту;
- разрешение на передвижение ТБ, перетаскиванием мыши;
- проверка клавиши *CTRL*, если нет нажатия, тогда вызов функции сброса выделения у остальных объектов.

Фрагмент кода, где *evt* – объект события *MouseEvent*, поле *BUTTON1* – код левой клавиши мыши, *isSelect* – переменная, показывающая постоянное выделение ТБ, *isNotSelect* – переменная для временного выделения ТБ, *source* – источник вызванного события, *flagDragged* – переменная разрешения перетаскивания ТБ:

```
JComponent source = (JComponent) evt.getSource();
if (evt.getButton() == MouseEvent.BUTTON1) {
    flagDragged = true;
    isNotSelect = true;
    isSelect = true;
    setForeground(Color.red);
    if (!evt.isControlDown()) {
        ... // вызов родителя, для сброса выделения
    }
} else {
    flagDragged = false;
}
```

mouseReleased–переопределенный метод класса *MouseAdapter*, предназначенный для обработки событий отжатия кнопкой мыши с объекта ТБ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение нет. Переопределение метода необходимо для обработки отжатия левой клавиши мыши, при котором выполняется:

- сброс стартовой позиции, относительно которой вычисляется перетаскивание;
- округление координаты левого верхнего угла и установка ТБ;
- сброс временного выделения;
- перерисовка ТБ;
- добавление в список истории событий, события перемещения ТБ, с помощью функции *addStep*.

Округление с помощью функции *rounding* класса *GlobalVars*.
Фрагмент кода, где *evt* – объект события *MouseEvent*, *isNotSelect* –

переменная для временного выделения ТБ, *flagDragged* – переменная разрешения перетаскивания ТБ:

```
isNotSelect = false;
flagDragged = false;
delta.setLocation(0, 0);
setLocation(rounding(getX(), stepVP, 0),
            rounding(getY(), stepVP, 0));
repaint(); addStep(2);
```

mouseDragged–переопределенный метод класса *MouseAdapter*, предназначенный для обработки событий перемещения с зажатой кнопкой мыши по объекту ТБ. Входной параметр – объект класса *MouseEvent*. Возвращаемого значение нет. Переопределение метода необходимо для реализации эффекта перетаскивания объекта левой клавиши мыши.

После изменения координат, вызывается перерисовка ТБ. Фрагмент кода, где *evt* – объект события *MouseEvent*, *source* – источник вызванного события, *p* – текущие координаты ТБ, *delta* – разница между текущей позицией ТБ и координатами указателя мыши, *flagDragged* – переменная разрешения перетаскивания ТБ:

```
if (flagDragged) {
    Point p = getLocation();
    if (delta.x == 0 && delta.y == 0) {
        delta.setLocation(evt.getPoint());
    }
    p.setLocation(p.x+evt.getX()-delta.x, p.y+evt.getY()-delta.y);
    setLocation(p);
}
```

paintComponent –переопределенный метод базового класса *JComponent*, предназначенный для отображения объекта ТБ. В качестве входных параметров принимает объект класса *Graphics*, методами которого отображается и прорисовывается объект. Возвращаемого значение нет. Переопределение необходимо для отображения пользовательского объекта с прозрачным фоном, а также выделение текста красным цветом, если объект

ТБ выделен. Фрагмент кода, где *g* – объект класса *Graphics*, *isSelected* – это флаг выделения:

```
super.paintComponent(g);
if (isSelected) {
    setForeground(Color.red);
} else {
    setForeground(Color.black);
}
Graphics2D g2d = (Graphics2D) g;
g2d.setRenderingHint(
    RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.S
RC_OVER, 0.0f));
```

Так как скрывать объект базового класса не нужно, то вначале вызывается метод его отображение, а уже после происходят модификации. Если объект ТБ выделен, то с помощью функции *setForeground* устанавливается черный или красный цвет текста ТБ. Далее установка с помощью функций класса *Graphics2D* палитры *RGBA*, которая и позволяет использовать полупрозрачность.

1.4.9 Класс Visio

Visio – класс, включающий основные компоненты графического редактора и их обработку. Обеспечивает взаимодействие с интерфейсом и с программно-логической моделью.

Таблица 4 – Переменные класса Visio

Название	Тип	Описание
<i>JTabbedPanel</i>	<i>javax.swing. JTabbedPane</i>	Объект класса <i>JTabbedPane</i> , реализующий панель вкладок.
<i>JTextField5</i>	<i>javax.swing. JTextField</i>	Объект класса <i>JTextField</i> , реализующий текстовое поле для ввода номера нового

		БМК.
<i>jPopupMenu-Block</i>	<i>JPopupMenu</i>	Объект класса <i>JPopupMenu</i> , реализующий контекстное меню для БМК.
<i>memForm</i>	<i>t3.Memory</i>	Объект класса <i>Memory</i> , для сохранения и загрузки данных в эмитируемую ОЗУ.
<i>spt</i>	<i>ArrayList</i> < <i>ScrollPane-Test</i> >	Массив рабочих областей. При работе со вкладками, сохранение или загрузки используется обход данного массива.
<i>ListBlock</i>	<i>LinkedList</i> < <i>Block</i> >	Связанный список БМК, один на все рабочие области, для исключения повторения блоков.
<i>jMenuReplace</i>	<i>javax.swing</i> <i>JMenu</i>	Контекстное меню для БМК, с операциями удаление, изменения и перемещения между рабочими областями.
<i>buttonGroup-Replace</i>	<i>javax.swing</i> <i>ButtonGroup</i>	Группа переключателей кнопок, для перемещения БМК между рабочими листа, так как БМК может находиться только на одной рабочей области.

Ниже приведены методы класса *Visio*.

Visio— конструктор класса, принимающий контейнер, панель вкладок, текстовое поле, ссылку на класс *Memory*, контекстные меню для вкладок и БМК, в качестве входных параметров. Инициализируются переданные параметры и список БМК, у контейнера убираются границы, устанавливается не прозрачный фон. Вызывается функция *initPopupMenu*, которая настраивает контекстные меню. Фрагмент кода создания вкладок по умолчанию:

```
for (int i = 0; i < 4; i++) {
    createTab(i);
}
jTabbedPane1.setSelectedIndex(0);
```

При переключении между вкладками, снимается выделение со всех объектов. Для этого переопределяется метод *stateChanged* слушателя *ChangeListener*. Фрагмент кода, где *index* – номер текущей активной вкладки, *spt* – массив рабочих областей, *JTabbedPane* – панель вкладок:

```
int index = 0;
@Override
public void stateChanged(javax.swing.event.ChangeEvent evt) {
    // Убрать выделения на листе, с которого переключились
    spt.get(index).vp.clearSelectAllObject();
    // Сохранения текущей активной вкладки
    index = jTabbedPane.getSelectedIndex();
}
```

При переключении вкладок вызовется метод *stateChanged*. В *index* храниться номер вкладки, с которой был переход. Рабочая область имеет тот же номер, что и вкладка. Получение рабочей области и вызов метода *clearSelectAllObject* для сброса выделения у всех объектов на этой области.

Обработка событий клавиатуры реализовано в этом классе, так как фокус может быть на любом компоненте рабочей области, а это не должно влиять на обработку событий клавиатуры. Для этого используется класс *KeyboardFocusManager*, который перехватывает все сгенерированные пользователем события. Метод *getCurrentKeyboardFocusManager* возвращает текущий менеджер фокуса клавиатуры, к которому и добавляется слушатель событий клавиатуры – *KeyEventDispatcher*. В этом слушателе переопределяется метод *dispatchKeyEvent*, в котором и будет обработка нажатий клавиатуры (см. Таблица 5).

Таблица 5 – Обрабатываемые клавиши рабочей области

Клавиша	Код	Описание
<i>Delete</i>	<i>KeyEvent.VK_DELETE</i>	Получение рабочей области и вызов метода удаление всех выделенных объектов.
<i>CTRL + A</i>	<i>KeyEvent</i>	Получение рабочей области и вызов метода

	<i>VK_A</i>	выделение всех объектов.
<i>Escape</i>	<i>KeyEvent.</i> <i>VK_ESCAPE</i>	Получение рабочей области и вызов метода снятия выделения со всех объектов.
<i>CTRL + Z</i>	<i>KeyEvent.</i> <i>VK_Z</i>	Отмена последнего действия с графическими объектами
<i>CTRL + Y</i>	<i>KeyEvent.</i> <i>VK_Y</i>	Повтора отменённого действия с графическими объектами

Если нажата другая клавиша или комбинация, то слушатель возвращает событие, если клавиша была обработана, то событие не возвращается.

initPopUpMenu—метод, предназначенный для инициализации контекстного меню на рабочей области. Входных и возвращаемых параметров не имеет. Создаются два контекстных пункта меню и одно вложенное меню совместно с группой кнопок. К каждому пункту меню добавляется слушатель *ActionListener*, который вызовет собственный переопределенный метод *actionPerformed* при нажатии. При нажатии на пункт «Изменить», в слушателе будет получена рабочая область, в которой вызовется метод *checkingAllBlock*. Этот метод проверяет выделенный БМК, и если в качестве входного аргумента логического типа было передана ЛЕ, то вызовется функция редактирования микрокоманды. При нажатии на пункт «Удалить» выполняются те же операции, что и для пункта «Изменить» с отличием в том, что в функцию *checkingAllBlock* передается ЛН, и вызова функции редактирования не происходит. После этого вызывается функция удаления выделенных объектов, а так как при срабатывании контекстного меню, выделяется только один объект, то происходит удаление именно его. Фрагмент кода, где *index* – номер выбранной рабочей области, *spt* – массив рабочих областей:

```
// Изменить
spt.get(index).vp.checkingAllBlock(true);
```

```
// Удалить
spt.get(index).vp.checkingAllBlock(false);
spt.get(index).vp.delete();
```

Вложенное меню добавлено с целью перемещения БМК между рабочими областями. Из-за того, что количество рабочих областей может изменяться, то появляется необходимость обновлять этот список во вложенном меню. Для этого добавлен слушатель вложенного меню *MenuListener*, у которого переопределен метод *menuSelected*. Этот метод вызовется при выборе вложенного пункта меню. Вызывается функция *checkingAllBlocks* параметром ЛН и возвращаемым объектом БМК. Далее устанавливается соответствующий номеру рабочей области у блока пункт вложенного контекстного меню. Фрагмент кода, где *index* – номер выбранной рабочей области:

```
@Override
public void menuSelected(javax.swing.event.MenuEvent evt) {
    // Получить индекс листа у блока
    Block blk = spt.get(index).vp.checkingAllBlock(false);
    // установить пункт соответствующего листа
    (jMenuReplace.getMenuComponents()[blk.indexList]).setSelected(1)
}
```

createTab – метод, предназначенный для:

- создания и настройка рабочей области;
- добавление ее в список рабочих областей;
- добавление в панель вкладок;
- настройка пользовательской вкладки;
- создание и добавление пункта, вложенного меню;
- добавление в группу переключателей;
- добавление слушателя к пункту меню;
- Обновление контекстного вложенного меню.

В качестве входных передается значение целочисленного типа, равное номеру вкладки и рабочей области. Размер области задается равным 1300

пикселей в ширину, и 925 пикселей в высоту, что является форматом А3. Шаг сетки рабочей области 20 пикселей. Фрагмент кода добавление на панель вкладок и массив рабочих областей, где *component* – созданная рабочая область:

```
spt.add(component);
jTabbedPane.addTab(name, component.scrollPane);
// Выбор последней добавленной вкладки
jTabbedPane.setSelectedIndex(jTabbedPane.getTabCount() - 1);
```

Для удобства работы со вкладками и создания интуитивно понятного интерфейса реализованы пользовательские вкладки. Класс *ButtonTabComponent* содержит текстовое поле и кнопку, для закрытия вкладки. Также добавляется контекстное меню для добавления, удаления и переименования вкладки. Подробнее описано в классе *ButtonTabComponent*. Фрагмент кода настройки пользовательской вкладки, где *jTabbedPane* – панель вкладок, *name* – имя новой вкладки, *jPopupMenuTabbed* – контекстное меню вкладки, *component* – рабочая область:

```
component.vp.indexList = jTabbedPane.getTabCount() - 1;
jTabbedPane.setTabComponentAt(jTabbedPane.getTabCount()-1, new
ButtonTabComponent(jTabbedPane, this, name, jPopupMenuTabbed));
```

Далее происходит добавление в перечень листов контекстного вложенного меню и добавление слушателя на этот пункт меню. Фрагмент кода создания пункта меню и установка текста:

```
JCheckBoxMenuItem jMenuItemReplace = new JCheckBoxMenuItem();
jMenuItemReplace.setText(name);
```

Фрагмент кода добавление слушателя, где *index* – номер выбранной рабочей области:

```
jMenuItemReplace.addActionListener(new ActionListener() {
@Override
public void actionPerformed(java.awt.event.ActionEvent evt) {
Component[] comp = jMenuItemReplace.getMenuComponents();
for (int i = 0; i < comp.length; i++) {
if (comp[i] == evt.getSource()) {
```

```
spt.get(index).vp.replaceBlock(i, spt.get(i).vp);
break;}}}});
```

Далее добавление пункта меню в группу переключателей, для выбор только одного пункта и обновление меню:

```
buttonGroupReplace.add(jMenuItemReplace);
updateIndexList();
```

updateIndexList – метод, обновляющий список рабочих областей во вложенном меню. Все рабочие области имеют сквозную нумерацию. При удалении области уменьшаются номера последующих рабочих областей на единицу. Фрагмент кода, где в цикле обходят весь список рабочих областей, и проверяют каждый номер области, если он не соответствует счетчику, то значит необходимо установить номер, равный значению счетчика:

```
for (int i = 0; i < spt.size(); i++) {
    int b = spt.get(i).vp.indexList;
    if (b != i) {
        buttonGroupReplace.remove(jMenuReplace.getItem(i));
        jMenuReplace.remove(i);
        for (Block blk : ListBlock)
            if (blk.indexList == b) {blk.indexList = i;}}
```

deleteTab– метод, удаляющий вкладку у панели вкладок, рабочую область из списка рабочих областей и все объекты на рабочей области и обновляющий контекстное вложенное меню. Перед удаление рабочей области, вызывается метод удаление всех объектов. Фрагмент кода, где *index* – номер текущей рабочей области, *component* – рабочая область:

```
int index = jTabbedPane1.getSelectedIndex();
if (index >= 0) {
    ScrollPaneTest component = spt.get(index);
    component.vp.deleteAllObject();
    jTabbedPane1.remove(index);
    spt.remove(index);
    updateIndexList(); }
```

добавляющий новый блок в рабочую область в список блоков. Устанавливает настройки блока по умолчанию, проверяет номера БМК на уникальность, отменяет создание копий БМК и подсчитывает их количество.

zoom – метод, реализующий масштабирование рабочей области. Вызывает у рабочей области методы, для преобразования размеров и координат и перерисовывает контейнер с панелью вкладок. Фрагмент кода, где *zoom* – переменная логического типа, определяющая направления изменения масштаба, *jPanel2* – контейнер с панелью вкладок:

```
if (flag) {  
    spt.get(jTabbedPane1.getSelectedIndex()).vp.addZoom();  
} else {  
    spt.get(jTabbedPane1.getSelectedIndex()).vp.subZoom();  
}  
jPanel2.repaint();
```

Вызов функции *zoom* с передачей параметров ЛЕ и ЛН происходит при нажатии кнопок масштабирования на панели управления, слушатели события клика вызовут методы *jButton2ActionPerformed* и *jButton3ActionPerformed* для увеличения и уменьшения соответственно.

resetPositionView – метод, возвращающее окно просмотра в исходную позицию. Для этого происходит получение активной вкладки с рабочей областью, установкой позиции «экрана» в координаты (0,0), вертикальной и горизонтальной прокрутки в нулевое положение. Фрагмент кода, где *s* – текущая рабочая область:

```
ScrollPaneTests = spt.get(jTabbedPane1.getSelectedIndex());  
s.viewport.setViewPosition(new Point(0, 0));  
s.scrollPane.setLocation(0, 0);  
s.scrollPane.getVerticalScrollBar().setValue(0);  
s.scrollPane.getHorizontalScrollBar().setValue(0);  
jPanel2.repaint();
```

saveProgrammPZU – метод, сохраняющий программу в память при переключении со вкладки графического редактора на любую другую.

```

Iterator<Block> iter = ListBlock.iterator();
while (iter.hasNext()) {
    Block b = iter.next();
    pzu[0][b.id] = b.m[1];
    pzu[1][b.id] = (b.m[6]*2048 + b.m[5] * 64 + b.m[3] * 4 +
    b.m[2]);
    pzu[2][b.id] = (b.m[13] * 16384 + b.m[10] * 2048 + b.m[9] * 256
    + b.m[8] * 64 + b.m[7] * 16 + b.m[4]);
    pzu[3][b.id]=(b.m[15]*256 + b.m[14] * 64 + b.m[12] * 8 +
    b.m[11]);
}

```

Save – метод, предназначенный для сохранения микропрограммы, памяти ОЗУ, графических объектов в файл. Формат файла *tact* был разработан для оптимального хранения всех необходимых данных (см. Рисунок 3).

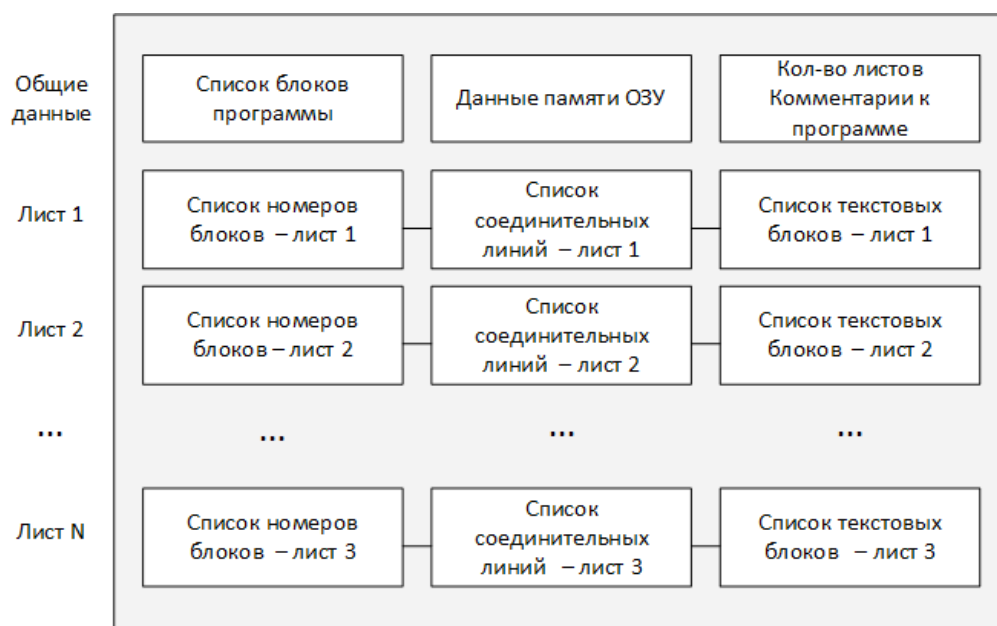


Рисунок 3 – Формат файла сохраненной программы

Фрагмент кода, где n – количество рабочих областей вместе с микропрограммой и памятью ОЗУ:

```
int n = spt.size() + 1;
Object[][] mas = new Object[n][3];
mas[0][0] = memForm.getDataTable();
mas[0][1] = pzu;
mas[0][2] = new Point(n, n);
```

Далее добавление в массив данных, которые получены с каждого рабочего листа. Нумерация в массиве начинается с 1, а в массиве рабочих областей с 0, так как по нулевому индексу записывается микропрограмма и память ОЗУ, поэтому необходимо вычитать 1 при обращении к массиву областей. Фрагменткода:

```
for (int i = 1; i < n; i++) {
    mas[i] = spt.get(i - 1).vp.getAllObject(); }

```

После формирования массива происходит сериализация – процесс преобразования массива данных в последовательность байтов, для последующего восстановления с сохранением структуры. За сериализацию отвечает класс *Serializer*. Метод *binarySerialization* сохраняет данные, переданные в качестве входных параметров в файл. Подробнее описано в

Serializer. Если данные успешно сохранены возвращается значение логического типа ЛЕ, если произошла ошибка, то ЛН или будет сгенерировано исключение, которое будет обработано в классе *iTact*. Фрагмент кода, где *mas* – данные для записи в файл, *n* – количество строк, *filePath* – путь и имя сохраняемого файла:

```
if (new Serializer().binarySerialization(mas, n, 3, filePath)) {
    System.out.println("Сохранено");
} else {
    System.out.println("Ошибка при сохранения данных");
}
```

Load – метод, предназначенный для загрузки микропрограммы, памяти ОЗУ, графических объектов из файла. Десериализации – создание массива данных из байтовой последовательности. Фрагмент кода, где *n* – количество рабочих областей вместе с микропрограммой и памятью ОЗУ:

```
Object[][] mas = newSerializer().binaryDeserialization(n, 3,
filePath);
```

Если произошла ошибка при десериализации, то возвращается *NULL* или генерируется исключение, которое обрабатывается в *iTact*. Если не *NULL*, то восстановление данных памяти ОЗУ и микропрограммы. Фрагмент кода, где *mas* – массива загруженных данных, *memForm* – объект класса для работы с ОЗУ, *pzu* – массив микропрограммы:

```
if (mas != null) {
    memForm.setDataTable((int[]) mas[0][0]);
    pzu = (short[][] mas[0][1];
    for (int i = 1; i < n; i++) {
        spt.get(i - 1).vp.addAllObject(mas[i]);
    }
    System.out.println("Загружено");
} else {
    System.out.println("Ошибка при чтении данных");
}
```

В цикле обход всех рабочих областей с вызовом функции – *addAllObject*, которая восстанавливает графические объекты из массива данных.

Generate – метод, генерирующий БМК по считанной микропрограмме. Упрощает переход от предыдущих версий программы к текущей. Фрагмент кода, где перед генерацией удаляются все блоки со всех рабочих областей:

```
for (ScrollPaneTest s : spt) {  
    s.vp.deleteAllObject(); }  

```

Сгенерированные блоки будут расставляться матрице, размеры которой задаются в переменной *maxList* типа *Point*, а счетчик прохода по этой матрице – переменная *curList* типа *Point*. Установка значения счетчика блоков и максимальное количество блоков в строке и колонке в следующем фрагменте кода:

```
int countList = 0;  
Point maxList = new Point(5, 6);  
Point curList = new Point(0, 0);  

```

Далее обход всей программы и выделение не нулевых микрокоманд. Переменная *check* устанавливает выделение только не пустых микрокоманд. Условия цикла *for*: индекс меньше размеру списка номер БМК; номер микрокоманды меньше 4096. В цикле извлекается очередная микрокоманда, с помощью функции *Extract()*, и записывается в массив *m*. Далее номер микрокоманды добавляется в список номеров микрокоманд, где проверяется, пустая она или нет. Если микрокоманда не пустая, добавляется новый БМК, иначе пропуск. Фрагмент реализации:

```
short[] m = new short[16];  
boolean check = true;  
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 = Extract(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)) {
            // код расположения новых БМК
        }
    }
}

```

Если микрокоманда удовлетворяет условию, то происходит создание графического объекта. Расположение вычисляется относительно матрицы *curList*. Фрагмент кода:

```

JLayeredPaneTestjp = spt.get(countList).vp;
int x = curList.x * jp.widthBlock + curList.x * 2 *
jp.poleSizeCur + 2 * jp.poleSizeCur;
int y = curList.y * jp.heightBlock + curList.y * 2 *
jp.poleSizeCur + 2 * jp.poleSizeCur;
jp.addBlock(n, new Point(x, y), false, m);
curList.x += 1;
if (curList.x == maxList.x) {
    curList.x = 0;
    curList.y += 1;
    if (curList.y == maxList.y) {
        countList += 1;
        curList.x = 0;
        curList.y = 0;
    }
}
}

```

1.4.10 Класс *ExportPDF*

ExportPDF – класс, реализующий сохранение графической представление программы в файл в формате *PDF* и *PNG*. Данный класс не имеет полей и содержит только одну функцию.

printImageToPDF– метод, предназначенный для сохранения рабочих областей вместе с графическими элементами в файл в формате *PDF* и *PNG*. Для работы с форматом PDF используются классы библиотеки *iTextpdf*:

- *Document* – для работы с документами;
- *PdfWriter* – для записи документа в формате PDF;
- *Image* – для преобразования размеров изображения, а также сохранения в формате PNG.

1.5 Описание способа организации пользовательского интерфейса

Микрокоманда в памяти имеет уникальный адрес, поэтому на всех листах каждый БМК должен иметь уникальный адрес.

Окно графического редактора представлено на рисунке 1.



Рисунок 1 – Графический редактор

В верхней части расположена рабочая область в которой располагаются блоки. Рабочая область имеет 4 листа, обычно используется не более 2-х. С помощью зажатой правой кнопкой мыши происходит перемещение по рабочему листу. Комбинация *CTRL* + колесико мыши масштабирует рабочую область.

Окно редактирования МК блока представлено на рисунке 2.

Рисунок 2 – Окно редактирования блока

Под рабочей областью располагается меню инструментов в котором осуществляется выбор тип добавляемого элемента (информационный блок или визуальная соединительная линия). Задается адрес добавляемого нового блока. Кнопки перемещения по рабочей области, сброса в начало документа.

В режиме «Блок» реализована возможность удаления, добавления, изменения блоков микрокоманды.

В режиме «Линии» реализована возможность добавления, удаления перемещения соединительных линий. Перемещать можно как целую линию, так и отдельную ее вершину. При наведении вершины линии на блок, она автоматически установиться на границу блока. С помощью зажатой клавиши CTRL можно выбрать несколько линий. Выделенная линия, отмечается желтыми кружками на концах и увеличением толщины (см. рисунок 3).

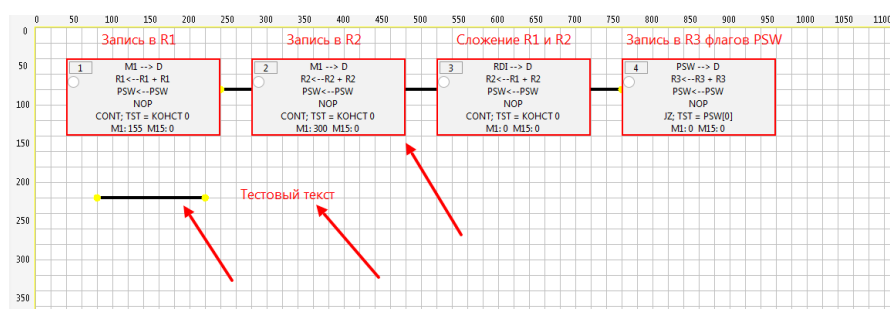


Рисунок 3 – Пример выделенной СЛ

Пример нескольких выделенных линий представлен на рисунке 4.

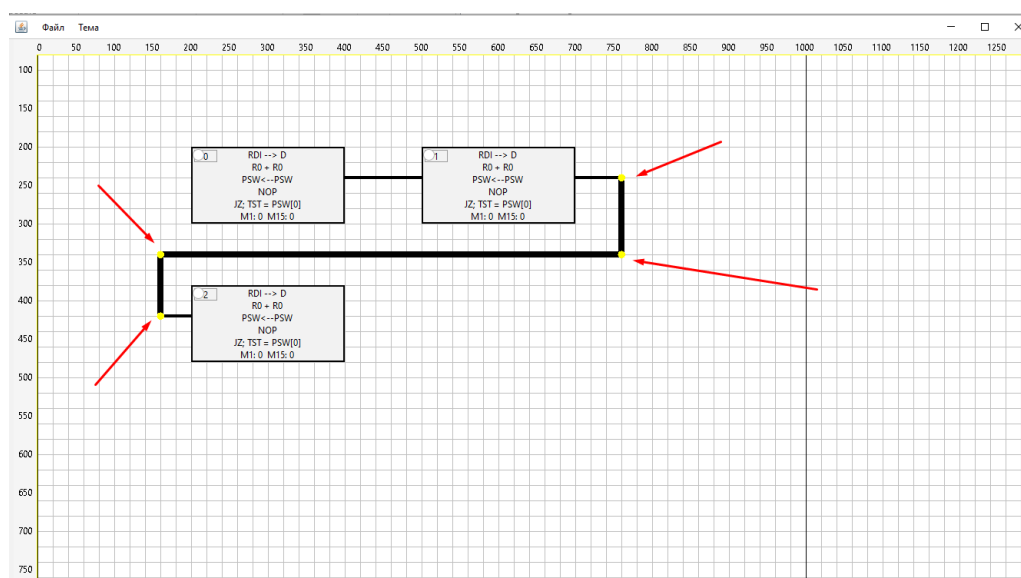


Рисунок 4 – Несколько выделенных линий

По нажатию на кнопку «Удалить» выбранные линии удаляются (см. рисунок 5).

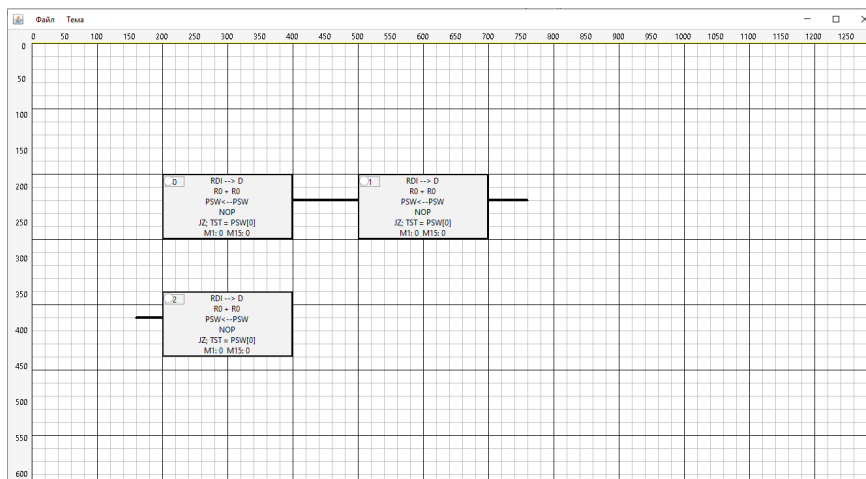


Рисунок 5 – Линии удалились

Реализована возможность создавать текстовые блоки. Нажимая кнопку «Добавить текст» в рабочее пространство добавляется поле с текстом по умолчанию (см. рисунок 6).

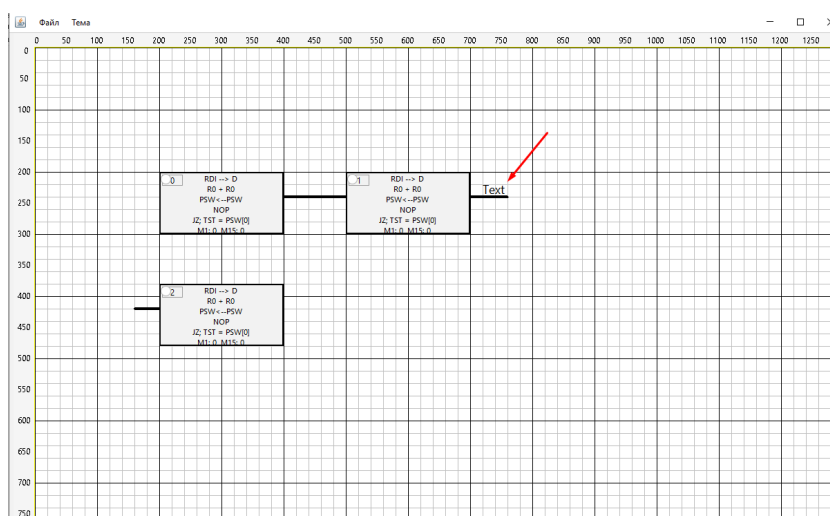


Рисунок 6 – Текстовый блок с установленным по умолчанию текстом

Чтобы изменить текст необходимо выделить его в блоке и написать необходимый текст. Пример представлен на рисунке 7.

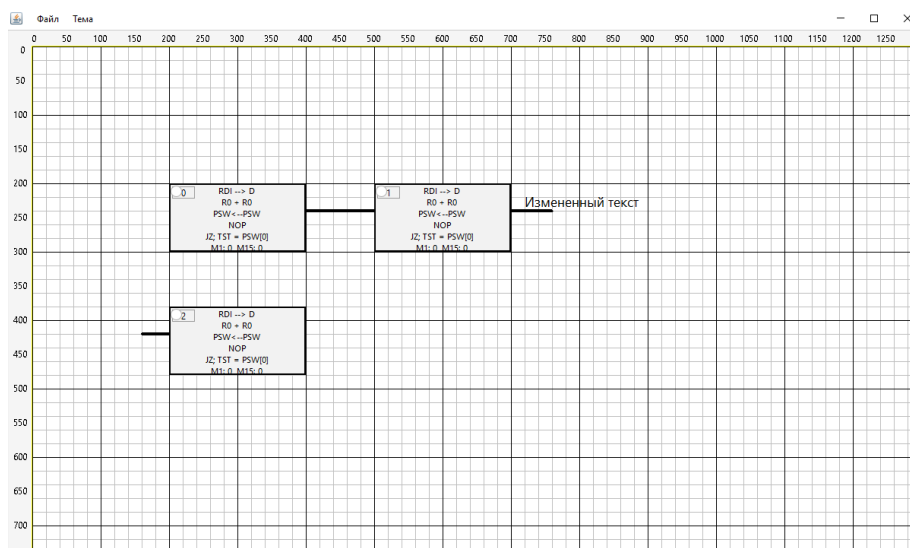


Рисунок 7 – Пользовательский текст

Если есть необходимость выделить несколько текстовых блоков, то следует зажать клавишу *CTRL*. Выделенные блоки подсвечиваются красным цветом (см. рисунок 8).

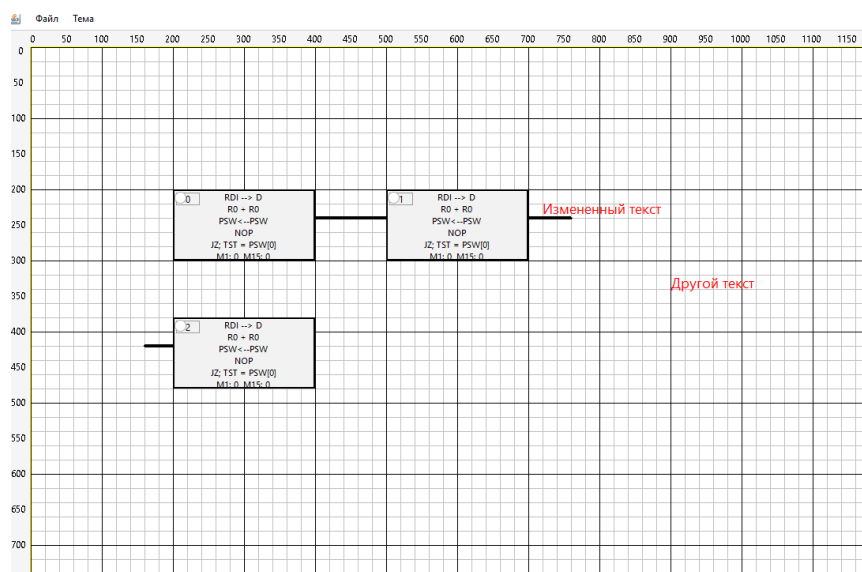


Рисунок 8 – Несколько выбранных текстовых блоков

На рисунке 9 представлено удаление выбранных текстовых блоков.

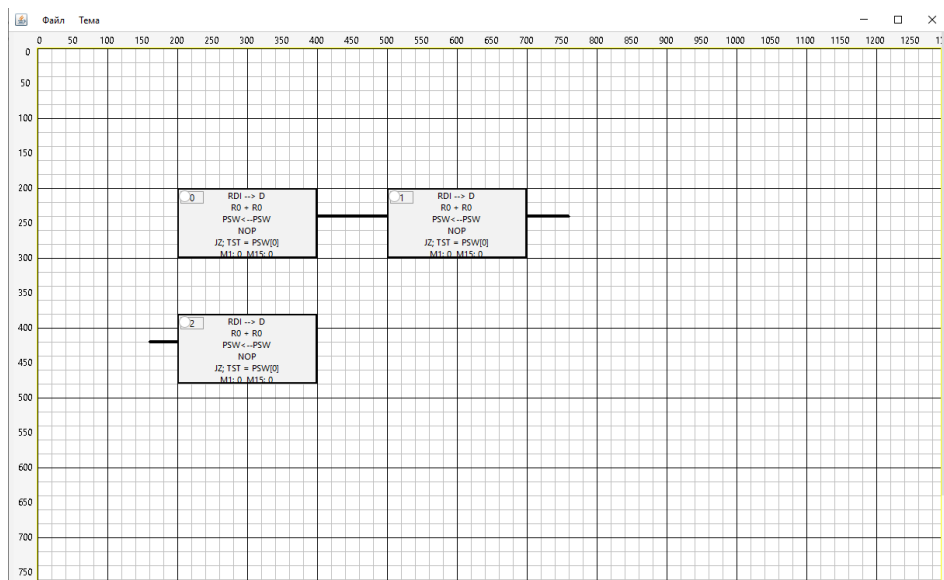


Рисунок 9 – Удаленные текстовые блоки

Одним из требований преподавателя в отчете – предоставление ДМЛ диаграммы реализованной программы, поэтому есть функция экспорта в файл формата *PDF* и *PNG*. При сохранении сохраняются все рабочие листы программы в формате A3. Пример *PDF* файла представлен на рисунке 10, *PNG* – рисунок 11.

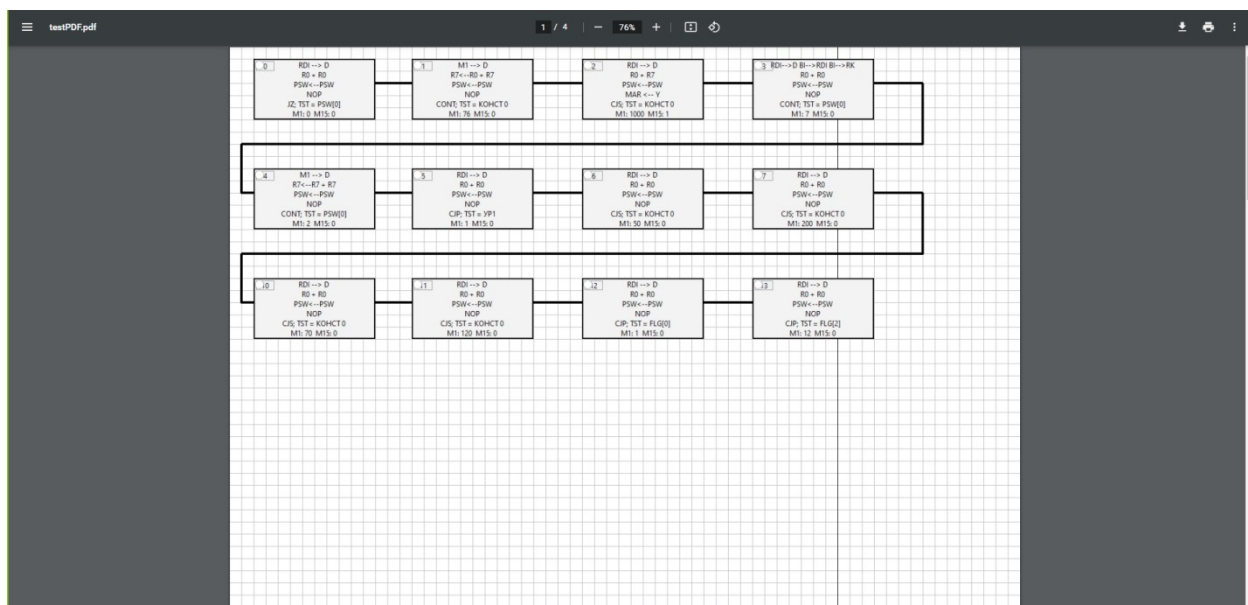


Рисунок 10 – ДМЛ диаграмма в формате *PDF*

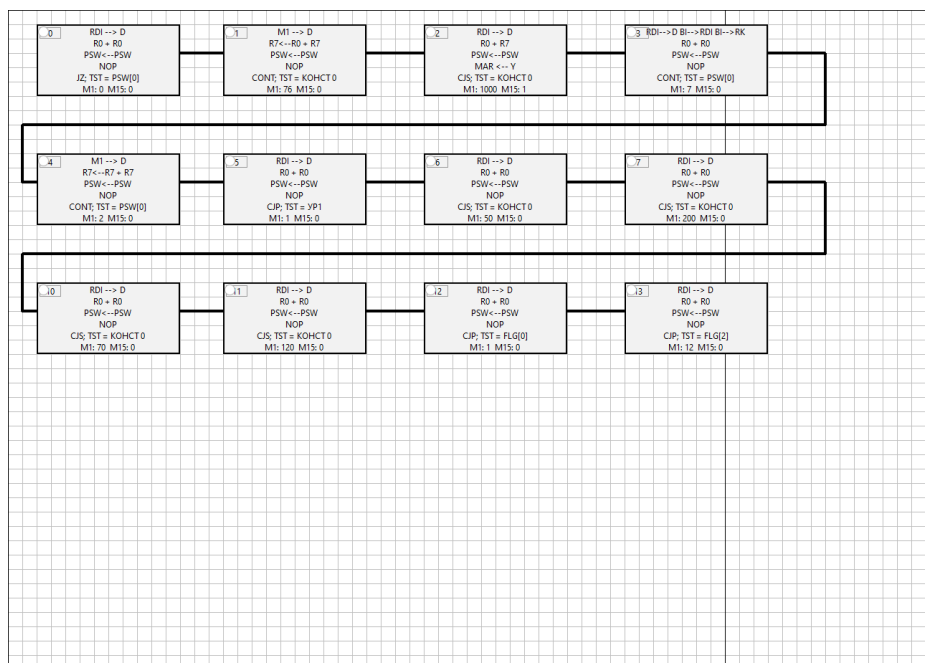


Рисунок 11 – ДМЛ диаграмма в формате *PNG*

После перехода из «Графического редактора» в другую вкладку происходит автоматическое сохранение программы в память, после чего ее можно отлаживать. Данная программа имеет свидетельства о регистрации [5,6].

1.6 Тестирование программы

Чтобы доказать работоспособность продукта, необходимо проверить возможные варианты использования:

- создание программы, БМК, СЛ, ТБ;
- изменение БМК;
- сохранение и загрузка программы;
- перемещение графических объектов на другие листы;
- выделение и удаление программы графических объектов;
- экспорт схемы программы в *PDF* и *PNG*;
- отмена и возврат действий с графическими объектами;
- импорт программ, предыдущих версий и генерация схемы.
- перемещение и масштабирование рабочего листа;

- создание и удаление рабочего листа.

Тестирование будет производиться под ОС *Windows 7* и *10*.

1.6.1 Создание программы

При открытии приложения, выбрана вкладка «Графический редактор» и первый рабочий лист (см. Рисунок 4). Отсутствие программы можно проверить во вкладке «Листинг» (см. Рисунок 5), где можно наблюдать значение 0 во всех полях микрокоманд. Далее, создание четырех БМК с адресами 1–4 (см. Рисунок 6).

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 4 – Главное окно редактора

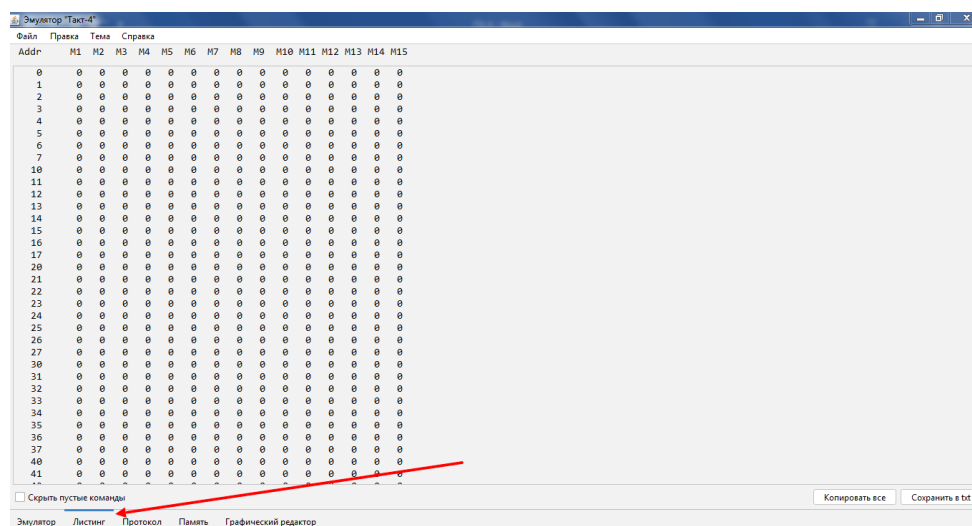


Рисунок 5 – Проверка отсутствия программы

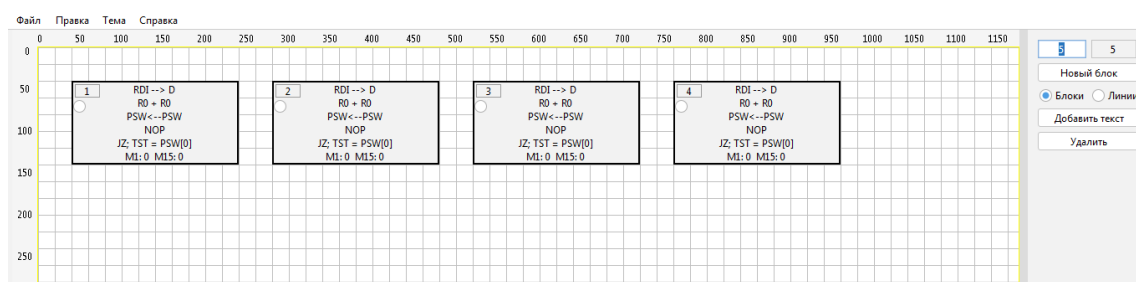


Рисунок 6 – Создание БМК

Далее, соединение БМК линиями связи и добавление текстовых полей с комментариями (см. Рисунок 7).

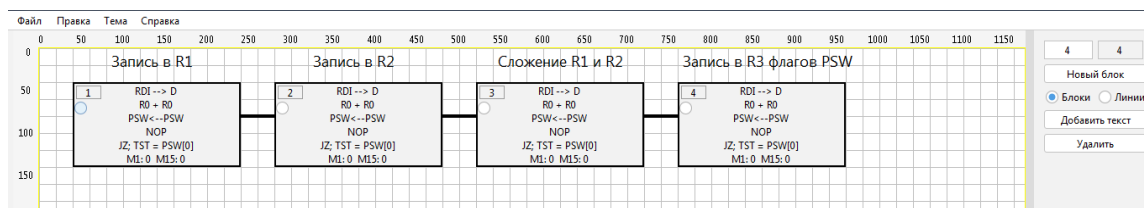


Рисунок 7 – Добавление СЛ и ТБ

1.6.2 Изменение БМК

Микрокоманда с первым адресом будет заносить значение из кода M1 в регистр *R1*. Занесение в любой регистр из поля M1 осуществляется через шину *D* и настраивается кодом M12 (см. Рисунок 8).

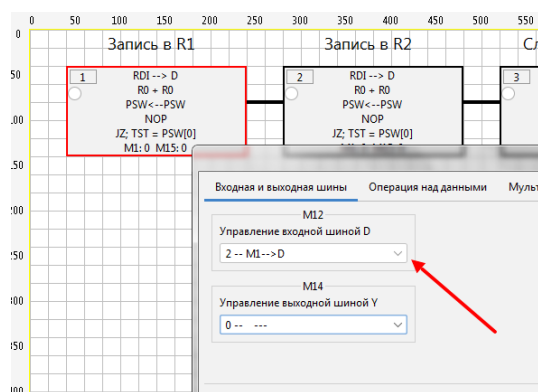


Рисунок 8 – Загрузка кода M1 на шину *D*

Далее, чтобы занести значение из шины *D* в регистр, необходимо их указать в качестве операндов *R* и *S* и сложить без бита сдвига, настраивается полями M8, M9 и M10 (см. Рисунок 9).

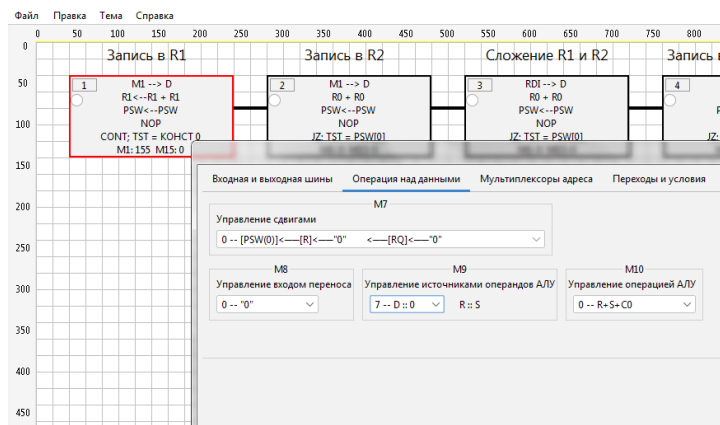


Рисунок 9 – Выбор операндов и их сложение

Так как эмулируемый процессор имеет 16 регистров, то необходимо выбрать один из них. В данном случае выбран первый регистр в полях M5 и M6, так как произойдет сложение с регистром R1 и результат будет записан в него же (см. Рисунок 10).

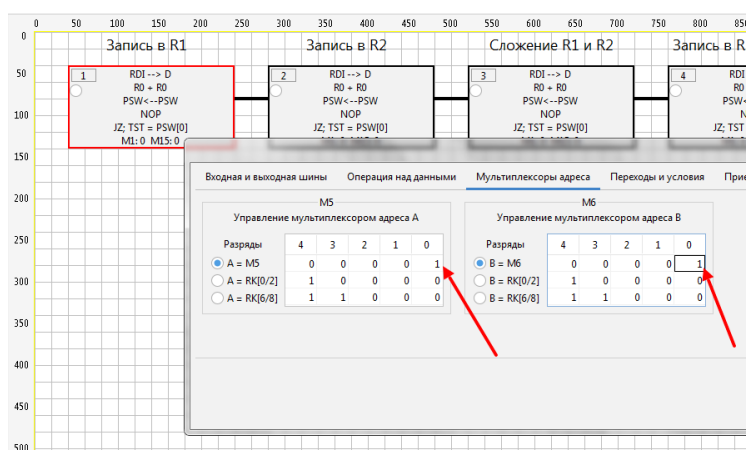


Рисунок 10 – Выбор регистров источника и приемника

После выполнения операция управление передается на следующую команду. За условия перехода, маскирование и сигналы отвечают полям M2 и M3 (см. Рисунок 11).

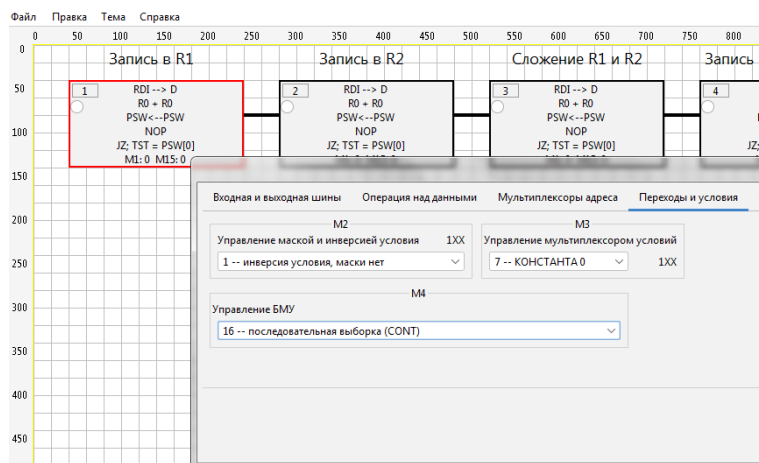


Рисунок 11 – Выбор адреса следующей команды и условия перехода

Так как на шину D подается код M1, то необходимо его задать значение. В поля констант установлено произвольное число 155 (см. Рисунок 12).

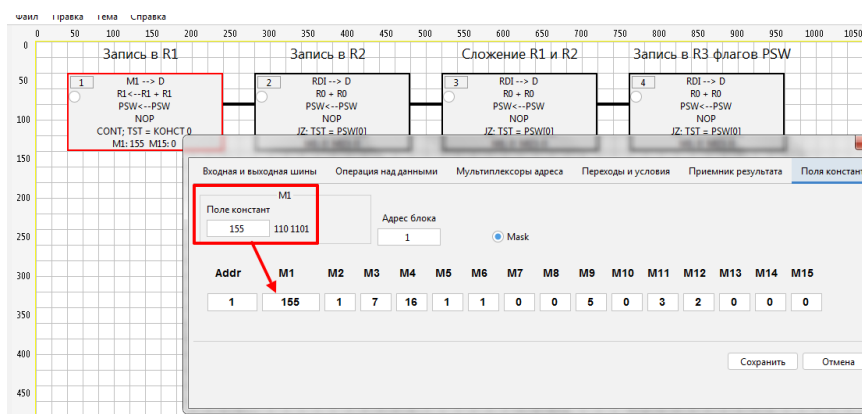


Рисунок 12 – Занесение в поле M1 значение для регистра

При закрытии окна редактирования микрокоманда сохраняется автоматически. Для проверки создания микрокоманды необходимо перейти во вкладку «Листинг» и по адресу 1 можно определить результат изменения (см. Рисунок 13).

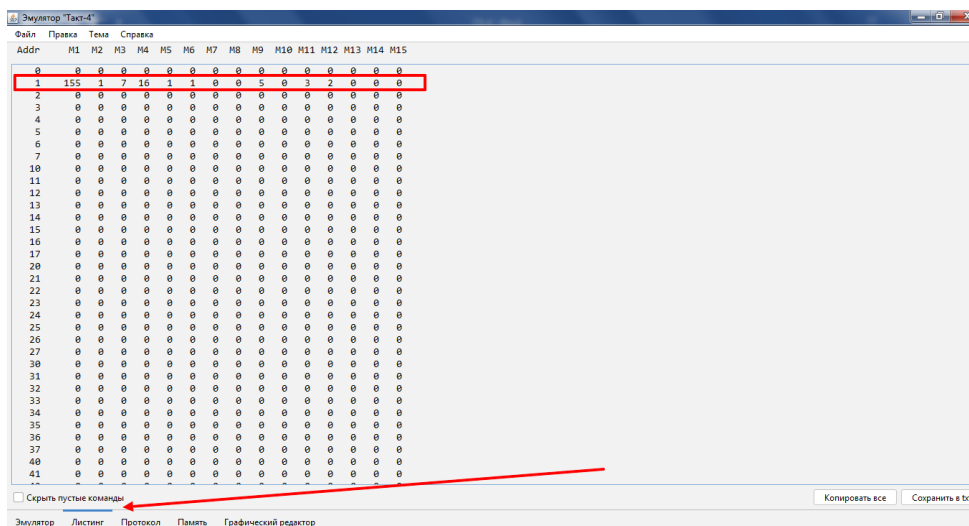


Рисунок 13 – Измененная микрокоманда

Изменение микрокоманд с адресами 2, 3 и 4 происходят аналогичным образом. В БМК с адресом 3 складываются два регистра $R1$ и $R2$, а с 4 адресом PSW заносится на шину D , которая складывается с 0 и заносится в регистр $R3$ (см. Рисунок 14). После выполнения команды с адресом управление будет передано на 0 адрес, который является адресом сброса или завершения выполнения программы.

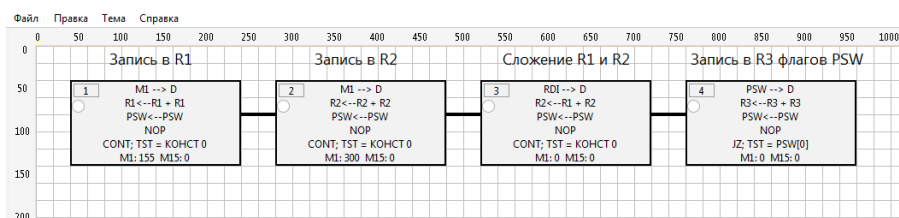


Рисунок 14 – Программа сложения двух чисел

Для проверки корректного сохранения программы используется вкладка «Листинг», в которой можно наблюдать строки под номерами 1–4 с непустыми значениями (см. Рисунок 15). Это означает, что изменение команд работает корректно.

Файл	Правка	Тема	Справка												
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	155	1	7	16	1	1	0	7	0	3	2	0	0	0	0
2	300	1	7	16	2	2	0	0	7	0	3	2	0	0	0
3	0	1	7	16	1	2	0	0	1	0	3	0	0	0	0
4	0	0	0	0	3	3	0	0	7	0	3	3	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	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
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	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
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

☐ Скрыть пустые ячейки

Экзидиру

Детские

Детские

Детские

Графический редактор

Копировать все

Сохранить в файл

Рисунок 15 – Программа сложения двух чисел во вкладке «Листинг»

1.6.3 Сохранение и загрузка программы

Сохраняемые данные:

- графические объекты;
- память программы;
- память ОЗУ.

Для тестирования необходимо заполнить эмулируемое ОЗУ произвольными значениями (см. Рисунок 16).

Файл	Правка	Тема	Справка					
Addr	0	2	4	6	10	12	14	16
0	<input type="text" value="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	0
100	4230	650	10	3213	3210	11110	0	0
120	0	0	0	0	0	0	0	0
140	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0
200	3210	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0
260	77770	6550	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	214	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	20	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
560	0	0	0	0	0	0	0	0
600	0	0	0	0	0	0	0	0
620	0	0	0	0	0	0	0	0
640	0	0	0	0	0	0	0	0
660	0	0	0	0	0	0	0	0

Адрес ячейки:

Очистить

Загрузить

Сохранить

Экспорт

Дисконт

Полтора

Паметь

Графический редактор

Рисунок 16 – Окно ОЗУ с некоторыми данными

Для того, чтобы сохранить программу с графическими объектами и всю схему в целом в файл, необходимо выбрать соответствующие пункты меню (см. Рисунок 17).

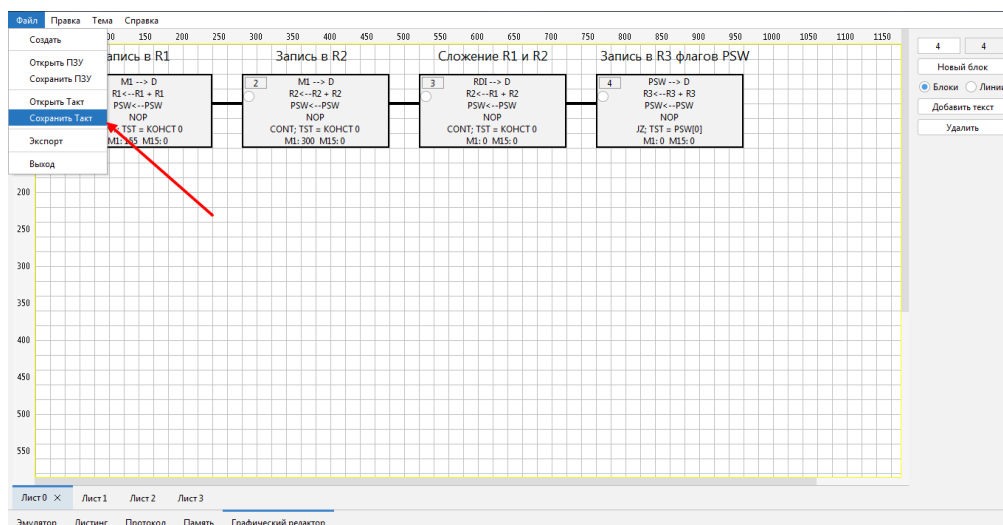


Рисунок 17 – Сохранение схемы в файл в формате *tact*

Далее в проводнике выбирается файл, в который будет сохранена схема (см. Рисунок 18). После успешного сохранения появляется окно подтверждения (см. Рисунок 19).

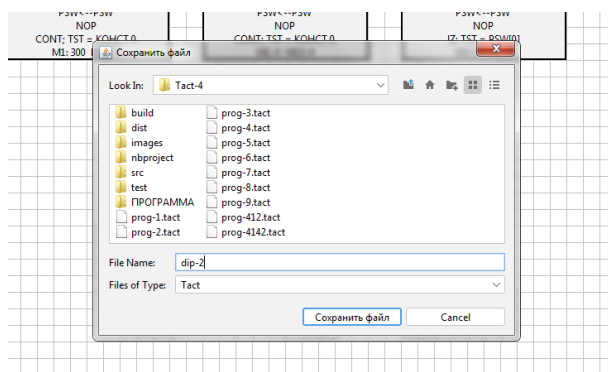


Рисунок 18 – Выбор файла сохранения

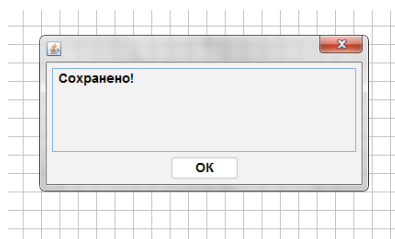


Рисунок 19 – Окно сообщения об успешном сохранении

После сохранения приложение было закрыто и запущено повторно, для полного удаления всех данных о предыдущей микропрограмме (см. Рисунок 20). Комментарий написан для проверки удаления его перед загрузкой микропрограммы.

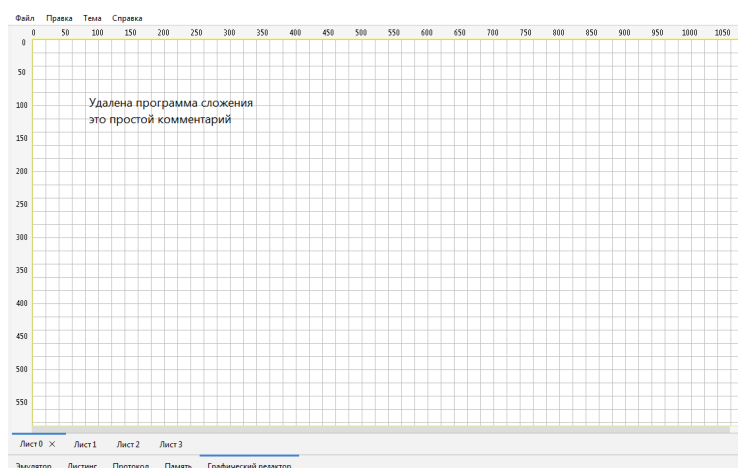


Рисунок 20 – Удаленная микропрограмма

Также в окне «Листинг» отсутствует микропрограмма (см. Рисунок 21).

Файл	Правка	Тема	Справка																
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
14	0	0	0	0	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				
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

☐ Скрывать пустые команды

Эмулятор Листинг Протокол Память Графический редактор

Рисунок 21 – Пустой листинг микропрограммы

Для того, чтобы открыть программу с графическими объектами и всю схему в целом в файл, необходимо выбрать соответствующие пункты меню (см. Рисунок 22).

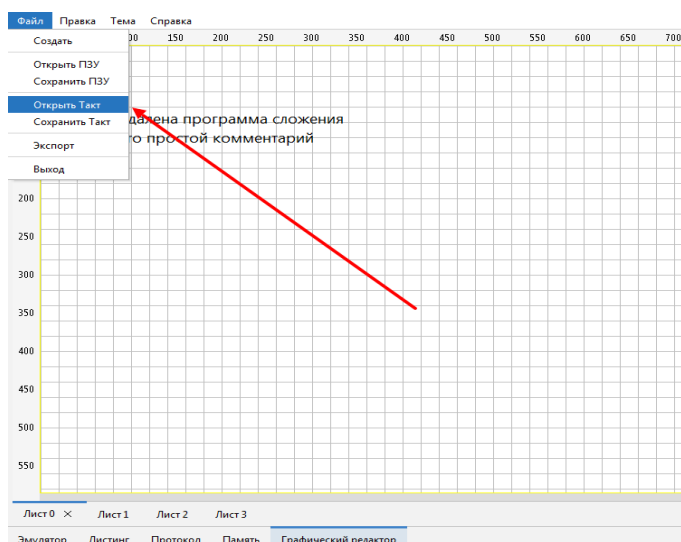


Рисунок 22 – Открытие файла с программой

Далее необходимо выбрать файл, ранее сохраненный (см. Рисунок 23).

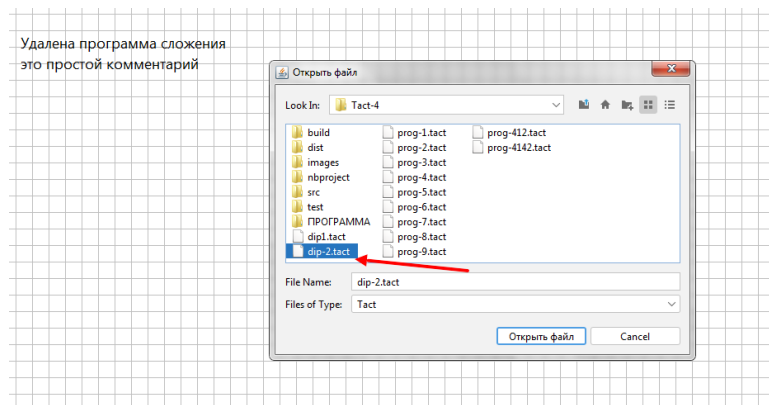


Рисунок 23 – Выбор файла загрузки

После успешной загрузки программы появляется окно подтверждения (см. Рисунок 24).

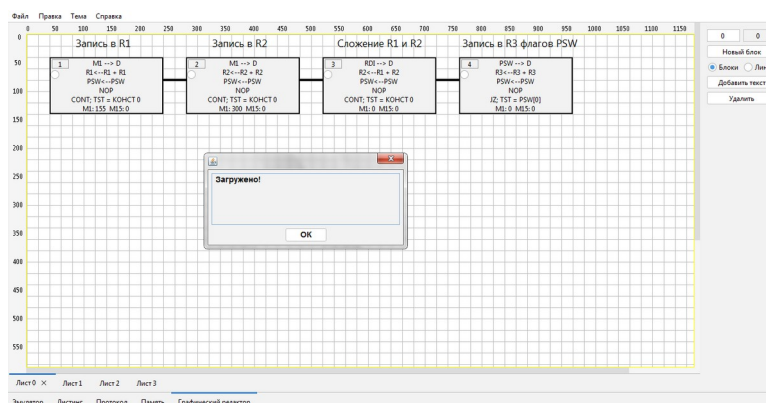


Рисунок 24 – Сообщение об успешной загрузке программы

Для проверки корректной загрузки микропрограммы необходимо обратиться во вкладку «Листинг», где на 1–4 строчка можно наблюдать загруженную микропрограмму, составленную в предыдущих пунктах (см. Рисунок 25). На вкладке «Память» можно проверить корректность загрузки данных ОЗУ (см. Рисунок 26). Исходя из тестов данного пункта можно сделать вывод, что сохранение и загрузка работают корректно.

Файл	Правка	Тема	Справка												
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	155	1	7	16	1	1	0	0	7	0	3	2	0	0	0
2	300	1	7	16	2	2	0	0	7	0	3	2	0	0	0
3	0	1	7	16	1	2	0	0	1	0	3	0	0	0	0
4	0	0	0	0	3	3	0	0	7	0	3	3	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	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
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

☐ Скрыть пустые команды

ЭмуляторЛистингПротоколПамятьГрафический редактор

Рисунок 25 – Проверка программы в окне «Листинг»

Файл

Правка

Тема

Справка

Addr	0	2	4	6	10	12	14	16
0								
20								
40								
60								
100	4230	650	10	3213	3210	11110		
120								
140								
160								
200	3210							
220								
240								
260	77770	6550						
300								
320								
340				214				
360								
400								
420								
440			20					
460								
500								
520								
540								
560								
600								
620								
640								
660								

Адрес ячейки: 0

Очистить

Загрузить

Сохранить

Эмулятор

Листинг

Протокол

Память

Графический редактор

Рисунок 26 – Загруженные данные ОЗУ

1.6.4 Перемещение между листами

Для тестирования перемещения созданы 4 новых БМК с адресами 4,5,6,7 (см. Рисунок 27).

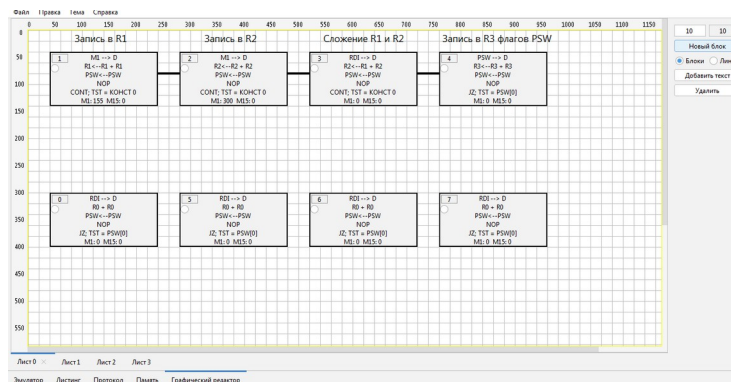


Рисунок 27 – Новые БМК для переноса на другие листы

Для переноса с вкладки «Лист 0» на вкладку «Лист 2» необходимо выбрать соответствующие пункты вложенного меню (см. Рисунок 28).

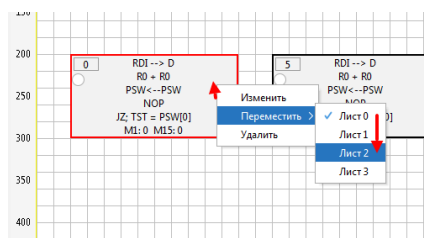


Рисунок 28 – Выбор листа для перемещения

После нажатия, БМК на вкладке «Лист 0» пропадает и появляется на вкладке «Лист 2» (см. Рисунок 30). Аналогичная операция производится с БМК с адресом 5 (см. Рисунок 29). БМК на вкладке «Лист 0» пропадает и появляется на вкладке «Лист 1» (см. Рисунок 31). БМК с адресом 7 перенес на вкладку «Лист 2» (см. Рисунок 32).

Изм.	Лист	№ докум.	Подпись	Дата

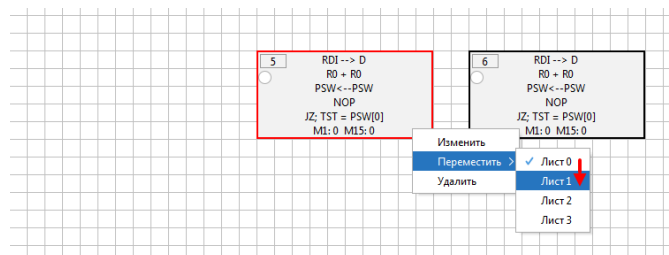


Рисунок 29 – Выбор листа для БМК с адресом 5

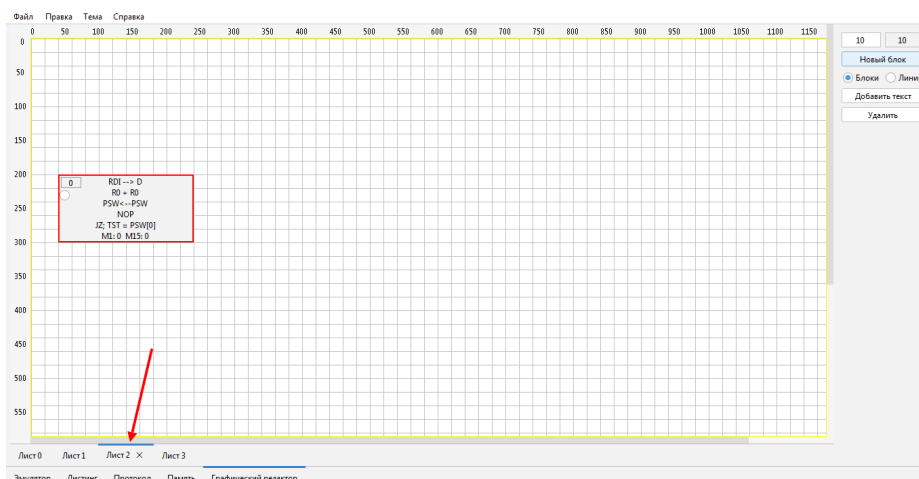


Рисунок 30 – Результат переноса БМК с адресом 4

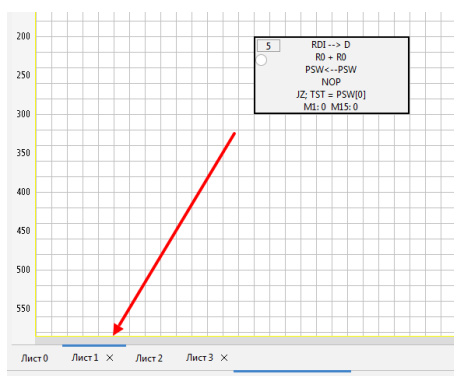


Рисунок 31 – Результат переноса БМК с адресом 5

Для перемещения нескольких БМК одновременно на один рабочий лист необходимо выделить эти блоки ЛКМ с зажатой клавишей *CTRL* и выбрать соответствующий пункт вложенного меню (см. Рисунок 33).

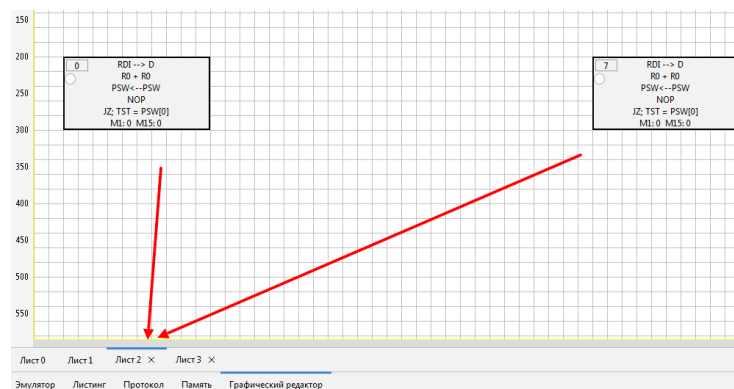


Рисунок 32 – Перенос БМК с адресом 7

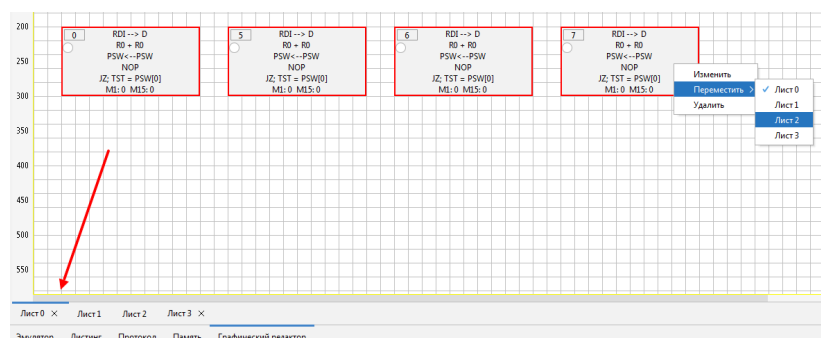


Рисунок 33 – Перемещение нескольких БМК

После нажатия, БМК на данной вкладке удаляются и появляются на вкладке «Лист 2» (см. Рисунок 34). Исходя из данных результатов, можно сделать вывод, что перемещение БМК работает корректно.

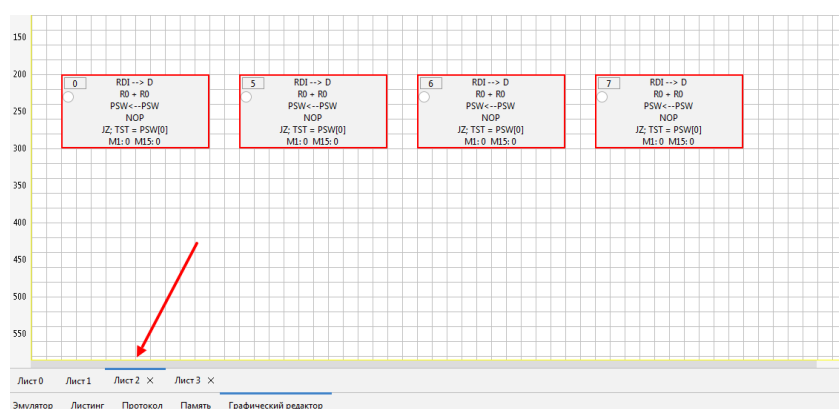


Рисунок 34 – Перемещенные одновременно БМК на лист

1.6.5 Выделение и удаление программы графических объектов

Для проверки выделения и удаления будут использован БМК созданные в предыдущем пункте. Выделение БМК с 5 и 6 адресом подсвечиваются (см. Рисунок 35). Далее выделение всех БМК, необходимых для удаления, и выбор соответствующего пункта контекстного меню (см. Рисунок 36). После нажатия, выделенные БМК пропадают (см. Рисунок 37).

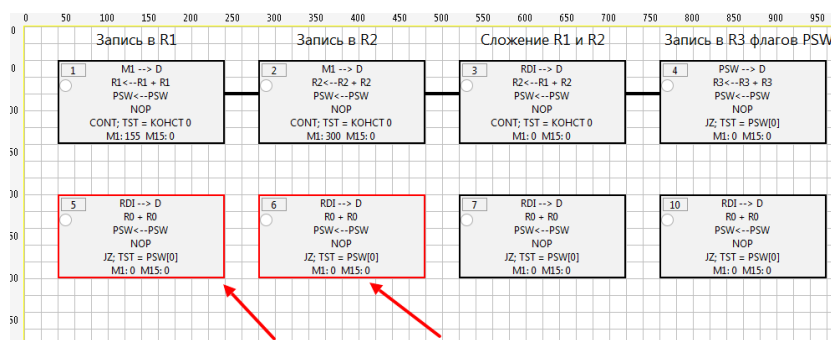


Рисунок 35 – Выделение БМК с адресами 5 и 6

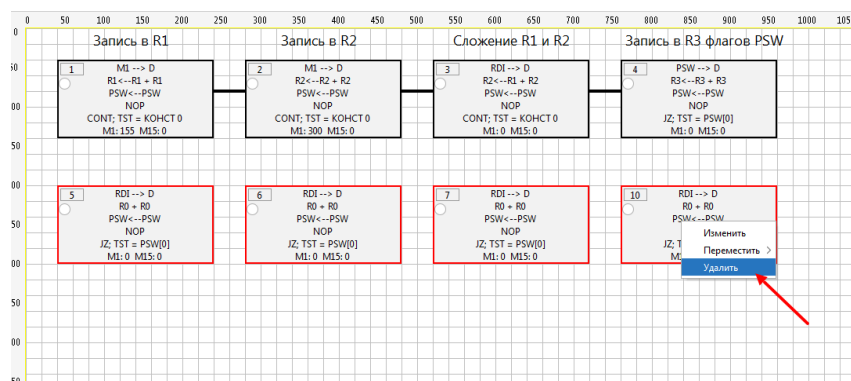


Рисунок 36 – Выделение всех БМК для удаления

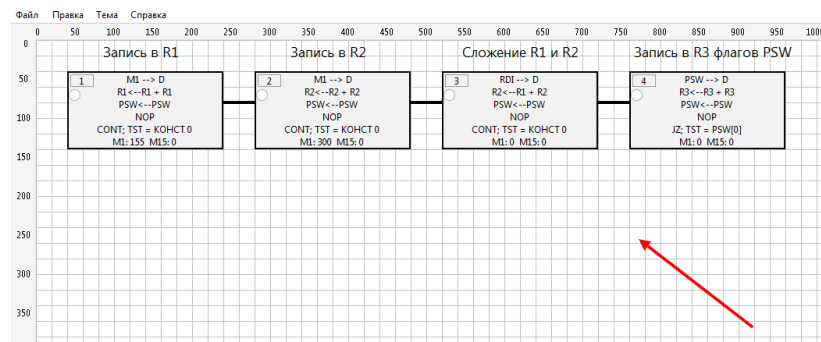


Рисунок 37 – Результат удаления БМК

Выделение всех объектов производится комбинацией *CTRL* + *A*. Для проверки выделения СЛ и ТБ добавлены дополнительные объекты (см. Рисунок 38). После нажатия клавиши *DELETE* все графические объекты удаляются (см. Рисунок 40). Для проверки корректности удаления БМК, необходимо перейти во вкладку «Листинг».

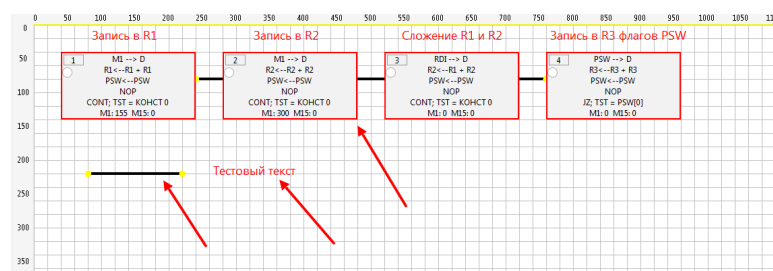


Рисунок 38 – Выделение всех объектов

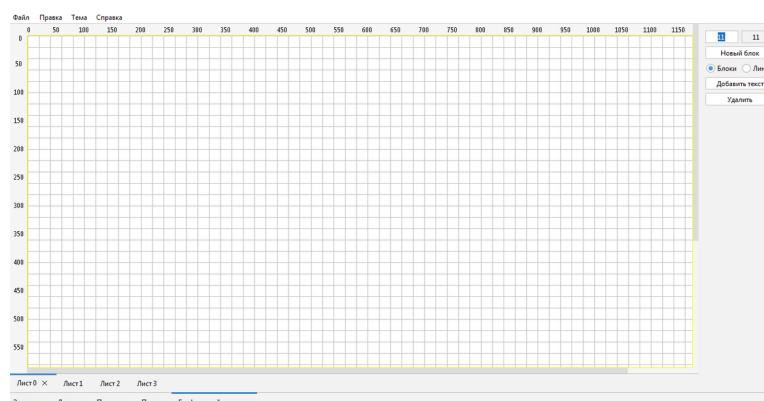


Рисунок 39 – Результат удаления всех объектов

Файл	Правка	Тема	Справка																
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
14	0	0	0	0	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				
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

Рисунок 40 – Пустые микрокоманды в листинге

1.6.6 Экспорт схемы программы

Для проверки экспорта программы в *PDF* и *PNG* программа из предыдущих пунктов была дополнена произвольными блоками на разных рабочих листах. Для экспорта необходимо нажать соответствующий пункт вложенного меню (см. Рисунок 41). После нажатия, необходимо выбрать папку для экспорта в окне диалога (см. Рисунок 42). После выбора папки происходит сохранение, и при успешном результате выводится подтверждение (см. Рисунок 43). Далее с помощью проводника можно проверить создание всех файлов (см. Рисунок 44). Создалось 4 файла *PNG* и один файл *PDF* с четырьмя страницами.

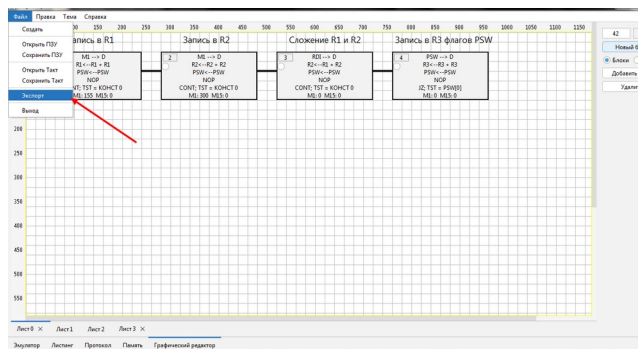


Рисунок 41 – Экспорт программы

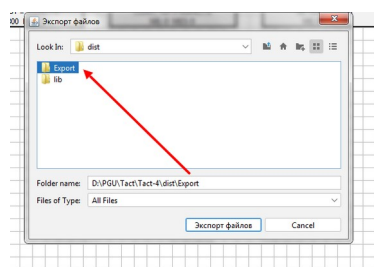


Рисунок 42 – Выбор папки сохранения

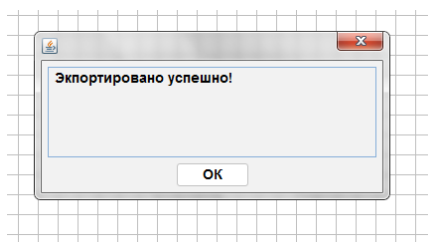


Рисунок 43 – Подтверждение успешного экспорта в файлы

Созданные файлы приведены в Приложение 3. Исходя из тестов в данном пункте, можно сделать вывод, что формирование отчетов и схемы программы работает корректно и удовлетворяет заданию.

	PDF	pdf	1 792 138
	image0	png	29 854
	image1	png	27 586
	image2	png	29 194
	image3	png	45 962

Рисунок 44 – Созданные файлы папке

1.6.7 Отмена и возврат действий с объектами

Для проверки отмены и возврата действий создан БМК, который перемещен два раза и удален. Создание БМК (см. Рисунок 45). Перемещение вправо БМК на координаты 300 по оси X и 40 по оси Y (см. Рисунок 46). Второе перемещение вправо БМК на координаты 540 по оси X и 40 по оси Y (см. Рисунок 47).



Рисунок 45 – Создание тестового БМК

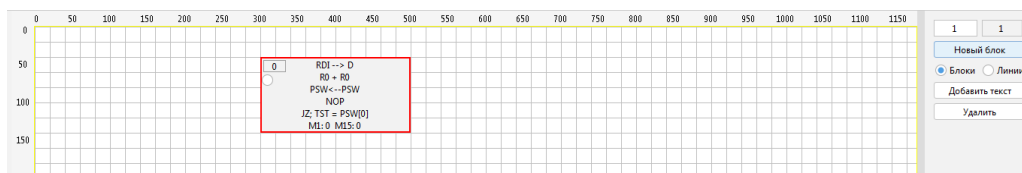


Рисунок 46 – Первое перемещение БМК вправо

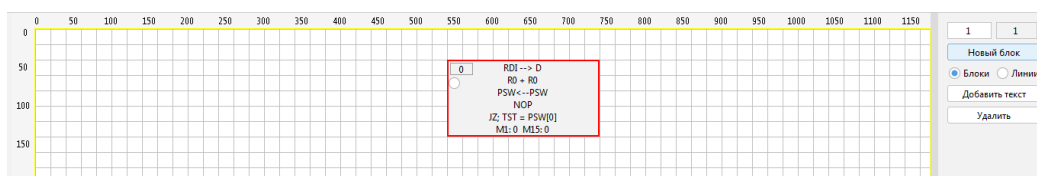


Рисунок 47 – Второе перемещение БМК вправо

И последний шаг в данном тесте – это удаление БМК по клавише *DELETE* (см. Рисунок 48).

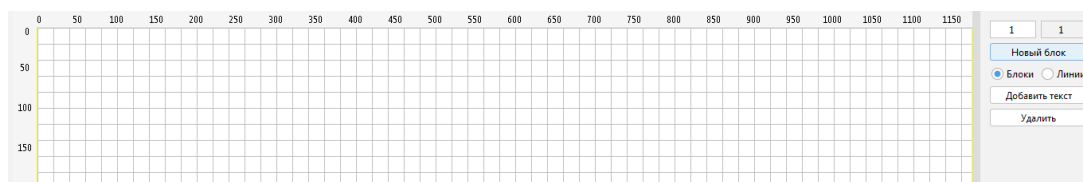


Рисунок 48 – Удаление БМК

Отмена действия производится нажатием комбинации клавиш *CTRL + Z*, после чего БМК восстановится на последнюю позицию (см. Рисунок 49). Следующая отмена действия – это отмена второго перемещения (см. Рисунок 50).

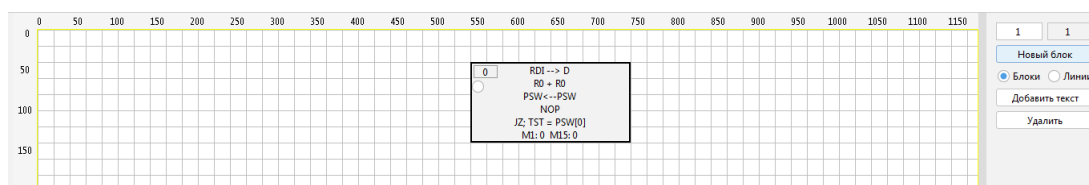


Рисунок 49 – Отмена удаления БМК

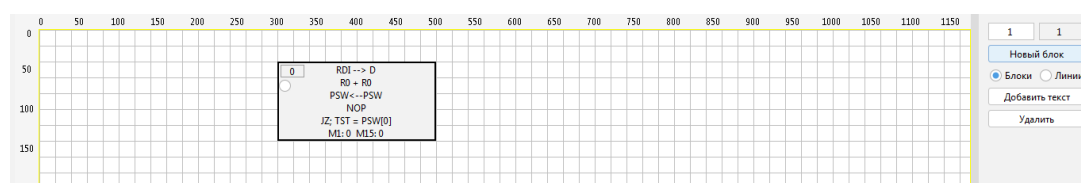


Рисунок 50 – Отмена второго перемещения

Аналогично в обратном порядке будет происходить отмена первого перемещения (см. Рисунок 51) и создания БМК (см. Рисунок 52).

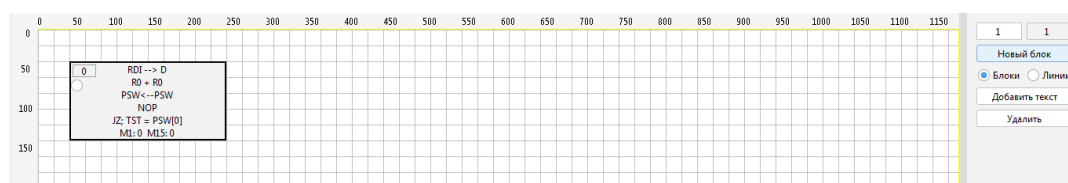


Рисунок 51 – Отмена первого перемещения

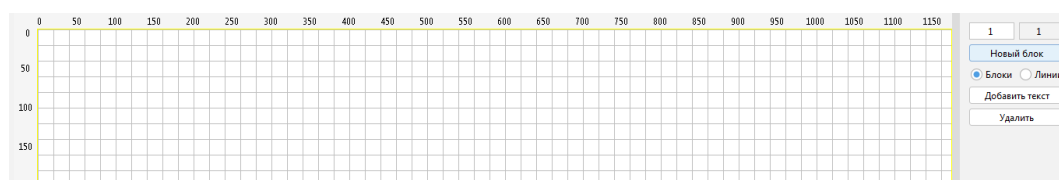


Рисунок 52 – Отмена создания БМК

Чтобы повторить действия, которые были отменены, нажимается комбинация *CTRL + Y*, добавление повтор двух перемещений(см. Рисунок 54 – Рисунок 56).

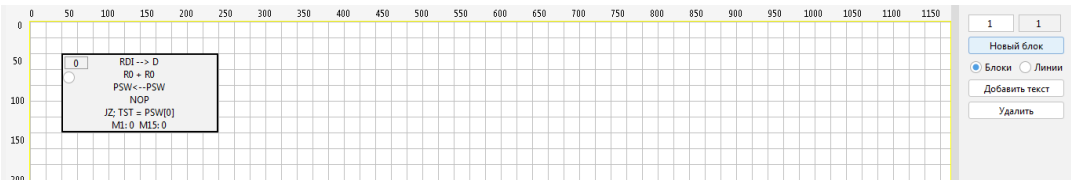


Рисунок 53 – Повтор добавления БМК

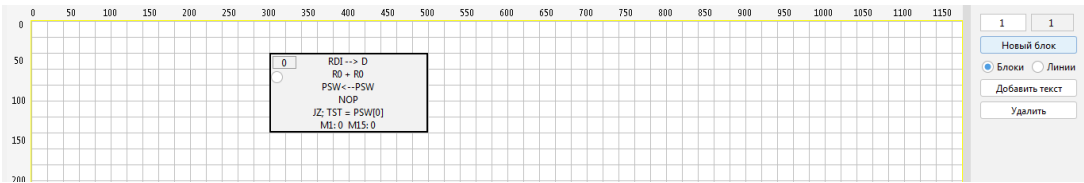


Рисунок 54 – Повтор первого перемещения

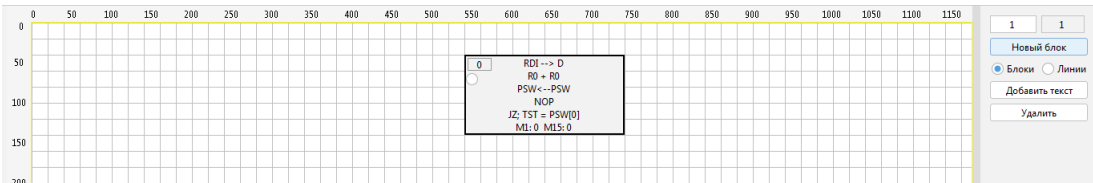


Рисунок 55 – Повтор второго перемещения

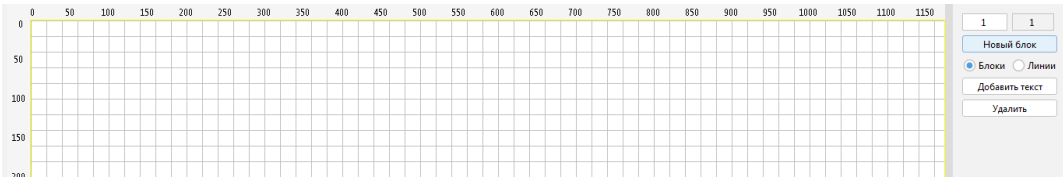


Рисунок 56 – Повтор удаления БМК

Отмена и возврат для нескольких БМК, СЛ, ТБ и их сочетаний работают аналогичным образом. Исходя из тестов, сделанных в данном пункте, можно сделать вывод что операция отмены и повтора действий для графических объектов работает корректно, что соответствует заданию.

1.6.8 Импорт программ и генерация схемы

[illegible]

Diagram showing 10 blocks of assembly code:

- Block 37:**

```

RDI -> D
RDX[R6] = RDX[R6]
PSW <- PSW
NOP
JZ TST = PSW[0]
MLI 37 MLI:0

```
- Block 100:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP1
MLI 100 MLI:0

```
- Block 101:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP2.1
MLI 101 MLI:0

```
- Block 102:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP2.2
MLI 102 MLI:0

```
- Block 103:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP3
MLI 103 MLI:0

```
- Block 104:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP4
MLI 104 MLI:0

```
- Block 105:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = VP5
MLI 105 MLI:0

```
- Block 106:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = RDX[9]
MLI 106 MLI:0

```
- Block 107:**

```

RDI -> D
R0 = R0
PSW <- PSW
NOP
JZ TST = RDX[3]
MLI 107 MLI:0

```
- Block 110:**

```

RDI -> D
RDI = RDI + RDI
PSW <- PSW
NOP
RDC TST = RLI[0]
MLI 110 MLI:0

```
- Block 111:**

```

RDI -> D
RDI = RDI - RDI - 1
PSW <- PSW
NOP
RDC TST = RLI[0]
MLI 111 MLI:0

```

Во вкладке «Листинг» корректно отображаются коды полей тестовой микропрограммы (см. Рисунок 59). Исходя из теста с загрузкой файла предыдущих версий, можно сделать вывод, что есть возможность переводить старые форматы в новый и работать с ними, что соответствует заданию.

Файл	Правка	Тема	Справка															
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	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
4	4	0	4	4	4	4	0	0	4	4	4	4	0	0	0			
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
10	10	0	10	10	10	10	0	0	0	0	0	0	0	0	0			
11	11	0	11	11	11	11	0	0	0	0	0	0	0	0	0			
12	12	0	12	12	12	12	0	0	0	0	0	0	0	0	0			
13	13	0	13	13	13	13	0	0	0	0	0	0	0	0	0			
14	14	0	14	14	14	14	0	0	0	0	0	0	0	0	0			
15	15	0	15	15	15	15	0	0	0	0	0	0	0	0	0			
16	16	0	16	16	16	16	0	0	0	0	0	0	0	0	0			
17	17	0	17	17	17	17	0	0	0	0	0	0	0	0	0			
20	20	0	0	0	20	20	0	0	0	0	0	0	0	0	0			
21	21	0	0	0	21	21	0	0	0	0	0	0	0	0	0			
22	22	0	0	0	22	22	0	0	0	0	0	0	0	0	0			
23	23	0	0	0	23	23	0	0	0	0	0	0	0	0	0			
24	24	0	0	0	24	24	0	0	0	0	0	0	0	0	0			
25	25	0	0	0	25	25	0	0	0	0	0	0	0	0	0			
26	26	0	0	0	26	26	0	0	0	0	0	0	0	0	0			
27	27	0	0	0	27	27	0	0	0	0	0	0	0	0	0			
30	30	0	0	0	30	30	0	0	0	0	0	0	0	0	0			
31	31	0	0	0	31	31	0	0	0	0	0	0	0	0	0			
32	32	0	0	0	32	32	0	0	0	0	0	0	0	0	0			
33	33	0	0	0	33	33	0	0	0	0	0	0	0	0	0			
34	34	0	0	0	34	34	0	0	0	0	0	0	0	0	0			
35	35	0	0	0	35	35	0	0	0	0	0	0	0	0	0			
36	36	0	0	0	36	36	0	0	0	0	0	0	0	0	0			
37	37	0	0	0	37	37	0	0	0	0	0	0	0	0	0			
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Рисунок 59 – Листинг загруженной программы

1.6.9 Перемещение и масштабирование рабочего листа

Перемещение по рабочему листу происходит перетаскиванием зажатого указателя ПКМ. Перетаскивание рабочего листа в правый нижний угол (см. Рисунок 60).

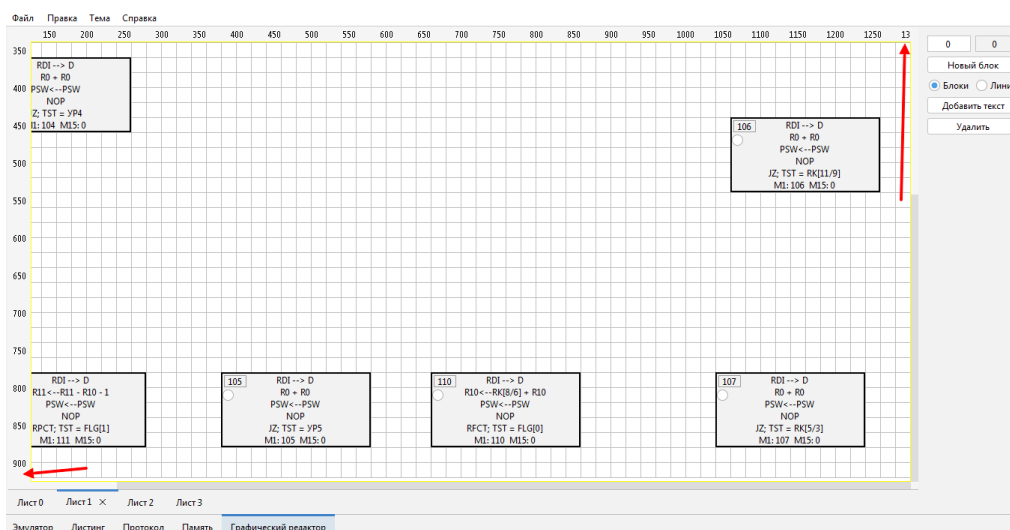


Рисунок 60 – Перемещение в правый нижний угол рабочего листа

Перемещение в верхний правый угол можно проверить по координатам на линейке – по оси Y должно быть 0, а по оси X максимальное значение (см. Рисунок 61). Перемещение в нижний левый угол можно проверить по координатам на линейке – по оси Y должно быть максимальное значение, а по оси X – 0 (см. Рисунок 62).

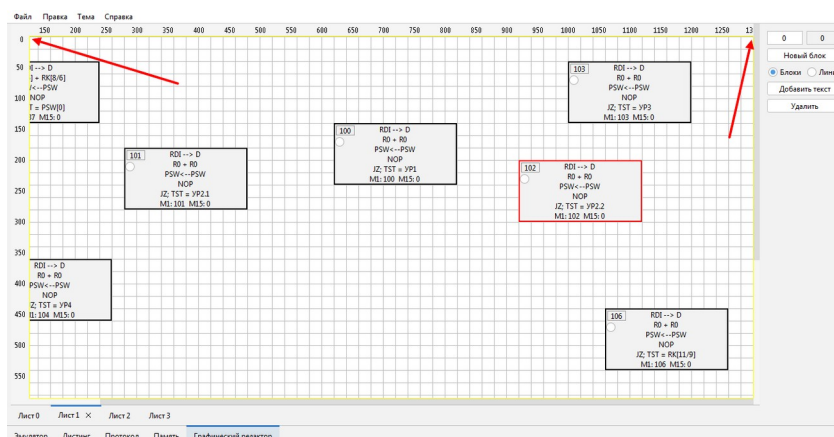


Рисунок 61 – Перемещение в верхний правый угол рабочего листа

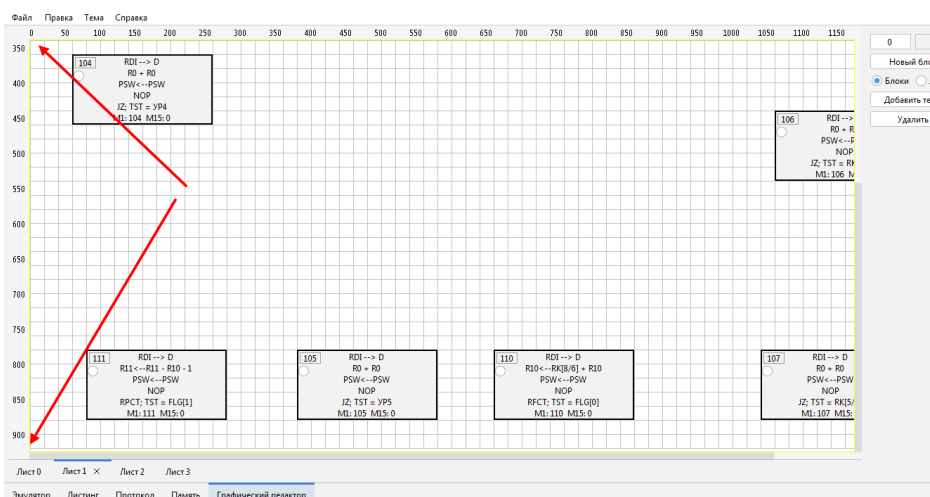


Рисунок 62 – Перемещение в левый нижний угол рабочего листа

Масштабирование происходит прокруткой колесика мыши при зажатой клавиши *CTRL* – уменьшается размер объект, сдвигаются ближе к левому верхнему углу пропорционально, изменяется шаг горизонтальной и вертикальной линейки рабочего листа (см. Рисунок 63 и Рисунок 64). Уменьшение и увеличение выбирается зависит от направление прокрутки колесика мыши.

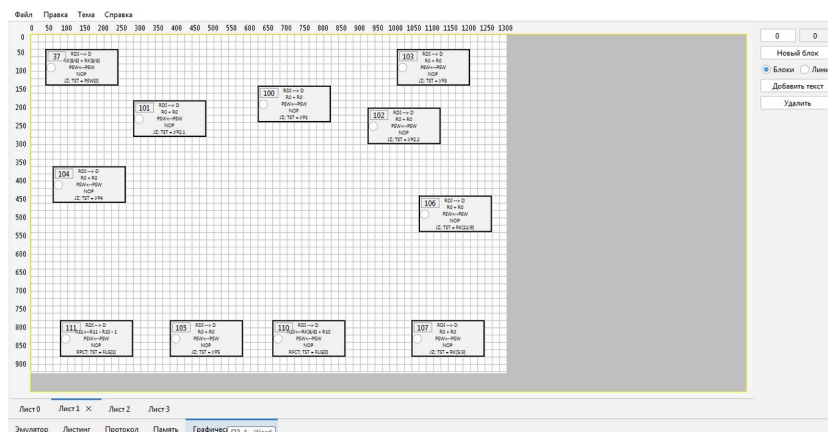


Рисунок 63 – Уменьшение размеров рабочего листа

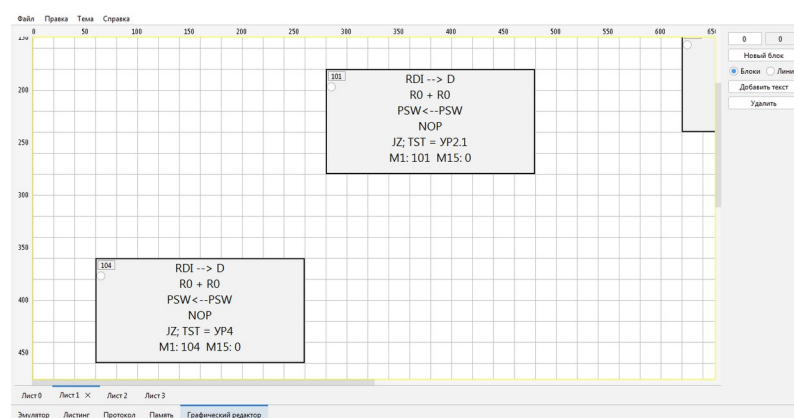


Рисунок 64 – Увеличение размеров рабочего листа

Тесты данного пункта показывают корректную работу с рабочим листом при перемещении и масштабировании.

1.6.10 Создание и удаление рабочего листа

Создание листа с помощью соответствующего пункта контекстного меню (см. Рисунок 65).

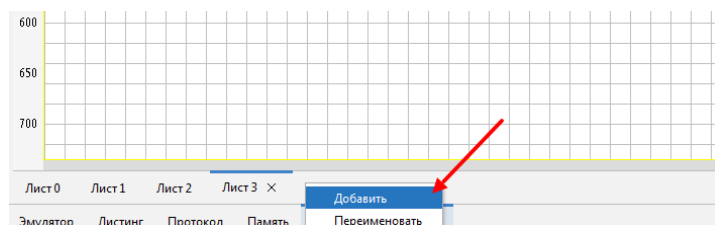


Рисунок 65 – Создание нового рабочего листа

Добавление блока на созданный рабочий лист, который будет удален вместе с удалением листа. Созданную микрокоманду можно проверить в окне «Листинг» на 5 адресе (см. Рисунок 67).

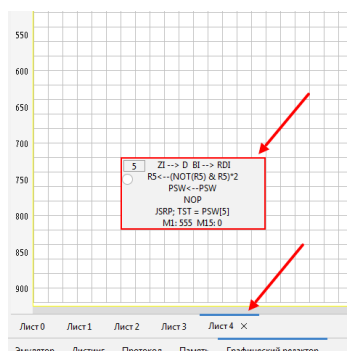


Рисунок 66 – Результат добавление рабочего листа и тестового БМК

Адрес	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	555	0	5	5	5	5	0	0	5	5	5	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	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
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	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
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 67 – Проверка созданной БМК с 5-ым адресом

Добавление 5 новых рабочих областей в программу (см. Рисунок 68).

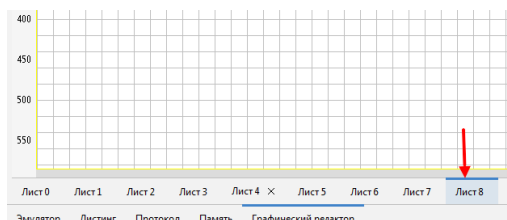


Рисунок 68 – Добавление новых рабочих областей

Удаление рабочих листов при нажатии соответствующей кнопки (см. Рисунок 69).

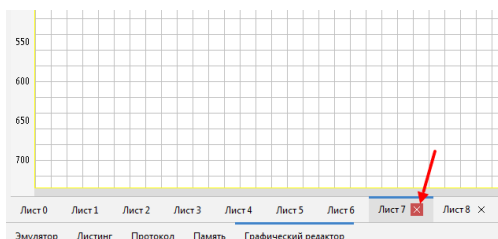


Рисунок 69 – Удаление рабочих областей

Далее удаление рабочей области вместе БМК, находящегося на ней (см. Рисунок 70).

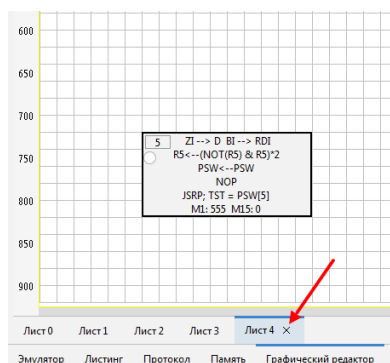


Рисунок 70 – Удаление рабочей области «Лист 4»

После нажатия вкладка исчезает из панели вкладок (см. Рисунок 71). В окне «Листинг» можно наблюдать удаление тестового БМК.

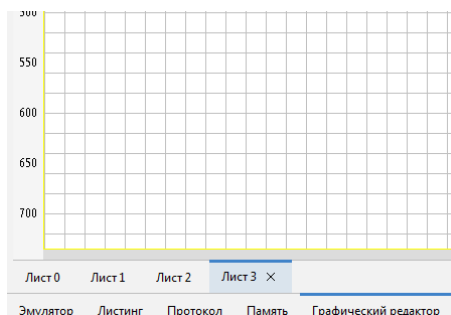


Рисунок 71 – Результат удаления рабочей области

Файл	Правка	Тема	Справка												
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	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
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

☐ Скрыть пустые команды

Эмулятор Листинг Протокол Память Графический редактор

Рисунок 72 – Проверка удаление БМК из памяти программы

Данным способом проверен корректность работы добавления и удаления БМК на рабочие области. Аналогичным образом работа с СЛ и ТБ, но так как они имеют привязку только к одной рабочей области, то проверить удаление таким способом невозможно.

Исходя из данных результатов можно сделать вывод, что графический редактор корректно добавляет, редактирует микрокоманды, сохраняет и загружает всю программу.

2 Безопасность жизнедеятельности

2.1 Эргономические требования к рабочему месту

Проектирование рабочих мест, снабженных видеотерминалами, относится к числу важных проблем эргономического проектирования в области вычислительной техники.

Рабочее место и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия: оптимальное размещение оборудования, входящего в состав рабочего места и достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения.

Эргономическими аспектами проектирования видеотерминальных рабочих мест, в частности, являются: высота рабочей поверхности, размеры пространства для ног, требования к расположению документов на рабочем месте (наличие и размеры подставки для документов, возможность различного размещения документов, расстояние от глаз пользователя до экрана, документа, клавиатуры и т.д.), характеристики рабочего кресла, требования к поверхности рабочего стола, регулируемость элементов рабочего места [26].

Главными элементами рабочего места программиста являются стол и кресло. Основным рабочим положением является положение сидя.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Моторное поле - пространство рабочего места, в котором могут осуществляться двигательные действия человека.

Максимальная зона досягаемости рук - это часть моторного поля рабочего места, ограниченного дугами, описываемыми максимально вытянутыми руками при движении их в плечевом суставе.

Оптимальная зона - часть моторного поля рабочего места, ограниченного дугами, описываемыми предплечьями при движении в локтевых суставах с опорой в точке локтя и с относительно неподвижным плечом.

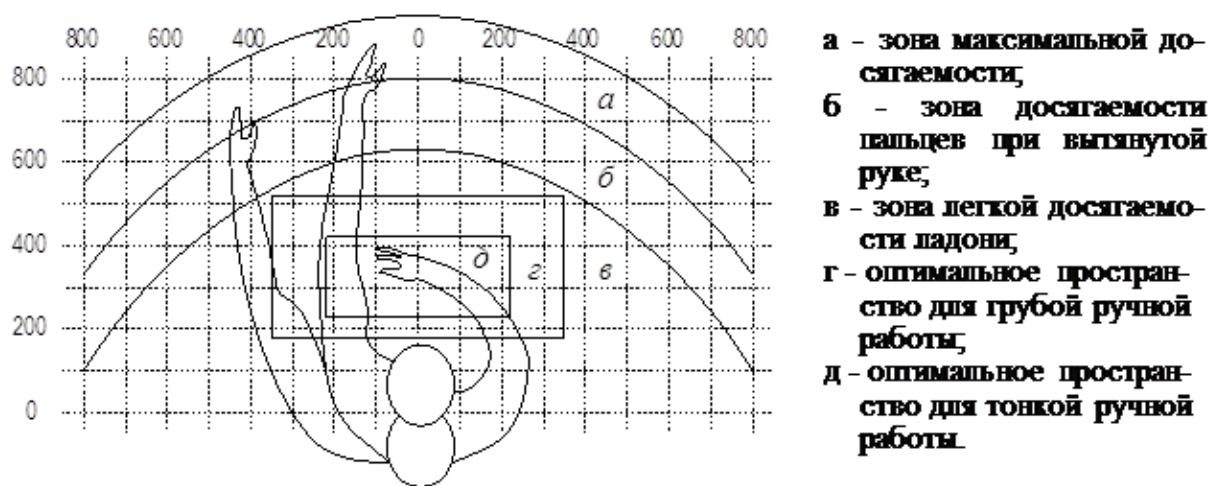


Рисунок 73 – Зоны досягаемости рук в горизонтальной плоскости

Пример размещения основных и периферийных составляющих ПК на рабочем столе программиста (см. Рисунок 74). Оптимальное размещение предметов труда и документации в зонах досягаемости:

- ДИСПЛЕЙ размещается в зоне а (в центре);
- СИСТЕМНЫЙ БЛОК размещается в предусмотренной нише стола;
- КЛАВИАТУРА - в зоне г/д;
- «МЫШЬ» - в зоне в справа;
- СКАНЕР в зоне а/б (слева);
- ПРИНТЕР находится в зоне а (справа);

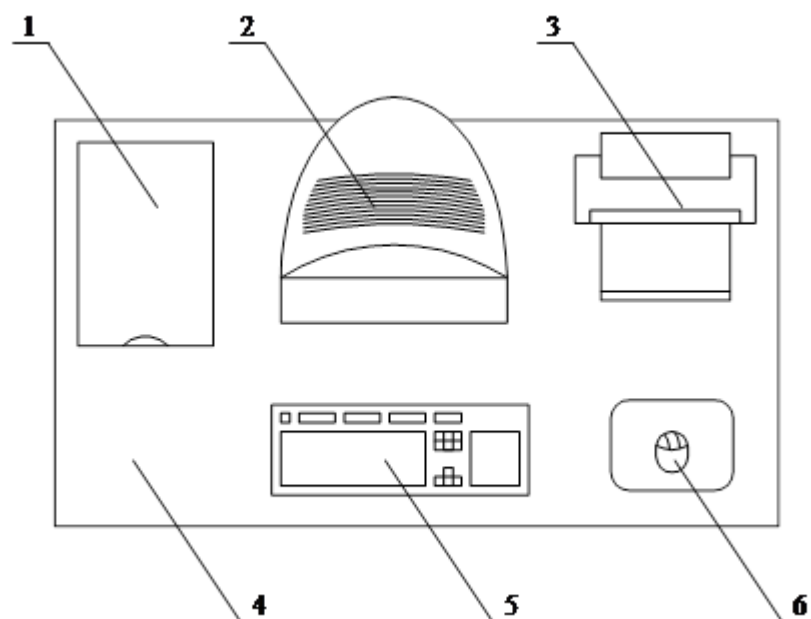


Рисунок 74 – Размещение основных периферийных устройств, составляющие ПК

1 – сканер, 2 – монитор, 3 – принтер, 4 – поверхность рабочего стола, 5 – клавиатура, 6 – манипулятор типа «мышь».

Для комфортной работы стол должен удовлетворять следующим условиям [26]:

- высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
- нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
- поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
- конструкция стола должна предусматривать наличие выдвижных ящиков (не менее 3 для хранения документации, листингов, канцелярских принадлежностей).
- высота рабочей поверхности рекомендуется в пределах 680-760мм. Высота поверхности, на которую устанавливается клавиатура, должна быть около 650мм.

Большое значение придается характеристикам рабочего кресла. Так, рекомендуемая высота сиденья над уровнем пола находится в пределах 420-550мм. Поверхность сиденья мягкая, передний край закругленный, а угол наклона спинки - регулируемый.

Необходимо предусматривать при проектировании возможность различного размещения документов: сбоку от видеотерминала, между монитором и клавиатурой и т.п. Кроме того, в случаях, когда видеотерминал имеет низкое качество изображения, например заметны мелькания, расстояние от глаз до экрана делают больше (около 700мм), чем расстояние от глаза до документа (300-450мм). Вообще при высоком качестве изображения на видеотерминале расстояние от глаз пользователя до экрана, документа и клавиатуры может быть равным.

Положение экрана определяется:

- расстоянием считывания (0,6...0,7м);
- углом считывания, направлением взгляда на 20° ниже горизонтали к центру экрана, причем экран перпендикулярен этому направлению.

Должна также предусматриваться возможность регулирования экрана:

- по высоте +3 см;
- по наклону от -10° до $+20^\circ$ относительно вертикали;
- в левом и правом направлениях.

Большое значение также придается правильной рабочей позе пользователя. При неудобной рабочей позе могут появиться боли в мышцах, суставах и сухожилиях. Требования к рабочей позе пользователя видеотерминала следующие:

- голова не должна быть наклонена более чем на 20° ,
- плечи должны быть расслаблены,
- локти - под углом $80^\circ \dots 100^\circ$,
- предплечья и кисти рук - в горизонтальном положении.

Причина неправильной позы пользователей обусловлена следующими факторами: нет хорошей подставки для документов, клавиатура находится слишком высоко, а документы - низко, некуда положить руки и кисти, недостаточно пространство для ног.

В целях преодоления указанных недостатков даются общие рекомендации: лучше передвижная клавиатура; должны быть предусмотрены специальные приспособления для регулирования высоты стола, клавиатуры и экрана, а также подставка для рук [7].

Существенное значение для производительной и качественной работы на компьютере имеют размеры знаков, плотность их размещения, контраст и соотношение яркостей символов и фона экрана. Если расстояние от глаз оператора до экрана дисплея составляет 60...80 см, то высота знака должна быть не менее 3мм, оптимальное соотношение ширины и высоты знака составляет 3:4, а расстояние между знаками – 15...20% их высоты. Соотношение яркости фона экрана и символов - от 1:2 до 1:15 [8].

Во время пользования компьютером медики советуют устанавливать монитор на расстоянии 50-60 см от глаз. Специалисты также считают, что верхняя часть видеодисплея должна быть на уровне глаз или чуть ниже. Когда человек смотрит прямо перед собой, его глаза открываются шире, чем когда он смотрит вниз. За счет этого площадь обзора значительно увеличивается, вызывая обезвоживание глаз. К тому же если экран установлен высоко, а глаза широко открыты, нарушается функция моргания. Это значит, что глаза не закрываются полностью, не омываются слезной жидкостью, не получают достаточного увлажнения, что приводит к их быстрой утомляемости.

Создание благоприятных условий труда и правильное эстетическое оформление рабочих мест на производстве имеет большое значение как для облегчения труда, так и для повышения его привлекательности, положительно влияющей на производительность труда.

2.2 Обеспечение электробезопасности

Электрический ток представляет собой скрытый тип опасности, т.к. его трудно определить в токо- и нетоковедущих частях оборудования, которые являются хорошими проводниками электричества. Смертельно опасным для жизни человека считают ток, величина которого превышает 0,05А, ток менее 0,05А – безопасен (до 1000 В). С целью предупреждения поражений электрическим током к работе должны допускаться только лица, хорошо изучившие основные правила по технике безопасности. В соответствии с правилами электробезопасности в служебном помещении должен осуществляться постоянный контроль состояния электропроводки, предохранительных щитов, шнуров, с помощью которых включаются в электросеть компьютеры, осветительные приборы, другие электроприборы.

Электрические установки, к которым относится практически все оборудование ЭВМ, представляют для человека большую потенциальную опасность, так как в процессе эксплуатации или проведении профилактических работ человек может коснуться частей, находящихся под напряжением. Специфическая опасность электроустановок – токоведущие проводники, корпуса стоек ЭВМ и прочего оборудования, оказавшегося под напряжением в результате повреждения (пробоя) изоляции, не подают каких-либо сигналов, которые предупреждают человека об опасности. Реакция человека на электрический ток возникает лишь при протекании последнего через тело человека. Исключительно важное значение для предотвращения электротравматизма имеет правильная организация обслуживания действующих электроустановок ВЦ, проведения ремонтных, монтажных и профилактических работ.

Основным физическим фактором, вызывающим тяжесть электротравмы, является сила тока — количество электричества, проходящего через тело человека в единицу времени. Важной характеристикой, определяющей исход воздействия тока, является

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

электрическое сопротивление тела человека, которое является суммой сопротивления кожи и сопротивления внутренних тканей. Ток, проходящий через тело человека ($I_{\text{чел}}$, А), условно определяют по закону Ома:

$$I_{\text{ч}} = \frac{U_{\text{пр}}}{R_{\text{ч}}}, \text{ где} \quad (2.1)$$

$U_{\text{пр}}$ - приложенное напряжение, В;

$R_{\text{чел}}$ - сопротивление тела человека, Ом.

Для расчетов обычно принимают, что $R_{\text{чел}} = 1000$ Ом. Основное сопротивление распространению тока оказывает кожа человека. В том случае, если кожа повреждена, увлажнена или загрязнена токопроводящей пылью (металлической или углеродной), сопротивление тела человека может быть и ниже 1000 Ом.

Заключение

В результате выполнения выпускной квалификационной работы был разработан графический редактор учебной системы проектирования микропрограммируемых ЭВМ, для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительно. Функционал программы соответствует заданию. Программа протестирована и готова к использованию в учебном процессе. В ходе работы был изучены аналоги, различные архитектуры приложений, язык программирования Java, технология SWING, на основе которых спроектирована система, а также эргономические требования к рабочему месту и обеспечение электробезопасности. Цели и задачи, поставленные на данную выпускную квалификационную работу, выполнены.

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Список используемых источников

1. 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>.
2. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум. – СПб. : Питер, 2018. – С. 816.
3. Шилдт, Г. *Java 8*. Полное руководство / Герберт Шилдт; – М.: «Питер». Вильямс, 2015. –С. 816 с.
4. Монахов, В.В. Язык программирования *Java* и среда *NetBeans*: [учеб. пособие] / В.В. Монахов. – СПб.: БХВ-Петербург, 2011. – С. 703.
5. 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.
6. 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.
7. Зинченко В.П. Основы эргономики. – М.: МГУ, 1979. – 179с.
8. Безопасность жизнедеятельности. /Под ред. Н.А. Белова - М.: Знание, 2000 - 364с.

Приложение А

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

Листинг программы

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		Н

Класс JScrollPaneTest

```
package t4;

import t4.JLayeredPane.JLayeredPaneTest;
import javax.swing.*;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.ContainerAdapter;
import java.awt.event.ContainerEvent;
import java.awt.event.ContainerListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.io.IOException;
import java.io.Serializable;
import java.util.LinkedList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JScrollPaneTest implements Serializable {

    public JViewport viewport;
    public JScrollPane scrollPane;
    public JLayeredPaneTest vp;
    public JTextField jtfZoom;
    public String name;

    public JScrollPaneTest(Dimension dm, Dimension dm2,
        LinkedList<Block> ListBlock,
        int step, int step2, JTextField jtfZoom,
        JPopupMenu jPopupMenuBlock, String name
    ) {
        try {
            vp = new JLayeredPaneTest(step, step2, dm2,
                ListBlock, jPopupMenuBlock);

            this.name = name;
            this.jtfZoom = jtfZoom;

            viewport = new JViewport();

            viewport.setView(vp);
            viewport.setViewSize(vp.getSize());
        }
    }
}
```

```

        // Определение видимого диапазона 30 = 20 + xxx
(число)
        viewport.setExtentSize(new Dimension(dm.width - 30,
dm.height));
        // Ограничение размера "видоискателя"
        viewport.setPreferredSize(new Dimension(dm.width -
30, dm.height));
        // Точка начала видимой области
        viewport.setViewPosition(new Point(0, 150));
        // Без этого мода не работает прокрутка, не
отрисовывает
        viewport.setScrollMode(0);
        viewport.setBackground(Color.LIGHT_GRAY);

        viewport.addMouseWheelListener(new
MouseWheelListenerPanel());

        //-----
        scrollPane = new JScrollPane(
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

        scrollPane.setViewport(viewport);

        // Устанавливаем заголовки
        scrollPane.setColumnHeaderView(new XHeader());
        scrollPane.setRowHeaderView(new YHeader());

scrollPane.setViewportBorder(BorderFactory.createLineBorder(Colo
r.yellow));
//        scrollPane.setViewportBorder(null);

// Определение свойств панели прокрутки
        scrollPane.setWheelScrollingEnabled(true);

        scrollPane.setBorder(null);

// Размещение кнопки печати в левом верхнем углу
        scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER,
null);

        scrollPane.setBounds(0, 0, dm.width, dm.height);

//        setBorder(null);
//        scrollPane.setLocation(0, 0);
//        setBounds(0, 0, 800, 300);
//        setBorder(null);
//        setLocation(0, 0);
//        setBounds(0, 0, dm.width, dm.height);
//        setVisible(false);
//        setBackground(Color.red);

```

```

        } catch (IOException ex) {

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

    // Заголовок по оси X
    class XHeader extends JPanel {

//        private static final long serialVersionUID = 1L;
//        Размер заголовка
        @Override
        public Dimension getPreferredSize() {
            return new Dimension(vp.sizeCurFrame.width, 20);
        }

//        Прорисовываем линейку
        @Override
        public void paintComponent(Graphics g) {

            int width = vp.sizeCurFrame.width;
            int y = 15;
            g.drawString("0", 0, y);

            int step = 50;
            if (vp.zoom < 0.25) {
                step = 200;
            } else if (vp.zoom < 0.45) {
                step = 100;
            } else if (vp.zoom > 2.25) {
                step = 10;
            }

            int step2 = (int) ((float) step * vp.zoom), step3 =
100, step4 = 0;

            Color clr = g.getColor();

            for (int i = step2 - 5, j = step; i < width; i +=
step2, j += step) {
                if (j >= step3) {
                    step3 *= 10;
                    i -= 4;
                    step4 += 4;
                }
                if (j % 50 == 0) {
                    g.drawString(String.format("%d", (j)), i,
y);
                } else {
                    g.setColor(Color.gray);

```

```

        g.drawString(String.format("%2d", (j %
100)), i + step4, y);
        g.setColor(clr);
    }
}

// Заголовок по оси Y
class YHeader extends JPanel {

//     private static final long serialVersionUID = 1L;
// Размер заголовка
@Override
public Dimension getPreferredSize() {
    return new Dimension(30, vp.sizeCurFrame.height);
}

// Прорисовываем линейку
@Override
public void paintComponent(Graphics g) {

    g.drawString(String.format("%6d", 0), 0, 10);

    int height = vp.sizeCurFrame.height;
    int step = 50;
    if (vp.zoom < 0.25) {
        step = 200;
    } else if (vp.zoom < 0.45) {
        step = 100;
    } else if (vp.zoom > 2.25) {
        step = 10;
    } else if (vp.zoom > 4.00) {
        step = 5;
    } else if (vp.zoom > 6.00) {
        step = 2;
    } else if (vp.zoom > 8.00) {
        step = 1;
    }

//     int step = 50;
//     if (vp.zoom < 0.45) {
//         step = 100;
//     } else if (vp.zoom > 1.45) {
//         step = 10;
//     }
    int step2 = (int) ((float) step * vp.zoom);

    Color clr = g.getColor();
//     String str = " ", str2 = " ";
    for (int i = step2 + 5, j = step; i < height; i +=
step2, j += step) {

```

```

        if (j % 50 == 0) {
            g.drawString(String.format("%5d", j), 0, i);
        } else {
            g.setColor(Color.gray);
//            g.drawString(" " + (j % 100), 0, i);
            g.drawString(String.format("%5d", (j %
100)), 0, i);
            g.setColor(clr);
        }
    }
}

class MouseWheelListenerPanel extends JPanel implements
MouseWheelListener {

    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
        if (e.isControlDown()) {
            if (e.getWheelRotation() < 0) {
                // System.out.println("mouse wheel Up");
                vp.addZoom(e.getPoint());
//                vp.setLocation(e.getPoint().x -
(vp.getWidth() / 2), e.getPoint().y - (vp.getHeight() / 2));
            } else {
                // System.out.println("mouse wheel Down");
                vp.subZoom(e.getPoint());
//                vp.setLocation(e.getPoint().x -
(vp.getWidth() / 2), e.getPoint().y - (vp.getHeight() / 2));
            }
            jtfZoom.setText(String.valueOf((int) (vp.zoom *
100.0)));
            Object ob = ((JComponent)
e.getSource()).getParent();
            ((JComponent) ob).dispatchEvent(e);
        } else {
//            vp.updateSelectPanel(e.getPoint().x ,
e.getPoint().y + e.getWheelRotation() * vp.poleSizeCur);

//            System.out.printf("\n %d %d %d",
e.getPoint().x, e.getPoint().y, e.getWheelRotation());
            Object ob = ((JComponent)
e.getSource()).getParent();
            ((JComponent) ob).dispatchEvent(e);

//            vp.addSelectPanel(e.getWheelRotation() *
vp.poleSizeCur, true);
        }
    }
}

```

```
}
```

Класс *Visio*

```
package t4;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.KeyEventDispatcher;
import java.awt.KeyboardFocusManager;
import java.awt.Point;
import java.awt.event.KeyEvent;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import javax.swing.ButtonGroup;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.plaf.basic.BasicButtonUI;
import javax.swing.plaf.basic.BasicTabbedPaneUI;
import static t3.GlobalVars.*;
import t3.Memory;
import t4.JLayeredPane.JLayeredPaneTest;
import t4.JLayeredPane.Serialize.Serializer;

public class Visio {

    private final javax.swing.JPanel jPanel2;
    public final javax.swing.JTabbedPane jTabbedPane1;
    private final javax.swing.JTextField jTextField3;
    private final javax.swing.JTextField jTextField4;
    private final javax.swing.JTextField jTextField5;

    private final javax.swing.JPopupMenu jPopupMenuBlock;
    private final javax.swing.JPopupMenu jPopupMenuTabbed;

    private final Memory memForm;

    public final ArrayList<ScrollPaneTest> spt = new
ArrayList<>();
    public final LinkedList<Block> ListBlock;
```

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

```

private JMenu jMenuReplace;
private ButtonGroup buttonGroupReplace;

public Visio(JPanel jPanel22,
             JTabbedPane jTabbedPane12,
             JTextField jTextField32,
             JTextField jTextField42,
             JTextField jTextField52,
             Memory memForm,
             JPopupMenu jPopupMenuBlock,
             JPopupMenu jPopupMenuTabbed
) throws InterruptedException {

    this.ListBlock = new LinkedList<Block>() {
        @Override
        public void clear() {
            super.clear(); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/Overridden
MethodBody
        }

        @Override
        public boolean remove(Object o) {
            return super.remove(o); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/Overridden
MethodBody
        }

    };
    this.jPanel2 = jPanel22;
    this.jTabbedPane1 = jTabbedPane12;
    this.jTextField3 = jTextField32;
    this.jTextField4 = jTextField42;
    this.jTextField5 = jTextField52;
    this.memForm = memForm;
    this.jPopupMenuBlock = jPopupMenuBlock;
    this.jPopupMenuTabbed = jPopupMenuTabbed;

    //      jPanel2.setBackground(Color.LIGHT_GRAY);
    jPanel2.setBorder(null);
    jPanel2.setOpaque(true);
    jPanel2.setLayout(new GridLayout(0, 1));

    jTextField4.setText(String.valueOf((int) (100.0)));
    jTextField5.setText(String.valueOf(ListBlock.size()));
    //-----
    //      jTabbedPane1.setUI(new BasicTabbedPaneUI() {
    //          @Override
    //          protected Insets getTabInsets(int tabPlacement,
    int tabIndex) {
    //              return new Insets(0, 0, 0, 0);

```

```

//          }
//      }
//      );

//-----
initPopupMenu();
//-----

for (int i = 0; i < 4; i++) {
    createTab(i);
}
jTabbedPane1.setSelectedIndex(0);

// Слушатель на переключение между вкладками
jTabbedPane1.addChangeListener(new
javax.swing.event.ChangeListener() {
    int index = 0; // текущая активная вкладка

    @Override
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        // Убрать выделения на листе, с которого
переключились
// System.out.println("size ListBlock " + ListBlock.size());
        spt.get(index).vp.clearSelectAllObject();
        // Сохранения текущей активной вкладки
        index = jTabbedPane1.getSelectedIndex();
    }
});
//-----

// Перехватчик событий клавиатуры
KeyboardFocusManager.getCurrentKeyboardFocusManager()
    .addKeyEventDispatcher(new KeyEventDispatcher()
{
    @Override
    public boolean dispatchKeyEvent(KeyEvent evt
    ) {

        if (evt.getID() == KeyEvent.KEY_PRESSED
&& evt.getKeyCode() != KeyEvent.VK_CONTROL && evt.getKeyCode() !=
KeyEvent.VK_ALT) {

//System.out.println(evt.paramString() + " " +
evt.getSource().toString());

            if (evt.getKeyCode() ==
KeyEvent.VK_DELETE) {

spt.get(jTabbedPane1.getSelectedIndex()).vp.delete();
            } else if (evt.getKeyCode() ==
KeyEvent.VK_A && evt.isControlDown()) {

```

```

spt.get(jTabbedPanel.getSelectedIndex()).vp.selectAllObject(true
);
                                } else if (evt.getKeyCode() ==
KeyEvent.VK_ESCAPE) {

spt.get(jTabbedPanel.getSelectedIndex()).vp.clearSelectAllObject
();
                                } else if (evt.getKeyCode() ==
KeyEvent.VK_Z && evt.isControlDown()) {
                                listEdit.Undo();
                                } else if (evt.getKeyCode() ==
KeyEvent.VK_Y && evt.isControlDown()) {
                                listEdit.Redo();
                                } else {
                                return false;
                                }
                                return true;
                                } else {
                                return false;
                                }
                                }
                                }
                                );

//      AddNewBlock();
//      resetPositionView();
jTabbedPanel.setSelectedIndex(0);
}

private void initPopupMenu() {
    JMenuItem jMenuItemDelete = new JMenuItem();
    JMenuItem jMenuItemEdit = new JMenuItem();

    jMenuItemEdit.setText("Изменить");
    jMenuItemEdit.addActionListener(new
java.awt.event.ActionListener() {
        @Override
        public void
actionPerformed(java.awt.event.ActionEvent evt) {

spt.get(jTabbedPanel.getSelectedIndex()).vp.checkingAllBlock(tru
e);
        }
    });
    jMenuItemEdit.add(jMenuItemEdit);

    buttonGroupReplace = new ButtonGroup();

    jMenuItemReplace = new JMenuItem();
    jMenuItemReplace.addMenuListener(new
javax.swing.event.MenuListener() {

```

```

        @Override
        public void menuCanceled(javax.swing.event.MenuEvent
evt) {
            }

            @Override
            public void
menuDeselected(javax.swing.event.MenuEvent evt) {
                }

                @Override
                public void menuSelected(javax.swing.event.MenuEvent
evt) {
                    // Получить индекс листа у блока
                    Block blk =
spt.get(jTabbedPane1.getSelectedIndex()).vp.checkingAllBlock(fal
se);

                    // установить пункт соответствующего листа
                    ((JCheckBoxMenuItem)
jMenuReplace.getMenuComponents()
[blk.indexList]).setSelected(true);
                }
            });
            jMenuReplace.setText("Переместить");

            jPopupMenuBlock.add(jMenuReplace);

            jMenuItemDelete.setText("Удалить");
            jMenuItemDelete.addActionListener(new
java.awt.event.ActionListener() {
                @Override
                public void
actionPerformed(java.awt.event.ActionEvent evt) {

spt.get(jTabbedPane1.getSelectedIndex()).vp.checkingAllBlock(fal
se);

spt.get(jTabbedPane1.getSelectedIndex()).vp.delete();
                }
            });
            jPopupMenuBlock.add(jMenuItemDelete);
            updateIndexList();
        }

        public void createTab(int idx) {
            String name = "Лист " + idx;
            Dimension dm1 = new Dimension(jTabbedPane1.getSize());
            double k1 = 1.0;
            int block = 20;
            int block2 = 50;

```

```

        Dimension d = new Dimension((int) (1300 * k1), (int)
(925 * k1));

        ScrollPaneTest component = new ScrollPaneTest(dm1, d,
ListBlock,
                block, block2, jTextField4, jPopupMenuBlock,
name);

        spt.add(component);

        jTabbedPane1.addTab(name, component.scrollPane);

        jTabbedPane1.setSelectedIndex(jTabbedPane1.getTabCount()
- 1);

        component.vp.indexList = jTabbedPane1.getTabCount() - 1;

jTabbedPane1.setTabComponentAt(jTabbedPane1.getTabCount() - 1,
new ButtonTabComponent(jTabbedPane1, this, name,
jPopupMenuTabbed));
        //
-----
----
        // добавление в перечень листов контекстного меню
JCheckBoxMenuItem jMenuItemReplace = new JCheckBoxMenuItem();

        // Установка текста, как название листа
jMenuItemReplace.setText(name);

        // Добавление слушателя, при нажатии на пункт меню
jMenuItemReplace.addActionListener(new
java.awt.event.ActionListener() {
            @Override
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                Component[] comp =
jMenuReplace.getMenuComponents();
                for (int i = 0; i < comp.length; i++) {
                    if (comp[i] == evt.getSource()) {

spt.get(jTabbedPane1.getSelectedIndex()).vp.replaceBlock(i,
spt.get(i).vp);

                                break;
                            }
                        }
                    }
                });

        jMenuReplace.add(jMenuItemReplace);        // Добавление в
группу меню
        buttonGroupReplace.add(jMenuItemReplace);    // и в
группу кнопок

```

```

        updateIndexList();
    }

    private void updateIndexList() {
        for (int i = 0; i < spt.size(); i++) {
            int b = spt.get(i).vp.indexList;
            if (b != i) {

buttonGroupReplace.remove(jMenuReplace.getItem(i));
                jMenuReplace.remove(i);
                for (Block blk : ListBlock) {
                    if (blk.indexList == b) {
                        blk.indexList = i;
                    }
                }
            }
        }
    }

    public void renameTab() {
        int index = jTabbedPane1.getSelectedIndex();
        if (index >= 0) {

            ButtonTabComponent b = (ButtonTabComponent)
jTabbedPane1.getSelectedComponent();
            if (b != null) {
//                b.label.setEditable(true);

            }
        }

    public void deleteTab() {
        int index = jTabbedPane1.getSelectedIndex();
        if (index >= 0) {
            ScrollPaneTest component = spt.get(index);
            component.vp.deleteAllObject();
            jTabbedPane1.remove(index);
            spt.remove(index);
            updateIndexList();
        }
    }

    public void AddNewBlock() throws NumberFormatException {

        short idx = (short)
Integer.parseInt(jTextField5.getText(), 8);

        boolean f;

        do {
            f = false;

```

```

        Iterator<Block> iter = ListBlock.iterator();
        while (iter.hasNext()) {
            if (iter.next().id == idx) {
                f = true;
                idx++;
                break;
            }
        }
    } while (f);

spt.get(jTabbedPane1.getSelectedIndex()).vp.addBlock(idx);

jTextField3.setText(decToOctStr(ListBlock.size()));
jTextField5.setText(decToOctStr(ListBlock.size()));

}

public void jButton1ActionPerformed() {
    AddNewBlock();
}

// Скрытие и показ блоков
public void DebugBlock(boolean flag) {
    Iterator<ScrollPaneTest> iter = spt.iterator();
    while (iter.hasNext()) {
        Component[] cmp = iter.next().vp.getComponents();
        for (Component cmp1 : cmp) {
            if (flag || !cmp1.equals(iter.next().vp)) {
                cmp1.setVisible(flag);
            }
        }
        iter.next().vp.repaint();
    }
}

public void jButton5ActionPerformed() {
    DebugBlock(false);
}

public void jButton6ActionPerformed() {
    DebugBlock(true);
}

public void jButton7ActionPerformed() {
    // Point p = jp.getLocation();
    // p.x += 10;
    // jp.setLocation(p);
}

public void jButton8ActionPerformed() {
    // Point p = jp.getLocation();

```

```

//      p.x -= 10;
//      jp.setLocation(p);
}

public void jButton9ActionPerformed() {
//      Point p = jp.getLocation();
//      p.y -= 10;
//      jp.setLocation(p);
}

public void jButton10ActionPerformed() {
//      Point p = jp.getLocation();
//      p.y += 10;
//      jp.setLocation(p);
}

public void zoom(boolean flag) {

    if (flag) {

spt.get(jTabbedPane1.getSelectedIndex()).vp.addZoom();
        } else {

spt.get(jTabbedPane1.getSelectedIndex()).vp.subZoom();
        }
        jTextField4.setText(String.valueOf((int)
(spt.get(jTabbedPane1.getSelectedIndex()).vp.zoom * 100.0)));
        jPanel2.repaint();
    }

public void jButton2ActionPerformed() {
    zoom(true);
}

public void jButton3ActionPerformed() {
    zoom(false);
}

public void jButton4ActionPerformed() {
//      if (jPanel2.isVisible()) {
//          printObj(jPanel2, "-----\n jPanel2");
//      }
//////      if (vp2.isVisible()) {
//////          printObj(vp2, "Vp2");
//////      }
//      if (vp2.scrollPane.isVisible()) {
//          printObj(vp2.scrollPane, "Scroll");
//      }
//      if (vp2.viewport.isVisible()) {
//          printObj(vp2.viewport, "viewport");
//      }
//      if (jp.isVisible()) {

```

```

//          printObj(jp, "jp");
//      }

      }

      public void printObj(JComponent comp, String str) {
          System.out.println("\n" + str
              + "\nLocScr: " +
String.valueOf(comp.getLocationOnScreen().x)
              + " " +
String.valueOf(comp.getLocationOnScreen().y)
              + "\nSize: " + String.valueOf(comp.getWidth())
              + " " + String.valueOf(comp.getHeight())
              + "\nLoc: " +
String.valueOf(comp.getLocation().x)
              + " " + String.valueOf(comp.getLocation().y)
          );
      }

      public void resetPositionView() {
          ScrollPaneTest s =
spt.get(jTabbedPane1.getSelectedIndex());
          s.viewport.setViewPosition(new Point(0, 0));
          s.scrollPane.setLocation(0, 0);
          s.scrollPane.getVerticalScrollBar().setValue(0);
          s.scrollPane.getHorizontalScrollBar().setValue(0);

          jPanel2.repaint();
      }

      public void jButton1ActionPerformed() {
          resetPositionView();
      }

      public void jTextField5ActionPerformed() {
          if (!jTextField5.getText().isEmpty()) {
              AddNewBlock();
          }
      }

      public void saveProgrammPZU() {

          Iterator<Block> iter = ListBlock.iterator();
          while (iter.hasNext()) {
              Block b = iter.next();
              pzu[0][b.id] = b.m[1];
              pzu[1][b.id] = (short) (b.m[6] * 2048 + b.m[5] * 64
+ b.m[3] * 4 + b.m[2]);
              pzu[2][b.id] = (short) (b.m[13] * 16384 + b.m[10] *
2048 + b.m[9] * 256 + b.m[8] * 64 + b.m[7] * 16 + b.m[4]);
              pzu[3][b.id] = (short) (b.m[15] * 256 + b.m[14] * 64
+ b.m[12] * 8 + b.m[11]);
          }
      }

```

```

    }

    /**
     * Кол-во листов фиксированное n = 4 + 1
     *
     * @param filePath
     * @throws IOException
     */
    public void save(String filePath) throws IOException {

        int n = spt.size() + 1;

        Object[][] mas = new Object[n][3];

        mas[0][0] = memForm.getDataTable();
        mas[0][1] = pzu;
        mas[0][2] = new Point(n, n);

        for (int i = 1; i < n; i++) {
            mas[i] = spt.get(i - 1).vp.getAllObject();
        }

        if (new Serializer().binarySerialization(mas, n, 3,
filePath)) {
            System.out.println("Сохранено");
        } else {
            System.out.println("Ошибка при сохранении данных");
        }
    }

    /**
     * Кол-во листов фиксированное n = 4 + 1
     *
     * @param filePath
     * @throws IOException
     */
    public void load(String filePath) throws IOException {

        int n = spt.size() + 1;

        Object[][] mas = new
Serializer().binaryDeserialization(n, 3, filePath);

        if (mas != null) {
            // ListBlock.clear();
            // idPzuArray.clear();

            memForm.setDataTable((int[]) mas[0][0]);
            pzu = (short[][]) mas[0][1];

            for (int i = 1; i < n; i++) {
                spt.get(i - 1).vp.addAllObject(mas[i]);
            }
        }
    }

```

```

        }

        System.out.println("Загружено");
    } else {
        System.out.println("Ошибка при чтении данных");
    }
}

public void generate() {

    for (ScrollPaneTest s : spt) {
        s.vp.deleteAllObject();
    }

    int countList = 0;
    Point maxList = new Point(5, 6);
    Point curList = new Point(0, 0);

    short[] m = new short[16];
    boolean check = true;
    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)) {

```



```

import javax.swing.JLayeredPane;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JViewport;
import javax.swing.Scrollable;
import t4.Block;
import t4.Rect;
import static t3.GlobalVars.pzu;
import t4.JLayeredPane.Serialize.SerializeBlock;
import t4.JLayeredPane.Serialize.SerializeConn;
import t4.JLayeredPane.Serialize.SerializeText;

public class JLayeredPaneTest extends JLayeredPane implements
Scrollable, Serializable { //KeyListener,

    private static final long serialVersionUID = 103L;
    short[] m = new short[16];

    public int indexList;

    public final int poleSizeConst;           // Размер ячейки
    public int poleSizeCur;                   // Размер ячейки
    public final int poleSizeHalf;

    public final Dimension sizeFrame;
    public final Dimension sizeCurFrame;
    public Point pStart = new Point(200, 200);

    public final LinkedList<TextBlock> ListTextBlock;
    public final LinkedList<Block> ListBlock;
    public final int interval = 0, widthBlock = 200 - interval,
heightBlock = 100 - interval;

    private Rect[][] rects;
    private final int widthRect, heightRect;
    private int widthArrayRect, heightArrayRect;

    public final LinkedList<Line> lines = new LinkedList<>();
    public final LinkedList<Connector> connectors = new
LinkedList<>();
    //=====
    private Point delta = new Point(0, 0);

    public double zoom = 1;
    private double zoomOld = 1;

    boolean flagDragged = false;
    //=====
    public boolean flagConnector = false;
    public boolean startConnector = false;
    private Connector conn;

```

```

private Point pointStart = new Point(0, 0);
//=====
public Circle circle = new Circle();
public JPanel panelLine; // для рисования соединительных
линий

//=====
private BufferedImage bImg;
JPanel panelSetka;
//=====

public boolean selectMouse = false;
public SelectPanel selectPanel;

//=====
private boolean flagAdd = false;

//=====
private final JPopupMenu jPopupMenuBlock;

//=====
public JLayeredPaneTest(
    int step,
    int step2,
    Dimension rc,
    LinkedList<Block> ListBlock,
    JPopupMenu jPopupMenuBlock
) throws IOException {

    this.ListBlock = ListBlock;

    ListTextBlock = new LinkedList<>();
    sizeFrame = rc;
    sizeCurFrame = new Dimension(rc.width, rc.height);
    poleSizeCur = step;
    poleSizeConst = step;
    poleSizeHalf = step / 2;

    widthRect = widthBlock / step;
    heightRect = heightBlock / step;

    setFocusable(true);
    setOpaque(true);
    setBackground(Color.white);
    setVisible(true);
    setSize(rc);
    setLocation(0, 0);

    //-----
    int x1 = getWidth(), y1 = getHeight();

    widthArrayRect = x1 / poleSizeCur;

```

```

        heightArrayRect = y1 / poleSizeCur;
        rects = new Rect[widthArrayRect][heightArrayRect];

        for (int i = 0, i2 = 0; i < x1 && i2 < widthArrayRect; i
+= poleSizeCur, i2++) {
            for (int j = 0, j2 = 0; j < y1 && j2 <
heightArrayRect; j += poleSizeCur, j2++) {
                rects[i2][j2] = new Rect(i, j, poleSizeCur,
poleSizeCur, Color.lightGray);
            }
        }

        //=====
        this.jPopupMenuBlock = jPopupMenuBlock;

        //        bImg = new BufferedImage(x1, y1,
BufferedImage.TYPE_INT_ARGB);
        //        Graphics2D g2d = bImg.createGraphics();
        //        g2d.setBackground(Color.white);
        //        g2d.setColor(Color.lightGray);
        //
        //        for (int i = 0, i2 = 0; i < x1 && i2 < widthArrayRect;
i += poleSizeCur, i2++) {
        //            for (int j = 0, j2 = 0; j < y1 && j2 <
heightArrayRect; j += poleSizeCur, j2++) {
        //                g2d.drawRect(i, j, poleSizeCur, poleSizeCur);
        //            }
        //        }
        //
        //        g2d.dispose();
        long usedBytes = (Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory()) / 1048576;
        System.out.printf("MB: %d \n", usedBytes);
        //-----
        //        for (int i = 0; i < x1; i += step2) {
        //            lines.add(new Line(i, 0, i, y1, Color.black));
        //        }
        //        for (int j = 0; j < y1; j += step2) {
        //            lines.add(new Line(0, j, x1, j, Color.black));
        //        }

        //-----
        addMouseMotionListener(new MouseAdapter() {

            Point p2 = new Point(0, 0);

            @Override
            public void mouseDragged(MouseEvent evt) {

                String text = "";
                int button = evt.getButton();
                int flag = -1;

```

```

switch (button) {
    case MouseEvent.BUTTON1:
        text = "Button 1";
        flag = -2;

        break;
    case MouseEvent.BUTTON2:
        text = "Button 2";

        break;
    case MouseEvent.BUTTON3:
        text = "Button 3";
        flag = -3;
        break;
    default:
        text = "Button default";
        break;
}

//          System.out.println(text);
if (flagDragged) {
    Point p = ((JViewport)
evt.getComponent().getParent()).getViewPosition();
    if (delta.x == 0 && delta.y == 0) {
        delta.setLocation(evt.getPoint());
    }
//          spt.viewport.setViewPosition(new Point(p.x
- delta.x, p.y - delta.y));
//          p.setLocation(p.x - evt.getX() + delta.x,
p.y - evt.getY() + delta.y);
//          p.setLocation(p.x - evt.getX(), p.y -
evt.getY());
//          setLocation(p);

    int a = p.x - evt.getX() + delta.x;
    if (a >= 0) {
        p.x = a;
    } else {
        a = -1;
    }
    int b = p.y - evt.getY() + delta.y;
    if (b >= 0) {
        p.y = b;
    } else {
        b = -1;
    }
    if (a != -1 || b != -1) {
        ((JViewport)
evt.getComponent().getParent()).setViewPosition(p);
//          spt.viewport.setViewPosition(p);
        repaint();
    }
}

```

```

        }
        if (selectMouse) { // выделение объектов
            // установить координаты перемещаемого угла
            updateSelectPanel(evt.getX(), evt.getY());
        }
    }

    @Override
    public void mouseMoved(MouseEvent evt) {
//        super.mouseMoved(e);
        Graphics g = getGraphics();
//        printComponent(g);
//        conn.print(g, conn.points.getLast(),
        evt.getPoint());
        boolean repaintCircle = false;
        for (Connector conn : connectors) {
            if (conn.circle != 0) {
                conn.circle = 0;
                repaintCircle = true;
            }
        }

        if (repaintCircle) {
            repaint();
        }

        if (startConnector || flagConnector) {
            // при нажатии в режиме линия

            // Нарисовать круг на пересечении
            if (getMousePosition() != null) {
                Point p = new Point(getMousePosition());

                p.x = rounding(p.x, 10, 5);
                p.y = rounding(p.y, 10, 5);

                circle.setLocation(p);
            }

        } else {

        }

    }

}

addMouseListener(new MouseAdapter() {

    @Override
    public void mousePressed(MouseEvent evt) {

```

```

String text;
int button = evt.getButton();
int flag = -1;

boolean repaintCircle = false;

for (TextBlock conn : ListTextBlock) {
    if (conn.isSelected && !conn.isNotSelect) {
        conn.transferFocus();
        conn.isSelected = false;
        repaintCircle = true;
    }
}

for (Connector conn : connectors) {
    if (conn.isSelected && !conn.isNotSelect) {

        conn.setSelect(false);
        repaintCircle = true;
    }
}

for (Block conn : ListBlock) {
    if (conn.isSelected && !conn.isNotSelect) {
        conn.rightClick = false;
        conn.isSelected = false;
        repaintCircle = true;
    }
}

if (repaintCircle) {
    repaint();
}

switch (button) {
    case MouseEvent.BUTTON1:
        text = "P Button 1";
        flag = -2;
        Point p2 = new
Point(rounding(evt.getX(), 10, 0), rounding(evt.getY(), 10, 0));

        if (flagConnector) {
            // conn = new Connector(getSize());
            /*conn = new Connector(new
Point2D.Double(20,20), new Point2D.Double(180,180));
            connectors.addLast(conn);
            add(conn);*/

            flagConnector = false;
            startConnector = true;
        } else if (startConnector) {

```

```

        conn = new Connector(pointStart, p2,
JLayeredPaneTest.this);

//
        panelLine.paint(getGraphics());
        panelLine.add(conn);
        connectors.addLast(conn);
        conn.addStep(0);
        conn.addStep(2);
        repaint();
    }

//
        conn.repaint();
        pointStart.setLocation(p2);

        selectMouse = true;        // началось
выделение
        selectPanel.setStart(evt.getPoint());
// фиксированная вершина прямоугольника выделения

        break;
    case MouseEvent.BUTTON2:
        text = "P Button 2";

        break;
    case MouseEvent.BUTTON3:
        text = "P Button 3";
        flag = -3;
        flagDragged = true;
        break;
    default:
        text = "P Button default";
        break;
}

//
        System.out.println("JLayeredPane Pressed");
//
        System.out.println(text);

}

@Override
public void mouseReleased(MouseEvent evt) {
    delta.setLocation(0, 0);
    boolean repaintCircle = false;
    flagDragged = false;
    String text;
    int button = evt.getButton();
    int flag = -1;

    Point p1 = evt.getPoint();
    Point p2 = new Point();

//
        for (Connector conn : connectors) {
//
            /* if (conn.IsLine(evt.getPoint())) {

```

```

//          conn.h = 3;
//          }*/ //проверка координаты линия
ВОССТАНОВИТЬ
//          }
//          System.out.println("JLayeredPane Release");
switch (button) {
    case MouseEvent.BUTTON1:
        text = "Button 1";
        flag = -2;
        p2.x = rounding(p1.x, 10, 0);
        p2.y = rounding(p1.y, 10, 0);
        addPointConnector(p2);
        repaint();
        break;
    case MouseEvent.BUTTON2:
        text = "Button 2";
        break;
    case MouseEvent.BUTTON3:
        text = "Button 3";
        flag = -3;
        if (startConnector) {
            startConnector = false;
            flagConnector = true;
        }
//          conn.finish();
        break;
    default:
        text = "Button default";
        break;
}

// Если выделение есть, то сброс прямоуголька
if (selectMouse) {
    selectPanel.setBounds(0, 0, 0, 0);
    selectMouse = false;
}

//
Iterator<Block> iter = ListBlock.iterator();
while (iter.hasNext()) {
    Block next = iter.next();
    if (evt.getX() > next.getX()
    && evt.getY() > next.getY()
    && evt.getX() < next.getX() + next.getWidth()
    && evt.getY() < next.getY() + next.getHeight()) {

        Point pBlock = new Point(next.getX(),
next.getY());

        Point pLine = new Point(evt.getX(),
evt.getY());

        //line.getP1().distance(x, y) <
depthDefault

```

```

        int min = next.getSize().width;

        for (int a = 0; a < 10; a++) {
            int tmp = (int)
pBlock.distance(evt.getX(), evt.getY());
            if (tmp < min) {
                min = tmp;
                pLine.setLocation(pBlock);
            }
            pBlock.x += poleSizeCur;
        }

        for (int a = 0; a < 5; a++) {
            int tmp = (int)
pBlock.distance(evt.getX(), evt.getY());
            if (tmp < min) {
                min = tmp;
                pLine.setLocation(pBlock);
            }

            pBlock.y += poleSizeCur;
        }

        for (int a = 0; a < 10; a++) {
            int tmp = (int)
pBlock.distance(evt.getX(), evt.getY());
            if (tmp < min) {
                min = tmp;
                pLine.setLocation(pBlock);
            }
            pBlock.x -= poleSizeCur;
        }

        for (int a = 0; a < 5; a++) {
            int tmp = (int)
pBlock.distance(evt.getX(), evt.getY());
            if (tmp < min) {
                min = tmp;
                pLine.setLocation(pBlock);
            }

            pBlock.y -= poleSizeCur;
        }

        for (Connector conn : connectors) {
            if (conn.isSelect && !
conn.isNotSelect) {

                if (conn.circle == 1) {
                    conn.line.x1 = pLine.x;
                    conn.line.y1 = pLine.y;

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        } else if (conn.circle == 2) {
            conn.line.x2 = pLine.x;
            conn.line.y2 = pLine.y;
        }

        repaintCircle = true;
    }

    }

    if (repaintCircle) {
        repaint();
    }
    next.repaint();
}

p1.x = p1.x - (p1.x % poleSizeCur);
p1.y = p1.y - (p1.y % poleSizeCur);
//      System.out.println(text);
//      System.out.println(String.valueOf("p1 == " +
p1));

for (int i = 0; i < widthArrayRect; i++) {
    for (int j = 0; j < heightArrayRect; j++) {
        p2 = rects[i][j].getLocation();
        if (p1.equals(p2)) {
            if (rects[i][j].flag == -1) {
                if (evt.isControlDown()) {
                    rects[i][j].flag = flag;
                } else {
                    rects[i][j].flag = flag;
                }
            }
            flagAdd = true;
            repaint();
            break;
        }
    }
}
pStart.setLocation(p1);

}

}
);

circle.setLocation(0, 0);
circle.setSize(10, 10);
circle.setVisible(false);
add(circle);

```

```

        //переопределили Панель для рисования линий
panelLine = new JPanel(new BorderLayout()) {
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Apply our own painting effect
        Graphics2D g2d = (Graphics2D) g;
        // 50% transparent Alpha
        g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_O
VER, 0.0f)); //задание прозрачности фона, НЕ ТРОГАТЬ!
    }

};

//Add Line To Panel
panelLine.setOpaque(false); // установка прозрачного
фона
panelLine.setSize(getSize()); // задание размеров
panelLine.setVisible(true); // делаем панель видимой
add(panelLine, 0);

selectPanel = new SelectPanel();
selectPanel.setVisible(true); // делаем панель видимой
add(selectPanel, 0);

//      conn = new Connector(new Point(600, 200), new
Point(700, 200));
//      panelLine.add(conn);
//      connectors.addLast(conn);
//      conn = new Connector(new Point(700, 200), new
Point(700, 400));
//      panelLine.add(conn);
//      connectors.addLast(conn);
//      conn = new Connector(new Point(700, 400), new
Point(600, 400));
//      panelLine.add(conn);
//      connectors.addLast(conn);
//      conn = new Connector(new Point(600, 400), new
Point(600, 600));
//      panelLine.add(conn);
//      connectors.addLast(conn);
//      conn = new Connector(new Point(600, 600), new
Point(800, 800));
//      panelLine.add(conn);
//      connectors.addLast(conn);
}

```

```

@Override

protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    //      long begin1 = System.nanoTime();
    //-----
    for (int i = 0; i < widthArrayRect; i++) {
        for (int j = 0; j < heightArrayRect; j++) {
            switch (rects[i][j].flag) {
                case -1:
                    g.setColor(rects[i][j].color);
                    g.drawRect(rects[i][j].x, rects[i][j].y,
rects[i][j].width, rects[i][j].height);
                    break;
                case -2:
                    rects[i][j].flag = -1;
                    g.setColor(Color.green);
                    g.fillRect(rects[i][j].x, rects[i][j].y,
rects[i][j].width, rects[i][j].height);

                    break;
                case -3:
                    rects[i][j].flag = -1;
                    g.setColor(Color.blue);
                    g.fillRect(rects[i][j].x, rects[i][j].y,
rects[i][j].width, rects[i][j].height);
                    break;
                default:
                    g.setColor(Color.red);
                    g.fillRect(rects[i][j].x, rects[i][j].y,
rects[i][j].width, rects[i][j].height);
                    break;
            }
        }
    }
    //      long end1 = System.nanoTime();
    //
    //      long begin = System.nanoTime();
    //      g.drawImage(bImg, 0, 0, null);
    //      long end = System.nanoTime();
    //
    //      System.out.printf("1-%d \n2-%d\n", end1 - begin1, end
- begin);

    //-----
    for (Line line : lines) {
        g.setColor(line.color);
        if (line.h == 1) {
            g.drawLine(line.x1, line.y1, line.x2, line.y2);
        } else {
            g.drawLine(line.x1 - 1, line.y1, line.x2 - 1,
line.y2);

```

```

        g.drawLine(line.x1, line.y1 - 1, line.x2,
line.y2 - 1);
        g.drawLine(line.x1 + 1, line.y1, line.x2 + 1,
line.y2);
        g.drawLine(line.x1, line.y1 + 1, line.x2,
line.y2 + 1);
    }
    }

    //-----
    // Условия, чтобы не перерисовывалась за пределами
панели
    if (circle.isVisible() && circle.getX() >= 0 && circle.getY() >=
0 && !flagDragged) {
        circle.paint(g);
    }

    //Данная последовательность Важна для прорисовки
    //Первый цикл для отрисовки всех линий
    //Второй для прорисовки кружков выбранных линий
    //-----
    for (Connectorconn : connectors) {
        conn.paintComponent(g); // вывод каждого
компонента(необходимо для мгновенной отрисовки линий)
    }
    for (Connectorconn : connectors) {
        conn.paintComponentLine(g); // вывод каждого
компонента(необходимо для мгновенной отрисовки линий)
    }
    //-----

    for (TextBlock textblk : ListTextBlock) {
        textblk.paintComponents(g);
    }
}

//=====
public boolean setLocationBlock(Point pNew, Point pOld, int
id) {

    boolean result = true;
    for (int i = 0; i < widthRect; i++) {
        for (int j = 0; j < heightRect; j++) {

            //            if (pNew.y + j > 30) {
            //                System.out.println("Индекс больше 30
[setBlock()]");
            //            }
            if (rects[pNew.x + i][pNew.y + j].flag != id &&
rects[pNew.x + i][pNew.y + j].flag != -1) {
                result = false;
                break;
            }
        }
    }
}

```

```

        }
    }
}
if (result) {
    for (int i = 0; i < widthRect; i++) {
        for (int j = 0; j < heightRect; j++) {
            rects[pOld.x + i][pOld.y + j].flag = -1;
            rects[pNew.x + i][pNew.y + j].flag = id;
        }
    }
    repaint();
}
return result;
}

public void setLocationBlock(Block blk) {
    for (int i = 0; i < widthRect; i++) {
        for (int j = 0; j < heightRect; j++) {
            rects[blk.idRect.x + i][blk.idRect.y + j].flag =
blk.id;
        }
    }
}

public void setSizeBlock(boolean flag) {

    poleSizeCur = (int) (poleSizeConst * zoom);

    int x1 = (int) (sizeFrame.width * zoom), y1 = (int)
(sizeFrame.height * zoom);

    setSize(x1, y1);
    sizeCurFrame.setSize(x1, y1);

    //      ((JViewport)this.getParent())
    //      setLocation(
    //          getX() + (int) ((zoomOld *
spt.viewport.getViewRect().getCenterX() - x1) / 2),
    //          getY() + (int) ((zoomOld *
spt.viewport.getViewRect().getCenterY() - y1) / 2)
    //      );
    setLocation(((int) ((JViewport)
this.getParent()).getViewRect().getCenterX() - x1 / 2),
                (int) ((JViewport)
this.getParent()).getViewRect().getCenterY() - y1 / 2);
    //      setLocation(((int)
spt.viewport.getViewRect().getCenterX() - x1 / 2), (int)
spt.viewport.getViewRect().getCenterY() - y1 / 2);

    for (int i = 0, i2 = 0; i < x1 && i2 < widthArrayRect; i
+= poleSizeCur, i2++) {

```

```

        for (int j = 0, j2 = 0; j < y1 && j2 <
heightArrayRect; j += poleSizeCur, j2++) {
            rects[i2][j2].setBounds(i, j, poleSizeCur,
poleSizeCur);
        }
    }

    int a = (int) (zoom * widthBlock), b = (int) (zoom *
heightBlock);
    float c = flag ? 1.0f : -1.0f;
    Iterator<Block> iter = ListBlock.iterator();
    while (iter.hasNext()) {
        Block next = iter.next();
        next.setMyLocation(a, b, c);
        setLocationBlock(next);
    }

    //-----
    lines.clear();

    int poleSize3 = poleSizeCur * 5;

    for (int i = 0; i < x1; i += poleSize3) {
        lines.add(new Line(i, 0, i, y1, Color.black));
    }
    for (int j = 0; j < y1; j += poleSize3) {
        lines.add(new Line(0, j, x1, j, Color.black));
    }

    zoomOld = zoom;
}

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 void addBlock(short idx) {

```

```

        addBlock(idx, pStart, true, m);
    }

    public void addBlock(int idx, Point loc) {
        addBlock(idx, loc, true, m);
    }

    public Block addBlock(int idx, Point loc, boolean restore,
short[] m1) {
        Extract((short) idx);

        Block blk = new Block(idx, loc, this, indexList, m1);

        blk.TEST1 = counterTest;
        counterTest++;

        SerializeBlock s1 = new SerializeBlock(blk, 0, this);

//        Connector conn2 = new Connector(new Point(10, 10),
pStart);
//        SerializeConn s2 = new SerializeConn(conn2);
//
//        listUndo.add(s1);
//        listUndo.add(s2);
//        listUndo.undo(0).undo();
//        listUndo.undo(0).redo();
//
//        listUndo.undo(1).undo();
//        listUndo.undo(1).redo();
//        blk.addComponentListener(new
java.awt.event.ComponentAdapter() {
//            @Override
//            public void
componentMoved(java.awt.event.ComponentEvent evt) {
////                if (flagAdd) {
////                    SerializeBlock s1 = new
SerializeBlock(blk);
////                    listEdit.addLast(s1);
////                    flagAdd = false;
////                }
//            }
//        });
//        blk.setBorder(getBorder());
//        blk.setBorder( BorderFactory.createMatteBorder(1, 5,
1, 1, Color.red));
        blk.setBounds((int) (loc.x * zoom), (int) (loc.y *
zoom), (int) (widthBlock * zoom), (int) (heightBlock * zoom));
//        blk.ba.setEditable(false);
//        blk.ba.setFocusable(false);
//        blk.ba.setMargin(new Insets(10, 10, 10, 10));
//        blk.ba.setAutoscrolls(false);

```

```

        blk.idRect.setLocation(loc.x / poleSizeCur, loc.y /
poleSizeCur);

        int a = (int) (zoom * widthBlock), b = (int) (zoom *
heightBlock);
        float c;
        if (ListBlock.isEmpty()) {
            c = this.getFont().getSize();
        } else {
            c = ListBlock.getFirst().getFont().getSize();
        }
        Font font = blk.getFont();
        blk.setFont(font.deriveFont((c)));
        blk.setEditable(false);
        blk.setFocusable(false);
        blk.setMargin(new Insets(10, 10, 10, 10));
        blk.setAutoscrolls(false);
        blk.setMyLocation(a, b, 0);
//
        blk.history = new SerializeBlock(blk, 0, this);
        if (restore) {
            blk.addStep(0);
            blk.addStep(2);
        }
        //          jPopupMenuBlock

        blk.setComponentPopupMenu(jPopupMenuBlock);
        //setLocationBlock(blk);
        if (idPzuArray.indexOf(new IdPzu((idx), (int) 0)) == -1)
        {
            idPzuArray.add(new IdPzu((idx), (int) 0));
        }
        ListBlock.add(blk);
        add(blk, 1);

        return blk;
    }

    //=====
    public Connector addConnector(Point p1, Point p2) {
        conn = new Connector(p1, p2, this);

        panelLine.add(conn);
        connectors.addLast(conn);
        repaint();
        //conn.history = new SerializeConn(conn, 0, this);

        return conn;
    }

    public void addPointConnector(Point p) {
        if (startConnector) {

```

```

        //conn.add(p);
    }
}

public void addPointConnector(Point p1, Point loc) {
    if (startConnector) {
        Point p = new Point(p1.x + loc.x, p1.y + loc.y);
        p.x = p.x - (p.x % poleSizeCur);
        p.y = p.y - (p.y % poleSizeCur);
        // conn.add(p);

    }
}

//=====
public void addNewTextBlock() {
    addNewTextBlock("Text", pStart);
}

public void addNewTextBlock(String text, Point loc) {
    TextBlock textBlk = new TextBlock(text, loc.x, loc.y,
getFont(), poleSizeCur);
    ListTextBlock.add(textBlk);
    add(textBlk);
}
//=====

public void addZoom() {
    zoom += (double) 0.10;
    correctZoom();
    setSizeBlock(true);
    repaint();
}

public void subZoom() {
    if (zoom > (double) 0.21) {
        zoom -= (double) 0.10;
        correctZoom();
        setSizeBlock(false);
        repaint();
    }
}

private void correctZoom() {
    int b = (int) (zoom * 100);
    if (b % 10 != 0) {
        zoom = (float) ((int) ((b + 5) / 10) * 10) / 100;
    }
}

//

public void addZoom(Point locat) {

```

```

        zoom += (double) 0.10;
        correctZoom();
        setSizeBlock(true, locat);
        repaint();
    }

    public void subZoom(Point locat) {
        if (zoom > (double) 0.41) {
            zoom -= (double) 0.10;
            correctZoom();
            setSizeBlock(false, locat);
            repaint();
        }
    }

    public void setSizeBlock(boolean flag, Point locat) {

        poleSizeCur = (int) (poleSizeConst * zoom);

        int x1 = (int) (sizeFrame.width * zoom), y1 = (int)
(sizeFrame.height * zoom);

        Point r1 = new Point(locat);
        Point r2 = new Point(((JViewport)
this.getParent()).getViewPosition());
        Point r3 = new Point(((JViewport)
this.getParent()).getViewPosition());
        //      Point r2 = new Point(spt.viewport.getViewPosition());
        //      Point r3 = new Point(spt.viewport.getViewPosition());
        //
        //      r1.x = (r1.x - r2.x);
        //      r1.y = (r1.y - r2.y);

        r2.x = r2.x - ((r2.x) * (int) zoomOld);
        r2.y = r2.y - ((r2.y) * (int) zoomOld);
        //
        //      r3.x = r3.x + (r3.x - r2.x);
        //      r3.y = r3.y + (r3.y - r2.y);

        //      r1.x = r1.x - locat.x;
        //      r1.y = r1.y - locat.y;
        setSize(x1, y1);
        sizeCurFrame.setSize(x1, y1);
        ((JViewport) this.getParent()).setViewPosition(r2);
        //      setLocation(
        //          getX() + (int) ((zoomOld *
(spt.viewport.getViewRect().getCenterX() - x1)) / 2),
        //          getY() + (int) ((zoomOld *
(spt.viewport.getViewRect().getCenterY() - y1)) / 2)
        //      );
    }

```

```

        //setLocation(((int)
spt.viewport.getViewRect().getCenterX() - x1 / 2), (int)
spt.viewport.getViewRect().getCenterY() - y1 / 2);
//setLocation(r1);

        for (int i = 0, i2 = 0; i < x1 && i2 < widthArrayRect; i
+= poleSizeCur, i2++) {
            for (int j = 0, j2 = 0; j < y1 && j2 <
heightArrayRect; j += poleSizeCur, j2++) {
                rects[i2][j2].setBounds(i, j, poleSizeCur,
poleSizeCur);
            }
        }

        int a = (int) (zoom * widthBlock), b = (int) (zoom *
heightBlock);
        float c = flag ? 1.0f : -1.0f;
        Iterator<Block> iter = ListBlock.iterator();
        while (iter.hasNext()) {
            Block next = iter.next();
            if (next.indexList == indexList) {
                next.setMyLocation(a, b, c);
                setLocationBlock(next);
            }
        }

        //-----
        for (Connector conn : connectors) {
            conn.setLocationLine(zoom);
        }
        for (TextBlock txtblk : ListTextBlock) {
            txtblk.setLocationTextBlock((float) zoom,
poleSizeCur);
        }
        panelLine.setSize(getSize()); // задание размеров

        //-----
        //
        // lines.clear();
        //
        // int poleSize3 = poleSizeCur * 5;
        //
        // for (int i = 0; i < x1; i += poleSize3) {
        //     lines.add(new Line(i, 0, i, y1, Color.black));
        // }
        // for (int j = 0; j < y1; j += poleSize3) {
        //     lines.add(new Line(0, j, x1, j, Color.black));
        // }
        zoomOld = zoom;
        repaint();
    }

    private final int CELL_SIZE = 10; // размер ячейки сетки

```

```

        // Предпочтительный размер компонента
        @Override
        public Dimension getPreferredSize() {
            return new Dimension(sizeCurFrame.width,
sizeCurFrame.height);
            //return new Dimension(CELL_SIZE * CELL_COUNT, CELL_SIZE
* CELL_COUNT);
        }

        @Override
        // Предпочтительный размер области прокрутки
        public Dimension getPreferredSizeScrollableViewportSize() {
            return getPreferredSize();
        }
        // Приращение при прокрутке на один элемент

        @Override
        public int getScrollableUnitIncrement(Rectangle visible, int
or, int dir) {
            return CELL_SIZE;
        }
        // Приращение при прокрутке "блоком"

        @Override
        public int getScrollableBlockIncrement(Rectangle visible,
int or, int dir) {
            return CELL_SIZE * 10;
        }
        // Контроль размера области прокрутки

        @Override
        public boolean getScrollableTracksViewportWidth() {
            return false;
        }

        @Override
        public boolean getScrollableTracksViewportHeight() {
            return false;
        }

        public void replaceBlock(int idx, JLayeredPaneTest jp) {
            // все выделенные
            if (idx == indexList) {
                return;
            }

            for (Block blk : ListBlock) {
                if (blk.indexList == indexList) {
                    if (blk.isSelect) {
                        blk.indexList = idx;
                        blk.vp = jp;
                    }
                }
            }
        }
    }

```

```

        for (int i = 0; i < widthRect; i++) {
            for (int j = 0; j < heightRect; j++) {
                rects[blk.idRect.x + i][blk.idRect.y
+ j].flag = -1;
            }
        }

        remove(blk);
        jp.add(blk);
    }
}
repaint();
}

public Block checkingAllBlock(boolean open) {
//    for (Connector conn : connectors) {
//        conn.isSelect = true;
//    }
//    for (TextBlock conn : ListTextBlock) {
//        conn.isSelect = true;
//    }
    Component[] components = getComponents();
    for (Component component : components) {
        if (ListBlock.indexOf(component) >= 0) {
            Block blk = ((Block) component);
            if (blk.rightClick) {
                blk.isSelect = true;
                if (open) {
                    blk.OpenAnketa();
                } else {
                    return blk;
                }
            }
            break;
        }
    }
    return null;
}

public void selectAllObject(boolean print) {
    for (Connector conn : connectors) {
        conn.isSelect = true;
    }
    for (TextBlock conn : ListTextBlock) {
        conn.isSelect = true;
    }
    Component[] components = getComponents();
    for (Component component : components) {
        if (ListBlock.indexOf(component) >= 0) {

            ((Block) component).isSelect = true;
        }
    }
}

```

```

        }
    }
    if (print) {
        repaint();
    }
}

public void clearSelectAllObject() {
    for (Connector conn : connectors) {
        conn.isSelect = false;
    }
    for (TextBlock conn : ListTextBlock) {
        conn.isSelect = false;
    }

    Component[] components = getComponents();
    for (Component component : components) {
        if (ListBlock.indexOf(component) >= 0) {

            ((Block) component).isSelect = false;
        }
    }

    repaint();
}

public void delete() {
    delete(true);
}

public void delete(boolean restore) {
    boolean repaintLine = false;

    for (Iterator<Connector> it = connectors.iterator();
it.hasNext();) {
        Connector conn = it.next();
        if (conn.isSelect) {
            panelLine.remove(conn);
            it.remove();
            repaintLine = true;
        }
        if (restore) {
            conn.addStep(1);
        }
    }
    for (Iterator<TextBlock> it = ListTextBlock.iterator();
it.hasNext();) {
        TextBlock conn = it.next();
        if (conn.isSelect) {
            remove(conn);
            it.remove();
            repaintLine = true;
        }
    }
}

```

```

        }
    }

    for (Iterator<Block> it = ListBlock.iterator();
it.hasNext();) {
        Blockconn = it.next();

        // Если выделен и находится на текущем листе
if (conn.isSelect && conn.indexList == this.indexList) {
            for (int i = 0; i < widthRect; i++) {
                for (int j = 0; j < heightRect; j++) {
                    rects[conn.idRect.x + i][conn.idRect.y +
j].flag = -1;
                }
            }
            remove(conn);

            if (restore) {
                conn.addStep(1);
            }

            //Обнуление МК после удаления блока
            idPzuArray.findRemove(conn.id);
            pzu[0][conn.id] = 0;
            pzu[1][conn.id] = 0;
            pzu[2][conn.id] = 0;
            pzu[3][conn.id] = 0;
            it.remove();
            repaintLine = true;
        }
    }

    if (repaintLine) {
        repaint();
    }
}

public void deleteAllObject() {

    selectAllObject(false);

    delete();

    ListTextBlock.clear();
    connectors.clear();
    lines.clear();
    panelLine.removeAll();
    repaint();
    //System.out.println(getComponentCount());
}

//=====

```

```

/**
 * addAllObject() - функция добавляет объекты на панель
 *
 * @param mas - массив трех списков сериализации [0] -
 список блоков [1] -
 * список линий [2] - список текстовых блоков
 */
public void addAllObject(Object[] mas) {

    deleteAllObject();    // Удаление всех объектов с
    панели

    LinkedList<SerializeBlock> serialBlk = new LinkedList<>();
    serialBlk.addAll((LinkedList) mas[0]); // Добавление в
    список блоков

    // Создание новых блоков по списку
    for (SerializeBlock blk : serialBlk) {
        addBlock(blk.id, blk.loc);
    }

    LinkedList<SerializeConn> serialConn = new
    LinkedList<>();
    serialConn.addAll((LinkedList) mas[1]); // Добавление в
    список линий
    // Создание новых линий по списку
    for (SerializeConn conn : serialConn) {
        Connector line = new Connector(conn.p1, conn.p2, this);
        panelLine.add(line);
        connectors.addLast(line);
    }

    LinkedList<SerializeText> serialText = new
    LinkedList<>();
    serialText.addAll((LinkedList) mas[2]); // Добавление в
    список текста
    // Создание новых текстовых блоков по списку
    for (SerializeText text : serialText) {
        addNewTextBlock(text.text, text.loc);
    }

    repaint();
}

/**
 * getAllObject() Функция возвращает данные: блоков, линий и
 текста на этом
 * листе в виде массива object[] для сериализации
 *
 * @return массив связанных списков классов для сериализации
 */
public Object[] getAllObject() {

```

```

        // Получение всех элементов на листе
Component[] components = getComponents();

        LinkedList<SerializeBlock> serialBlk = new
LinkedList<>();
        for (Component component : components) {
            // Является ли объект component Блоком
            int i = ListBlock.indexOf(component);
            if (i>= 0) {
                // Если это блок, то добавление его данных в
список для сериал.
serialBlk.add(new SerializeBlock(ListBlock.get(i), 0, this));
            }
        }

        // Добавление данных о линиях в список для сериализации
линий
LinkedList<SerializeConn> serialConn = new LinkedList<>();
        for (Connector conn : connectors) {
            serialConn.add(new SerializeConn(conn, 0, this));
        }

        // Добавление данных о текстовых блоках в список для
сериал. текстБлоках
LinkedList<SerializeText> serialText = new LinkedList<>();
        for (TextBlock text : ListTextBlock) {
            serialText.add(new SerializeText(text));
        }

        Object[] mas = new Object[3];
        mas[0] = serialBlk;        // Блоки
        mas[1] = serialConn;       // Линии
        mas[2] = serialText;       // Текст
        return mas;
    }

    //=====
    /**
     * *
     * Функция для установки координат и размеров прямоугольника
выделения и
     * обход объектов и проверка их на попадание в прямоугольник
Если объект
     * попадает, то он выделяется.
     *
     * @param X0 - координата X перемещаемого угла
прямоугольника
     * @param Y0 - координата Y перемещаемого угла
прямоугольника
     */
    public void updateSelectPanel(int X0, int Y0) {

```

```

        selectPanel.setEnd(X0, Y0);

        int x = selectPanel.getX();
        int y = selectPanel.getY();
        int width = x + selectPanel.getWidth();
        int height = y + selectPanel.getHeight();

        for (TextBlock next : ListTextBlock) {
            next.containsArea(x, y, width, height);
        }
        for (Block next : ListBlock) {
            next.containsArea(x, y, width, height);
        }
        for (Connector next : connectors) {
            next.containsArea(x, y, width, height);
        }
        selectPanel.repaint();
        repaint();
    }

    /**
     * *
     * Не работает! Функция для изменения прямоугольника
    выделения при скролле
     *
     * @param offset - смещение при скролле
     * @param b - true вертикаль, false горизонталь
     */
    public void addSelectPanel(int offset, boolean b) {
        if (b) {
            updateSelectPanel(selectPanel.getX(),
selectPanel.getY() + offset);
        } else {
            updateSelectPanel(selectPanel.getX() + offset,
selectPanel.getY());
        }
    }
}

```

Класс *Block*

```

package t4;

import t4.JLayeredPane.JLayeredPaneTest;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.ActionEvent;

```

```

import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.Serializable;
import javax.swing.BorderFactory;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JTextPane;
import javax.swing.SwingUtilities;
import javax.swing.Timer;
import javax.swing.border.Border;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
import static t3.GlobalVars.listEdit;
import static t3.GlobalVars.pzu;
import t4.JLayeredPane.Serialize.SerializeBlock;

public class Block extends JTextPane implements Serializable {

    //    private static final long serialVersionUID = 101L;
    public short[] m = new short[16];
    public int id;
    String[] m3;
    String[] m4;
    String[] m56;
    String[] m12;
    String[] m13;
    String[] m14;

    String textBlock;

    public int indexList;

    public Point idRect;

    private Point pnt, h, delta;

    private boolean flagDragged = false;

    public transient JLayeredPaneTest vp;
    public LabelId jLabelId;
    public JRadioButton jrb;

    public boolean isSelect = false, isNotSelect = false,
isSelected = false;
    private final Border border, borderSelect;

    public SerializeBlock history;

```

```

public int TEST1 = -1;

public boolean rightClick = false;

public Block(int id, Point pStart, JLayeredPaneTest vp, int
indexList, short[] m1) {
    textBlock = "";
    this.m12 = new String[]{
        "RDI --> D",
        "  ZI --> D",
        "  M1 --> D",
        "PSW --> D",
        "RDI --> D  BI --> RDI",
        "  ZI --> D  BI --> RDI",
        "  M1 --> D  BI --> RDI",
        "RDI-->D BI-->RDI BI-->RK"
    };
    this.m13 = new String[]{
        "PSW<--PSW",
        "PSW<--C16,OVR,Z,F15",
        "PSW<--OVR,Z,F15[+]OVR",
        "PSW<--Y"
    };
    this.m14 = new String[]{
        "NOP",
        "RDO <-- Y",
        "MAR <-- Y",
        "NOP"
    };
    this.m4 = new String[]{
        "JZ",
        "CJS",
        "JMAP",
        "CJP",
        "PUSH",
        "JSRP",
        "CJV",
        "JRP",
        "RFCT",
        "RPCT",
        "CRTN",
        "CJPP",
        "LDCT",
        "LOOP",
        "CONT",
        "JP"
    };
    this.m56 = new String[]{
        "R0",
        "R1",
        "R2",

```

```

        "R3",
        "R4",
        "R5",
        "R6",
        "R7",
        "R10",
        "R11",
        "R12",
        "R13",
        "R14",
        "R15",
        "R16",
        "R17",
        "RK[2/0]",
        "RK[8/6]"

};

this.m3 = new String[]{
    "PSW[0]",
    "PSW[1]",
    "PSW[2]",
    "PSW[3]",
    "PSW[4]",
    "PSW[5]",
    "ПЕРЕНОС СТ",
    "КОМСТ 0", //7
    "FLG[0]", //8
    "FLG[1]",
    "FLG[2]",
    "FLG[3]",
    "FLG[4]",
    "FLG[5]",
    "FLG[6]",
    "FLG[7]", //15
    "УР1", //16
    "УР2.1",
    "УР2.2",
    "УР3",
    "УР4",
    "УР5",
    "RK[11/9]",
    "RK[5/3]"
};

System.arraycopy(m1, 0, this.m, 0, 16);
this.id = id;
this.vp = vp;
this.indexList = indexList;

idRect = new Point(0, 0);
delta = new Point(0, 0);
pnt = new Point(pStart);

```

```

        h = new Point(pStart);

//        ba = new JTextArea();
        jrb = new JRadioButton();
        jrb.setSelected(false);

        jrb.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent e) {
                printMy(pnt);
            }
        });

        jrb.setBounds(0, 20, 20, 20);
        add(jrb);

        jLabelId = new JLabelId();
        jLabelId.setText("0");
        jLabelId.setHorizontalAlignment((int) CENTER_ALIGNMENT);
        jLabelId.setBounds(4, 4, 30, 16);
        int em1 = 1;
//        Border cmp = BorderFactory.createRaisedBevelBorder();
        Border redline2 = BorderFactory.createMatteBorder(em1,
em1, em1, em1, new Color(156, 156, 156));
//        Border cmp = BorderFactory.createCompoundBorder(cmp,
redline2);
        jLabelId.setBorder(redline2);
        add(jLabelId);
//        ba.setEditable(false);
//        ba.setFocusable(false);
////        ba.setMargin(new Insets(10, 10, 10, 10));
//        ba.setAutoscrolls(false);
//        ba.setBounds(5, 5, 120, 90);
//        var b = ba.getMouseListeners();
//        int a = ba.getMouseListeners().length;
//        for (int i = 0; i < a; i++) {
//            removeMouseListener(b[i]);
//        }
//        var c = ba.getMouseMotionListeners();
//        a = ba.getMouseMotionListeners().length;
//        for (int i = 0; i < a; i++) {
//            removeMouseMotionListener(c[i]);
//        }
//
//
//        add(ba);
        setTextBlock();

        printMy(pnt);

//-----
        addMouseListener(new MouseAdapterBlock());

```

```

        addMouseListener(new NodeMouseListener());

        addMouseMotionListener(new MouseMotionAdapterBlock());

        int em = 2;

        Border cmp2 = BorderFactory.createRaisedBevelBorder();
        Border redline = BorderFactory.createMatteBorder(em, em,
em, em, new Color(0, 0, 0));
        cmp2 = BorderFactory.createCompoundBorder(cmp2,
redline);
        border = BorderFactory.createMatteBorder(em, em, em, em,
new Color(0, 0, 0));
        borderSelect = BorderFactory.createMatteBorder(em, em,
em, em, new Color(255, 0, 0));
        StyledDocument doc = getStyledDocument();
        SimpleAttributeSet center = new SimpleAttributeSet();
        StyleConstants.setAlignment(center,
StyleConstants.ALIGN_CENTER);
        //      StyleConstants.setLineSpacing(center, (float) -0.25);
        doc.setParagraphAttributes(0, doc.getLength(), center,
false);

        setBorder(border);

    }

    private void printMy(Point p) {
        jLabelId.setText(Integer.toOctalString(id));
        if (jrb.isSelected()) {
            setText("\n[p]      " + String.valueOf(p.x) + "  " +
String.valueOf(p.y)
                    + "\n[h]      " + String.valueOf(h.x) + "  " +
String.valueOf(h.y)
                    + "\n[idR]    " + String.valueOf(idRect.x) + "
" + String.valueOf(idRect.y)
                );
        } else {
            setText(textBlock);
        }
    }

    public void setMyLocation(int a, int b, float c) {

        // Этот кусок весь добавить надо !
        Point s = new Point(idRect.x * vp.poleSizeCur, idRect.y *
vp.poleSizeCur);
        setBounds(s.x, s.y, a, b);
        Font font = getFont();
        setFont(font.deriveFont((font.getSize() + c)));
        h.setLocation(getLocation());
        printMy(h);
    }

```

```

}

private void setTextBlock() {
    String s5 = "";
    if (m[5] >= 0b11000) {
        s5 += m56[17];
    } else if (m[5] >= 0b10000) {
        s5 += m56[16];
    } else {
        s5 += m56[m[5]];
    }

    String s6 = "";
    if (m[6] >= 0b11000) {
        s6 += m56[17];
    } else if (m[6] >= 0b10000) {
        s6 += m56[16];
    } else {
        s6 += m56[m[6]];
    }

    String s56 = "";
    switch (m[10]) {
        case 0:
            s56 = String.format("%s + %s", s5, s6);
            break;
        case 1:
            s56 = String.format("%s - %s - 1", s6, s5);
            break;
        case 2:
            s56 = String.format("%s - %s - ", s5, s6);
            break;
        case 3:
            s56 = String.format("%s v %s", s5, s6);
            break;
        case 4:
            s56 = String.format("%s & %s", s5, s6);
            break;
        case 5:
            s56 = String.format("NOT(%s) & %s", s5, s6);
            break;
        case 6:
            s56 = String.format("%s [+] %s", s5, s6);
            break;
        case 7:
            s56 = String.format("NOT(%s[+] %s)", s5, s6);
            break;
    }

    if (m[11] == 4 || m[11] == 5) {
        s56 = String.format("(%s)*2", s56);
    } else if (m[11] == 6 || m[11] == 7) {

```



```

        * @param h - координата Y правого нижнего угла
прямоугольника выделения h =
        * Высота+Y
        */
        public void containsArea(int x, int y, int w, int h) {
            if (x < getX() && y < getY()
&& w > getX() + getWidth()
&& h > getY() + getHeight()) {
                isSelect = true;
            } else {
                isSelect = false;
            }
        }

        public void OpenAnketa() {
            Anketa anketa = new Anketa(m);
            System.arraycopy(anketa.m, 0, m, 0, 16);
            id = m[0];
            setTextBlock();
            printMy(pnt);
            pzu[0][id] = m[1];
            pzu[1][id] = (short) (m[6] * 2048 + m[5] * 64 + m[3] * 4
+ m[2]);
            pzu[2][id] = (short) (m[13] * 16384 + m[10] * 2048 +
m[9] * 256 + m[8] * 64 + m[7] * 16 + m[4]);
            pzu[3][id] = (short) (m[15] * 256 + m[14] * 64 + m[12] *
8 + m[11]);
        }

        @Override
        protected void paintComponent(Graphics g) {

            if (!isSelected && isSelect && !vp.selectMouse) {
                isSelected = true;
                setBorder(borderSelect);
            } else if (isSelected && !isSelect && !vp.selectMouse) {
                isSelected = false;
                setBorder(border);
            }
            super.paintComponent(g);
        }

        private class NodeMouseListener extends MouseAdapter
implements Serializable {

            //        private static final long serialVersionUID = 120L;
            private boolean singleClick = true;
            private final Timer timer;

            private final int delay = 200;

            NodeMouseListener() {

```

```

        ActionListener al = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent evt) {
//                if (singleClick) {
//                    // processSingleClick();
//                }
            };
            timer = new Timer(delay, al);
            timer.setRepeats(false);
        }

        @Override
        public void mouseClicked(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();

            if (e.getClickCount() == 1) {
                singleClick = true;
//                System.out.println("Sin");
                timer.start();
            } else {
                singleClick = false;
//                System.out.println("Double");
                OpenAnketa();
            }
        }

        private void processSingleClick() {

        }

    }

    private class MouseAdapterBlock extends MouseAdapter
    implements Serializable {

        @Override
        public void mouseReleased(MouseEvent evt
        ) {
            flagDragged = false;
            isNotSelect = false;
            delta.setLocation(0, 0);

            Point p = getLocation();          // Положение блока

            //-----
            // Если от 0 до 4, то в меньшую сторону, иначе в
большую
            // По горизонтали
            // Вывнести в глобальную функцию

```

```

int a1 = p.x % vp.poleSizeCur;

    if (a1 >= vp.poleSizeHalf) {
        p.x += vp.poleSizeCur;
    }
    p.x -= a1;

    // По вертикали
    a1 = p.y % vp.poleSizeCur;

    if (a1 >= vp.poleSizeHalf) {
        p.y += vp.poleSizeCur;
    }
    p.y -= a1;

    Point p1 = getLocation(); // Положение блока
    if (!(p.x == p1.x && p.y == p1.y)) {
        setLocationBlock(p);
        addStep(2);
    }

    //vp.moveToFront(evt.getComponent());
    int button = evt.getButton();
    switch (button) {
        case MouseEvent.BUTTON1:
            vp.addPointConnector(evt.getPoint(),
getLocation());
            break;
        case MouseEvent.BUTTON2:
            break;
        case MouseEvent.BUTTON3:
            break;
        default:
            break;
    }
    JComponent source = (JComponent) evt.getSource();
    if (!evt.isControlDown()) {

source.getParent().dispatchEvent(SwingUtilities.convertMouseEven
t(source, evt, source.getParent()));
    }

    }

    @Override
    public void mouseClicked(MouseEvent evt
    ) {
        // vp.moveToFront(evt.getComponent());

//            clicksCount++;
//
//            if (clicksCount == 2) { //Or clicksCount%2==0

```

```

//                                int index =
myJList.locationToIndex(evt.getPoint());
//                                if (index >= 0) {
//                                Object o =
theList.getModel().getElementAt(index);
//                                System.out.println("Double-clicked on:
" + o.toString());
//                                }
//                                clicksCount = 0; //If you use clickCounts
%2==0 you don't need this line
//                                }
    }

@Override
public void mousePressed(MouseEvent evt) {
    boolean r = isSelected;
    rightClick = false;
    isSelected = true;
    isNotSelect = true;
    flagDragged = false;

    int button = evt.getButton();
    switch (button) {
        case MouseEvent.BUTTON1:
            flagDragged = true;
            break;
        case MouseEvent.BUTTON3:
            rightClick = true;
            break;
        default:
            break;
    }
    //vp.moveToFront(evt.getComponent());
    JComponent source = (JComponent) evt.getSource();
    if (rightClick && r) {
    } else if (!evt.isControlDown()) {

source.getParent().dispatchEvent(SwingUtilities.convertMouseEven
t(source, evt, source.getParent()));
    }
    source.getParent().repaint();
    }
};

private class MouseMotionAdapterBlock extends MouseAdapter
implements Serializable {

//    private static final long serialVersionUID = 122L;
@Override
public void mouseDragged(MouseEvent evt
) {
    if (flagDragged) {

```

```

        Point p = getLocation();

        if (delta.x == 0 && delta.y == 0) {
            delta.setLocation(evt.getPoint());
        }

        p.setLocation(p.x + evt.getX() - delta.x, p.y +
evt.getY() - delta.y);
        setLocation(p);

        printMy(p);
    }
}

@Override
public void mouseMoved(MouseEvent e) {
    super.mouseMoved(e);
    JComponent source = (JComponent) e.getSource();
    //setBackground(Color.red);

    source.getParent().dispatchEvent(SwingUtilities.convertMouseEven
t(source, e, source.getParent()));
}

};

private class LabelId extends JLabel implements Serializable
{
    //      private static final long serialVersionUID = 124L;
    LabelId() {
        super();
    }
}

};

public void setLocationBlock(Point p) {
    pnt.setLocation(p.x / vp.poleSizeCur, p.y /
vp.poleSizeCur);

    if (p.x >= 0 && p.y >= 0 && vp.setLocationBlock(pnt,
idRect, id)) {
        h.setLocation(p);
        idRect.setLocation(pnt);
    }
    setLocation(h);          // Устанавливаем новую позицию
    //-----
printMy(p);
}

public void addStep(int a) {
    SerializeBlock history2;
    history2 = new SerializeBlock(this, a, vp);
    listEdit.add(history2);
}

```

```

        history = history2;
        //listEdit.restore(history);
    }
}

```

Класс *ExportPDF*

```

package t4;

import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Image;
import com.itextpdf.text.Rectangle;
import com.itextpdf.text.pdf.PdfWriter;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.swing.JFileChooser;
import t4.JLayeredPane.JLayeredPaneTest;

/**
 *
 * @author Вячеслав
 */
public class ExportPDF {

    public boolean printImageToPDF(ArrayList<ScrollPaneTest>
spt, String fileName) throws IOException {

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

            fileopen.setSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            if (fileopen.showDialog(null, "Экспорт файлов") ==
JFileChooser.APPROVE_OPTION) {
                String path =
fileopen.getSelectedFile().getPath();

                //String path = "file/test";
                // Создаем документ укащывая размер А3 в
ПИКЕСЕЛЯХ и отступы полей
Document d = new Document(new Rectangle(1191, 842), 0, 0, 0, 0);
                PdfWriter writer = PdfWriter.getInstance(d, new
FileOutputStream(path + "/PDF.pdf"));

                d.open();
            }
        }
    }
}

```

```

        //Сохраняем каждую страницу в ПДФ и PNG
for (int i = 0; i < spt.size(); i++) {

        JLayeredPaneTest panel = spt.get(i).vp;
//получаем страницу

        BufferedImage bImg = new
BufferedImage(panel.getWidth(),
                panel.getHeight(),
BufferedImage.TYPE_INT_RGB);
        panel.paint(bImg.getGraphics()); //важно
обычный paint

        d.newPage(); // переход на новую страницу
Image iTextImage = Image.getInstance(writer, bImg, 1);
        //Расчет коэффициента масштабирования
        iTextImage.scalePercent(100.0F /
(panel.getWidth() / 1191.0F));
        d.add(iTextImage);

        FileOutputStream f = new
FileOutputStream(path + "/image" + String.valueOf(i) + ".png",
false);

        ImageIO.write(bImg, "png", f);
        f.close();
    }

    d.close();
    writer.close();
    System.out.printf("Экпортировано в PDF и PNG
(%s)\n", path);
    return true;
}
} catch (DocumentException e) {
    // Выход исключения выше
throw new IOException("Ошибка! (DocumentException)");
}
return false;
}
}

```

Класс *Connector*

```

package t4.JLayeredPane;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseAdapter;

```

```

import java.awt.event.MouseEvent;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.Rectangle;
import java.io.Serializable;
import javax.swing.JComponent;
import javax.swing.SwingUtilities;
import static t3.GlobalVars.listEdit;
import static t3.GlobalVars.rounding;
import t4.JLayeredPane.Serialize.SerializeConn;

/**
 *
 * @author Алексей
 */
public class Connector extends JComponent implements
Serializable {

    //    private static final long serialVersionUID = 102L;
    private final transient JLayeredPaneTest vp;

    public Line2D.Double line = new Line2D.Double(0, 0, 0, 0);

    private Point2D.Double temp = new Point2D.Double(0, 0);

    private float depthDefault = 4.0F;        // постоянный размер
    private float depth = depthDefault;        // переменный размер

    public boolean isSelect = false, isNotSelect = false;

    public short circle = 0;                    // показатель
выбранной вершины

    double zoomOld = 1.0;

    double radius = 5;

    public SerializeConn history;

    /**
     * Конструктор
     *
     * @param start первая вершина линии
     * @param end вторая вершина линии
     */
    public Connector(Point start, Point end, JLayeredPaneTest jvp) {
        this.vp = jvp;
        line = new Line2D.Double(start.x, start.y, end.x,
end.y);
        temp = new Point2D.Double(0, 0);
    }

```

```

        setSelect(false);

        addMouseListener(new MouseAdapterLine());

        addMouseMotionListener(new MouseMotionAdapterLine());
        //addStep(0);
        //addStep(2);

        repaint();
    }

    public Connector(JLayeredPaneTest jvp) {
        this.vp = jvp;
        line = new Line2D.Double(0, 0, 0, 0);
        temp = new Point2D.Double(0, 0);

        setSelect(false);

        addMouseListener(new MouseAdapterLine());

        addMouseMotionListener(new MouseMotionAdapterLine());
        //addStep(0);
        //addStep(2);
        repaint();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.BLACK); // цвет

        g2.setStroke(new BasicStroke(depth,
        BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND)); // задание
        толшины
        g2.draw(line); // отрисовка линии с заданными параметрам

    }

    void paintComponentLine(Graphics g) {

        int rad = (int) radius;
        int diameter = rad + rad;
        if (circle != 0 || isSelect) {
            Graphics2D g2 = (Graphics2D) g;
            g2.setColor(Color.YELLOW);
            if (circle == 1) {
                g2.fillOval((int) line.x1 - rad, (int) line.y1 -
rad, diameter, diameter);
            } else if (circle == 2) {

```

```

        g2.fillOval((int) line.x2 - rad, (int) line.y2 -
rad, diameter, diameter);
    } else if (circle == 3 || isSelected) {
        g2.fillOval((int) line.x1 - rad, (int) line.y1 -
rad, diameter, diameter);
        g2.fillOval((int) line.x2 - rad, (int) line.y2 -
rad, diameter, diameter);

    }

}

}

//Получение размеров по умолчанию
@Override
public Dimension getPreferredSize() {

    Rectangle bounds = line.getBounds();

    int width = bounds.x + bounds.width;
    int height = bounds.y + bounds.height;

    return new Dimension(width, height);
}

/**
 * Проверка принадлежности точка линии depthDefault - радиус
захвата
 *
 * @param x координата мыши X
 * @param y координата мыши Y
 * @return true/false (включает/нет)
 */
@Override
public boolean contains(int x, int y) {
    double distance = line.ptSegDist((double) x, (double)
y);
    returndistance<depthDefault;
}

/**
 * Проверка к какой точке ближе координата
 *
 * @param x координата мыши X
 * @param y координата мыши Y
 * @return 1 - ближе к 1 точке; 2 - ближе к 2 точке; 3 -
далеко от вершин
 * линий, перенос всей линии;
 *
 */
public short containsEquals(int x, int y) {

```

```

        if (line.getP1().distance(x, y) < depthDefault) {
            return 1;
        } else if (line.getP2().distance(x, y) < depthDefault) {
            return 2;
        } else {
            return 3;
        }
    }

    /**
     * Установка начальных параметров линии
     *
     * @param flag true/false (установка/сброс выделения)
     */
    public void setSelect(boolean flag) {
        if (flag) {
            depth = depthDefault * 2.0f;
        } else {
            depth = depthDefault;
        }
        isSelect = flag;
    }

    /**
     * *
     * Функция, для проверки линии, на попадание ее в
     * прямоугольник выделения
     *
     * @param x - координата X левого верхнего угла
     * прямоугольника выделения
     * @param y - координата Y левого верхнего угла
     * прямоугольника выделения
     *
     * @param w - координата X правого нижнего угла
     * прямоугольника выделения w =
     * Ширина+X
     * @param h - координата Y правого нижнего угла
     * прямоугольника выделения h =
     * Высота+Y
     *
     * У линии две вершины, поэтому необходимо определить какая
     * левее и выше
     * относительно нее и будем вычислять
     *
     * Если первая левее и выше, то выполнение первого условия,
     * иначе второго
     * Если точки принадлежат прямоугольнику, значит выделяем
     * линию
     */
    public void containsArea(int x, int y, int w, int h) {
        if (line.x1 < line.x2 || line.y1 < line.y2) {
            if (x < line.x1

```

```

    && y < line.y1
    && w > line.x2
    && h > line.y2) {
        if (!isSelect) {
            setSelect(true);    // Установка выделения
        }
        } else {
            setSelect(false);    // Сброс выделения
        }
    } else {
        if (x < line.x2
    && y < line.y2
    && w > line.x1
    && h > line.y1) {
            if (!isSelect) {
                setSelect(true);    // Установка выделения
            }
            } else {
                setSelect(false);    // Сброс выделения
            }
        }
    }

    public void setLocationLine(double zoom) {

        line.x1 = line.x1 / zoomOld * zoom;
        line.y1 = line.y1 / zoomOld * zoom;
        line.x2 = line.x2 / zoomOld * zoom;
        line.y2 = line.y2 / zoomOld * zoom;

        depthDefault = (float) (depthDefault / zoomOld * zoom);
        setSelect(isSelect);

        radius = (radius / zoomOld * zoom);

        zoomOld = zoom;
    }

    private class MouseAdapterLine extends MouseAdapter
    implements Serializable {

        //      private static final long serialVersionUID = 125L;
        @Override
        public void mousePressed(MouseEvent evt) {
            super.mousePressed(evt);
            if (evt.getButton() == MouseEvent.BUTTON1) {
                temp.setLocation(evt.getX(), evt.getY());

                circle = containsEquals(evt.getX(), evt.getY());

                setSelect(true);
                isNotSelect = true;
            }
        }
    }

```

```

        JComponent source = (JComponent)
evt.getSource();

        //Если не нажат CTRL то сброс предыдущего
выделения
if (!evt.isControlDown()) {

    source.getParent().getParent().dispatchEvent(SwingUtilities.conv
ertMouseEvent(source, evt, source.getParent().getParent()));
    }
    source.getParent().getParent().repaint();

    } else {
        circle = 0;
    }

}

@Override
public void mouseReleased(MouseEvent evt) {
    super.mouseReleased(evt);
    isNotSelect = false;
    if (evt.getButton() == MouseEvent.BUTTON1) {
        if (circle == 1 || circle == 3) {
            line.x1 = rounding(line.getP1().getX(), 10,
0);
            line.y1 = rounding(line.getP1().getY(), 10,
0);
        }
        if (circle == 2 || circle == 3) {
            line.x2 = rounding(line.getP2().getX(), 10,
0);
            line.y2 = rounding(line.getP2().getY(), 10,
0);
        }

        temp.setLocation(0, 0);

        JComponent source = (JComponent)
evt.getSource();
        source.getParent().getParent().repaint();

        source.getParent().getParent().dispatchEvent(SwingUtilities.conv
ertMouseEvent(source, evt, source.getParent().getParent()));
        }
        circle = 0;
        addStep(2);
    }

};

```

```

private class MouseMotionAdapterLine extends MouseAdapter
implements Serializable {

//      private static final long serialVersionUID = 126L;
@Override
public void mouseDragged(MouseEvent evt) {
    super.mouseDragged(evt);
    if (circle != 0) {
        //если ближе к началу, то изменяем начало
        //если ближе к концу, то изменяем координату
второй точки
        /**
         * слушатель
         */
        switch (circle) {
            case 1:
                line.x1 = evt.getX();
                line.y1 = evt.getY();
                break;
            case 2:
                line.x2 = evt.getX();
                line.y2 = evt.getY();
                break;
            default:
                int offsetX = (int) temp.x - evt.getX();
                int offsetY = (int) temp.y - evt.getY();
                line.x1 -= offsetX;
                line.y1 -= offsetY;

                line.x2 -= offsetX;
                line.y2 -= offsetY;

                temp.x = evt.getX();
                temp.y = evt.getY();
                break;
        }

        //для постоянной перерисовки, обращение к
родителю(работает только так!)
JComponent source = (JComponent) evt.getSource();

source.getParent().getParent().dispatchEvent(SwingUtilities.conv
ertMouseEvent(source, evt, source.getParent().getParent()));
        source.getParent().getParent().repaint();

    }
}

@Override
public void mouseMoved(MouseEvent evt) {
    super.mouseMoved(evt);
}

```

```

        circle = containsEquals(evt.getX(), evt.getY());

        JComponent source = (JComponent) evt.getSource();

source.getParent().getParent().dispatchEvent(SwingUtilities.convertMouseEvent(source, evt, source.getParent().getParent()));
        source.getParent().getParent().repaint();

    }

};

public void addStep(int a) {
    listEdit.add(history);
    history = new SerializeConn(this, a, vp);
    listEdit.restore(history);
}

}

```

Класс *Circle*

```

package t4.JLayeredPane;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;
import javax.swing.JPanel;

public class Circle extends JPanel implements Serializable {

//    private static final long serialVersionUID = 106L;
    Circle() {

    }

    @Override
    protected void paintComponent(Graphics g) {
        g.setColor(Color.PINK);
        g.fillOval(0, 0, 10, 10);
    }

}

```

Класс *Line*

```

package t4.JLayeredPane;

import java.awt.Color;
import java.io.Serializable;

public class Line implements Serializable {

```

```

        final int x1;
        final int y1;
        final int x2;
        final int y2;
        final Color color;
        final int h;

        public Line(int x1, int y1, int x2, int y2, Color color) {
            this.x1 = x1;
            this.y1 = y1;
            this.x2 = x2;
            this.y2 = y2;
            this.h = 1;
            this.color = color;
        }
    }
}

```

Класс *Select*

```

package t4.JLayeredPane;

import java.awt.Color;
import java.awt.Point;
import javax.swing.JPanel;

/**
 * @author Алексей
 * Прямоугольник выделения объектов
 */
public class SelectPanel extends JPanel {

    SelectPanel() {
        // установка полупрозрачного фона
        setBackground(new Color(0, 100, 220, 100));
    }

    public boolean selectMouse = false;

    /**
     * *
     * Начальной точка прямоугольника, относительно которой
     * вычисляется размер
     * прямоугольника и его положение. Задается при нажатии
     * MousePress в
     * JLayeredPane
     * *
     * @see JLayeredPaneTest
     */
    private int startX = 0, startY = 0;

    /**

```

```

    * Установка начальной точки прямоугольника выделения
    * @param point - координаты вершины прямоугольника
    * Сброс размеров прямоугольника, установка координат
прямоугольника
    *
    */
    public void setStart(Point point) {
        setLocation(point);
        setSize(0, 0);
        startX = point.x;
        startY = point.y;
    }

    /**
     * Функция устанавливает размеры и координаты прямоугольника
выделения
     * @param x - координата X, (мыши) (любой крайний угол,
который изменяется)
     * @param y - координата Y, (мыши) (любой крайний угол,
который изменяется)
     *
     * Если Изменений нет, то выход из функции
     *
     * Если перемещаемый угол прямоугольника правее точки
старта,
     * значит просто увеличивает размер прямоугольника. Левая
вехняя вершина
     * сохраняет координату x. Иначе (если левее), необходимо
переместить
     * прямоугольни в координаты (x) и задать новую ширину.
     *
     * Аналогично вычисляется высота и координата y.
     */
    public void setEnd(int x, int y) {
        if (startX != x || startY != y) {    // если без
изменений, то пропуск
            int width;
            if (startX < x) {
                width = x - startX;
                x = startX;
            } else {
                width = startX - x;
            }

            int height;
            if (startY < y) {
                height = y - startY;
                y = startY;
            } else {
                height = startY - y;
            }
        }
    }

```

```

        setBounds(x, y, width, height);    // установка
вычисленных значений
        repaint();
    }
}
height = startY - endY;

```

Класс *TextBlock*

```

package t4.JLayeredPane;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.Serializable;
import javax.swing.JComponent;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import static t3.GlobalVars.rounding;

/**
 * @author Вячеслав
 */
public class TextBlock extends JTextArea implements Serializable
{
    //    private static final long serialVersionUID = 108L;
    private boolean flagDragged = false;
    private Point pnt, h, delta;
    public boolean isSelect = false, isNotSelect = false;
    private int stepVP;

    TextBlock(String text, int x, int y, Font font, int step) {

        setText(text);
        setFont(new Font(font.getFamily(), font.getStyle(),
18));
        setSize(getPreferredSize());
        setLocation(x, y);
        delta = new Point(0, 0);
        pnt = new Point(x, y);
        h = new Point(x, y);
    }
}

```

```

        stepVP = step;

        setOpaque(false); // установка прозрачного фона
//        setBackground(new Color(255, 255, 255));

//        setSize(getSize()); // задание размеров
        setVisible(true); // делаем панель видимой
//        setBackground(Color.red);

        // jTextArea.setBackground(null);
//        //-----
        addKeyListener(new KeyListenerText());

        addMouseListener(new MouseAdapterText());

        addMouseMotionListener(new MouseMotionAdapterText());
    }

    public void containsArea(int x, int y, int w, int h) {
        if (x < getX() && y < getY()
        && w > getX() + getWidth()
        && h > getY() + getHeight()) {
            isSelect = true;
        } else {
            isSelect = false;
        }
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        if (isSelect) {
            setForeground(Color.red);
        } else {
            setForeground(Color.black);
        }

        //Apply our own painting effect
        Graphics2D g2d = (Graphics2D) g;
        // 50% transparent Alpha
        g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_O
VER, 0.0f)); //задание прозрачности фона, НЕ ТРОГАТЬ!
        //        g2d.drawString(getText(), getX(), getY());

        // jTextArea.paintComponents(g2d);
    }

```

```

        private class KeyListenerText implements KeyListener,
Serializable {

//            private static final long serialVersionUID = 130L;
            @Override
            public void keyTyped(KeyEvent e) {
                setSize(getPreferredSize().width + 50,
getPreferredSize().height + 50);
            }

            @Override
            public void keyPressed(KeyEvent e) {
                setSize(getPreferredSize().width + 50,
getPreferredSize().height + 50);
            }

            @Override
            public void keyReleased(KeyEvent e) {
                setSize(getPreferredSize().width + 50,
getPreferredSize().height + 50);
            }

        };

        private class MouseAdapterText extends MouseAdapter
implements Serializable {

//            private static final long serialVersionUID = 131L;
            private int clicksCount = 0;

            @Override
            public void mouseReleased(MouseEvent evt
            ) {
                isNotSelect = false;
                flagDragged = false;
                delta.setLocation(0, 0);

                setLocation(rounding(getX(), stepVP,
0),rounding(getY(), stepVP, 0));
                Point p = getLocation();          // Положение блока
                repaint();
            }

            @Override
            public void mousePressed(MouseEvent evt) {

                JComponent source = (JComponent) evt.getSource();
                if (evt.getButton() == MouseEvent.BUTTON1) {
                    flagDragged = true;
                    isNotSelect = true;
                    isSelect = true;
                }
            }
        }
    }

```

```

        setForeground(Color.red);

        if (!evt.isControlDown()) {

source.getParent().dispatchEvent(SwingUtilities.convertMouseEvent
(source, evt, source.getParent()));
        }
        } else {
            flagDragged = false;
        }

        //source.getParent().getParent().repaint();
    }

};

private class MouseMotionAdapterText extends MouseAdapter
implements Serializable {

//        private static final long serialVersionUID = 132L;
@Override
    public void mouseDragged(MouseEvent evt
    ) {
        if (flagDragged) {
            Point p = getLocation();

            if (delta.x == 0 && delta.y == 0) {
                delta.setLocation(evt.getPoint());
            }

            p.setLocation(p.x + evt.getX() - delta.x, p.y +
evt.getY() - delta.y);
            setLocation(p);
        }

        //jTextArea.repaint();
        //jTextArea.paintComponents(getGraphics());
    }

};

float zoomOld = 1.0f;

public void setLocationTextBlock(float zoom, int step) {

    int X = (int) (getX() / zoomOld * zoom), Y = (int)
(getY() / zoomOld * zoom);

    setLocation(rounding(X, step, 0), rounding(Y, step, 0));

    Font font = getFont();

```

```

        setFont(font.deriveFont((font.getSize() / zoomOld *
zoom))));

        setSize(getPreferredSize());
        zoomOld = zoom;
        stepVP = step;
    }

}

```

Класс *Anketa*

```

package t4;

import java.text.DecimalFormat;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JRadioButton;
import javax.swing.JTable;
import javax.swing.JTextField;
import t3.GlobalVars;
import static t3.GlobalVars.decToOctStr;
import static t3.GlobalVars.octToDecInt;

public class Anketa extends javax.swing.JDialog {

    short[] m = new short[16];
    short[] mBuffer;
    private boolean flagM7 = false;
    boolean mask = true;
    JTextField[] jTextArray;

    public Anketa(short[] m1) {

        initComponents();

        mBuffer = m1;
        //this.m = m1;
        System.arraycopy(m1, 0, m, 0, 16);

        //-----
        // Для отладки
        // Вывод полученной МК в поля
        jTextArray = new JTextField[]{
            TextAddrNext,
                TextM21,
                TextM22,
                TextM23,
                TextM24,

```

```

        TextM25,
        TextM26,
        TextM27,
        TextM28,
        TextM29,
        TextM30,
        TextM31,
        TextM32,
        TextM33,
        TextM34,
        TextM35
    };

    //-----
    printM();
    //-----
    setTextFieldM1(m[1]);
    //-----
    setComboBoxM2(m[2], true);

    setComboBoxM3(m[2], m[3], true);
    //-----
    jComboBoxM4.setSelectedIndex(m[4]);
    //-----
    setComboBoxM56(m[5], false);
    setComboBoxM56(m[6], true);
    //-----
    // FlagM7 исключает повторное добавление элементов в
    поле M7
    flagM7 = ((m[11] & (short) 2) == 0);
    setComboBoxM7(m[7], m[11], true);
    //-----
    jComboBoxM8.setSelectedIndex(m[8]);
    jComboBoxM9.setSelectedIndex(m[9]);
    jComboBoxM10.setSelectedIndex(m[10]);
    jComboBoxM11.setSelectedIndex(m[11]);
    jComboBoxM12.setSelectedIndex(m[12]);
    jComboBoxM13.setSelectedIndex(m[13]);
    jComboBoxM14.setSelectedIndex(m[14]);
    jComboBoxM15.setSelectedIndex(m[15]);

jScrollPaneSettingFull.setVerticalScrollBar().setUnitIncrement(2
0);    // по умолчанию было 1

//      int w = jTabbedPane1.getWidth();
//      int h = jTabbedPane1.getHeight();
//      JPanel[] jp = new JPanel[7];
//
//      jp[0] = func1(jPanel0, w, h);
//      jp[1] = func1(jPanel1, w, h);
//      jp[2] = func1(jPanel2, w, h);

```

```

//      jp[3] = func1(jPanel3, w, h);
////      jp[4] = func1(jPanel4, w, h);
//      for (int i = 4; i < 7; i++) {
//          jp[i] = new JPanel();
//          jp[i].setVisible(true);
//          jp[i].setBounds(0, 0, w, h);
//      }
//      jTabbedPane1.addTab("Входная и выходная шина", jp[0]);
//      jTabbedPane1.addTab("Мультиплексор адреса", jp[1]);
//      jTabbedPane1.addTab("Операция над данными", jp[2]);
//      jTabbedPane1.addTab("Формирование PSW", jp[3]);
//      jTabbedPane1.addTab("Приемник результата", jp[4]);
//      jTabbedPane1.addTab("Переходы и условия переходов",
jp[5]);
//      jTabbedPane1.addTab("Поля констант", jp[6]);
//
//      jTabbedPane1.setVisible(false);
//      jComboBox13.setEditor(new MyComboBoxEditor());
//      jComboBox13.setEditable(true);
//      jTextFieldM0.setInputVerifier(new
GlobalVars.InputVerifierOct());
//      jTextFieldM0.setText(decToOctStr(m[0]));
//      setLocationRelativeTo(null);
//      pack();

//      jScrollPaneSettingFull.setSize(700, 500);
//      jScrollPaneSettingFull.setLocation(0, 0);
//      jScrollPaneSettingFull.setVisible(true);
//      setVisible(true);
}

//-----
// Для отладки
// Печать полученной МК каждого поля отдельно в jTextField
private void printM() {
    for (int i = 0; i < 16; i++) {
        jTextArea[i].setText(Integer.toString(m[i], 8));
    }
}

private void
jButtonSaveActionPerformed(java.awt.event.ActionEvent evt)
{
    //GEN-FIRST:event_jButtonSaveActionPerformed
    Long l = Long.parseLong(jTextFieldM1.getText(), 8);
    setTextFieldM1((short) l.shortValue());
    m[0] = (short) octToDecInt(jTextFieldM0.getText());
    getComboBoxM56();
    System.arraycopy(m, 0, mBuffer, 0, 16);
    this.dispose();
    //GEN-LAST:event_jButtonSaveActionPerformed
}

```

```

        private void
jButtonCancelActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_jButtonCancelActionPerformed

        this.dispose();
    } //GEN-LAST:event_jButtonCancelActionPerformed

// При завершении ввода в поле M1, проверка 8CC и <= 0xffff
private void jTextFieldM1KeyPressed(java.awt.event.KeyEvent evt)
{ //GEN-FIRST:event_jTextFieldM1KeyPressed

        // Верификацию сделать
        //         int fifteen =
Integer.parseInt(jTextField2.getText(), 8);
        //         if (fifteen > (int)65535 ) {
        //
        //     }
    } //GEN-LAST:event_jTextFieldM1KeyPressed

private void
jTextFieldM1ActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_jTextFieldM1ActionPerformed
        Long l = Long.parseLong(jTextFieldM1.getText(), 8);
        setTextFieldM1((short) l.shortValue());
        // m[1] = (short) l.shortValue();
    } //GEN-LAST:event_jTextFieldM1ActionPerformed

private void
jRadioButton4ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jRadioButton4ItemStateChanged
        mask = !mask;
        int sel = jComboBoxM3.getSelectedIndex();
        setComboBoxM2((short) 0, false);
        if (jComboBoxM2.getSelectedIndex() >= 0) {
            setComboBoxM3((short)
jComboBoxM2.getSelectedIndex(), (short) sel, true);
        }
    } //GEN-LAST:event_jRadioButton4ItemStateChanged

private void
jComboBoxM2ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jComboBoxM2ItemStateChanged
        int sel = jComboBoxM3.getSelectedIndex(); // !!!!
        m[2] = (short) jComboBoxM2.getSelectedIndex();
        if (m[2] >= 0 && sel >= 0 && jComboBoxM3.getItemCount()
> 15) {
            setComboBoxM3((short) m[2], (short) sel, true);
        }
    } //GEN-LAST:event_jComboBoxM2ItemStateChanged

```

```

        private void
jComboBoxM3ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jComboBoxM3ItemStateChanged
    if (jComboBoxM3.getSelectedIndex() >= 0) {
        m[3] = (short) jComboBoxM3.getSelectedIndex();
        printM();
    }
} //GEN-LAST:event_jComboBoxM3ItemStateChanged


        private void
jComboBoxM11ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jComboBoxM11ItemStateChanged
    m[11] = (short) jComboBoxM11.getSelectedIndex();
    setComboBoxM7((short) 0, (short)
jComboBoxM11.getSelectedIndex(), false);
    printM();
} //GEN-LAST:event_jComboBoxM11ItemStateChanged


        private void jTable4MousePressed(java.awt.event.MouseEvent
evt) { //GEN-FIRST:event_jTable4MousePressed
    int col = jTable4.getSelectedColumn();
    int row = jTable4.getSelectedRow();
    if (col >= 0 && row >= 0) {
        if (col > 1 || col == 1 && row == 0) {
            int value = (int)
jTable4.getModel().getValueAt(row, col);
            jTable4.getModel().setValueAt((value + 1) % 2,
row, col);
        }

        m[6] = 0;
        for (int i = 0; i < 5; i++) {
            int j = (int) jTable4.getModel().getValueAt(row,
i);

            m[6] += (short) j << (4 - i);
            printM();
        }
    }
} //GEN-LAST:event_jTable4MousePressed


        private void jTableM5MousePressed(java.awt.event.MouseEvent
evt) { //GEN-FIRST:event_jTableM5MousePressed
    int col = jTableM5.getSelectedColumn();
    int row = jTableM5.getSelectedRow();
    if (col >= 0 && row >= 0) {
        if (col > 1 || col == 1 && row == 0) {
            int value = (int)
jTableM5.getModel().getValueAt(row, col);
            jTableM5.getModel().setValueAt((value + 1) % 2,
row, col);

```

```

        }
        m[5] = 0;
        for (int i = 0; i < 5; i++) {
            int j = (int)
jTableM5.getModel().getValueAt(row, i);
            m[5] += (short) j << (4 - i);
            printM();
        }
    }
    ///GEN-LAST:event_jTableM5MousePressed

    // Вывод значения из поля M1, в 2СС в текстовое поле
private void setTextFieldM1(short value) {
    DecimalFormat df = new
DecimalFormat("####,####,####,####");
    //      DecimalFormat df = new
DecimalFormat("0000,0000,0000,0000");
    String s1 = Integer.toBinaryString(value);
    try {
        Number n = df.parse(s1);
        jLabelM1.setText(df.format(n.longValue()));
    } catch (ParseException ex) {

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

    //      jLabelM1.setText(df.format(Long.parseLong(s1)));
    jTextFieldM1.setText(Integer.toString(value, 8));
    m[1] = value;
printM();
    }

    // Установка Поля M2, value - значение
    // flag - устанавливать value в combobox или нет
private void setComboBoxM2(short value, boolean flag) {
    //      JComboBox jComboBoxM2 = jComboBoxM2;
    jComboBoxM2.removeAllItems();

    // Зависит от маски (в старой версии всегда mask = true)
    if (mask) {
jComboBoxM2.addItem("0 -- нет инверсии условия и нет маски");
jComboBoxM2.addItem("1 -- инверсия условия, маски нет");
jComboBoxM2.addItem("2 -- нет инверсии условия, маскирование
адреса");
jComboBoxM2.addItem("3 -- инверсия условия, маскирование
адреса");
jLabelM2Mask.setText("1XX");
    } else {
jComboBoxM2.addItem("0 -- нет инверсии условия");
jComboBoxM2.addItem("1 -- инверсия условия");
jComboBoxM2.addItem("2 -- нет инверсии условия");
    }
}

```

```

jComboBoxM2.addItem("3 -- инверсия условия");
jLabelM2Mask.setText("0XX");
    }

    // Если flag = true, то необходимо установить value в
ComboBox
if (flag) {
    jComboBoxM2.setSelectedIndex(value);
    m[2] = value;
printM();
}
}

// Установка Поля М3, зависит от М2
// m2 - значение поля М2, m3 - значение поля М3,
// flag - устанавливать value в combobox или нет
private void setComboBoxM3(short m2, short m3, boolean flag) {

    JComboBox cb = jComboBoxM3;
    cb.removeAllItems();

// PSW, C0 и Конст не зависят от mask (маски)
cb.addItem("0 -- PSW[0]");
    cb.addItem("1 -- PSW[1]");
    cb.addItem("2 -- PSW[2]");
    cb.addItem("3 -- PSW[3]");
    cb.addItem("4 -- PSW[4]");
    cb.addItem("5 -- PSW[5]");
    cb.addItem("6 -- ПЕРЕНОС СТ");
    cb.addItem("7 -- КОНСТАНТА 0");

    if (mask) { // Подсказка варианта
        jLabelM3Mask.setText("1XX");
    } else {
        jLabelM3Mask.setText("0XX");
    }

    // Если mask = true, то проверяется поле М2.
// Если М2 выбрано 2 или 3, то установка Ур
// Иначе или при mask = false, установка внешних
сигналов
if (mask && m2 > 1) {
    cb.addItem("10 -- УР1");
    cb.addItem("11 -- УР2.1");
    cb.addItem("12 -- УР2.2");
    cb.addItem("13 -- УР3");
    cb.addItem("14 -- УР4");
    cb.addItem("15 -- УР5");
    cb.addItem("16 -- RK[11/9]");
    cb.addItem("17 -- RK[5/3]");
} else {
    cb.addItem("10 -- FLG[0]");
}
}

```

```

        cb.addItem("11 -- FLG[1]");
        cb.addItem("12 -- FLG[2]");
        cb.addItem("13 -- FLG[3]");
        cb.addItem("14 -- FLG[4]");
        cb.addItem("15 -- FLG[5]");
        cb.addItem("16 -- FLG[6]");
        cb.addItem("17 -- FLG[7]");
    }

    // Если flag = true, то необходимо установить value в
    ComboBox
    // И кол-во новых полей больше, устанавливаемый индекс
    if (flag && cb.getItemCount() > m3) {
        cb.setSelectedIndex(m3);

        printM();
    }
}

// Установка полей M5 и M6. value значение
// Если flag = false, значит поле M5
// Если flag = true, значит поле M6
private void setComboBoxM56(short value, boolean flag) {
    JRadioButton jrb1 = jRadioButtonM5;
    JRadioButton jrb2 = jRadioButtonM5RK02;
    JRadioButton jrb3 = jRadioButtonM5RK68;

    JTable jtb = jTableM5;

    // Если полей M6, то указать другие компоненты
    if (flag) {
        jrb1 = jRadioButtonM6;
        jrb2 = jRadioButtonM6RK02;
        jrb3 = jRadioButtonM6RK68;
        jtb = jTable4;
    }

    // Всего 3 строки, по умолчанию нулевая
    int row = 0;
    if (value >= 0b11000) { // Если >= 24
        jrb3.setSelected(true); // Установка 3 флажка
        row = 2;
    } else if (value >= 0b10000) { // Если >= 16 и < 24
        jrb2.setSelected(true); // Установка 2 флажка
        row = 1;
    } else { // Иначе выбрана 0
        jrb1.setSelected(true); // Установка 1 флажка А-
    }
}

```

```

        jtb.getModel().setValueAt(((value & 0b01000) != 0 ?
1 : 0), row, 1);
    }

    // Установка 2-4 битов, из xx000 -> [x][x][0][0][0]
jtb.getModel().setValueAt(((value & 0b00100) != 0 ? 1 : 0), row,
2);
    jtb.getModel().setValueAt(((value & 0b00010) != 0 ? 1 :
0), row, 3);
    jtb.getModel().setValueAt(((value & 0b00001) != 0 ? 1 :
0), row, 4);
    }

private void getComboBoxM56() {
    JRadioButton jrb1 = jRadioButtonM5;
    JRadioButton jrb2 = jRadioButtonM5RK02;
    JRadioButton jrb3 = jRadioButtonM5RK68;

    int index = 0;
    if (jrb2.isSelected()) {
        index = 1;
    } else if (jrb3.isSelected()) {
        index = 2;
    }

    m[5] = 0;
    for (int i = 0; i < 5; i++) {
        int j = (int) jTableM5.getModel().getValueAt(index,
i);

        m[5] += (short) j << (4 - i);
        printM();
    }

    // Если полей M6, то указать другие компоненты
jrb1 = jRadioButtonM6;
jrb2 = jRadioButtonM6RK02;
jrb3 = jRadioButtonM6RK68;

    if (jrb2.isSelected()) {
        index = 1;
    } else if (jrb3.isSelected()) {
        index = 2;
    } else {
        index = 0;
    }

    m[6] = 0;
    for (int i = 0; i < 5; i++) {
        int j = (int) jTable4.getModel().getValueAt(index,
i);

        m[6] += (short) j << (4 - i);
        printM();
    }

```

```

    }

    }

    // Установка Поля М7, зависит от М11
    // m7 - значение поля М7, m11 - значение поля М11,
    // flag - устанавливать value в combobox или нет
private void setComboBoxM7(short m7, short m11, boolean flag) {

    JComboBoxcb = jComboBoxM7;

    // Добавление элементов, в зависимости от 1 бита поля
    М11 (2 = xx1x)
    // 0145 вправо, 2367 влево
    if ((m11 & (short) 2) == 0 && flagM7) {
        cb.removeAllItems();
        cb.addItem("0
-- \"0\"-->[R]-->[PSW(0)]      \"0\"-->[RQ]-->");
        cb.addItem("1 -- -->[R]-->[PSW(0)]-->[R]
-->[RQ]-->[RQ]-->");
        cb.addItem("2 --
[PSW(0)]-->[R]----->[RQ]-->[PSW(0)]");
        cb.addItem("3 --
[(F15[+]OVR)]-->[R]----->[RQ]-->[PSW(0)]");

    // Исключение повторного добавления (2-ой раз в этот блок не
    зайдет)
    flagM7 = false;

        } else if ((m11 & (short) 2) != 0 && !flagM7) {
            cb.removeAllItems();
            cb.addItem("0 -- [PSW(0)]<--[R]<--\"0\"
<--[RQ]<--\"0\"");
            cb.addItem("1 -- <--[R]<--[PSW(0)]<--[R]<--
<--[RQ]<--[RQ]<--");
            cb.addItem("2 --
[PSW(0)]<--[R]<-----[RQ]<--\"1\"");
            cb.addItem("3 --
[PSW(0)]<--[R]<-----[RQ]<--\"0\"");

    // Исключение повторного добавления (2-ой раз в этот блок не
    зайдет)
    flagM7 = true;
        }

    // Если flag = true, то необходимо установить m7 в
    ComboBox
    if (flag) {
        cb.setSelectedIndex(m7);
        m[7] = m7;
        printM();
    }

}

```

```

private void
jComboBoxM4ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM4ItemStateChanged
    m[4] = (short) jComboBoxM4.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM4ItemStateChanged

private void
jComboBoxM8ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM8ItemStateChanged
    m[8] = (short) jComboBoxM8.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM8ItemStateChanged

private void
jComboBoxM9ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM9ItemStateChanged
    m[9] = (short) jComboBoxM9.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM9ItemStateChanged

private void
jComboBoxM10ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM10ItemStateChanged
    m[10] = (short) jComboBoxM10.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM10ItemStateChanged

private void
jComboBoxM12ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM12ItemStateChanged
    m[12] = (short) jComboBoxM12.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM12ItemStateChanged

private void
jComboBoxM13ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM13ItemStateChanged
    m[13] = (short) jComboBoxM13.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM13ItemStateChanged

private void
jComboBoxM14ItemStateChanged(java.awt.event.ItemEvent evt)
{//GEN-FIRST:event_jComboBoxM14ItemStateChanged
    m[14] = (short) jComboBoxM14.getSelectedIndex();
    printM();
}//GEN-LAST:event_jComboBoxM14ItemStateChanged

```

```

        private void
jComboBoxM15ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jComboBoxM15ItemStateChanged
    m[15] = (short) jComboBoxM15.getSelectedIndex();
    printM();
} //GEN-LAST:event_jComboBoxM15ItemStateChanged

        private void
jRadioButtonM5ActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_jRadioButtonM5ActionPerformed
    printM();
} //GEN-LAST:event_jRadioButtonM5ActionPerformed

        private void
jRadioButtonM6ActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_jRadioButtonM6ActionPerformed
    printM();
} //GEN-LAST:event_jRadioButtonM6ActionPerformed

        private void
jTextFieldM0ActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_jTextFieldM0ActionPerformed

    m[0] = (short) octToDecInt(jTextFieldM0.getText());
    printM();
} //GEN-LAST:event_jTextFieldM0ActionPerformed

        private void jTextFieldM0KeyPressed(java.awt.event.KeyEvent
evt) { //GEN-FIRST:event_jTextFieldM0KeyPressed
    // TODO add your handling code here:
} //GEN-LAST:event_jTextFieldM0KeyPressed

        private void
jComboBoxM7ItemStateChanged(java.awt.event.ItemEvent evt)
{ //GEN-FIRST:event_jComboBoxM7ItemStateChanged
    m[7] = (short) jComboBoxM7.getSelectedIndex();
    printM();
} //GEN-LAST:event_jComboBoxM7ItemStateChanged

```

Приложение Б

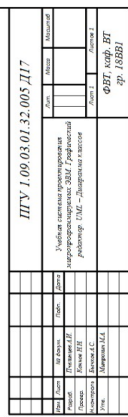
(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

UML – диаграмма классов

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						183
Изм.	Лист	№ докум.	Подпись	Дата		



Приложение В

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

UML – диаграмма деятельности

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						185
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Г

(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

UML – диаграмма последовательности

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						187
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Д

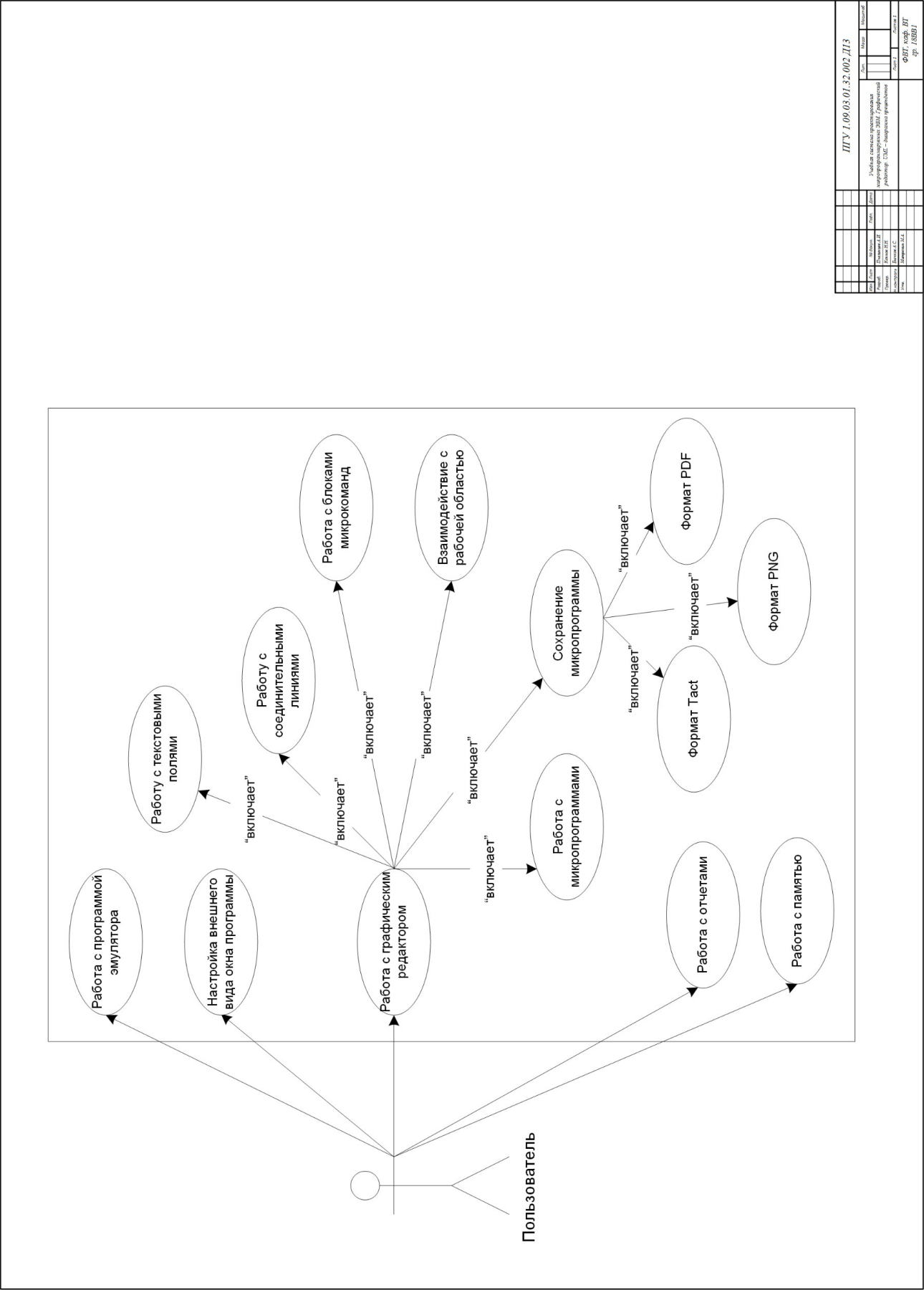
(Обязательное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

UML – диаграмма прецедентов

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						189
Изм.	Лист	№ докум.	Подпись	Дата		



ПГУ 1.09.03.01.32.002 Л13									
Изм.	Лист	№ докум.	Подпись	Дата	Имя	Фамилия	Отчество	Пол	Возраст
Учебная программа по дисциплине «Информационные технологии в проектировании»									
Учебно-методический комплекс (УМК) по дисциплине «Информационные технологии в проектировании»									
Формат PDF, RT, EPUB, MOBI, FB2, DJVU, XPS, SVG, EPS, AI, PSD, RAW, DNG, CR2, CR3, NEF, ARW, DCR, PEF, MPO, K25, K26, K27, K28, K29, K30, K31, K32, K33, K34, K35, K36, K37, K38, K39, K40, K41, K42, K43, K44, K45, K46, K47, K48, K49, K50, K51, K52, K53, K54, K55, K56, K57, K58, K59, K60, K61, K62, K63, K64, K65, K66, K67, K68, K69, K70, K71, K72, K73, K74, K75, K76, K77, K78, K79, K80, K81, K82, K83, K84, K85, K86, K87, K88, K89, K90, K91, K92, K93, K94, K95, K96, K97, K98, K99, K100									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Е

(Справочное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

Презентация

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						191
Изм.	Лист	№ докум.	Подпись	Дата		



Пензенский
государственный
университет



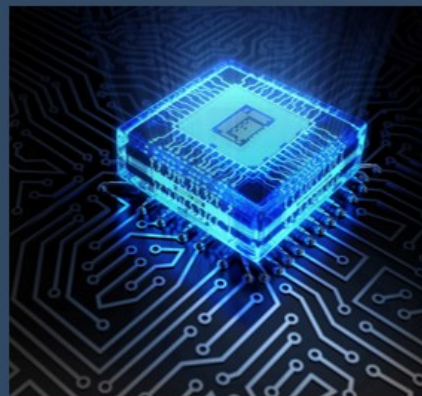
Кафедра
вычислительной
техники

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Учебная система проектирования микропрограммируемых ЭВМ. Графический редактор

АВТОР РАБОТЫ: СТУДЕНТ ГРУППЫ 18ВВ1 Пчелинцев А.И.

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

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.32.001 ПЗ

Лист

492

1
Разработан под
Windows платформу

2
Невозможность
редактирования МК
в процессе
выполнения

3
Отсутствие
возможности
работы с ОЗУ

4
Загрузка программ
разработанных
только в MS Visio

5
Отсутствие
возможности
обработки
перерываний

Недостатки известных эмуляторов

Цели и задачи

разработка графического редактора учебной системы проектирования микропрограммируемых ЭВМ. Редактор предназначен для использования в учебном процессе подготовки бакалавров и специалистов в области вычислительной техники с целью выработки компетенций по проектированию алгоритмов работы микропрограммируемых процессоров и других средств вычислительной техники..

Программа должна быть кроссплатформенной.

Выбор решения



Кроссплатформенность

Выбор объектно-ориентированного языка программирования Java

Редактирование МК

Доработка интерфейса программы

Работа с ОЗУ

Разработка модуля и алгоритма взаимодействия с ОЗУ

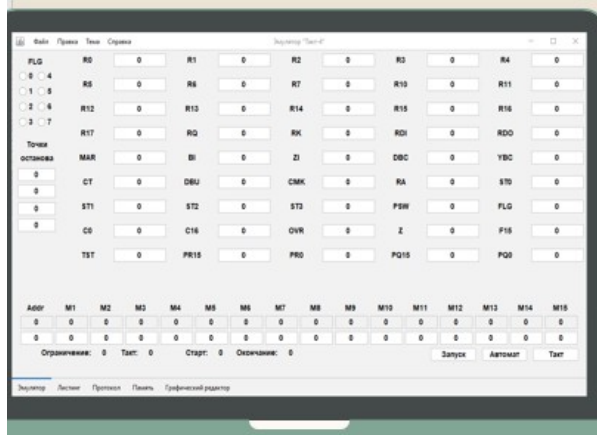
Загрузка программы

Интеграция графического редактора

Прерывания

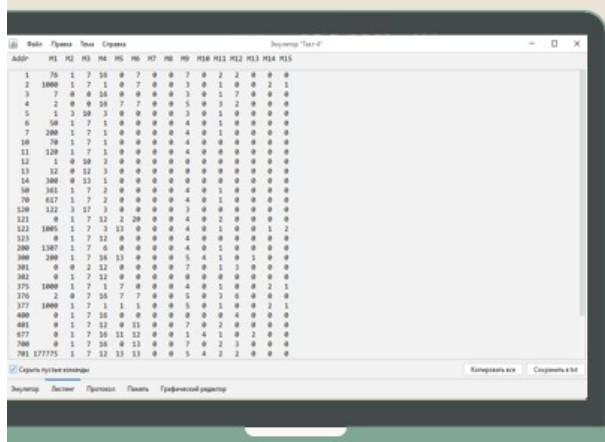
Разработка алгоритма обработки прерывания

Главное окно



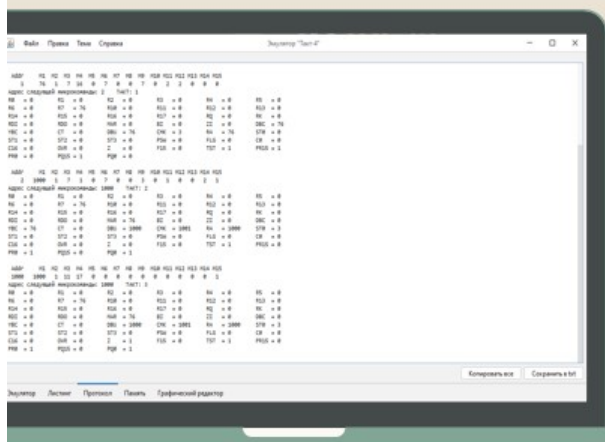
- Порты ввода внешних сигналов
- Точки останова
- Регистры микропроцессора
- Текущая и следующая микрокоманда
- Органы управления ходом отладки

Окно листинга



- Таблица из 15 полей
- Возможность скрывтия 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

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 1.09.03.01.32.001 ПЗ	Лист
						196
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Ж

(Справочное)

Учебная система проектирования микропрограммируемых ЭВМ.

Графический редактор

Свидетельства

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						197
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 0x02A5CFB5C00B1ACF59A40A2F08092E9A118
Владелец **Ивлиев Григорий Петрович**
Действителен с 15.01.2021 по 15.01.2035

Г.П. Ивлиев

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.32.001 ПЗ

Лист

198

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

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.32.001 ПЗ

Лист

499



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.32.001 ПЗ

Лист

200

Приложение 3

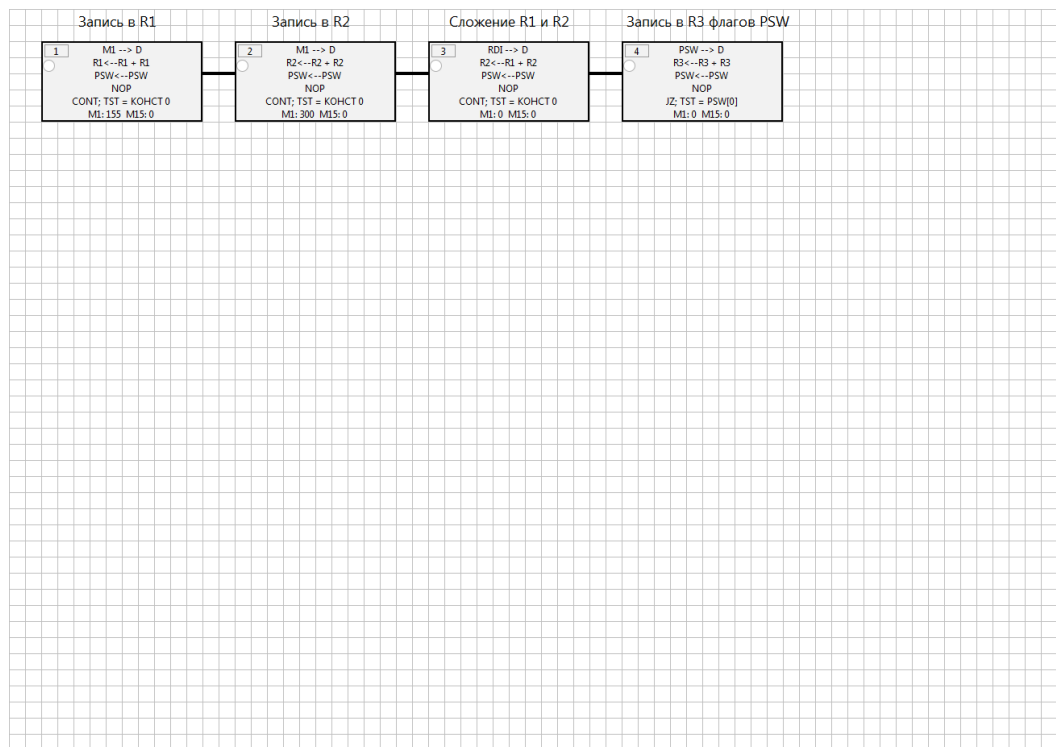
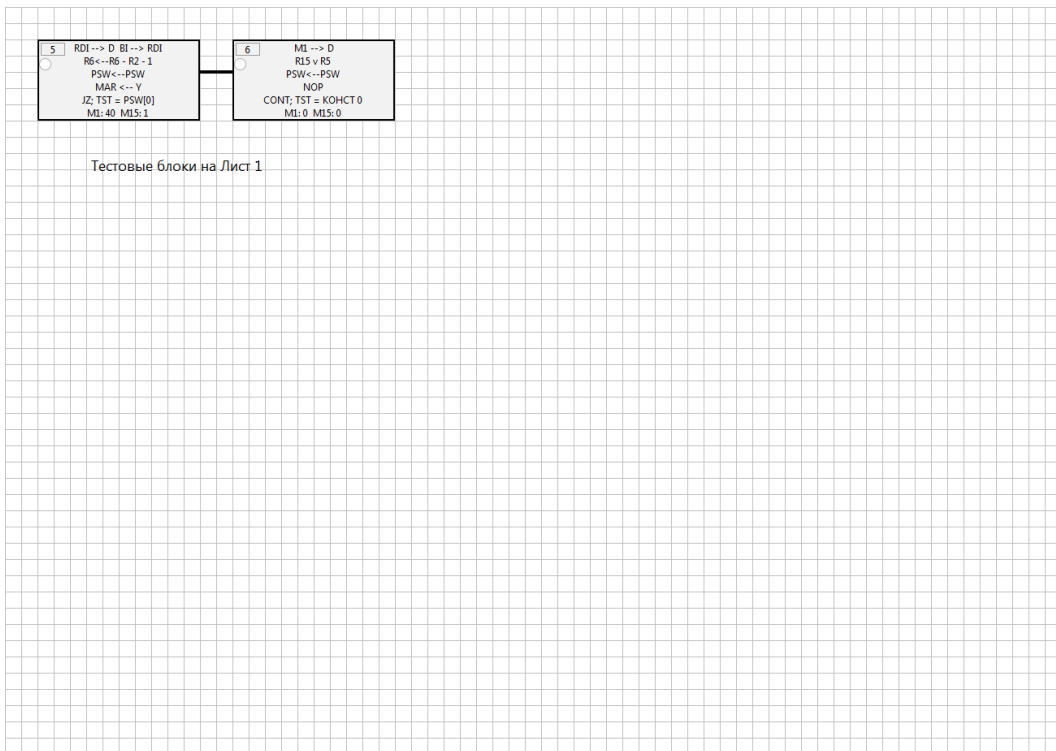
(Справочное)

Учебная система проектирования микропрограммируемых ЭВМ.

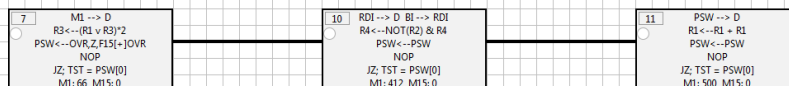
Графический редактор

Схема программы ДМЛ

					ПГУ 1.09.03.01.32.001 ПЗ	Лист
						201
Изм.	Лист	№ докум.	Подпись	Дата		



Тестовые блоки на Лист 1



Тестовые блоки на Листе 2

