МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Вятский государственный университет»**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Допущено к защите

Руководитель проекта

\_\_\_\_\_\_\_\_/Караваева О.В./

(подпись) (Ф.И.О)

«\_\_»\_\_\_\_\_\_\_\_\_\_2023г.

Разработка программного продукта для создания тестов проверки блоков релейной логики

Пояснительная записка курсового проекта по дисциплине

«Комплекс знаний бакалавра в области программного и аппаратного обеспечения вычислительной техники»

ТПЖА.09.03.01.487 ПЗ

Разработал студент группы ИВТ-31 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М./

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Караваева О.В./

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (дата)

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Долженкова М.Л. /

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Кошкин О.В./

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Чистяков Г.А./

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Чудинов А.А./

(подпись)

Киров 2023

РЕФЕРАТ

Птахова А.М.. Разработка программного продукта для создания тестов автоматизированной проверки блоков релейной логики. ТПЖА.09.03.01.487ПЗ: Курс. проект/ ВятГУ, каф. ЭВМ; рук. Караваева О.В. – Киров, 2023. - ПЗ 157с., 6 табл., 26 рис., 7 прил.

РЕЛЕЙНАЯ ЛОГИКА, ПРОГРАММНАЯ МОДЕЛЬ, КОМПИЛЯТОР, СРЕДА РАЗРАБОТКИ, ТЕСТИРОВАНИЕ

Программная модель, разработанная в рамках данного курсового проекта – продукт для написания тестов автоматизированной проверки блоков релейной логики.

В ходе выполнения курсового проекта был выполнен анализ проблемной области, проектирование и написание компилятора и среды разработки.

# ГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 3](#_Toc135301816)

[ВВЕДЕНИЕ 5](#_Toc135301817)

[1. АНАЛИЗ ПРОБЛЕМНОЙ ОБЛАСТИ 6](#_Toc135301818)

[1.1. Обзор языков программирования 6](#_Toc135301819)

[1.1.1. Ассемблер 6](#_Toc135301820)

[1.1.2. С для Arduino 6](#_Toc135301821)

[1.1.3. Scratch 7](#_Toc135301822)

[1.2. Актуальность разработки 8](#_Toc135301823)

[1.3. Техническое задание 8](#_Toc135301824)

[1.3.1. Наименование системы 8](#_Toc135301825)

[1.3.2. Краткая характеристика области применения 8](#_Toc135301826)

[1.3.3. Цели создания 9](#_Toc135301827)

[1.3.4. Требования к программе 9](#_Toc135301828)

[1.3.5. Требования к программной документации 10](#_Toc135301829)

[2. РАЗРАБОТКА КОМПИЛЯТОРА 11](#_Toc135301830)

[2.1. Разработка структуры компилятора 11](#_Toc135301831)

[2.2. Разработка анализаторов и генераторов 12](#_Toc135301832)

[2.2.1. Разработка лексического анализатора 12](#_Toc135301833)

[2.2.2. Разработка синтаксического анализатора 20](#_Toc135301834)

[2.2.3. Семантический анализатор 38](#_Toc135301835)

[2.2.4. Разработка генератора промежуточного кода 42](#_Toc135301836)

[2.2.5. Разработка генератора кода 48](#_Toc135301837)

[2.3. Описание поведения 54](#_Toc135301838)

[2.4. Программная реализация 56](#_Toc135301839)

[ЗАКЛЮЧЕНИЕ 57](#_Toc135301840)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № докум | Подпись | Дата |
| Разраб | | Птахова А. М |  |  | Разработка программного продукта для создания тестов проверки блоков релейной логики | Литера | | | Лист | Листов |
| Пров | | Долженкова М. Л. |  |  |  |  |  | 3 | 157 |
|  | |  |  |  |  | | | | |
|  | |  |  |  |
| Реценз. | |  |  |  |

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 58](#_Toc135301841)

[Приложение А 59](#_Toc135301842)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 4 |
| Изм. | Лист | № докум | Подпись | Дата |

[Приложение Б 72](#_Toc135301843)

[Приложение В 73](#_Toc135301844)

[Приложение Г 90](#_Toc135301845)

[Приложение Д 100](#_Toc135301846)

[Приложение Е 119](#_Toc135301847)

[Приложение Ж 153](#_Toc135301848)

# ВВЕДЕНИЕ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 5 |
| Изм. | Лист | № докум | Подпись | Дата |

Существует устройство для автоматизированной проверки блоков релейной логики, позволяющее подавать сигналы на входы проверяемого устройства и получать сигналы с его выходов. Результат о исправности или не исправности блока формируется за счет сравнения данных, полученных на его выходах с эталонными значениями. Основным недостатком разработанной модели было отсутствие возможности изменения программы проверки. Для решения данной проблемы возможно создание языка программирования, предоставляющего инструменты для написания программы тестов. Основной концепцией разрабатываемого языка является простота, позволяющая писать программы тестов, не имея большого опыта в программировании.

# 1. АНАЛИЗ ПРОБЛЕМНОЙ ОБЛАСТИ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 6 |
| Изм. | Лист | № докум | Подпись | Дата |

На первом этапе работы было произведено сравнение существующих языков программирования для микроконтроллеров, выявлены их достоинства и недостатки. На основе анализа было сформировано техническое задание.

## 1.1. Обзор языков программирования

### 1.1.1. Ассемблер

Команды языка программирования - инструкции процессора. Каждая команда состоит из названия операции и операндов, к которым будет применена выбранная операция. Операндами могут являться как регистры, так и числа.

Так как любая команда подразумевает взаимодействие с регистрами, то пользователю необходимо обладать знаниями об архитектуре процессора для написания корректно работающей программы. Кроме того, необходимо иметь опыт написания программ на языках низкого уровня, так как они имеют ряд особенностей: наличие команды переходов для возможности организации циклов и ветвлений; наличие типов данных и т.д.

С другой стороны, приближенность команд к инструкциям процессора позволяет создавать компактные и эффективные исполняемые модули. Это достигается за счет отсутствия приведения команд к инструкции, выполняемой процессором. И как следствие, отсутствие избыточных или ненужных команд.

### 1.1.2. С для Arduino

Предоставляет набор команд для взаимодействия с микроконтроллером. Каждая команда является функцией, описание которой скрыто от пользователя и подключается только в момент компиляции программы. Написание программы подразумевает вызов той или иной функции с передачей в нее параметров. Это значительно упрощает процесс программирования из-за чего пользователем может быть человек, не имеющего опыт в программировании. Но опыт программирования все же должен быть, так как параметрами функции могут быть переменные, чей тип необходимо указывать. Кроме того, название программ на английском, что еще сильнее усложняет процесс программирования для русскоговорящего пользователя.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 7 |
| Изм. | Лист | № докум | Подпись | Дата |

Основным недостатком такого подхода является отдаленность команд от инструкций процессора. Следовательно, при создании исполняемого модуля необходима трансляция всех выражений в эти инструкции. Из-за этого возникают избыточные команды, дублирующие друг друга команды, что сказывается на объеме исполняемого модуля.

### 1.1.3. Scratch

Синтаксические конструкции представляют собой визуальные блоки, которые можно последовательно соединять, тем самым задавая алгоритм работы микроконтроллера. Визуальные блоки имеют такой же принцип работы, как функции на С для Arduino, но параметры функции не имеют явной типизации. Это позволяет сделать написание кода доступным для людей, не имеющих опыта в программировании. Кроме того, синтаксические конструкции написаны на русском языке, что упрощает процесс программирования для русскоговорящего населения.

При создании исполняемого модуля все блоки сначала транслируются в команды на языке С, а после в команды процессора. Такая многоуровневая трансляция сказывается на скорости создания исполняемого модуля, на его компактности и эффективности.

Сравнение языков программирования представлено в таблице 1

Таблица 1 – Сравнение языков программирования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 8 |
| Изм. | Лист | № докум | Подпись | Дата |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ассемблер | С | Scratch |
| 1. Поддержка русского языка | - | - | + |
| 2. Простота написания программы | - | + | + |
| 3. Компактность исполняемого модуля | + | - | - |
| 4. Эффективность исполняемого модуля | + | - | - |

## 1.2. Актуальность разработки

Существующие языки программирования имеют ряд преимуществ: поддержку русского языка, простоту написания программы, компактность исполняемого модуля и его эффективность. Но ни один из рассмотренных языков не содержит в себе все эти достоинства.

В результате было принято решение о разработке языка программирования, включающего в себя все преимущества рассмотренных языков.

## 1.3. Техническое задание

### 1.3.1. Наименование системы

Полное наименование системы: система для создания тестов автоматизированной проверки блоков релейной логики.

### 1.3.2. Краткая характеристика области применения

Продукт предназначен для возможности написания тестов автоматизированной проверки блоков релейной логики

### 1.3.3. Цели создания

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 9 |
| Изм. | Лист | № докум | Подпись | Дата |

Функциональным назначением программы является предоставление возможности написания тестов для устройства проверки блоков релейной логики.

Программа должна эксплуатироваться на ПК, установленных в кабинетах на предприятии. Особые требования к конечному пользователю не предъявляются.

### 1.3.4. Требования к программе

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1) функции написания программы;

2) функции сохранения и открытия программы;

3) функции проверки исправности написанной программы;

4) функции компиляции программы в объектный код;

5) функции загрузки объектного кода в микроконтроллер

В состав технических средств должен входить ПК, включающий в себя:

32 или 64 разрядный процессор с тактовой частотой не меньше 1.0

ГГц;

2) дисплей;

3) не менее 1 Гб оперативной памяти;

4) не менее 100 мегабайт свободного дискового пространства;

5) клавиатура и мышь.

Системные программные средства, используемые программой, должны быть представлены ОС Windows 7/8/10;

Программа должна обеспечивать взаимодействие с пользователем по средствам графического интерфейса и предоставлять возможность выполнять наиболее часто используемые операции с помощью сочетаний клавиш на клавиатуре.

### 1.3.5. Требования к программной документации

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 10 |
| Изм. | Лист | № докум | Подпись | Дата |

Состав программной документации должен включать в себя:

1) техническое задание;

2) руководство пользователя;

3) исходный код.

1.3.6. Стадии и этапы разработки

Разработка должна быть приведена в 3 стадии:

1) разработка технического задания;

2) проектирование;

3) внедрение.

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и разработки технического задания.

На стадии проектирования необходимо выполнить следующие этапы:

1) разработка программы;

2) разработка программной документации;

3) испытание программы.

На этапе внедрения выполняется передача программы заказчику.

# 2. РАЗРАБОТКА КОМПИЛЯТОРА

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 11 |
| Изм. | Лист | № докум | Подпись | Дата |

На данном этапе работы необходимо в соответствии с требованиями, указанными в техническом задании разработать структуру компилятора, выполнить разработку соответствующего анализатора или генератора для каждого из этапов компиляции, рассмотреть поведение компилятора в контексте времени и на основании этого произвести программную реализацию компилятора.

## 2.1. Разработка структуры компилятора

Для обеспечения трансляции исходного кода в исполняемый модуль разработана структура компилятора, представляющая собой диаграмму прецедентов. Диаграмма прецедентов представлена на рисунке 1.

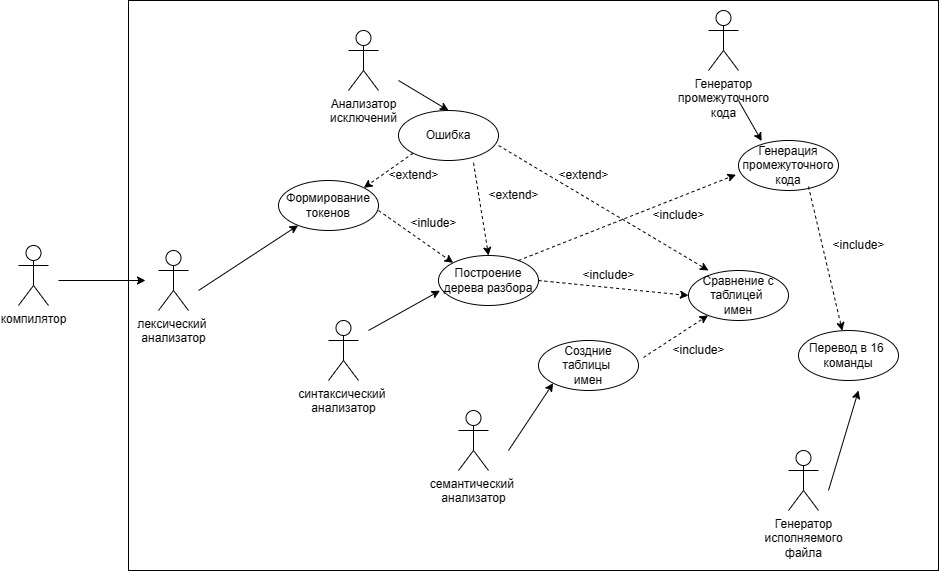


Рисунок 1 – Диаграмма прецедентов

Инициализатором трансляции исходного кода является компилятор, передающий управление лексическому анализатору.

В свою очередь лексический анализатор выполняет формирование токенов за счет выделения слова из текста программы, определения его типа и записи этих данных в структуру. Во время выполнения может возникнуть ошибка при некорректно введенном слове.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 12 |
| Изм. | Лист | № докум | Подпись | Дата |

Действие «ошибка» инициализирует анализатор исключений.

Синтаксический анализатор может выполнять действие построения дерева разбора, но только в том случае, если до этого было произведено разбиение текста программа на массив токенов. Построение дерева выполняется по разработанной грамматике. Следовательно, при несоответствии анализируемого токена с тем, который задан грамматикой выполняется формирование ошибки.

Семантический анализатор инициализирует действие сравнения с таблицей имен, которое включает в себя создание таблицы имен и построение дерева разбора.

Генерация промежуточного кода выполняет формирование промежуточного кода, которое включает в себя этап синтаксического и лексического анализатора.

Генератор промежуточного может выполнять действие перевода команды в 16-ричный формат. Данное действие включает в себя предыдущие этапы компиляции.

## 2.2. Разработка анализаторов и генераторов

### 2.2.1. Разработка лексического анализатора

Во входном тексте программы возможно использование следующих элементов:

1. Идентификаторы или имена переменных
2. Числа
3. Знаки сравнения
4. Разделители
5. Скобки
6. Ключевые слова

Для выделения этих элементов, проверку их на корректность необходимо выполнить описание поведения лексического анализатора, выполнить программную реализацию и провести тестирование.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 13 |
| Изм. | Лист | № докум | Подпись | Дата |

#### 2.2.1.1. Разработка структуры лексического анализатора

В качестве структуры лексического анализатора была выбрана диаграмма переходов состояний, так как при помощи ее состояний можно представлять различные ситуации, которые могут возникнуть в процессе сканирования входного текста.

Если в начале разбора нового входного слова встречается цифра, причем она может быть представлена в 16-чном формате, то происходит запись символа в буфер и переход в состояние «число». В этом состоянии происходит дальнейшее сканирование текста и проверка новых символ на соответствие цифре. В случае непрохождения проверки, происходит переход в состояние «ошибка». Проверка и запись в буфер продолжается до тех пор, пока не встретиться разделитель. После происходит создание токена, очистка буфера и переход в начальное состояние. Диаграмма переходов состояний для распознавания числа представлена на рисунке 2.

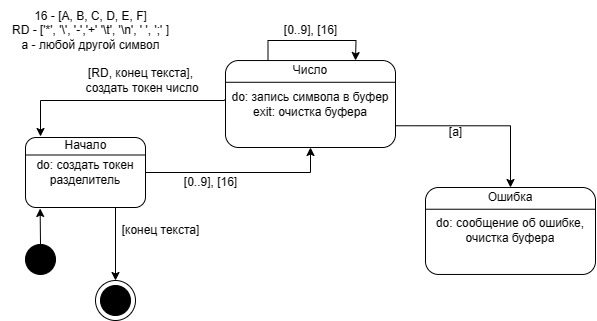


Рисунок 2 – Диаграмма переходов состояний для распознавания числа

Правильной числовой последовательностью можно считать следующие примеры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 14 |
| Изм. | Лист | № докум | Подпись | Дата |

- 856

- AFF

Неправильным примером числовой последовательности является:

- 8ва24

Если при распознавании нового слова встретилась буква, то происходит запись символа в буфер и переход в состояние «слово» с последующим сканированием текста программы и запись в буфер до момента обнаружения символа – разделителя. Далее полученный буфер сравнивается со списком ключевых слов, если находится соответствие, то создается токен с указанием «ключевое слово» в качестве типа или «лев\_скобка» и «прав\_скобка» для слов «н» и «к» соответственно. Если же соответствие не найдено, то создается токен с типом идентификатор. Очистка буфера и переход в начальное состояние. Диаграмма переходов состояний представлена на рисунке 3.

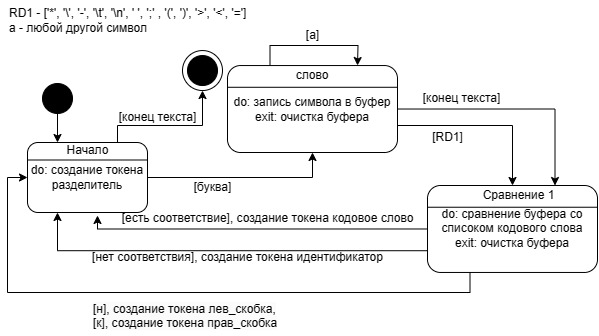


Рисунок 3 – Диаграмма переходов для распознавания слова

Примеры определения типов следующие:

- определить – ключевое слово

- н – лев\_скобка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 15 |
| Изм. | Лист | № докум | Подпись | Дата |

- опред - идентификатор

При обнаружении разделителя происходит создание токена с указанием его типа в качестве типа.

Примеры символов и их типы могут быть следующие:

- “ - ” - минус

- “ \* ” - звезда

В случае получения символов “ ( ” или “ ) ” создаются токены с типом прав\_скобка и лев\_скобка соответственно.

Если при сканировании встречается символ “=” или “!”, то происходит запись в буфер и переход в состояние «разделитель». В нем происходит запись в буфер до момента, пока не входной символ не будет являться числом, или буквой, или пробелом. В этом случае происходит анализ содержимого буфера, если там хранится “=”, то создается токен с типом присваивания. Если же содержимым является “ != ” или “==”, то токен имеет тип сравнение. В любом другом случае происходит переход в состояние «ошибка». Диаграмма переходов состояний представлена на рисунке 4.

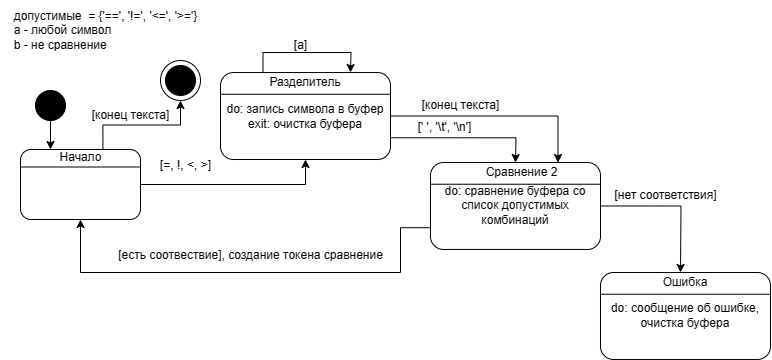


Рисунок 4 – Диаграмма переходов для распознавания разделителя

Неправильными вариантами последовательности могут служить следующие примеры:

- !в

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 16 |
| Изм. | Лист | № докум | Подпись | Дата |

- ===

В состоянии ошибка бросается исключение, в котором указывается лексема, в которой обнаружена ошибка и № строки, в которой она допущена.

Общая диаграмма переходов состояний представлена на рисунке 5.

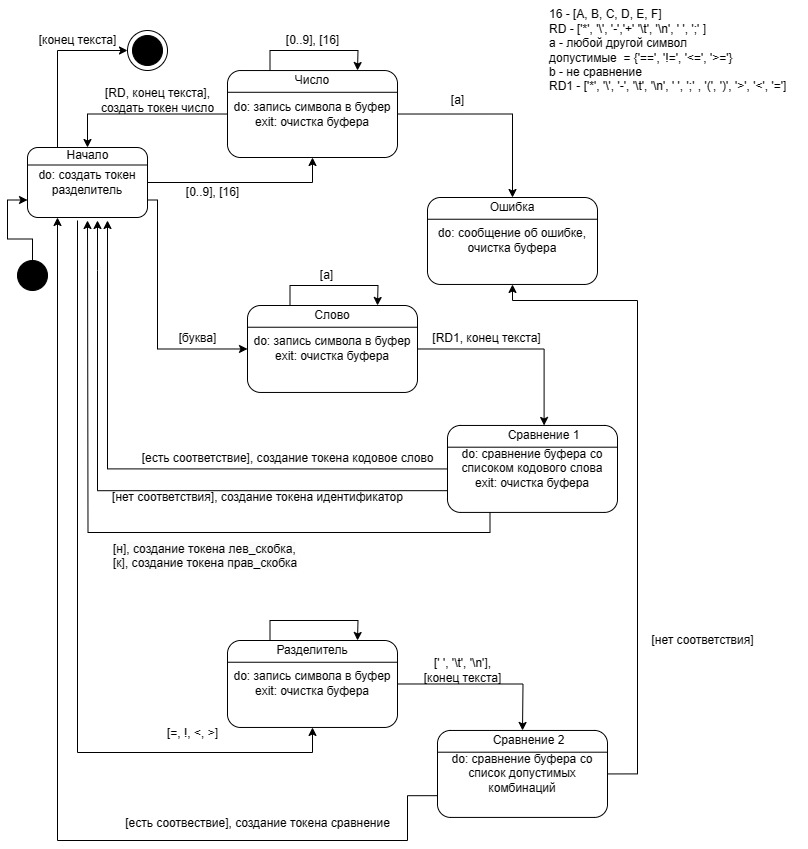


Рисунок 5 – Диаграмма переходов общая

#### 2.2.1.2. Программная реализация

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 17 |
| Изм. | Лист | № докум | Подпись | Дата |

В ходе выполнения программной реализации было произведено кодирование автомата для распознавания лексем, заданного диаграммой переходов состояний.

В качестве состояний автомата был создан класс перечислений States, в котором указаны названия состояний. Данный класс представляет собой ограниченный список идентификаторов, что позволяет выполнять быстрые переходы между ними. Программный код класса States представлен ниже:

enum class States // состояния автомата

{

S, //начальное

NUM, // число

DLM, //разделитель

WORD, //слово

ER, // ошибка

FIN, //конечное

};

Такой же класс был разработан для определения типа токена. Его программный код представлен ниже:

enum class token\_type

{

ID, //переменная

NUMBER, //число

KEY\_WORD, //ключевое слово

DLM, // , ;

ASSGN, // =

COM, // != == > <

LPAR, // ( н

RPAR, // ) к

PAR, // "

MINUS, // -

PLUS,

METKA, // <name>:

OR,

AND,

NOT,

STAR,

SHIFT, // << >>

DIV,

};

Для определения является ли символ разделителем был создан массив, в котором описаны все разделители, которые могут встретиться в тексте программы, а также функция, проверяющая наличие символа в массиве разделителей. Их программная реализация представлена ниже:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 18 |
| Изм. | Лист | № докум | Подпись | Дата |

18

char delimetr[] { '(', ')', ';', ' ', '-', '\n', '=', ':', '>', '<', '\t', '/', ',',

'!', '+', '\*'};

bool isDelimetr(char symbol)

{

bool exists = find(begin(delimetr), end(delimetr), symbol) != end(delimetr);

if (exists) return true;

else return false;

}

Программная реализация конечного автомата для распознавания числа следующая:

для каждого символа строки:

{

выбор (состояние):

{

состояние:: начальное:

{

если (символ – число)

{

запись в буфер;

состояние: = число;

}

}

состояние:: число:

{

если (символ – число)

{

запись в буфер;

состояние: = число;

}

иначе

{

если (символ – RD)

{

создать токен число;

}

Иначе

{

состояние:= ошибка;

}

}

}

состояние:: ошибка:

{

выдать сообщение об ошибке;

очистить буфер; } }}

Полностью листинг программы лексического анализатора представлен в приложении А.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 19 |
| Изм. | Лист | № докум | Подпись | Дата |

#### 2.2.1.3. Тестирование

Тестирование по методу белого ящика представлено в таблице 2.

Таблица 2 – Тестирование методом белого ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| Ввод 10 числа | 98 (в) | Type: NUMBER | Type: NUMBER | Успешно |
| 8к (н\в) | Ошибка 8к | Ошибка 8к | Успешно |
| Ввод 16 числа | 08FF (в) | Type: NUMBER | Type: NUMBER | Успешно |
| 08ff (н\в) | Ошибка 08f | Ошибка 08f | Успешно |
| 08АА (н\в) | Ошибка 08А | Ошибка 08А | Успешно |
| Ввод имени | С\_рег (в) | Type: ID | Type: ID | Успешно |
| 2имя (в) | Ошибка 2и | Ошибка 2и | Успешно |
| Ввод унарной операции | 8 – 7 (в) | Type: MINUS | Type: MINUS | Успешно |
| 8 % 7 (в) | Ошибка % | Ошибка % | Успешно |
| Ввод двойной операции | 8 << 7 (в) | Type: SHIFT | Type: SHIFT | Успешно |
| 8 <+ 7 (н\в) | Ошибка <+ | Ошибка <+ | Успешно |
| Ввод скобок | ( (в) | Type: LPAR | Type: LPAR | Успешно |
| { (н\в) | Ошибка { | Ошибка { | Успешно |
| Ввод ключевых слов | Определить (в) | Type: KEY\_WORD | Type: KEY\_WORD | Успешно |
| Опрделить (н\в) | Type: ID | Type: ID | Успешно |

### 2.2.2. Разработка синтаксического анализатора

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 20 |
| Изм. | Лист | № докум | Подпись | Дата |

На данном этапе работ необходимо разработать грамматику в соответствии с формой Бекуса – Наура, разработать алгоритм построения дерева разбора, выполнить программную реализацию и провести тестирование.

#### 2.2.2.1. Разработка грамматики

Для создания грамматики необходимо задать четверку:

,где

*N* – множество нетерминальных символов. В разрабатываемой грамматике оно представлено следующим образом:

*N* = {code, define, proc, macros, preproc, parametr, name, value, proc\_code, proc, operation, action, if\_statment, else\_statment, op, if\_code, else\_code, op2, op3, condition, first\_cond, metka}

*T* – множество терминальных символов. Совпадает с алфавитом языка, задаваемого грамматикой, имеет следующий вид:

*T* = {определить, н, к, идентификатор, число, И, ИЛИ, НЕ, \*, - , + , / , >>, <<, вернуть, переход, если, иначе, конец, (, ), ==, !=}

*P* – конечное множество правил порождения. Подробное описание их создания приведено ниже:

Исходный код программы представляет собой функции и директивы препроцессора. Последние определяется за счет ключевого слова «определить». Грамматика для определения функций и директив имеет следующий вид:

<code> := определить <define> | <proc>

Но так как в программе может быть функция и процедура вместе, не одна директива и/или не одна функция, то следует доработать грамматику и тогда она примет следующий вид:

<code> := определить <define> <code> | <proc> <code> | e

Такое правило означает, что символ <code> может заменяться на одну из последовательностей в зависимости от типа и значения текущего токена. То есть при значении токена равному «определить» происходит выбор последовательности, связанной с разбором директивы препроцессора. После разбора <define> происходит переход к следующему символу последовательности, так как он равен <code>, то снова происходит выбор последовательности. В случае, если токенов больше нет, то нетерминальный символ заменяется на e, обозначающий пустую последовательность и символизирующую окончание раскрытия нетерминального символа <code>.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 21 |
| Изм. | Лист | № докум | Подпись | Дата |

В свою очередь нетерминальный символ <define>, отвечает за разбор директивы подстановки. Как известно, директива подстановки состоит из 2х частей: макроса и части, которая будет подставлена во время компиляции. Для простоты назовем последнюю препроцессором. Тогда грамматика для разбора директивы препроцессора будет следующей:

<define> := <macros> н <preproc> к

Макросом, как правило, является переменная или функция c параметрами. Но в любом из этих случаев сначала всегда идет идентификатор, а потом могут идти параметры, которые указываются в скобках, если это функция. Тогда грамматика для разбора будет выглядеть так:

<macros>:= <name> | <name> ( <parametr>

Грамматика для идентификатора имеет вид:

<name> := идентификатор

Что касается параметров, то их либо нет и тогда скобок не будет, либо их один и больше. В последнем случае скобки обязательно будут присутствовать. Это в значительной степени упрощает разработку грамматики для определения параметров и выглядит она так:

<parametr> := <value> | <value><parametr> | )

<value> := идентификатор | число

Как можно заметить, <value> и <name> отличаются лишь тем, что в первом дополнительно есть определение числа. Это было сделано с целью исключить лишнюю проверку на правильность введенных данных. То есть <name> применяется там, где должно быть только имя: имя переменной, которой присваивают значение; имя функции и т.д. А <value> используется там, где допустимо использование чисел. Например, при описании параметров.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 22 |
| Изм. | Лист | № докум | Подпись | Дата |

В разрабатываем языке препроцессором может являться число, причем число может быть отрицательным, поэтому грамматика такая:

<preproc> := -<value> | <value>

Как было описано выше, программа может состоять из функций и для ее определения использовался символ <proc>. Так же, было отмечено, что функции могут иметь параметры. Кроме того, стоит помнить, что любая функция должна иметь реализацию, то есть тело. Учитывая это все, получается следующая грамматика:

<proc>:=<name> н <proc\_code> | <name> (<parametr> н <proc\_code>

Что касается тела функции, то оно состоит из различных операций:

<proc\_code> := <operation> к | <operation><proc\_code>

Так как количество операций, выполняющихся внутри функции может быть различным, то наиболее логичным вариантом для определения всех внутренних операций будет «отлов» конца тела функции, задаваемого скобкой «к». В случае, когда конец был обнаружен происходит завершение обработки кода процедуры и управление передается в последовательность, которая содержала <proc\_code> для дальнейшего разбора.

Операции, поддерживаемые языком следующие: арифметические (сложение, вычитание, сдвиги, деление, умножение), условие, переход, метка и выход из подпрограммы (конец), присваивание. Следовательно, грамматика разбора для операций следующая:

<operation>:=<action> | <metka> | вернуть | переход <name>| конец | если <if\_statment> | иначе <else\_statment>

<action>:= <name> | <name> (<operation> | <name> () | <name> = <value>| <name>=<name>() | <name>=<name>(<parametr>| <name> <op> <value>

<op>:=И | ИЛИ | НЕ | \* | - | + | / | >> | <<

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 23 |
| Изм. | Лист | № докум | Подпись | Дата |

Далее, будет разработана грамматика условия. В простом случае условный оператор формируется из самого условия, которое записывается в скобках, а потом следует код, который будет выполняться при выполнении заданного условия.

<if\_statment> := ( <condition> н <if\_code>

Условие может быть простым, то есть состоять из сравнения 2х операндов, а может быть составным, то есть состоять из сравнения нескольких операндов. В разрабатываем языке составные условия представляют собой логическое сравнение простых условий между собой.

Например, (a==b И a!=c).

Тогда грамматика будет иметь следующий вид:

<condition> := <first\_cond> <op2><condition> | <first\_cond>

<first\_cond> := <name> <op3><value>

<op2>:= И | ИЛИ

<op3>:= И | ИЛИ | == | !=

Что касается части для невыполнения условия, то ее грамматика:

<else\_statment> := н <else\_code>

<if\_code> и <else\_code> определяет операции, происходящие внутри условия, грамматика построена по принципу <proc\_code>.

Стоит отметить, что <else\_statment> и <if\_statment> были вынесены в разные последовательности одного правила для возможности осуществления вложенных условий.

Правила перехода в общем виде представлены в приложении Б.

S – начальное состояние, с которого начинается разбор, задается нетерминальным символом. В разрабатываемой грамматике – code.

#### 2.2.2.2. Разработка алгоритма построения дерева разбора

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 24 |
| Изм. | Лист | № докум | Подпись | Дата |

Результатом выполнения синтаксического разбора является построение дерева разбора, для его построения необходимо разработать соответствующий алгоритм. Схема алгоритма представлена на рисунках 6 – 14.

Дополнительно стоит отметить, что для наглядности схемы было принято решение об объединение действия «создать new\_node» для всех процедур и операций. В действительности же для каждой процедуры и операции будет создаваться свой new\_node, отличающийся типом создаваемого узла.

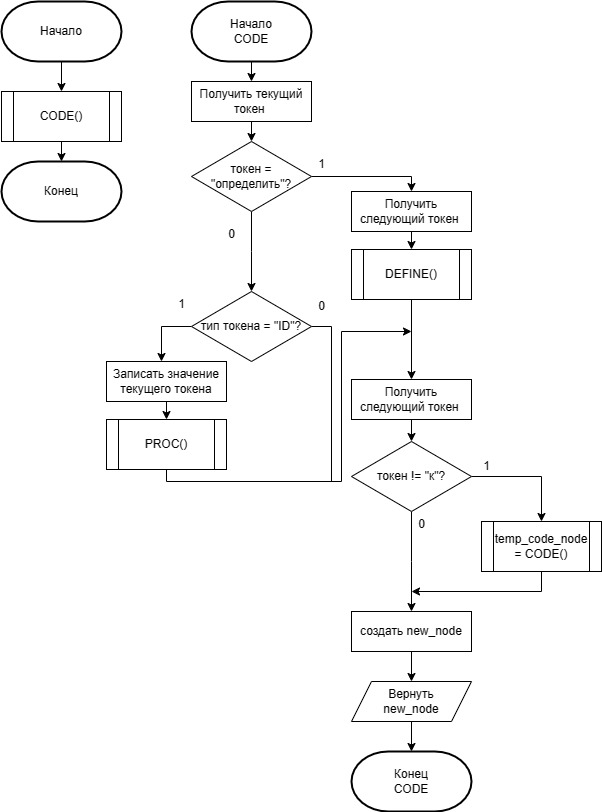


Рисунок 6 – Схема алгоритма, часть 1

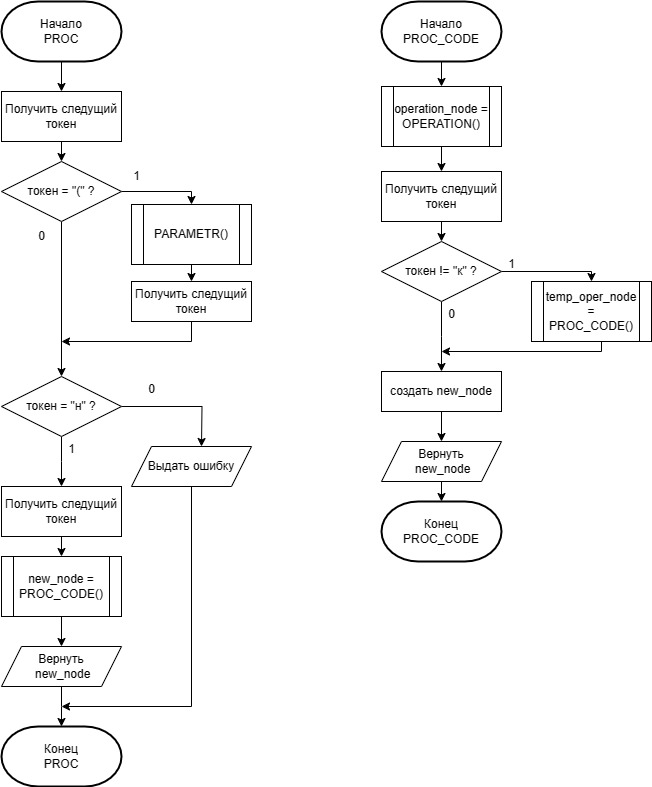


Рисунок 7 – Схема алгоритма, часть 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 25 |
| Изм. | Лист | № докум | Подпись | Дата |

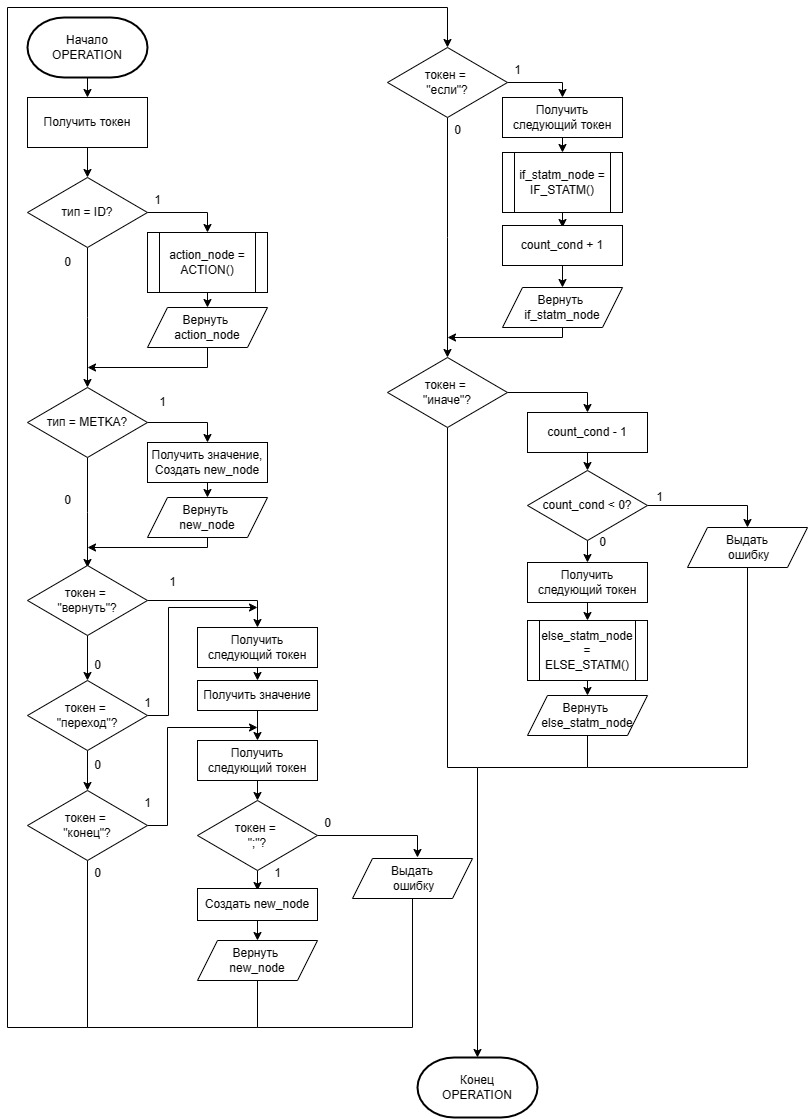


Рисунок 8 – Схема алгоритма, часть 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 26 |
| Изм. | Лист | № докум | Подпись | Дата |

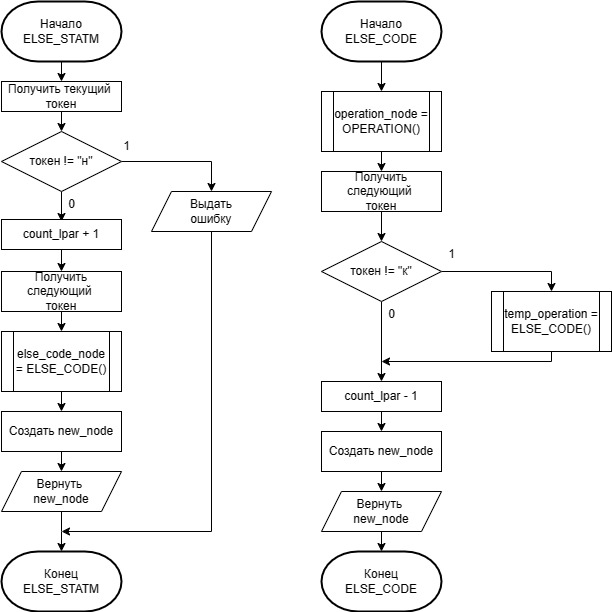


Рисунок 9 – Схема алгоритма, часть 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 27 |
| Изм. | Лист | № докум | Подпись | Дата |

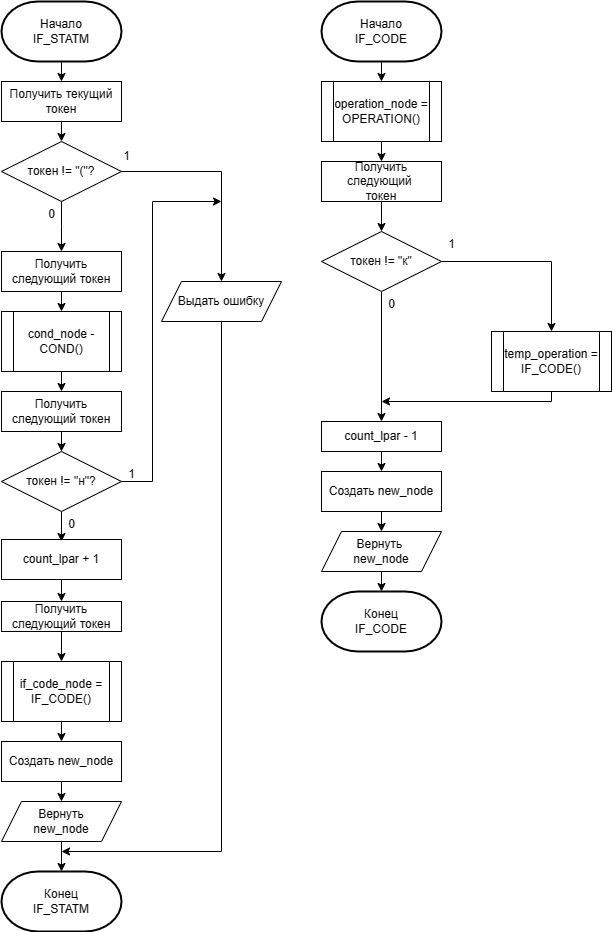


Рисунок 10 – Схема алгоритма, часть 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 28 |
| Изм. | Лист | № докум | Подпись | Дата |

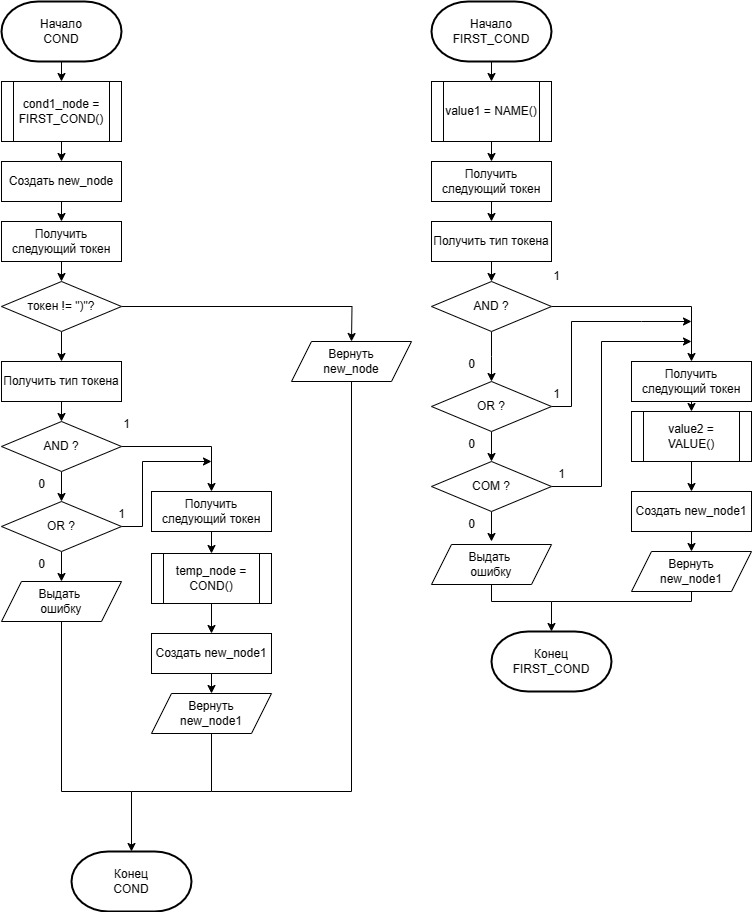


Рисунок 11 – Схема алгоритма, часть 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 29 |
| Изм. | Лист | № докум | Подпись | Дата |

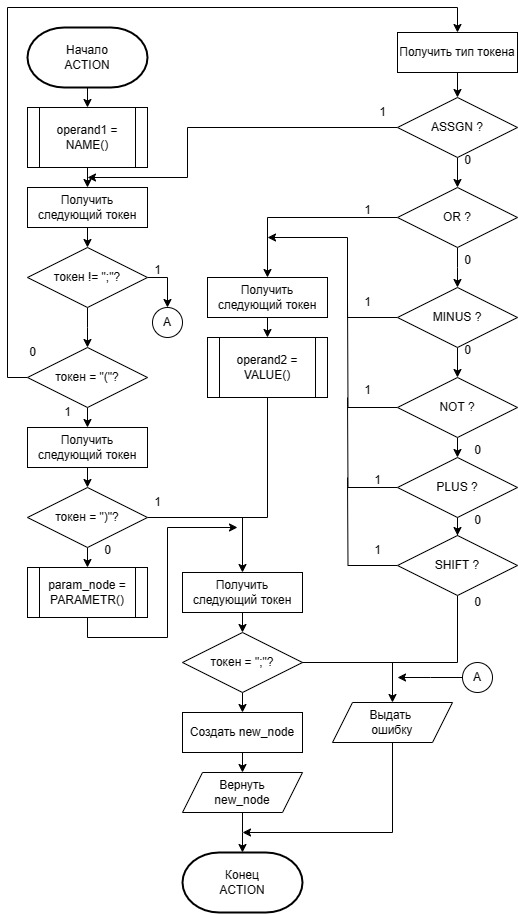


Рисунок 12 – Схема алгоритма, часть 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 30 |
| Изм. | Лист | № докум | Подпись | Дата |

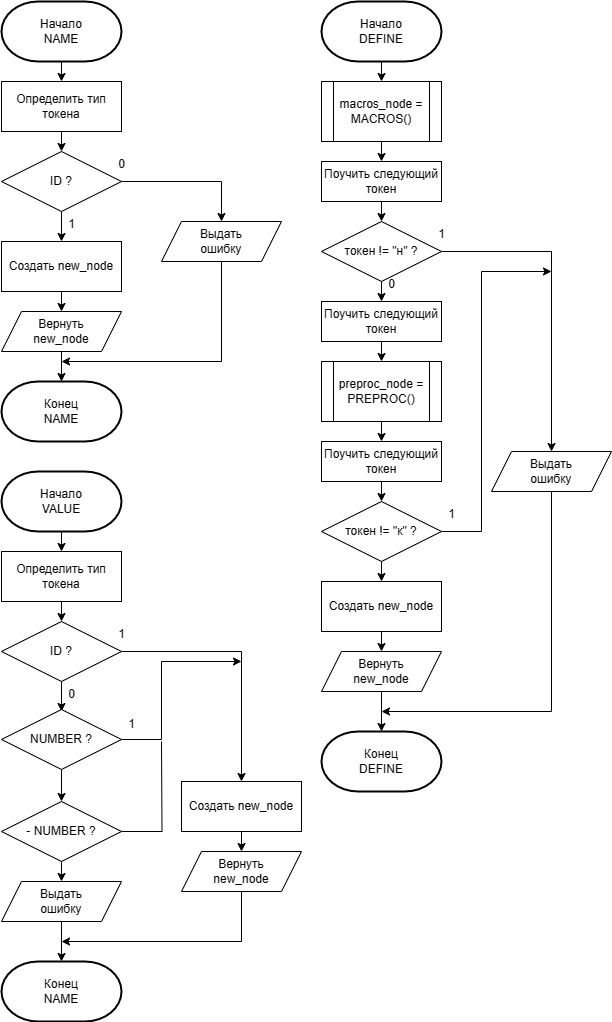


Рисунок 13 – Схема алгоритма, часть 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 31 |
| Изм. | Лист | № докум | Подпись | Дата |

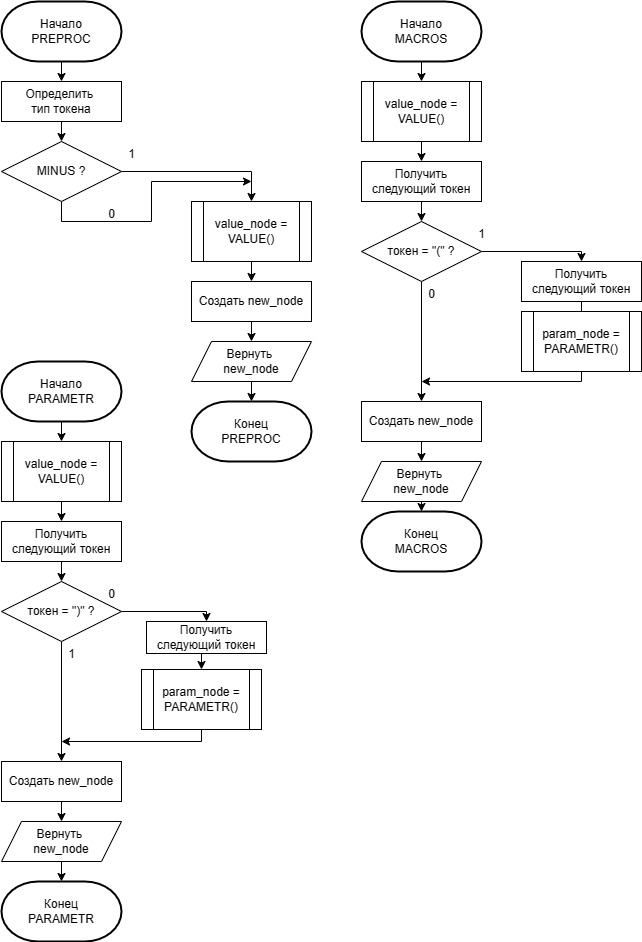


Рисунок 14 – Схема алгоритма, часть 9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 32 |
| Изм. | Лист | № докум | Подпись | Дата |

#### 2.2.2.3. Программная реализация

Для построения дерева разбора использовался стандартный алгоритм рекурсивного спуска. Как было отмечено выше для каждого узла выполняется действие «создать new\_node». В свою очередь new\_node является экземпляром класса node. Реализация класса node представлена ниже:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 33 |
| Изм. | Лист | № докум | Подпись | Дата |

class node

{

private:

string value;

node\_type type;

node\* operand1;

node\* operand2;

public:

node(node\_type type, const string& value = "",

node\* operand1 = nullptr, node\* operand2 = nullptr)

{

this->type = type;

this->value = value;

this->operand1 = operand1;

this->operand2 = operand2;

}

};

Для построения дерева был разработан класс AST, экземпляр которого создавался в классе лексического анализатора, Parser, и вызывал функцию code. Диаграмма классов представлена на рисунке 15.

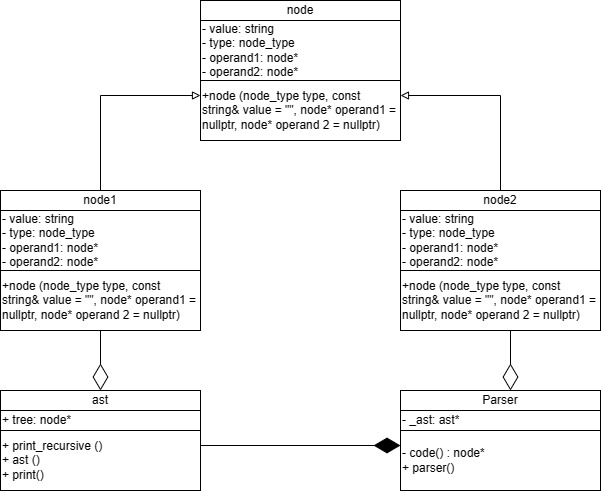


Рисунок 15 – Диаграмма классов

Класс node описывает узел синтаксического дерева. Каждый узел может обладать типом, значением и ссылками на 2 дочерних узла. Все эти параметры имеют режим доступа «private». Также стоит отметить, что обязательным параметром для узла является только тип, остальных параметров может и не быть, именно поэтому в конструкторе остальные параметры заданы, как параметры по умолчанию.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 34 |
| Изм. | Лист | № докум | Подпись | Дата |

Классы node1 и node2 являются наследниками класса node, создающиеся для использования в классах ast и parser соответственно.

В свою очередь класс ast реализует проход по дереву для отображения всех его узлов. Следовательно, класс ast является контейнером для экземпляров класса node, а отношение между ними – агрегация.

Класс parser необходим для формирования дерева разбора. Во время прохода по лексемам создаются узлы при помощи класса node. Они в свою очередь помещаются в структуру дерево. Следовательно, связь между node2 и parser – агрегация.

Отношение между ast и parser – композиция, потому что класс ast не будет сформирован, а его методы не будут использованы, если класс parser до это был уничтожен.

Пример работы синтаксического анализатора представлен на рисунках 16 -17.

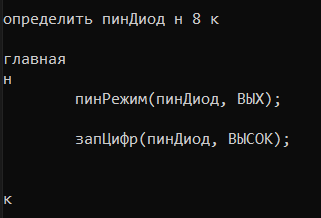


Рисунок 16 – Исходный код программы

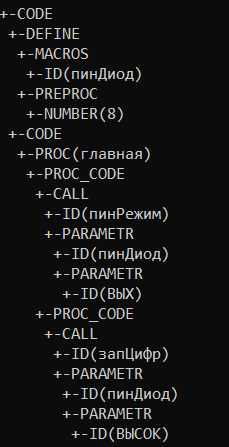


Рисунок 17 – Дерево разбора

#### 2.2.2.4. Тестирование

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 36 |
| Изм. | Лист | № докум | Подпись | Дата |

Тестирование по методу черного ящика приведено в таблице 3.

Таблица 3 – Тестирование по методу черного ящика

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 37 |
| Изм. | Лист | № докум | Подпись | Дата |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| Директива переменной | определить пинДиод н 8 к | Node::DEFINE | Node::DEFINE | Успешно |
| определить н 8 к | Ожидалось имя | Ожидалось имя | Успешно |
| определить пинДиод 8 к | Ожидалось «н» | Ожидалось «н» | Успешно |
| определить пинДиод н к | Ожидалось значение | Ожидалось значение | Успешно |
| определить пинДиод н 8 | Ожидалось «к» | Ожидалось «к» | Успешно |
| Директива функции | определить пинРег(пин) н -пин\_рег к | Node::DEFINE | Node::DEFINE | Успешно |
| определить пинРег() н -пин\_рег к | Ожидались параметры | Ожидались параметры | Успешно |
| определить пинРег пин) н -пин\_рег к | Ожидалось «н» | Ожидалось «н» | Успешно |
| определить пинРег(пин н -пин\_рег к | Ожидалось «)» | Ожидалось «)» | Успешно |

Таблица 3 - Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тело функции | Главная  н a=6; к | Node::PROC | Node::PROC | Успешно |
| Главная(пин) н а=6; к | Node::PROC | Node::PROC | Успешно |
| Главная () н а=6; к | Ожидалось значение | Ожидалось значение | Успешно |
| Главная пин) н а=6; к | Ожидалось «н» | Ожидалось «н» | Успешно |
| Главная а=6; к | Ожидалось «н» | Ожидалось «н» | Успешно |
| Главная н к | Ожидалось описание | Ожидалось описание | Успешно |
| Ввод операции | а = 6; | Node::ASSGN | Node::ASSGN | Успешно |
| a = 6 | Ожидалось «;» | Ожидалось «;» | Успешно |
| Условный оператор | Если ( а==5) н а+1; к | Node::IF | Node::IF | Успешно |
| Если ( а==5) н а+1; к иначе н а-1; к | Node:: IF, ELSE | Node::IF, ELSE | Успешно |
| иначе н а-1; к | Нет «если» | Нет «если» | Успешно |
| Если ф == 5) | Ожидалось «(« | Ожидалось «(« | Успешно |
| Если ( ф == 5 | Ожидалось «)» | Ожидалось «)» | Успешно |
| Если ( ф == 5) а+1; к | Ожидалось «н» | Ожидалось «н» | Успешно |
| Если (ф == 5) н а+1; | Ожидалось «к» | Ожидалось «к» | Успешно |
| Если ( ф == 5) н к | Ожидалось описание | Ожидалось описание | Успешно |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 38 |
| Изм. | Лист | № докум | Подпись | Дата |

Таблица 3 - Продолжение

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 39 |
| Изм. | Лист | № докум | Подпись | Дата |

39

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Условие | а != 5 | Node::COND | Node::COND | Успешно |
| a != 5 И б == 2 | Node::COND | Node::COND | Успешно |
| a != 5 == б != 2 | «==» недопустимо | «==» недопустимо | Успешно |
| а = 5 | «=» недопустимо | «=» недопустимо | Успешно |
| а == 5 И в | н\в кол-во | н\в кол-во | Успешно |
| Вложенный условный оператор | Если ( а == 5) н если ( а == 4) н а + 1; к к | Node:: IF | Node::IF | Успешно |
| Если ( а == 5) н если ( а == 4) н а + 1; к | Ожидалось «к» | Ожидалось «к» | Успешно |

### 2.2.3. Семантический анализатор

Данный этап вводится для осуществления контроля инициализации переменных, используемых в программе. Для выполнения этого действия необходимо разработать алгоритм, выполнить программную реализацию и провести тестирование.

#### 2.2.3.1. Разработка алгоритма

В разрабатываемом языке инициализацией переменной является связывание переменной с номером порта на плате Arduino через создание директивы препроцессора, а также присваивание переменной какого-то значения. Для упрощения дальнейшей разработки было принято решение о записи номера регистра в таблицу имен вместе с переменной присваивания. Номер регистра определяется свободно доступными регистрами и увеличивается каждый раз, когда происходит присваивание. Тогда проверка инициализации необходима только в параметрах и условиях. Схема алгоритма представлена на рисунке 18.

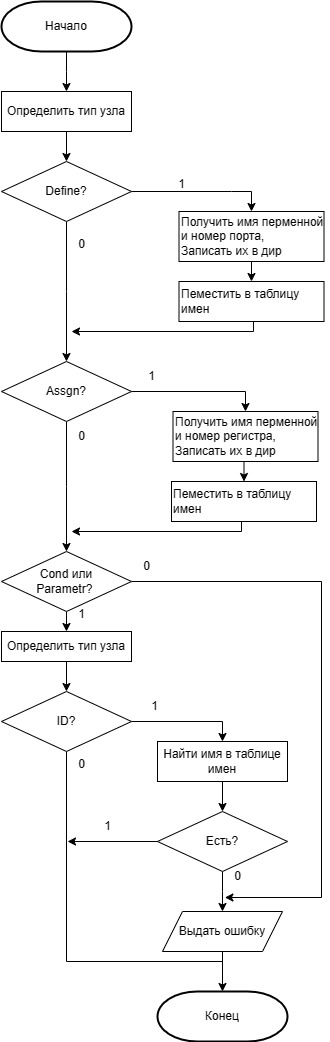


Рисунок 18 – Схема алгоритма семантического анализатора

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 39 |
| Изм. | Лист | № докум | Подпись | Дата |

Первоначально необходимо определить тип узла. Если тип равен «define» или «assgn», происходит запись имени переменной в таблицу имен, имеющую структуру, представленную на рисунке 19.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 40 |
| Изм. | Лист | № докум | Подпись | Дата |



Рисунок 19 – Структура таблицы имен

Как можно заметить, запись переменной в таблицу имен осуществляется при помощи создания элемента из 2х полей. В первом поле указывается имя переменной, а во втором – номер порта (значение), в случае если это директива или номер регистра, если тип рассматриваемого узла равен операции присваивания.

Если же тип равен «cond» или «parametr», то необходимо найти переменную и сравнить ее с таблицей имен. Для ее нахождения необходимо определить узел с типом ID. Сравнение с таблицей имен подразумевает поиск имени в таблице.

#### 2.2.3.2. Программная реализация

В разрабатываемом языке инициализация переменной должна происходить до момента ее использования через директиву или во время ее использования в операции присваивания. Такого порядка следования можно достичь за счет использования алгоритма рекурсивного обхода дерева разбора. Кроме того, использование дерева разбора позволяет обращаться напрямую к узлам для получения значений, так как порядок узлов известен.

Например, для получения значения имени макроса возможно следующее обращение:

node\* macros = current\_node->operand1;

string name\_macros = macros->operand1->value;

Для возможности создания директивы и таблицы имен были разработаны соответствующие классы. Диаграмма классов представлена на рисунке 20.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 41 |
| Изм. | Лист | № докум | Подпись | Дата |

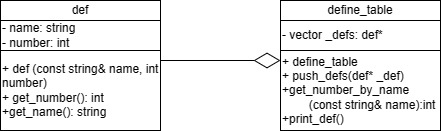


Рисунок 20 – Диаграмма классов семантического анализатора

Класс define\_table является таблицей имен, сохраняющий экземпляры класса def в массив при помощи метода push\_defs. Для демонстрации этой зависимости использовалось отношение агрегации между классами.

Что касается класса def, то он разрабатывался для хранения информации об одной переменной: имени и номера регистра или порта. Оба этих параметра имеют режим доступа «private». Для их получения были разработаны методы get.

Листинг программы, содержащий описание классов и алгоритм семантического анализа приведен в приложении В.

#### 2.2.3.3. Тестирование

Тестирование по методу черного ящика приведено в таблице 4.

Таблица 4 – Тестирование методом черного ящика

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 42 |
| Изм. | Лист | № докум | Подпись | Дата |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| Использование  переменной | определить пин н 8 к  чтЦифр(пин); | Node::CALL | Node::CALL | Успешно |
| определить пин н 8 к  чтЦифр(пин1); | Неизвестно пин1 | Неизвестно пин1 | Успешно |
| в = 5; если (в ==6) | Node::ASSGN, IF | Node::ASSGN, IF | Успешно |
| в = 5; если (г ==6) | Неизвестно г | Неизвестно г | Успешно |
| Кол-во параметров | чтЦифр(пин) | Node::CALL | Node::CALL | Успешно |
| чтЦифр() | Ожидались параметры | Ожидались параметры | Успешно |
| чтЦифр(пин, выс) | Н/в кол-во параметров | Н/в кол-во параметров | Успешно |
| Правильность параметров | пинРежим(пин, ВЫХ) | Node::PROC | Node::PROC | Успешно |
| пинРежим(пин, пин1) | Ожидалось «ВХ/ВЫХ» | Ожидалось «ВХ/ВЫХ» | Успешно |
| пинРежим(ВЫХ, ВЫХ) | Ожидалось имя пина | Ожидалось имя пина | Успешно |

### 2.2.4. Разработка генератора промежуточного кода

Для последовательности генерации исполняемого файла вводится дополнительный этап генерации промежуточного представления кода, который преобразует дерево синтаксического разбора в ассемблерные команды. Кроме того, на данном этапе происходит присоединение пользовательских функций.

На данном этапе работы необходимо описать присоединение пользовательских функций, разработать алгоритм генерации, выполнить программную реализацию и провести тестирование.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 43 |
| Изм. | Лист | № докум | Подпись | Дата |

#### 2.2.4.1. Присоединение пользовательских функций

Как было отмечено ранее, пользовательские функции имеют стандартное определение, то есть оно не изменяется ни при каких обстоятельствах. Следовательно, выполнять проверку на корректность их написание излишне. Именно поэтому их подключение было вынесено в этап генерации промежуточного кода.

Для подключения функций создается список, содержащий имена всех функций программы. Список имеет следующее содержание: вектора\_прерыв, лин\_конец, порт\_вых, порт\_вх, цифр\_пин\_порт, цифр\_пин\_маска, цифр\_пин\_таймер, иниц\_конец, очичтка\_неиниц\_данных, нет\_прерыв, вклТаймер, чтЦифр, запЦифр, пинРежим, микро, задержка, РегистрВых, РегистрВх, проверка, вектор\_16, главная, конец, стоп.

Определение всех функций с комментариями, поясняющими назначение операций приведено в приложении Г.

В функцию «главная» упаковывается программа написанная пользователем.

#### 2.2.4.2. Разработка алгоритма генерации

Схема алгоритма для генерации промежуточного кода представлена на рисунке 21.

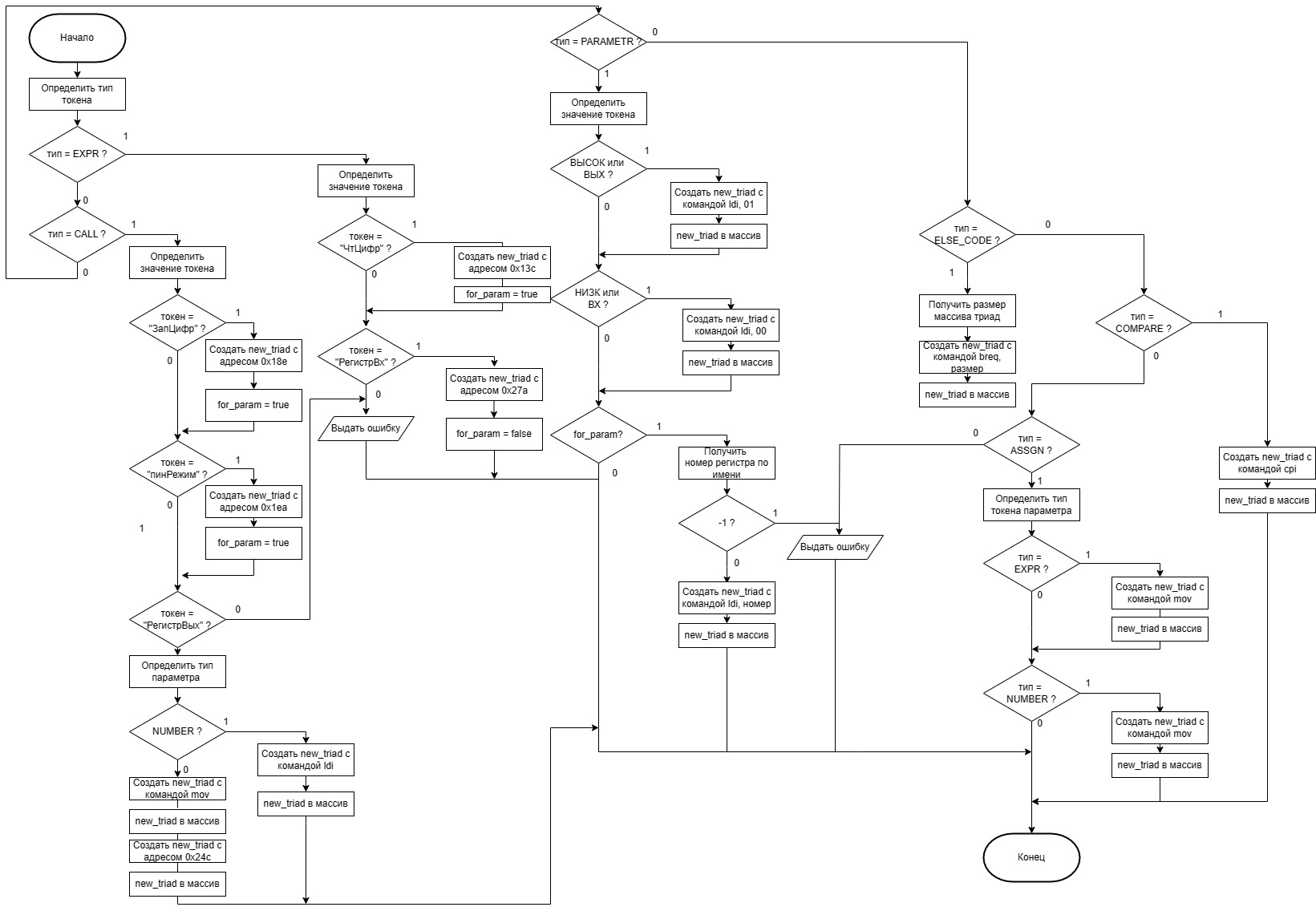
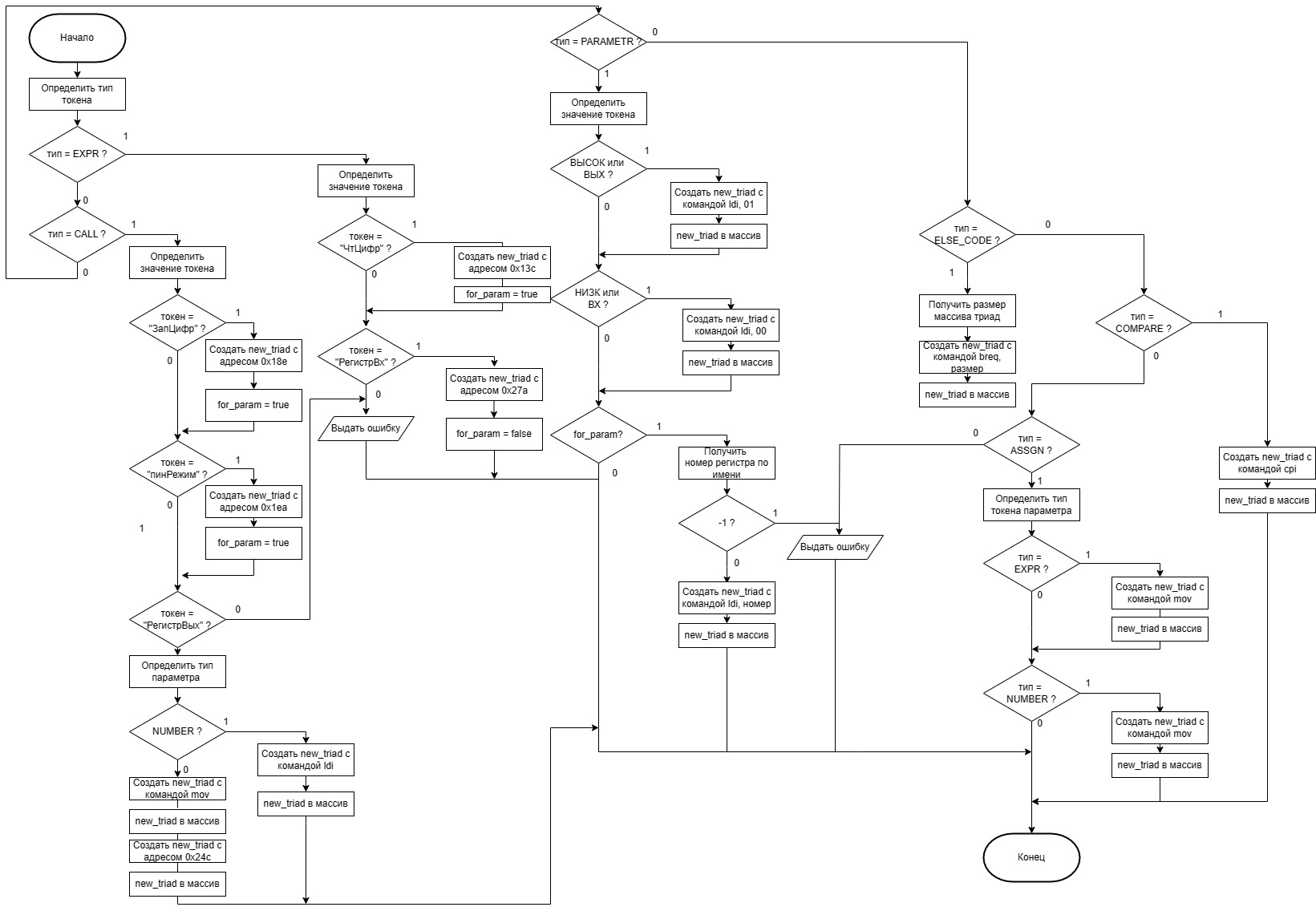


Рисунок 21 – Схема алгоритма генерации промежуточного кода

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 44 |
| Изм. | Лист | № докум | Подпись | Дата |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 45 |
| Изм. | Лист | № докум | Подпись | Дата |

Первоначально происходит определение типа токена. Если его тип равен EXPR или CALL, то создается new\_triad, в которой в качестве параметров передается «call, <адрес начала функции>». Для повышения читаемости схемы алгоритма описание всех параметров было опущено, но описание основного параметра (адреса – для вызова функции, команды – для операции) оставлено. После определения типа происходит определение имени функции, это необходимо для того, чтобы подставить правильный адрес начала, а также для того, чтобы выполнить проверку на корректность. Так как EXPR возможен только в функциях, которые возвращают значение – чтЦифр и РегистрВх, то во всех остальных случаях ошибка. В случае с CALL все наоборот: функции не должны возвращать значение.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 46 |
| Изм. | Лист | № докум | Подпись | Дата |

Дополнительно стоит отметить, что в функции РегистрВых происходит определение не всех параметров, а только того, который отвечает за данные. И для корректного отображения в команды ассемблера необходимо определить его тип: если тип – число, то команда ldi, иначе – команда mov.

Также в алгоритме был введен флаг for\_param, необходимый для пропуска разбора параметров, относящихся к функции РегистрВых. Его использование обусловлено тем, что пропуск узла с данной функцией и ее потомками при обходе дерева невозможен, так как это приведет к усложнению разработки и отладки кода.

Продолжая описание разбора параметров, стоит отметить, что при попадании в этот узел первоначально происходит проверка на наличие ключевых слов (НИЗК, ВЫСОК и т.д.) и формирования соответствующей команды. Если же это не ключевое слово, то происходит попытка получить регистр параметра, через таблицу имен. Если такого параметра нет, то выдается ошибка, иначе происходит формирование команды с проверкой флага, о котором было написано выше. Если флаг установлен в состояние, позволяющее выполнить формирование команды, то в качестве параметров в команду передается полученный номер регистра или число, означающее номер порта.

Команда для операции ELSE\_CODE – breq, так как в разрабатываемом языке допустимы лишь операции сравнения на равно или не равно. В качестве параметра передается число, означающее кол-во операций, выполняющихся в теле if.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 47 |
| Изм. | Лист | № докум | Подпись | Дата |

#### 2.2.4.3. Программная реализация

Для реализации алгоритма был выбран рекурсивный обход дерева разбора. Кроме того, при обходе необходимо выполнять проверку с таблицей имен для получения регистра адреса. В пункте про семантический анализ это было описано и реализовано. И в качестве минимизации времени выполнения программы было принято решение об объединении семантического анализа и генерацию промежуточного кода в одну функцию.

Что касается вычисления параметра для команды breq, то для решения этой задачи был создан массив, временно хранящий набор команд, с момента обнаружения узла условия до момента узла ELSE\_CODE. Далее вычисляется размер списка, и передается в качестве параметра с командой breq.

Листинг программы приведен в приложении В.

Результатом выполнения данного этапа компиляции является 2 списка: имен функций и ассемблерных команд для главной функции. Они будут использованы на этапе генерации кода.

2.2.4.4. Тестирование

Тестирование по методу белого ящика с использованием стратегии перекрытия операторов приведено в таблице 5.

Таблица 5 – Тестирование методом белого ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| Вызов функции с возвращаемым значением | ЧтЦифр | Call 0x13c | Call 0x13c | Успешно |
| РегистрВых | Call 0x27a | Call 0x27a | Успешно |

Таблица 5 - Продолжение

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 48 |
| Изм. | Лист | № докум | Подпись | Дата |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вызов функции без возвращаемого значения | запЦифр | Call 0x18e | Call 0x18e | Успешно |
| РегистрВых(12) | Call 0x24c, ldi 24, 12 | Call 0x24c, ldi 24, 12 | Успешно |
| РегистрВых(в) | Call 0x24c, mov 24, 8\* | Call 0x24c, mov 24, 8 | Успешно |
| пинРежим | Call 0x1ea | Call 0x1ea | Успешно |
| Параметры | ВЫСОК | Ldi 22, 01 | Ldi 22, 01 | Успешно |
| ВХ | Ldi 22, 00 | Ldi 22, 00 | Успешно |
| For\_param = true; number = -1; | Неизвестный параметр | Неизвестный параметр | Успешно |
| For\_param = true; number = 1; | Ldi 24, 01 | Ldi 24, 01 | Успешно |

\* - следует понимать, что 8 – порт, который был определен, как переменная «в», через директиву.

### 2.2.5. Разработка генератора кода

Последним этапом компиляции является генерация исполняемого кода за счет перевода ассемблерной команды в 16 формат и запись полученных цифр в файл hex формата. Для осуществления этих действий необходимо разработать соответствующие алгоритмы, описать формирование контрольной суммы, выполнить программную реализацию и провести тестирование.

#### 2.2.5.1. Разработка алгоритмов

Для перевода ассемблерной команды необходимо разработать алгоритм преобразования. Кроме того, для записи чисел в файл также необходимо разработать алгоритм.

Алгоритм перевода ассемблерной команды в 16 формат представлен на рисунке 22.

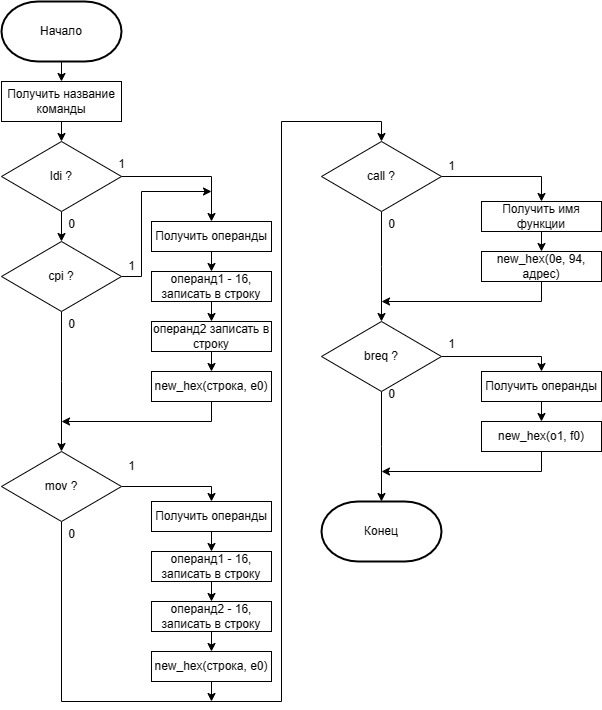


Рисунок 22 – Схема алгоритма перевода команды

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 49 |
| Изм. | Лист | № докум | Подпись | Дата |

Несмотря на то, что команды ассемблера имеют следующий вид: <название команды> <номера регистров>, в таком же порядке происходит вычисление 16-ричного формата, записывается команда справа налево, то есть сначала будет записан 16-ричный формат регистров, а потом номер команды.

Команда ldi выполняет запись числа в регистр. Для преобразования ее в 16-ричный формат необходимо получить первый операнд, отвечающий за номер регистра, вычесть из него 16, так как команда может работать только с РОН 16-31, полученный результат записать в строку. Далее получить второй операнд, преобразовать его в строку и записать в строку с первым регистром. Это и будет 1-ой частью 16-ричной команды. Вторая часть – е0 – код операции.

Аналогично для команды cpi. Единственное код операции – 30.

Что касается команды mov, то она подразумевает работу с 2мя регистрами. Следовательно, вычислять первую часть 16-чной команды необходимо вычитанием 16 из значения операндов и запись полученных значений в одну строку.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 50 |
| Изм. | Лист | № докум | Подпись | Дата |

Для команд, содержащих непосредственный адрес в качестве параметров формирование 16-ричного кода происходит при помощи разделения адреса на символы и запись их в соответствующие поля в формате команды.

Дополнительно стоит отметить, что в арабской записи подлежат первые 16 бит команды. Если же формат команды представлен 32 разрядами, то справа налево запишется только одна часть.

Формат для преобразования команды call в 16-чный формат представлена на рисунке 23.

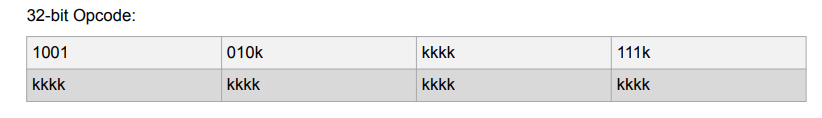


Рисунок 23 – Формат для преобразования команды call

В качестве примера будет рассмотрен вызов функции по адресу 0х18e. Первоначально происходит перевод адреса в 2СС - 000 0001 1000 1110. Первый нуль отбрасывается при формировании. Далее идет подстановка символов в формат команды:

1-ый байт: как можно заметить, тут отсутствует буква k. Это значит, никаких символов подставлять не надо.

2-ой байт: тут содержится одна буква k. Из адреса функции извлекается первый символ и записывается в формат.

0 | 00 0001 1000 1110

010k 🡪 0100

3-ий байт: 4 буквы k – из адреса извлекаются следующие 4 цифры.

00 00 | 01 1000 1110

kkkk 🡪 0000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 51 |
| Изм. | Лист | № докум | Подпись | Дата |

4-ый байт: 1 буква k

0 | 1 1000 1110

111k 🡪 1110 и т.д.

Единственно, о чем стоит помнить, что при нехватке цифр в адресе функции в конец дописываются нули, которые дальше используются при формировании команды.

Результатом преобразования команды call 0x18e в 16-ричный формат является следующая последовательность: 0e 94 c7 00

Аналогичным образом будет происходить преобразование для команды breq. Единственное отличие будет заключатся в размерности команды.

Алгоритм записи значений в файл представлен на рисунке 24.

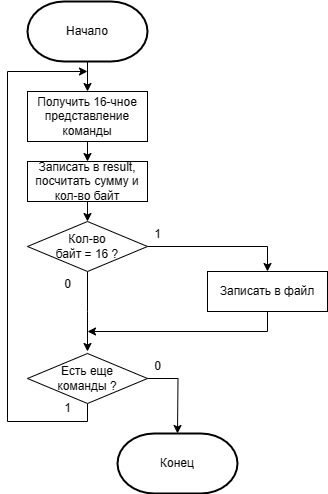


Рисунок 24 – Алгоритм записи значений в файл

Сначала получаем команду, потом записываем ее в результирующую строку. Одновременно происходит подсчет кол-ва символов (байт) и контрольной суммы. Если символов 16, то происходит запись результирующей строки в файл вместе с контрольной суммой. Далее происходит проверка на наличие команд. Если команд больше нет, то происходит запись содержимого результирующей строки и символа конца в файл.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 52 |
| Изм. | Лист | № докум | Подпись | Дата |

Формат строки в hex фале представлен на рисунке 25.

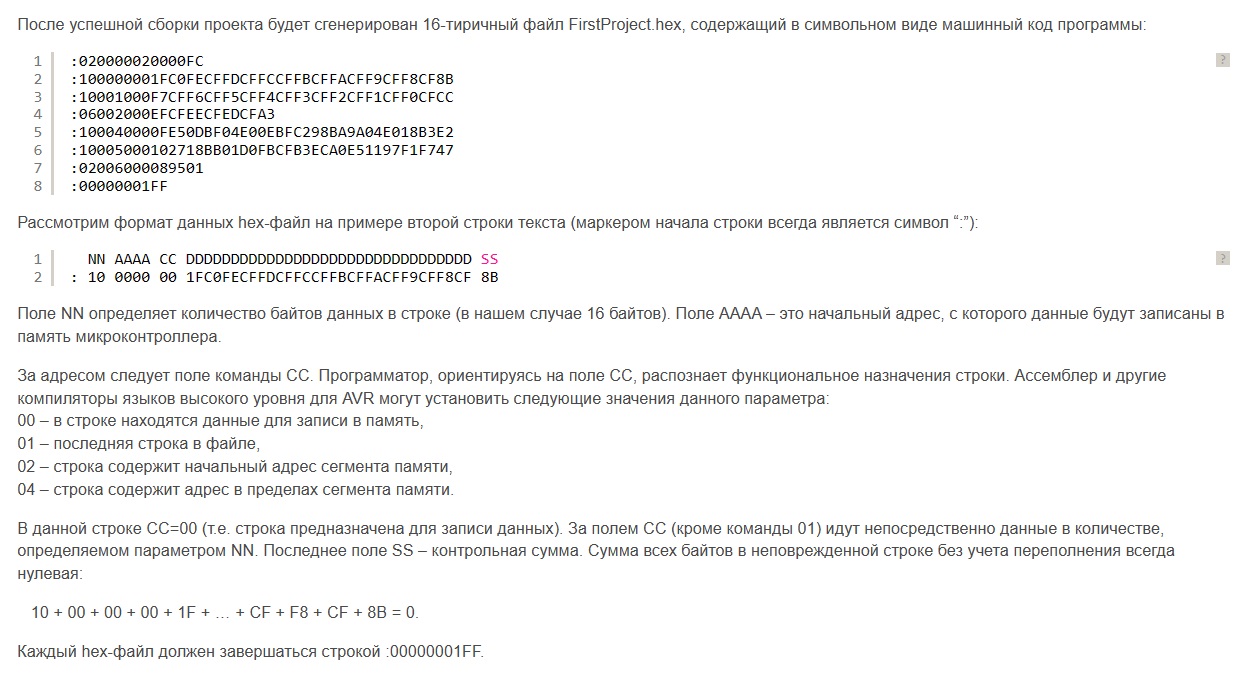


Рисунок 25 – Формат строки в hex файле

NN – количество байт в данной строке

АААА – начальный адрес

СС – функциональное назначение строки

00 – в строке находятся данные для записи в память,

01 – последняя строка в файле,

02 – строка содержит начальный адрес сегмента памяти,

04 – строка содержит адрес в пределах сегмента памяти.

DD - данные

SS – контрольная сумма.

2.2.5.2. Контрольная сумма

Как было сказано ранее, при формировании hex файла в конец строки записывается контрольная сумма. В случае с созданием hex файла для прошивки МК контрольная сумма не подразумевает восстановление некорректных данных. Она служит лишь для проверки их правильности.

Алгоритм формирования контрольной суммы следующий:

1. Строка разбивается на байты;
2. Вычисляется сумма байтов;
3. Полученная сумма инвертируется;
4. К результату добавляется 1;
5. Из полученного числа извлекают последние 2 цифры – контрольная сумма.

#### 2.2.5.3. Программная реализация

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 53 |
| Изм. | Лист | № докум | Подпись | Дата |

Как было отмечено ранее, к моменту запуска генератора кода уже сформированы 2 списка, содержащих имена функций и ассемблерные команды для «главной» функции.

Реализация генерации исполняемого кода подразумевает проход по первому списку и запись набора 16-ричных команд в hex файл. Набор команд определяется именем функции. Это подразумевает, что все операции, содержащиеся в теле функции, были заранее переведены в 16-ричный формат и полученный набор цифр был «зарезервирован» за именем функции.

Если во время прохода по списку обнаружено, что имя функции «главная», то формирование 16-ричных команд происходит в соответствии с алгоритмом, описанном выше.

Для того, чтобы получить последние 2 цифры контрольной суммы она первоначально записывается в файл, потом считываются 2 последние цифры, дописываются в результирующую строку, которая потом записывается в исполняемый файл.

Листинг кода приведен в приложении Г.

#### 2.2.5.4. Тестирование

Тестирование по методу белого ящика с использованием стратегии перекрытия операторов представлено в таблице 6.

Таблица 6 – Тестирование методом белого ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| ldi | Ldi 24, 4 | 84 | 84 | Успешно |
| Ldi 24, 12 | 8c | 8c | Успешно |

Таблица 6 - Продолжение

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 54 |
| Изм. | Лист | № докум | Подпись | Дата |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| call | 398 | 0e 94 c7 00 | 0e 94 c7 00 | Успешно |
| 490 | 0e 94 f5 00 | 0e 94 f5 00 | Успешно |
| 316 | 0e 94 9e 00 | 0e 94 9e 00 | Успешно |
| 588 | 0e 94 26 01 | 0e 94 26 01 | Успешно |
| 634 | 0e 94 3d 01 | 0e 94 3d 01 | Успешно |
| cpi | Cpi 18, 4 | 24 | 24 | Успешно |
| mov | Mov 19, 18 | 32 | 32 | Успешно |

## 2.3. Описание поведения

Для описания поведения компилятора с течением времени была разработана диаграмма последовательности, представленная на рисунке 26.

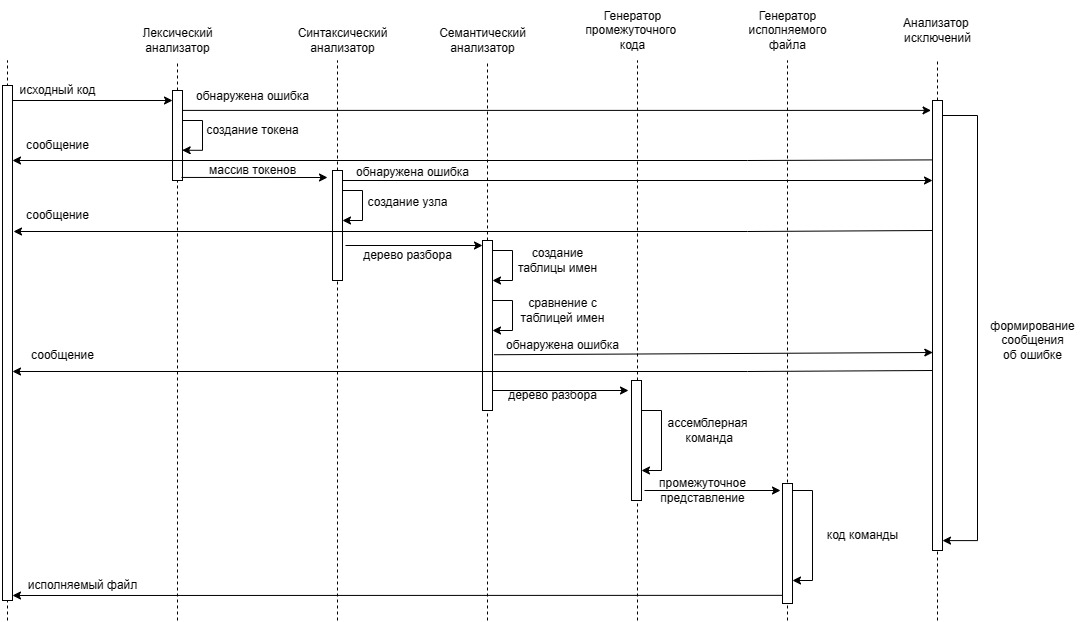


Рисунок 26 – Диаграмма последовательности

Инициализатором трансляции исходного кода является компилятор, передающий управление лексическому анализатору.

В свою очередь лексический анализатор выполняет разбиение текста программы на слова, определение типа введенного слова и создание элемента, содержащего само слово и его тип. Такой элемент называется токеном. Кроме того, анализатор проверяет корректность введенных данных. В случае возникновения некорректных данных управление передается анализатору исключений. Если же введенные данные корректны, то управление передается синтаксическому анализатору, инициализируя при это действие передачи массива токенов.

Анализатор исключений инициализирует действие о формировании сообщения об ошибки и передает управление компилятору.

При получении управления синтаксический анализатор проверяет правильность полученных конструкций при помощи заданной грамматики, последовательно перебирая массив токенов. В результате выполнения анализа формируется дерево разбора. В случае несоответствия полученного токена с тем, который описан в грамматике управление передается анализатору исключения. При отсутствии ошибок управление передается семантическому анализатору, инициализируя при этом действие передачи дерева разбор.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 55 |
| Изм. | Лист | № докум | Подпись | Дата |

Что касается семантического анализатора, то он проверяет корректность программы, но с той точки зрения были ли объявлены переменны до того, как их начали использовать. Для выполнения этого этапа происходит инициализация действия создания таблицы имен, при котором происходит обход синтаксического дерева и записывания переменных, имеющих определение, в таблицу. Далее инициализируется действие сравнение с таблицей имен. В случае обнаружения неинициализированной переменной управление передается анализатору исключений, иначе – генератору промежуточного кода.

Генератор промежуточного кода, при получении управления, формирует ассемблерные команды, построенные на основе синтаксического дерева, чью инициализацию действия передачи выполнил семантический анализатор. В свою очередь генератор является инициализатором действия передачи промежуточного представления генератору исполняемого файла.

Генератор исполняемого файла инициализирует действие перевода каждой ассемблерной команды в 16-ричный код и запись значений в файл. Далее управление передается компилятору.

## 2.4. Программная реализация

Как видно из диаграммы последовательности, все этапы должны выполняться последовательно; при возникновении ошибки последующие этапе не выполняются. Для реализации такого поведения возможно использование конструкции try – catch. В качестве обработчика исключений можно использовать уже разработанный класс Exception. Листинг программы представлен ниже.

try

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 56 |
| Изм. | Лист | № докум | Подпись | Дата |

{

\_lex->split();

//\_lex->print();

\_ast->tree = code();

\_ast->print();

\_all\_triads->create\_obj(\_ast->tree);

\_all\_triads->print\_triads();

}

catch (Exception ex)

{

cout << ex.getMessage() << '\n';

}

# ЗАКЛЮЧЕНИЕ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 57 |
| Изм. | Лист | № докум | Подпись | Дата |

В ходе выполнения курсового проекта был разработан программный продукт для создания тестов автоматизированной проверки блоков релейной логики, отличающийся от существующих аналогов одновременной поддержкой русского языка, простотой написания программы, компактностью исполняемого модуля и его эффективностью.

Для оценки эффективности исполняемого модуля использовалась компиляция 2х одинаковых программ на разных языках: С для Arduino, разработанный язык. В результате объем исполняемого кода, сформированного компилятором Arduino, составил 4Кб, в то время как у разработанного компилятора – 3Кб.

В качестве направления дальнейшего развития можно выбрать создание среды разработки.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 58 |
| Изм. | Лист | № докум | Подпись | Дата |

1. Документация по C++ [Электронный курс] – режим доступа [https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160](https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160%20)

2. ГОСТ «Отчет по НИР» [Электронный курс] – режим доступа <https://www.tsu.ru/upload/medialibrary/235/gost_7.32_2017.pdf>

3. Wirth N. Compiler Construction, Addison Wesley, 1996. – 176 c.

4. Свердлов, С.З., Конструирование компиляторов, Изд-во «Lambert Academic Publishing», 2015. – 575 с.

5. Datasheet ATmega328P [Электронный курс] – режим доступа

<https://www.microchip.com/content/dam/mchp/documents/MCU08/ProductDocuments/DataSheets/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>

6. Instruction Set Manual [Электронный курс] – режим доступа <https://www.microchip.com/content/dam/mchp/documents/MCU08/ProductDocuments/ReferenceManuals/AVR-InstructionSet-Manual-DS40002198.pdf>

# Приложение А

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 59 |
| Изм. | Лист | № докум | Подпись | Дата |

(обязательное)

**Листинг кода**

Token.h

#pragma once

#include <iostream>

#include <string>

//#include "Lexer.h"

using namespace std;

enum class token\_type;

class token

{

public:

string lexeme;

token\_type type;

token(const string& lexeme, token\_type type)

{

this->lexeme = lexeme;

this->type = type;

}

};

Lexer.h

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include "Token.h"

#include "Exception.h"

using namespace std;

string key\_word[] { "н", "к", "И", "ИЛИ", "НЕ", "определить", "если", "иначе", "переход", "прервать", "выбор", "конец", "вернуть"};

enum class token\_type

{

ID, //переменная

NUMBER, //число

KEY\_WORD, //ключевое слово

DLM, // , ;

ASSGN, // =

COM, // != == > <

LPAR, // ( н

RPAR, // ) к

PAR, // "

MINUS, // -

PLUS,

METKA, // <name>:

OR,

AND,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 60 |
| Изм. | Лист | № докум | Подпись | Дата |

NOT,

STAR,

SHIFT, // << >>

DIV,

};

char delimetr[] { '(', ')', ';', ' ', '-', '\n', '=', ':', '>', '<', '\t', '/', ',', '!', '+', '\*'};

char invalid\_del[]{'(', '=', ':', '>', '<', '!'};

string composite\_del[]{ "<<", ">>", "==", "=", "!=" };

enum class States // состояния автомата

{

S, //начальное

NUM, // число

DLM, //разделитель

WORD, //слово

ER, // ошибка

FIN, //конечное

};

char hexFormat[]{ 'A', 'B', 'C', 'D', 'E', 'F' };

bool isDigit(char symbol)

{

bool exist = find(begin(hexFormat), end(hexFormat), symbol) != end(hexFormat);

if (symbol < '0' || symbol>'9')

{

if (exist) return true;

else return false;

}

else return true;

}

bool isWord(char symbol)

{

if (symbol < 'а' || symbol >'я')

{

if (symbol < 'А' || symbol >'Я') return false;

else return true;

}

else return true;

}

bool isDelimetr(char symbol)

{

bool exists = find(begin(delimetr), end(delimetr), symbol) != end(delimetr);

if (exists) return true;

else return false;

}

bool isInvalidDelimetr(char symbol)

{

bool exists = find(begin(invalid\_del), end(invalid\_del), symbol) != end(invalid\_del);

if (exists) return true;

else return false;

}

bool isCompositeDelimetr(const string& del)

{

bool exists = find(begin(composite\_del), end(composite\_del), del) != end(composite\_del);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 61 |
| Изм. | Лист | № докум | Подпись | Дата |

if (exists) return true;

else return false;

}

int NumberKey(const string& temp\_token)

{

auto exsist = find(begin(key\_word), end(key\_word), temp\_token);

int res = distance( key\_word, exsist);

return res;

}

bool isKeyWord(const string& temp\_token)

{

bool exists = find(begin(key\_word), end(key\_word), temp\_token) != end(key\_word);

if (exists) return true;

else return false;

}

class lexer

{

private:

string code;

vector <token\*> \_tokens;

int current\_token\_index = 0;

bool flag = true;

public:

int number\_string = 0;

lexer(const string& code)

{

this->code = code;

}

void split() //разделяем код на токены

{

//переменная для хранения символов, найденных до разделителя

string temp\_token;

States \_state = States::S;//начало

for (char symbol : code) //по символу строки

{

if (flag == true)

{

switch (\_state)

{

case States::S: //+

{

if (isDigit(symbol))

{

temp\_token += symbol;

\_state = States::NUM;

}

if (isWord(symbol) )

{

temp\_token += symbol;

\_state = States::WORD;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 62 |
| Изм. | Лист | № докум | Подпись | Дата |

}

// составные разделители

if (symbol == '=')

{

temp\_token += symbol;

\_state = States::DLM;

}

if (symbol == '!')

{

temp\_token += symbol;

\_state = States::DLM;

}

if (symbol == '<')

{

temp\_token += symbol;

\_state = States::DLM;

}

if (symbol == '>')

{

temp\_token += symbol;

\_state = States::DLM;

}

if (symbol == '-')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::MINUS);

\_tokens.push\_back(new\_token);

}

if (symbol == '+')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::PLUS);

\_tokens.push\_back(new\_token);

}

if (symbol == '\*')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::STAR);

\_tokens.push\_back(new\_token);

}

if (symbol == ';')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::DLM);

\_tokens.push\_back(new\_token);

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 63 |
| Изм. | Лист | № докум | Подпись | Дата |

if (symbol == '(')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::LPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == ')')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::RPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '\n')

{

number\_string++;

}

if (!isDelimetr(symbol) && !isWord(symbol) && !isDigit(symbol))

{

temp\_token += symbol;

\_state = States::ER;

break;

}

break;

}

case States::NUM: //+

{

if (isDigit(symbol))

{

temp\_token += symbol;

\_state = States::NUM;

}

else

{

if (isDelimetr(symbol))

{

if (isInvalidDelimetr(symbol))

{

\_state = States::ER;

}

else

{

token\* new\_token = new token(temp\_token, token\_type::NUMBER);

\_tokens.push\_back(new\_token);

temp\_token.clear();

if (symbol == ';' || symbol == ',')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::DLM);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 64 |
| Изм. | Лист | № докум | Подпись | Дата |

\_tokens.push\_back(new\_token);

}

if (symbol == '\*')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::STAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '(')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::LPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == ')')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::RPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '\n')

{

number\_string++;

}

\_state = States::S;

}

}

else

{

temp\_token += symbol;

\_state = States::ER;

}

}

break;

}

case States::DLM: //+

{

if (isDigit(symbol) || isWord(symbol) || symbol == ' ')

{

if (temp\_token == "=")

{

token\* new\_token = new token(temp\_token, token\_type::ASSGN);

\_tokens.push\_back(new\_token);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 65 |
| Изм. | Лист | № докум | Подпись | Дата |

}

else if (temp\_token == "==" || temp\_token == "!=")

{

token\* new\_token = new token(temp\_token, token\_type::COM);

\_tokens.push\_back(new\_token);

}

else if (temp\_token == ">" || temp\_token == "<")

{

token\* new\_token = new token(temp\_token, token\_type::COM);

\_tokens.push\_back(new\_token);

}

else if (temp\_token == ">>" || temp\_token == "<<")

{

token\* new\_token = new token(temp\_token, token\_type::SHIFT);

\_tokens.push\_back(new\_token);

}

else

{

\_state = States::ER;

break;

}

temp\_token.clear();

if (symbol != ' ')

{

temp\_token += symbol;

}

if (symbol == '(' || symbol == '{')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::LPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == ')' || symbol == '}')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::RPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '\*')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::STAR);

\_tokens.push\_back(new\_token);

}

\_state = States::S;

}

else

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 66 |
| Изм. | Лист | № докум | Подпись | Дата |

{

temp\_token += symbol;

}

break;

}

case States::WORD:

{

if (isDelimetr(symbol))

{

if (isKeyWord(temp\_token))

{

switch (NumberKey(temp\_token))

{

case 0:

{

token\* new\_token = new token(temp\_token, token\_type::LPAR);

\_tokens.push\_back(new\_token);

break;

}

case 1:

{

token\* new\_token = new token(temp\_token, token\_type::RPAR);

\_tokens.push\_back(new\_token);

break;

}

case 2:

{

token\* new\_token = new token(temp\_token, token\_type::AND);

\_tokens.push\_back(new\_token);

break;

}

case 3:

{

token\* new\_token = new token(temp\_token, token\_type::OR);

\_tokens.push\_back(new\_token);

break;

}

case 4:

{

token\* new\_token = new token(temp\_token, token\_type::NOT);

\_tokens.push\_back(new\_token);

break;

}

default:

{

token\* new\_token = new token(temp\_token, token\_type::KEY\_WORD);

\_tokens.push\_back(new\_token);

break;

}

}

\_state = States::S;

}

else

{

if (symbol == ':')

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 67 |
| Изм. | Лист | № докум | Подпись | Дата |

{

token\* new\_token = new token(temp\_token, token\_type::METKA);

\_tokens.push\_back(new\_token);

}

else

{

token\* new\_token = new token(temp\_token, token\_type::ID);

\_tokens.push\_back(new\_token);

}

\_state = States::S;

}

temp\_token.clear();

if (symbol == '=' || symbol == '!')

{

temp\_token += symbol;

\_state = States::DLM;

}

else

{

if (symbol == ';' || symbol == ',')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::DLM);

\_tokens.push\_back(new\_token);

}

if (symbol == '(')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::LPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == ')')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::RPAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '\*')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::STAR);

\_tokens.push\_back(new\_token);

}

if (symbol == '+')

{

string symbol\_token(1, symbol);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 68 |
| Изм. | Лист | № докум | Подпись | Дата |

token\* new\_token = new token(symbol\_token, token\_type::PLUS);

\_tokens.push\_back(new\_token);

}

if (symbol == '/')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::DIV);

\_tokens.push\_back(new\_token);

}

if (symbol == '-')

{

string symbol\_token(1, symbol);

token\* new\_token = new token(symbol\_token, token\_type::MINUS);

\_tokens.push\_back(new\_token);

}

if (symbol == '\n')

{

number\_string++;

}

\_state = States::S;

}

}

else

{

temp\_token += symbol;

}

break;

}

case States::ER:

{

string message = "ошибка в строке № " + to\_string(number\_string) + " " + temp\_token + '\n';

throw Exception{ message };

//cout << "ошибка в строке № " << number\_string <<" "<< temp\_token << '\n';

temp\_token.clear();

\_state = States::FIN;

break;

}

case States::FIN:

{

cout << "End\n";

flag = false;

break;

}

default:

break;

}

}

}

}

void print()

{

for (const auto& token : \_tokens)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 69 |
| Изм. | Лист | № докум | Подпись | Дата |

{

cout << token->lexeme << " with type: ";

switch ((int)token->type)

{

case 0:

{

cout << "ID\n";

break;

}

case 1:

{

cout << "NUMBER\n";

break;

}

case 2:

{

cout << "KEY\_WORD\n";

break;

}

case 3:

{

cout << "DELLIMETR\n";

break;

}

case 4:

{

cout << "ASSIGN\n";

break;

}

case 5:

{

cout << "COMPARE\n";

break;

}

case 6:

{

cout << "LPAR\n";

break;

}

case 7:

{

cout << "RPAR\n";

break;

}

case 8:

{

cout << "PAR\n";

break;

}

case 9:

{

cout << "MINUS\n";

break;

}

case 10:

{

cout << "PLUS\n";

break;

}

case 11:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 70 |
| Изм. | Лист | № докум | Подпись | Дата |

{

cout << "METKA\n";

break;

}

case 12:

{

cout << "OR\n";

break;

}

case 13:

{

cout << "AND\n";

break;

}

case 14:

{

cout << "NOT\n";

break;

}

case 15:

{

cout << "STAR\n";

break;

}

case 16:

{

cout << "SHIFT\n";

break;

}

case 17:

{

cout << "DIV\n";

break;

}

default:

break;

}

}

}

void next\_token()

{

if (current\_token\_index < \_tokens.size() - 1)

{

current\_token\_index++;

}

}

void prev\_token()

{

if (current\_token\_index > 0)

{

current\_token\_index--;

}

}

string current\_token()

{

return \_tokens[current\_token\_index]->lexeme;

}

token\_type current\_token\_type()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 71 |
| Изм. | Лист | № докум | Подпись | Дата |

{

return \_tokens[current\_token\_index]->type;

}

~lexer()

{

\_tokens.clear();

}

};

# Приложение Б

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 72 |
| Изм. | Лист | № докум | Подпись | Дата |

(Обязательное)

**Грамматика**

<code> := определить <define> <code> | <proc> <code> | e

<define> := <macros> н <preproc> к

<macros>:= <name> | <name> ( <parametr>

<name> := идентификатор

<parametr> := <value> | <value><parametr> | )

<value> := идентификатор | число

<preproc> := -<value> | <value>

<proc>:=<name> н <proc\_code> | <name> (<parametr> н <proc\_code>

<proc\_code> := <operation> к | <operation><proc\_code>

<operation>:=<action> | <metka> | вернуть | переход <name>| конец | если <if\_statment> | иначе <else\_statment>

<action>:= <name> | <name> (<parametr> | <name> () | <name> = <value>| <name>=<name>() | <name>=<name>(<parametr>| <name> <op> <value>

<op>:=И | ИЛИ | НЕ | \* | - | + | / | >> | <<

<if\_statment> := ( <condition> н <if\_code>

<condition> := <first\_cond> <op2><condition> | <first\_cond>

<first\_cond> := <name> <op3><value>

<op2>:= И | ИЛИ

<op3>:= И | ИЛИ | == | !=

<else\_statment> := н <else\_code>

<else\_code> := <operation> к | <operation><else\_code>

<if\_code> := <operation> к | <operation><if\_code>

# Приложение В

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 73 |
| Изм. | Лист | № докум | Подпись | Дата |

(обязательное)

**Листинг кода**

AST.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

enum class node\_type

{

CODE,

PROC,

PROC\_CODE,

IF\_CODE,

COND,

ELSE\_CODE,

//директории

DEFINE,

PREPROC,

MACROS,

PARAMETR,

EXPR,

RET,

RETURN,

//операции

SUB,

ASSGN,

ADD,

MULT,

DIV,

AND,

METKA,

CALL,

OR,

NOT,

SHIFT,

GOTO,

IF,

COMPARE,

ELSE,

//значения

ID,

NUMBER,

};

class node

{

public:

string value;

node\_type type;

node\* operand1;

node\* operand2;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 74 |
| Изм. | Лист | № докум | Подпись | Дата |

node(node\_type type, const string& value = "",

node\* operand1 = nullptr, node\* operand2 = nullptr)

{

this->type = type;

this->value = value;

this->operand1 = operand1;

this->operand2 = operand2;

}

};

class ast

{

public :

node\* tree;

public:

ast()

{

tree = nullptr;

}

void print\_recursive(node\* current\_node, size\_t level)

{

if (current\_node == nullptr) return;

for (int i = 0; i < level; i++)

{

cout << " ";

}

cout << "+-";

switch (current\_node->type)

{

case node\_type::ADD:

{

cout << "ADD";

break;

}

case node\_type::AND:

{

cout << "AND";

break;

}

case node\_type::ASSGN:

{

cout << "ASSGN";

break;

}

case node\_type::CALL :

{

cout << "CALL";

break;

}

case node\_type::CODE:

{

cout << "CODE";

break;

}

case node\_type::COMPARE:

{

cout << "COMPARE";

cout << "(" << current\_node->value << ")";

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 75 |
| Изм. | Лист | № докум | Подпись | Дата |

break;

}

case node\_type::COND:

{

cout << "COND";

break;

}

case node\_type::DEFINE:

{

cout << "DEFINE";

break;

}

case node\_type::DIV:

{

cout << "DIV";

break;

}

case node\_type::ELSE:

{

cout << "ELSE";

break;

}

case node\_type::ELSE\_CODE:

{

cout << "ELSE CODE";

break;

}

case node\_type::EXPR:

{

cout << "EXPR";

break;

}

case node\_type::GOTO:

{

cout << "GOTO";

cout << "(" << current\_node->value << ")";

break;

}

case node\_type::ID:

{

cout << "ID";

cout << "(" << current\_node->value << ")";

break;

}

case node\_type::IF:

{

cout << "IF";

break;

}

case node\_type::IF\_CODE:

{

cout << "IF CODE";

break;

}

case node\_type::MACROS:

{

cout << "MACROS";

break;

}

case node\_type::METKA:

{

cout << "METKA";

cout << "(" << current\_node->value << ")";

break;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 76 |
| Изм. | Лист | № докум | Подпись | Дата |

}

case node\_type::MULT:

{

cout << "MULT";

break;

}

case node\_type::NOT:

{

cout << "NOT";

break;

}

case node\_type::NUMBER:

{

cout << "NUMBER";

cout << "(" << current\_node->value << ")";

break;

}

case node\_type::OR:

{

cout << "OR";

break;

}

case node\_type::PARAMETR:

{

cout << "PARAMETR";

break;

}

case node\_type::PREPROC:

{

cout << "PREPROC";

break;

}

case node\_type::PROC:

{

cout << "PROC";

cout<<"(" << current\_node->value << ")";

break;

}

case node\_type::PROC\_CODE:

{

cout << "PROC\_CODE";

break;

}

case node\_type::RET:

{

cout << "RET";

break;

}

case node\_type::RETURN:

{

cout << "RETURN";

break;

}

case node\_type::SHIFT:

{

cout << "SHIFT";

break;

}

case node\_type::SUB:

{

cout << "SUB";

break;

}

default:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 77 |
| Изм. | Лист | № докум | Подпись | Дата |

break;

}

cout << '\n';

print\_recursive(current\_node->operand1, level + 1);

print\_recursive(current\_node->operand2, level + 1);

}

void print()

{

print\_recursive(tree, 0);

}

};

Parser.h

#pragma once

#include <iostream>

#include <string>

#include "Lexer.h"

#include "AST.h"

#include "Exception.h"

#include "TriadList.h"

using namespace std;

class parser

{

private:

lexer\* \_lex;

ast\* \_ast;

triadList\* \_all\_triads;

int count\_lpar = 0; //кол-во открывающихся скобок

int count\_condition = 0; //кол-во операторов if

string message = "Ошибка в строке № ";

public:

parser(const string& code)

{

\_lex = new lexer(code);

\_ast = new ast();

\_all\_triads = new triadList();

}

private:

node\* code()

{

//директива

if (\_lex->current\_token() == "определить")

{

\_lex->next\_token();

node\* define\_node = define();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

node\* temp\_code\_node = code();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 78 |
| Изм. | Лист | № докум | Подпись | Дата |

node\* new\_node = new node(node\_type::CODE, "", define\_node, temp\_code\_node);

return new\_node;

}

node\* new\_node = new node(node\_type::CODE, "", define\_node);

return new\_node;

}

//процедура

if (\_lex->current\_token\_type() == token\_type::ID)

{

node\* proc\_node = procedure();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

node\* temp\_code\_node = code();

node\* new\_node = new node(node\_type::CODE, "", proc\_node, temp\_code\_node);

return new\_node;

}

node\* new\_node = new node(node\_type::CODE, "", proc\_node);

return new\_node;

return new\_node;

}

}

node\* procedure()

{

string name\_proc = \_lex->current\_token();

if (\_lex->current\_token\_type() != token\_type::ID)

{

message = message + to\_string(\_lex->number\_string)+": "+ "ожидалось имя процедуры";

throw Exception{ message};

}

//параметры

\_lex->next\_token();

if (\_lex->current\_token() == "(")

{

\_lex->next\_token();

node\* parametr\_node = parametr();

\_lex->next\_token();

if (\_lex->current\_token() != "н")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'н'";

throw Exception{ message };

//error("'н' exepted");

}

\_lex->next\_token();

if (\_lex->current\_token() == "к")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось описание функции";

throw Exception{ message };

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 79 |
| Изм. | Лист | № докум | Подпись | Дата |

}

node\* proc\_node = proc\_code();

node\* new\_node = new node(node\_type::PROC, name\_proc, parametr\_node, proc\_node);

return new\_node;

}

//нет параметров

if (\_lex->current\_token() != "н")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'н'";

throw Exception{ message };

//error("'н' exepted");

}

\_lex->next\_token();

if (\_lex->current\_token() == "к")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось описание функции";

throw Exception{ message };

}

node\* proc\_node = proc\_code();

node\* new\_node = new node(node\_type::PROC, name\_proc, proc\_node);

return new\_node;

}

node\* proc\_code()

{

node\* operation\_node = operation();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

node\* temp\_operation\_node = proc\_code();

node\* new\_node = new node(node\_type::PROC\_CODE, "", operation\_node, temp\_operation\_node);

return new\_node;

}

node\* new\_node = new node(node\_type::PROC\_CODE, "", operation\_node);

return new\_node;

}

node\* operation()

{

if (\_lex->current\_token\_type() == token\_type::ID)

{

node\* action\_node = action();

return action\_node;

}

if (\_lex->current\_token\_type() == token\_type::METKA)

{

string metka\_name = \_lex->current\_token();

node\* new\_node = new node(node\_type::METKA, metka\_name);

return new\_node;

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 80 |
| Изм. | Лист | № докум | Подпись | Дата |

if (\_lex->current\_token() == "вернуть")

{

\_lex->next\_token();

node\* value1 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::RETURN, "", value1);

return new\_node;

}

if (\_lex->current\_token() == "переход")

{

//имя перехода

\_lex->next\_token();

string name\_goto = \_lex->current\_token();

node\* new\_node = new node(node\_type::GOTO, name\_goto);

\_lex->next\_token();

return new\_node;

}

if (\_lex->current\_token() == "конец")

{

node\* new\_node = new node(node\_type::RET, "");

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

return new\_node;

}

if (\_lex->current\_token() == "если")

{

\_lex->next\_token();

node\* if\_statment\_node = if\_statment();

count\_condition++;

//cout << "кол-во 'если':" << count\_condition << '\n';

return if\_statment\_node;

}

if (\_lex->current\_token() == "иначе")

{

count\_condition--;

if (count\_condition < 0)

{

message = message + to\_string(\_lex->number\_string) + ": " + "кол-во 'иначе' не совпадет с кол-ом 'если'";

throw Exception{ message };

//error("never count if - else");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 81 |
| Изм. | Лист | № докум | Подпись | Дата |

}

\_lex->next\_token();

node\* else\_statment\_node = else\_statment();

return else\_statment\_node;

}

}

node\* else\_statment()

{

if (\_lex->current\_token() != "н")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'н'";

throw Exception{ message };

//error("'н' exepted");

}

count\_lpar++;

\_lex->next\_token();

node\* else\_code\_node = else\_code();

node\* new\_node = new node(node\_type::ELSE, "", else\_code\_node);

return new\_node;

}node\* else\_code()

{

node\* operation\_node = operation();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

node\* temp\_operation = else\_code();

node\* new\_node = new node(node\_type::ELSE\_CODE, "", operation\_node, temp\_operation);

return new\_node;

}

count\_lpar--;

node\* new\_node = new node(node\_type::ELSE\_CODE, "", operation\_node);

return new\_node;

}

node\* if\_statment()

{

if (\_lex->current\_token() != "(")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось '('";

throw Exception{ message };

//error("'(' exepted");

}

//условие

\_lex->next\_token();

node\* condition\_node = condition();

\_lex->next\_token();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 82 |
| Изм. | Лист | № докум | Подпись | Дата |

if (\_lex->current\_token() != "н")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'н'";

throw Exception{ message };

//error("'н' exepted");

}

count\_lpar++;

\_lex->next\_token();

node\* if\_code\_node = if\_code();

node\* new\_node = new node(node\_type::IF, "", condition\_node, if\_code\_node);

return new\_node;

}

node\* if\_code()

{

node\* operation\_node = operation();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

node\* temp\_operation = if\_code();

node\* new\_node = new node(node\_type::IF\_CODE, "", operation\_node, temp\_operation);

return new\_node;

}

count\_lpar--;

node\* new\_node = new node(node\_type::IF\_CODE, "", operation\_node);

return new\_node;

}

node\* condition()

{

node\* cond1\_node = first\_condit();

node\* new\_node\_cond1 = new node(node\_type::COND, "", cond1\_node);

\_lex->next\_token();

if (\_lex->current\_token() != ")")

{

switch (\_lex->current\_token\_type())

{

case token\_type::AND:

{

\_lex->next\_token();

node\* temp\_node = condition();

node\* new\_node = new node(node\_type::AND, "",new\_node\_cond1, temp\_node);

return new\_node;

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 83 |
| Изм. | Лист | № докум | Подпись | Дата |

case token\_type::OR:

{

\_lex->next\_token();

node\* temp\_node = condition();

node\* new\_node = new node(node\_type::OR, "", new\_node\_cond1, temp\_node);

return new\_node;

}

default:

{

message = message + to\_string(\_lex->number\_string) + ": " + "недопустимая команда";

throw Exception{ message };

//error("undef");

break;

}

}

}

return new\_node\_cond1;

}

node\* first\_condit()

{

node\* value1 = name();

//операция

\_lex->next\_token();

switch (\_lex->current\_token\_type())

{

case token\_type::AND:

{

\_lex->next\_token();

node\* value2 = value();

node\* new\_node = new node(node\_type::AND, "", value1, value2);

return new\_node;

}

case token\_type::COM:

{

string compare = \_lex->current\_token();

\_lex->next\_token();

node\* value2 = value();

node\* new\_node = new node(node\_type::COMPARE, compare, value1, value2);

return new\_node;

}

case token\_type::OR:

{

\_lex->next\_token();

node\* value2 = value();

node\* new\_node = new node(node\_type::OR, "", value1, value2);

return new\_node;

}

default:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 84 |
| Изм. | Лист | № докум | Подпись | Дата |

break;

}

}

node\* action()

{

node\* operand1 = name();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

//операции

if (\_lex->current\_token() == "(")

{

\_lex->next\_token();

//вызов функции без параметров

if (\_lex->current\_token() == ")")

{

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("';' exepted2");

}

node\* new\_node = new node(node\_type::CALL, "", operand1);

return new\_node;

}

//вызов функции с параметрами

else

{

node\* parametr\_node = parametr();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("';' exepted2");

}

node\* new\_node = new node(node\_type::CALL, "", operand1, parametr\_node);

return new\_node;

}

}

switch (\_lex->current\_token\_type())

{

case token\_type::AND:

{

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 85 |
| Изм. | Лист | № докум | Подпись | Дата |

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("';' exepted2");

}

node\* new\_node = new node(node\_type::AND, "", operand1, operand2);

return new\_node;

}

case token\_type::ASSGN:

{

\_lex->next\_token();

node\* operand2 = value();

//параметры

\_lex->next\_token();

if (\_lex->current\_token() == "(")

{

\_lex->next\_token();

if (\_lex->current\_token() == ")")

{

\_lex->next\_token();

//функция без параметров

node\* funct\_node = new node(node\_type::EXPR, "", operand2);

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("';' exepted");

}

node\* new\_node = new node(node\_type::ASSGN, "", operand1, funct\_node);

return new\_node;

}

else

{

node\* parametr\_node = parametr();

node\* function\_node = new node(node\_type::EXPR, "", operand2, parametr\_node);

node\* assgn\_node = new node(node\_type::ASSGN, "", operand1, function\_node);

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("exepted ';'");

}

return assgn\_node;

}

}

if (\_lex->current\_token() != ";")

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 86 |
| Изм. | Лист | № докум | Подпись | Дата |

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::ASSGN, "", operand1, operand2);

return new\_node;

}

case token\_type::MINUS:

{

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::SUB, "", operand1, operand2);

return new\_node;

}

case token\_type::NOT:

{

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::NOT, "", operand1, operand2);

return new\_node;

}

case token\_type::OR:

{

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::OR, "", operand1, operand2);

return new\_node;

}

case token\_type::PLUS:

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 87 |
| Изм. | Лист | № докум | Подпись | Дата |

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

//error("';' exepted3");

}

node\* new\_node = new node(node\_type::ADD, "", operand1, operand2);

return new\_node;

}

case token\_type::SHIFT:

{

\_lex->next\_token();

node\* operand2 = value();

\_lex->next\_token();

if (\_lex->current\_token() != ";")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось ';'";

throw Exception{ message };

}

node\* new\_node = new node(node\_type::SHIFT, "", operand1, operand2);

return new\_node;

}

default:

break;

}

}

//\_lex->prev\_token();

return operand1;

}

node\* name()

{

string value\_expr = \_lex->current\_token();

if (\_lex->current\_token\_type() == token\_type::ID)

{

node\* new\_node = new node(node\_type::ID, value\_expr);

return new\_node;

}

else

{

\_lex->next\_token();

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось имя";

throw Exception{ message };

//error("identification exepted");

}

}

node\* define()

{

node\* macros\_node = macros();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 88 |
| Изм. | Лист | № докум | Подпись | Дата |

\_lex->next\_token();

if (\_lex->current\_token() != "н")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'н'";

throw Exception{ message };

//error("'н' exepted");

}

\_lex->next\_token();

node\* preproc\_node = preproc();

\_lex->next\_token();

if (\_lex->current\_token() != "к")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось 'к'";

throw Exception{ message };

//error("'к' exepted");

}

node\* new\_node = new node(node\_type::DEFINE, "", macros\_node, preproc\_node);

return new\_node;

}

node\* preproc()

{

if (\_lex->current\_token\_type() == token\_type::MINUS)

{

//\_lex->next\_token();

node\* value\_node = value();

node\* new\_node = new node(node\_type::PREPROC, "-", value\_node);

return new\_node;

}

node\* value\_node = value();

node\* new\_node = new node(node\_type::PREPROC, "", value\_node);

return new\_node;

}

node\* macros()

{

node\* value\_node = name();

\_lex->next\_token();

if (\_lex->current\_token() == "(")

{

//параметры

\_lex->next\_token();

if (\_lex->current\_token() == ")")

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидались параметры";

throw Exception{ message };

}

node\* parametr\_node = parametr();

node\* new\_node = new node(node\_type::MACROS, "", value\_node, parametr\_node);

return new\_node;

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 89 |
| Изм. | Лист | № докум | Подпись | Дата |

\_lex->prev\_token();

node\* new\_node = new node(node\_type::MACROS, "", value\_node);

return new\_node;

}

node\* parametr()

{

node\* value\_node = value();

\_lex->next\_token();

if (\_lex->current\_token() != ")")

{

\_lex->next\_token();

node\* temp\_node = parametr();

node\* new\_node = new node(node\_type::PARAMETR, "", value\_node, temp\_node);

return new\_node;

}

node\* new\_node = new node(node\_type::PARAMETR, "", value\_node);

return new\_node;

}

node\* value()

{

string value\_expr = \_lex->current\_token();

if (\_lex->current\_token\_type() == token\_type::ID)

{

node\* new\_node = new node(node\_type::ID, value\_expr);

return new\_node;

}

else if (\_lex->current\_token\_type() == token\_type::NUMBER)

{

node\* new\_node = new node(node\_type::NUMBER, value\_expr);

return new\_node;

}

else if (\_lex->current\_token\_type() == token\_type::MINUS)

{

\_lex->next\_token();

string value = "-"+\_lex->current\_token();

node\* new\_node = new node(node\_type::ID, value);

return new\_node;

}

else

{

message = message + to\_string(\_lex->number\_string) + ": " + "ожидалось какае-то значение";

throw Exception{ message };

}

} };

# Приложение Г

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 90 |
| Изм. | Лист | № докум | Подпись | Дата |

(обязательное)

**Листинг кода**

Define.h

#pragma once

#include <string>

using namespace std;

class def

{

string name;

int number;

public:

def(const string& name, int number)

{

this->name = name;

this->number = number;

}

int get\_number()

{

return number;

}

string get\_name()

{

return name;

}

};

Define\_table.h

#pragma once

#include <string>

#include <iostream>

#include <vector>

#include "Define.h"

using namespace std;

class define\_table

{

vector <def\*> \_defs;

public:

define\_table() = default;

void push\_defs(def\* \_def)

{

\_defs.push\_back(\_def);

}

int get\_number\_by\_name(const string& name)

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 91 |
| Изм. | Лист | № докум | Подпись | Дата |

for (auto& def : \_defs)

{

if (def->get\_name() == name)

{

return def->get\_number();

}

}

return -1;

}

int size()

{

return \_defs.size();

}

void print\_def()

{

for (auto& def : \_defs)

{

cout << def->get\_name() << " n: " << def->get\_number()<<'\n';

}

}

};

Triad.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class triad

{

string operation;

int number\_reg;

int number\_or\_reg;

public:

triad(const string& operation, int numb\_reg, int number\_or\_reg)

{

this->operation = operation;

this->number\_reg = numb\_reg;

this->number\_or\_reg = number\_or\_reg;

}

string get\_operation()

{

return operation;

}

int get\_num\_reg()

{

return number\_reg;

}

int get\_num\_or\_reg()

{

return number\_or\_reg;

}

};

TriadList.h

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 92 |
| Изм. | Лист | № докум | Подпись | Дата |

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include "Triad.h"

#include "AST.h"

#include "Name\_func\_table.h"

#include "Hex\_code\_table.h"

#include "Define\_table.h"

#include "Exception.h"

using namespace std;

class triadList

{

define\_table\* \_defs;

vector <triad\*> \_all\_triads;

vector <triad\*> \_temp\_triads;

vector <name\_func\*> \_names\_func;

hex\_code\_table\* \_hexes;

int number\_reg\_for\_ans = 16;

int number\_command = 0;

bool for\_param = true;

int number\_operand = 0;

void reverse\_triad(vector <triad\*>& \_all\_triads, vector <triad\*>& \_temp\_triads)

{

reverse(begin(\_temp\_triads), end(\_temp\_triads));

\_all\_triads.insert(\_all\_triads.end(), \_temp\_triads.begin(), \_temp\_triads.end());

\_temp\_triads.clear();

}

public:

triadList()

{

\_defs = new define\_table();

\_hexes = new hex\_code\_table();

}

void generate\_triad(node\* current\_node)

{

if (current\_node == nullptr) return;

switch (current\_node->type)

{

case node\_type::DEFINE:

{

node\* macros = current\_node->operand1;

string name\_macros = macros->operand1->value;

int number\_pin=0;

node\* preproc = current\_node->operand2;

if (preproc->operand1->type == node\_type::NUMBER)

{

number\_pin = stoi(preproc->operand1->value, 0);

}

else

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 93 |
| Изм. | Лист | № докум | Подпись | Дата |

{

string op = preproc->operand1->value;

if (op == "А0")

{

number\_pin = 14;

}

if (op == "А1")

{

number\_pin = 15;

}

if (op == "А2")

{

number\_pin = 16;

}

if (op == "А3")

{

number\_pin = 17;

}

}

def\* new\_def = new def(name\_macros, number\_pin);

\_defs->push\_defs(new\_def);

break;

}

case node\_type::EXPR:

{

string name\_call = current\_node->operand1->value;

if (name\_call == "чтЦифр")

{

triad\* new\_triad = new triad("call", 0x13c, 0);

\_temp\_triads.push\_back(new\_triad);

for\_param = true;

}

if (name\_call == "РегистрВх")

{

//number\_reg\_for\_ans++;

triad\* new\_triad = new triad("call", 0x27a, 0);

\_temp\_triads.push\_back(new\_triad);

for\_param = false;

}

break;

}

case node\_type::CALL:

{

reverse\_triad(\_all\_triads, \_temp\_triads);

string name\_call = current\_node->operand1->value;

if (name\_call == "запЦифр")

{

triad\* new\_triad = new triad("call", 0x18e, 0);

\_temp\_triads.push\_back(new\_triad);

for\_param = true;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 94 |
| Изм. | Лист | № докум | Подпись | Дата |

}

if (name\_call == "РегистрВых")

{

node\* param = current\_node->operand2->operand2->operand2->operand2->operand1;

if (param->type == node\_type::NUMBER)

{

triad\* new\_triad = new triad("ldi", 24, stoi(param->value, 0));

\_all\_triads.push\_back(new\_triad);

}

else

{

int number = \_defs->get\_number\_by\_name(param->value);

triad\* new\_triad = new triad("mov", 24, number);

\_all\_triads.push\_back(new\_triad);

}

triad\* new\_triad = new triad("call", 0x24c, 0);

\_all\_triads.push\_back(new\_triad);

for\_param = false;

}

if (name\_call == "пинРежим")

{

triad\* new\_triad = new triad("call", 0x1ea, 0);

\_temp\_triads.push\_back(new\_triad);

for\_param = true;

}

break;

}

case node\_type::PARAMETR:

{

string value\_parametr = current\_node->operand1->value;

if (value\_parametr == "ВЫСОК" || value\_parametr == "ВЫХ")

{

triad\* new\_triad = new triad("ldi", 22, 01);

\_temp\_triads.push\_back(new\_triad);

}

else if (value\_parametr == "НИЗК" || value\_parametr == "ВХ")

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 95 |
| Изм. | Лист | № докум | Подпись | Дата |

{

triad\* new\_triad = new triad("ldi", 22, 00);

\_temp\_triads.push\_back(new\_triad);

}

else

{

if (for\_param)

{

int number = \_defs->get\_number\_by\_name(value\_parametr);

if (number == -1)

{

string message = "неизвестный параметр: " + value\_parametr;

throw Exception{ message };

}

else

{

triad\* new\_traid = new triad("ldi", 24, number);

\_temp\_triads.push\_back(new\_traid);

}

}

}

break;

}

case node\_type::ELSE\_CODE:

{

/\*triad\* new\_traid = new triad("nop", 0, 0);

\_temp\_triads.push\_back(new\_traid);\*/

int goto\_adress = \_temp\_triads.size()+1;

cout << goto\_adress << '\n';

triad\* new\_traid = new triad("breq", goto\_adress, 0);

\_temp\_triads.push\_back(new\_traid);

reverse\_triad(\_all\_triads, \_temp\_triads);

break;

}

case node\_type::COMPARE:

{

reverse\_triad(\_all\_triads, \_temp\_triads);

node\* operand = current\_node;

string op2 = operand->operand2->value;

if (op2 == "ВЫСОК")

{

string name\_compare = operand->operand1->value;

int number\_reg\_compare = \_defs->get\_number\_by\_name(name\_compare);

triad\* new\_traid = new triad("cpi", number\_reg\_compare, 1);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 96 |
| Изм. | Лист | № докум | Подпись | Дата |

\_temp\_triads.push\_back(new\_traid);

}

//запуск счетчика команд до else

break;

}

case node\_type::ASSGN:

{

string name = current\_node->operand1->value;

reverse\_triad(\_all\_triads, \_temp\_triads);

if (current\_node->operand2->type == node\_type::EXPR)

{

triad\* new\_traid = new triad("mov", number\_reg\_for\_ans, 24);

\_temp\_triads.push\_back(new\_traid);

def\* new\_def = new def(name, number\_reg\_for\_ans);

\_defs->push\_defs(new\_def);

number\_reg\_for\_ans++;

}

if (current\_node->operand2->type == node\_type::NUMBER)

{

int number = stoi(current\_node->operand2->value);

def\* new\_def = new def(name, number\_reg\_for\_ans);

\_defs->push\_defs(new\_def);

triad\* new\_traid = new triad("ldi", number\_reg\_for\_ans, number);

\_temp\_triads.push\_back(new\_traid);

}

break;

}

default:

break;

}

generate\_triad(current\_node->operand1);

generate\_triad(current\_node->operand2);

}

void triad\_for\_circle()

{

reverse\_triad(\_all\_triads, \_temp\_triads);

triad\* new\_triad = new triad("sbiw", 28, 00);

\_all\_triads.push\_back(new\_triad);

new\_triad = new triad("brne", 2, 0);

\_all\_triads.push\_back(new\_triad);

new\_triad = new triad("rjmp", -144, 0);

\_all\_triads.push\_back(new\_triad);

new\_triad = new triad("call", 0, 00);

\_all\_triads.push\_back(new\_triad);

new\_triad = new triad("rjmp", -150, 00);

\_all\_triads.push\_back(new\_triad);

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 97 |
| Изм. | Лист | № докум | Подпись | Дата |

void create\_names\_func()

{

name\_func\* new\_name;

new\_name = new name\_func("векторы\_прерыв");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("лин\_конец");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("порт\_вых");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("порт\_вх");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("цифр\_пин\_порт");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("цифр\_пин\_бит\_маска");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("цифр\_пин\_таймер");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("иниц");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("иниц\_данных");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("нет\_прерыв");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("таймер");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("чтЦифр");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("запЦифр");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("пинРежим");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("РегистрВых");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("РегистрВх");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("вектор\_16");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("иниц\_глав");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("код\_основ");

\_names\_func.push\_back(new\_name);

new\_name = new name\_func("выход");

\_names\_func.push\_back(new\_name);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 98 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_name = new name\_func("стоп\_прогр");

\_names\_func.push\_back(new\_name);

}

void create\_obj(node\* current\_node)

{

generate\_triad(current\_node);

reverse\_triad(\_all\_triads, \_temp\_triads);

cout << "def\n";

\_defs->print\_def();

create\_names\_func();

//\_names\_func->print\_name();

\_hexes->generate\_hex\_code(\_names\_func, \_all\_triads, \_defs);

\_hexes->print\_hex\_code();

ofstream outf("code.hex", ios::app);

outf << ":00000001FF" << '\n';

outf.close();

}

void print\_triads()

{

for (auto& triad : \_all\_triads)

{

cout << triad->get\_operation() << ' ';

cout << triad->get\_num\_reg() << ' ';

cout << triad->get\_num\_or\_reg() << '\n';

}

}

};

Name\_func.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class name\_func

{

string name;

public:

name\_func(const string& name)

{

this->name = name;

}

string get\_name()

{

return name;

}

};

Name\_func\_table.h

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 99 |
| Изм. | Лист | № докум | Подпись | Дата |

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include "Name\_func.h"

using namespace std;

class name\_func\_table

{

vector <name\_func\*> \_name\_call;

public:

name\_func\_table() = default;

void push\_name\_table(name\_func\* \_name)

{

\_name\_call.push\_back(\_name);

}

bool find\_name(const string& name)

{

}

void print\_name()

{

for (auto& names : \_name\_call)

{

cout << names->get\_name() << '\n';

}

}

};

Exception.h

#include <iostream>

using namespace std;

class Exception

{

public:

Exception(const string& message)

{

this->message = message;

}

string getMessage()

{

return message;

}

private:

string message;

};

# Приложение Д

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 100 |
| Изм. | Лист | № докум | Подпись | Дата |

(обязательное)

**Определение функций с комментариями**

1. <векторы\_прерыв>

jmp 0xc2 <иниц>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0x3fe <вектор\_16>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

jmp 0xfc <нет\_прерыв>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 101 |
| Изм. | Лист | № докум | Подпись | Дата |

2. <лин\_конец>

0;

0;

а;

б;

в;

3. Названия выходного порта <порт\_вых >

нет\_порта,

нет\_порта,

portb,

portc,

portd

4. Названия входного порта <порт\_вх>

нет\_порта,

нет\_порта,

portb,

portc,

portd,

5. Названия цифровых выходов <цифр\_пин\_порт>

pd;

pd;

pd;

pd;

pd;

pd;

pd;

pd;

pb;

pb;

pb;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 102 |
| Изм. | Лист | № докум | Подпись | Дата |

pb;

pb;

pb;

pc;

pc;

pc;

pc;

pc;

pc;

6. Названия битовой маски портов <цифр\_пин\_маска>

м0;

м1;

м2;

м3;

м4;

м5;

м6;

м7;

м0;

м1;

м2;

м3;

м4;

м5;

м0;

м1;

м2;

м3;

м4;

м5;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 103 |
| Изм. | Лист | № докум | Подпись | Дата |

6. ШИМ сигнал на выходах <цифр\_пин\_таймер>

нет\_таймера;

нет\_таймера;

нет\_таймера;

таймер\_8;

нет\_таймера;

таймер\_2;

таймер\_1;

нет\_таймера;

нет\_таймера;

таймер\_3;

таймер\_4;

таймер\_8;

нет\_таймера;

нет\_таймера;

нет\_таймера;

нет\_таймера;

нет\_таймера;

нет\_таймера;

нет\_таймера;

нет\_таймера;

7. инициализация <иниц>

- Регистр sreg=0

eor r1, r1

out 0x3f, r1

- загрузка адреса (самая последняя запись в SRAM)

ldi r28, 0xFF

ldi r29, 0x08

- присовение адреса регистрам SPH, SPL

out 0x3e, r29

out 0x3d, r28

2. Копирование инициализированных переменных <иниц\_данных>

\* нужно назначить регистрам область памяти

- один регистр в SRAM

ldi r18, 0x01

- один в динамической памяти (там 32 - регистра)

ldi r26, 0x00

- один регистр в SRAM

ldi r27, 0x01

\* 01- адрес SRAM начинается с 1000

00 - 0000 начало памяти

- проверка, что мы загрузка во флеш память

\* Объем динамической памяти 9 байт, там хранятся глобальные переменные

cpi r26, 0x09

- проверить, что загрузка из ОЗУ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 104 |
| Изм. | Лист | № докум | Подпись | Дата |

cpc r27, r18 (тут проверка, что они равны)

- если одно из условий не выполнено, уходим на цикл

brne .-8

- если же все нормально, то вызов функции «главная»

call 0x2ba

- если вообще все пошло не так (или окончание проги), то переход «стоп»

jmp 0x398

4. переадресация прерывания <нет\_прерыв>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 105 |
| Изм. | Лист | № докум | Подпись | Дата |

- переход на 0 вектор прерывания (resert)

jmp 0

5. Настройка таймера прерываний <таймер>

Выбор таймера и тактовой частоты

Если (таймер == TIMER1A)

cpi r24, 0x03

breq .+32

если (таймер!= TIMER1A)

brcc .+10

если (таймер == TIMER1B)

cpi r24, 0x01

breq .+38

если (таймер == TIMER1С)

cpi r24, 0x02

breq .+42

ret

если (таймер ==TIMER2)

cpi r24, 0x07

breq .+42

если (таймер == TIMER0A)

cpi r24, 0x08

breq .+50

если (таймер !=TIMER0B)

cpi r24, 0x04

brne .+44

lds r24, 0x0080 t = TIMER0B;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 106 |
| Изм. | Лист | № докум | Подпись | Дата |

andi r24, 0xDF t & 223

rjmp .+6 переход переадрес

COM1A1:

lds r24, 0x0080

andi r24, 0x7F

переадрес:

sts 0x0080, r24

ret

COM1B1:

in r24, 0x24 ; 36

andi r24, 0x7F ; 127

переадрес2:

out 0x24, r24 ; 36

ret

COM1C1:

in r24, 0x24 ; 36

andi r24, 0xDF ; 223

rjmp .-10

COM21:

lds r24, 0x00B0

andi r24, 0x7F ; 127

переадрес3:

sts 0x00B0, r24

ret

COM0A1:

lds r24, 0x00B0

andi r24, 0xDF ; 223

rjmp .-14

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 107 |
| Изм. | Лист | № докум | Подпись | Дата |

6. <запЦифр>

- функция, проверяющая есть ли на ПИНе возможность ШИМа

таймер = цифрПинТаймер(pin);

ldi r30, 0xAE //загрузка адреса функции digitalPinToTimer

ldi r31, 0x00 //загрузка второй части адреса функции

lpm r18, Z /запись результата

- функция переводит номер ПИНа на плате в номер ПИНа по даташиту

бит = цифрПинМаска(pin);

ldi r30, 0x9A //загрузка адреса функции digitalPinToTimer

ldi r31, 0x00 //загрузка второй части адреса функции

lpm r25, Z /запись результата

- функция переводит номер ПИНа в букву ПИНа по даташиту (PORTB, PORTC, PORTD)

порт = цифрПинПорт(pin);

ldi r30, 0x86 //загрузка адреса функции digitalPinToTimer

ldi r31, 0x00 //загрузка второй части адреса функции

lpm r30, Z /запись результата

- проверка наличия порта (PORTX - такого не существует, то ничего не делаем)

if (порт == нет\_порт) return;

and r30, r30

breq .+50

- если у ПИНа нет ШИМа, то функция не поключается

if (таймер != нет\_таймера) Таймер(таймер);

and r18, r18

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 108 |
| Изм. | Лист | № докум | Подпись | Дата |

breq .+14

- присвоение полученного ранее адреса указателю

\* название порта написано, как PB, а эта функция возвращает PORTB

вых = портВыхРегистр(порт);

ldi r31, 0x00 ; 0

add r30, r30

adc r31, r31

subi r30, 0x8E ; 142

sbci r31, 0xFF ; 255

lpm r26, Z+

lpm r27, Z

- копирование регистра состояния (SREG)

oldSREG = SREG;

in r18, 0x3f

- отключение прерываний с помощью cli()

cli

- подготовка регистра - запись в регистр состояние

ld r30, X

- проверка, чему равно значение

cpse r24, r1

rjmp .+76

- настройка регистра

com r25 //это инверсия

- \*вых &= ~бит;

and r25, r30

- \*вых |= бит;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 109 |
| Изм. | Лист | № докум | Подпись | Дата |

or r25, r30

rjmp .-76

- сохранение результата

st X, r25

- возвращение регистру состояний то, что было

out 0x3f, r18

- конец

ret

7. <micros>

- копия значений регистра флагов (SREG) - запоминаем, были ли включены прерывания

in r19, 0x3f

- отключение прерываний

cli

- загрузка вектора прерывания переполнения таймера

m = timer0\_overflow\_count; //счет переполнения

lds r24, 0x0105 //1байт

lds r25, 0x0106 //2 байт

lds r26, 0x0107//3 байт

lds r27, 0x0108//4 байт - потому что тип long

- запись вектора прерывания (TCNT0 - регистр таймер-счетчика)

t=TCNT0 //считать содержимое счетного регистра

in r18, 0x26

- проверка

если (TIFR0 & \_BV(TOV0) & t < 255)

sbis 0x15, 0 посмотреть 0 бит, там записан флаг переполнения счетчика таймера

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 110 |
| Изм. | Лист | № докум | Подпись | Дата |

rjmp .+10

cpi r18, 0xFF (сравниваем с 255)

breq .+6

- m++ - инкремент по переполнению

adiw r24, 0x01 (+1)

adc r26, r1 (если переполнение, то учитываем это)

adc r27, r1 (если переполнение, то учитываем это)

- восстановить значение регистра состояний

out 0x3f, r19

- возвращение значения m, которое тут же вычислить

return ((m << 8) + t) \* (64 / clockCyclesPerMicrosecond());

mov r27, r26 //пересылка - сдвиг

mov r26, r25

mov r25, r24

eor r24, r24 //очистить r24

movw r22, r24 //объединяем - для того, чтобы выдать, как одно число, а не

movw r24, r26

add r22, r18 //складываем с t

adc r23, r1 //если переполнение, то учитываем

adc r24, r1

adc r25, r1

ldi r20, 0x02 //2

add r22, r22 //сложить

adc r23, r23

adc r24, r24

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 111 |
| Изм. | Лист | № докум | Подпись | Дата |

adc r25, r25

dec r20 //типо делаем умножение на 4

brne .-12 //если не ноль, то снова складываем

- конец

ret

8. <задержка>

push r8

push r9

push r10

push r11

push r12

push r13

push r14

push r15

- вызов функции micros

call 0x170

- запись

старт = micros();

movw r8, r22

movw r10, r24

ldi r24, 0xF4

mov r12, r24

eor r13, r13

inc r13

mov r14, r1

mov r15, r1

- while (ms>0 && (micros() - start) >= 1000)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 112 |
| Изм. | Лист | № докум | Подпись | Дата |

call 0x170 ; 0x170 <micros>

sub r22, r8 // вычитаем старт

sbc r23, r9

sbc r24, r10

sbc r25, r11

cpi r22, 0xE8 ; 232 //-1000

sbci r23, 0x03 ; 3

cpc r24, r1

cpc r25, r1

brcs .-22 //проверяем на не ноль

- ms--

ldi r18, 0x01 ; 1

sub r12, r18

sbc r13, r1

sbc r14, r1

sbc r15, r1

- start+=1000

ldi r24, 0xE8 ; 232

add r8, r24

ldi r24, 0x03 ; 3

adc r9, r24

adc r10, r1

adc r11, r1

- while (ms>0)

cp r12, r1

cpc r13, r1

cpc r14, r1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 113 |
| Изм. | Лист | № докум | Подпись | Дата |

cpc r15, r1

brne .-54

- возвращение из стека

pop r15

pop r14

pop r13

pop r12

pop r11

pop r10

pop r9

pop r8

r et

9. <запЦифр>, <пинРежим>, <РегистрВых>, <РегистрВх>

Имеют такое же содержание, как и функция <чтЦифр>

10. вектор прерывания по переполнению <вектор\_16>

- все бросаем в стек

push r1

push r0

in r0, 0x3f ; 63

push r0

eor r1, r1

push r18

push r19

push r24

push r25

push r26

push r27

- м = timer0\_millis;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 114 |
| Изм. | Лист | № докум | Подпись | Дата |

lds r24, 0x0101 ; 0x800101 <timer0\_millis>

lds r25, 0x0102 ; 0x800102 <timer0\_millis+0x1>

lds r26, 0x0103 ; 0x800103 <timer0\_millis+0x2>

lds r27, 0x0104 ; 0x800104 <timer0\_millis+0x3>

- ф = timer0\_fract;

lds r19, 0x0100 ; 0x800100 <\_edata>

- ф += FRACT\_INC;

ldi r18, 0x03 ; 3

add r18, r19

- если (f >= FRACT\_MAX)

cpi r18, 0x7D

brcc .+86 (тыц)

- м += MILLIS\_INC;

adiw r24, 0x01 ; 1

adc r26, r1

adc r27, r1

- timer0\_fract = ф;

sts 0x0100, r18 ; 0x800100 <\_edata>

-timer0\_millis = м;

sts 0x0101, r24 ; 0x800101 <timer0\_millis>

sts 0x0102, r25 ; 0x800102 <timer0\_millis+0x1>

sts 0x0103, r26 ; 0x800103 <timer0\_millis+0x2>

sts 0x0104, r27 ; 0x800104 <timer0\_millis+0x3>

- timer0\_overflow\_count++;

lds r24, 0x0105 ; 0x800105 <timer0\_overflow\_count>

lds r25, 0x0106 ; 0x800106 <timer0\_overflow\_count+0x1>

lds r26, 0x0107 ; 0x800107 <timer0\_overflow\_count+0x2>

lds r27, 0x0108 ; 0x800108 <timer0\_overflow\_count+0x3>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 115 |
| Изм. | Лист | № докум | Подпись | Дата |

adiw r24, 0x01 ; 1

adc r26, r1

adc r27, r1

sts 0x0105, r24 ; 0x800105 <timer0\_overflow\_count>

sts 0x0106, r25 ; 0x800106 <timer0\_overflow\_count+0x1>

sts 0x0107, r26 ; 0x800107 <timer0\_overflow\_count+0x2>

sts 0x0108, r27 ; 0x800108 <timer0\_overflow\_count+0x3>

- возвращаем значения из стека

pop r27

pop r26

pop r25

pop r24

pop r19

pop r18

pop r0

out 0x3f, r0 ; 63

pop r0

pop r1

reti

- (тыц) ф -= FRACT\_MAX;

ldi r18, 0x86 ; 134

add r18, r19

- м += 1;

adiw r24, 0x02 ; 2

adc r26, r1

adc r27, r1

rjmp .-92

11. <иниц\_главн>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 116 |
| Изм. | Лист | № докум | Подпись | Дата |

- sei

sei

- подрубаем вектора прерывания по таймеру для разных архитектур

sbi(TCCR0A, WGM01);

in r24, 0x24 ; 36

ori r24, 0x02 ; 2

out 0x24, r24 ; 36

sbi(TCCR0A, WGM00);

in r24, 0x24 ; 36

ori r24, 0x01 ; 1

out 0x24, r24 ; 36

sbi(TCCR0B, CS01);

in r24, 0x25 ; 37

ori r24, 0x02 ; 2

out 0x25, r24 ; 37

sbi(TCCR0B, CS00);

in r24, 0x25 ; 37

ori r24, 0x01 ; 1

out 0x25, r24 ; 37

sbi(TIMSK0, TOIE0);

lds r24, 0x006E

ori r24, 0x01 ; 1

sts 0x006E, r24

TCCR1B = 0;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 117 |
| Изм. | Лист | № докум | Подпись | Дата |

sts 0x0081, r1

sbi(TCCR1B, CS11);

lds r24, 0x0081

ori r24, 0x02

sts 0x0081, r24

sbi(TCCR1B, CS10);

lds r24, 0x0081

ori r24, 0x01

sts 0x0081, r24

sbi(TCCR1A, WGM10);

lds r24, 0x0080

ori r24, 0x01

sts 0x0080, r24

sbi(TCCR2B, CS22);

lds r24, 0x00B1

ori r24, 0x04

sts 0x00B1, r24

sbi(TCCR2A, WGM20);

lds r24, 0x00B0

ori r24, 0x01 ; 1

sts 0x00B0, r24

sbi(ADCSRA, ADPS2);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 118 |
| Изм. | Лист | № докум | Подпись | Дата |

lds r24, 0x007A

ori r24, 0x04

sts 0x007A, r24

sbi(ADCSRA, ADPS1);

lds r24, 0x007A

ori r24, 0x02

sts 0x007A, r24

sbi(ADCSRA, ADPS0);

lds r24, 0x007A

ori r24, 0x01 ; 1

sts 0x007A, r24

sbi(ADCSRA, ADEN);

lds r24, 0x007A

ori r24, 0x80 ; 128

sts 0x007A, r24

UCSR0B = 0;

sts 0x00C1, r1

12. <выход>

Cli

13. <стоп\_прогр>

rjmp .-2

# Приложение Е

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 119 |
| Изм. | Лист | № докум | Подпись | Дата |

(Обязательное)

**Листинг кода**

Hex\_code.h

#pragma once

#include <iostream>

class hex\_code

{

string hc1;

string hc2;

string hc3;

string hc4;

public:

hex\_code(const string& hc1, const string& hc2, const string& hc3 = "", const string& hc4 = "")

{

this->hc1 = hc1;

this->hc2 = hc2;

this->hc3 = hc3;

this->hc4 = hc4;

}

string get\_hc1()

{

return hc1;

}

string get\_hc2()

{

return hc2;

}

string get\_hc3()

{

return hc3;

}

string get\_hc4()

{

return hc4;

}

};

Hex\_code\_table.h

#pragma once

#include <iostream>

#include <vector>

#include <fstream>

#include <format>

#include "Hex\_code.h"

#include "Name\_func\_table.h"

#include "Define\_table.h"

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 120 |
| Изм. | Лист | № докум | Подпись | Дата |

using namespace std;

class hex\_code\_table

{

vector <hex\_code\*> \_codes;

char DigitToHex(int N)

{

switch (N)

{

case 0: return '0'; break;

case 1: return '1'; break;

case 2: return '2'; break;

case 3: return '3'; break;

case 4: return '4'; break;

case 5: return '5'; break;

case 6: return '6'; break;

case 7: return '7'; break;

case 8: return '8'; break;

case 9: return '9'; break;

case 10: return 'A'; break;

case 11: return 'B'; break;

case 12: return 'C'; break;

case 13: return 'D'; break;

case 14: return 'E'; break;

case 15: return 'F'; break;

default:

return NULL;

break;

}

}

string to\_hex\_2(int number)

{

string res = "";

int x;

while (number > 0)

{

x = number % 16;

number = number / 16;

if (x < 16) res = DigitToHex(x) + res;

}

return res;

}

string to\_hex(int number)

{

string res = "";

int x;

while (number > 0)

{

x = number % 16;

number = number / 16;

if (x < 16) res = DigitToHex(x) + res;

}

int count\_number = res.length();

//cout << count\_number<<'\n';

switch (count\_number)

{

case 0:

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 121 |
| Изм. | Лист | № докум | Подпись | Дата |

return "0000";

break;

}

case 1:

{

return "000" + res;

break;

}

case 2:

{

return "00" + res;

break;

}

case 3:

{

return "0" + res;

break;

}

default:

break;

}

}

public:

hex\_code\_table()

{

}

void generate\_hex\_code(vector <name\_func\*>& names, vector <triad\*>& \_all\_triads, define\_table\* \_defs)

{

hex\_code\* new\_hex;

for (auto& name : names)

{

if (name->get\_name() == "векторы\_прерыв")

{

new\_hex = new hex\_code("0c", "94", "61", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 122 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "5d", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "73", "00");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "лин\_конец")

{

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("24", "00", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 123 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("27", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2a", "00", "", "");

\_codes.push\_back(new\_hex);

/\*new\_hex = new hex\_code("04", "07", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);\*/

/\*new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);\*/

}

if (name->get\_name() == "порт\_вых")

{

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("25", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("28", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2b", "00", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "порт\_вх")

{

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("23", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("26", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("29", "00", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "цифр\_пин\_порт")

{

new\_hex = new hex\_code("04", "04", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 124 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("02", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("02", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("02", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("03", "03", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("03", "03", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("03", "03", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "цифр\_пин\_бит\_маска")

{

new\_hex = new hex\_code("01", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "20", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("40", "80", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "20", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "20", "", "");

\_codes.push\_back(new\_hex);

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 125 |
| Изм. | Лист | № докум | Подпись | Дата |

if (name->get\_name() == "цифр\_пин\_таймер")

{

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "02", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "03", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("04", "07", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "иниц")

{

new\_hex = new hex\_code("11", "24", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("1f", "be", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "ef", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d8", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("de", "bf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cd", "bf", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "иниц\_данных")

{

new\_hex = new hex\_code("21", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "e0", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 126 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("1d", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a9", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b2", "07", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e1", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "a7", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "94", "63", "02");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "нет\_прерыв")

{

new\_hex = new hex\_code("0c", "94", "00", "00");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "таймер")

{

new\_hex = new hex\_code("83", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("28", "f4", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("99", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("82", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a9", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("87", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a9", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "30", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 127 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("c9", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "f4", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "80", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "7d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("03", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "80", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "77", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "80", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "77", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "bd", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "7d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("fb", "cf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "b0", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "77", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "b0", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "b0", "00");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 128 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "7d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "cf", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "чтЦифр")

{

new\_hex = new hex\_code("cf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("28", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("30", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e2", "55", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e6", "56", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ea", "57", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cc", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "11", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 129 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "75", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ee", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "1f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e4", "58", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a5", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ed", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("09", "f4", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("fa", "cf", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 130 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "запЦифр")

{

new\_hex = new hex\_code("1f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("28", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("30", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e2", "55", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e6", "56", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f9", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ea", "57", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cc", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a9", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("16", "2f", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 131 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "11", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "75", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ee", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "1f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ee", "58", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a5", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "b7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f8", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("11", "11", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d0", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("de", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("dc", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "bf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "91", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 132 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("1f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("de", "2b", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f8", "cf", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "пинРежим")

{

new\_hex = new hex\_code("cf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("fc", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e6", "56", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("24", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8a", "57", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("fc", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d1", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("99", "1f", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 133 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("fc", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e8", "59", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a5", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("fc", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ee", "58", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "4f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c5", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d4", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("61", "11", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "b7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f8", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8c", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e2", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e0", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8e", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8c", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("28", "81", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e2", "23", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a8", "83", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 134 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "bf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "b7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f8", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e2", "2b", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ec", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "bf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f6", "cf", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "РегистрВых")

{

new\_hex = new hex\_code("cf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d8", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c8", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("6d", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("61", "70", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8d", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d6", "95", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 135 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("61", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8b", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("60", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8b", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c1", "50", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "РегистрВх")

{

new\_hex = new hex\_code("1f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("61", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("86", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 136 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("85", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "9e", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0c", "2e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0a", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ea", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("18", "2b", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("60", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("86", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("21", "96", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c8", "30", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d1", "05", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("59", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("1f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "микросекунды")

{

new\_hex = new hex\_code("3f", "b7", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 137 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f8", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("05", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "91", "06", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("A0", "91", "07", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b0", "91", "08", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("26", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a8", "9b", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("05", "c0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2f", "3f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("19", "f0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "96", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("3f", "bf", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ba", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a9", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("98", "2f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "27", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("bc", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cd", "01", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 138 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("62", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("71", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("91", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("42", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("66", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("77", "1f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "1f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("99", "1f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("4a", "95", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d1", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "задержка")

{

new\_hex = new hex\_code("8f", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("af", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("bf", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ef", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "db", "00");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 139 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("4b", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("5c", "01", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "ee", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c8", "2e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("83", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d8", "2e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e1", "2c", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f1", "2c", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0e", "94", "db", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("68", "19", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("79", "09", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8a", "09", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9b", "09", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("68", "3e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("73", "40", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "05", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("91", "05", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a8", "f3", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("21", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c2", "1a", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 140 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("d1", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e1", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f1", "08", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "ee", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("88", "0e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("83", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("98", "1e", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "1c", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "1c", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("c1", "14", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d1", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("e1", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("f1", "04", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("29", "f7", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ff", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("ef", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("df", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("cf", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("bf", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("af", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "90", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 141 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("08", "95", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "вектор\_16")

{

new\_hex = new hex\_code("1f", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "b6", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "92", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("11", "24", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("3f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("af", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("bf", "93", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "01", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "91", "02", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a0", "91", "03", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b0", "91", "04", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("30", "91", "00", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "96", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "1d", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 142 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("23", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("23", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2d", "37", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("26", "e8", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("23", "0f", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("02", "96", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("20", "93", "00", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "01", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "93", "02", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a0", "93", "03", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b0", "93", "04", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "05", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "91", "06", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a0", "91", "07", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b0", "91", "08", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("01", "96", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b1", "1d", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "05", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("90", "93", "06", "01");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 143 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("a0", "93", "07", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("b0", "93", "08", "01");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("00", "00", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("bf", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("af", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("9f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("8f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("3f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("2f", "91", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "be", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("0f", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("1f", "90", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("18", "95", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "иниц\_глав")

{

new\_hex = new hex\_code("78", "94", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("82", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "bd", "", "");

\_codes.push\_back(new\_hex);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 144 |
| Изм. | Лист | № докум | Подпись | Дата |

new\_hex = new hex\_code("84", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "bd", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("85", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("82", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("85", "bd", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("85", "b5", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("85", "bd", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "6e", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "6e", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "92", "81", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "81", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("82", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "81", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "81", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "81", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "80", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "80", "00");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 145 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "b1", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "b1", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "b0", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "b0", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("84", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("82", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("81", "60", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "91", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "68", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("80", "93", "7a", "00");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("10", "92", "c1", "00");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "код\_основ")

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 146 |
| Изм. | Лист | № докум | Подпись | Дата |

int count\_def = \_defs->size();

for (auto& triad : \_all\_triads)

{

string operation=triad->get\_operation();

if (operation == "ldi")

{

if (count\_def == 0)

{

new\_hex = new hex\_code("c0", "e0", "", "");

\_codes.push\_back(new\_hex);

new\_hex = new hex\_code("d0", "e0", "", "");

\_codes.push\_back(new\_hex);

}

int operand1 = triad->get\_num\_reg();

int command1 = operand1 - 16;

string command = to\_string(command1);

int operand2 = triad->get\_num\_or\_reg();

if (operand2 < 10)

{

command += to\_string(operand2);

}

else

{

switch (operand2)

{

case 10:

{

command += "a";

break;

}

case 11:

{

command += "b";

break;

}

case 12:

{

command += "c";

break;

}

case 13:

{

command += "d";

break;

}

case 14:

{

command += "e";

break;

}

case 15:

{

command += "f";

break;

}

default:

{

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 147 |
| Изм. | Лист | № докум | Подпись | Дата |

throw Exception{ "слишком большое число" };

break;

}

}

}

cout << command << ' ' << "0e\n";

new\_hex = new hex\_code(command, "e0");

\_codes.push\_back(new\_hex);

}

if (operation == "call")

{

int adress\_call = triad->get\_num\_reg();

if (adress\_call == 398) //digitalwrite

{

cout << "0e 94 c7 00\n";

new\_hex = new hex\_code("0e", "94", "c7", "00");

\_codes.push\_back(new\_hex);

}

if (adress\_call == 490) //pinMode

{

count\_def--;

new\_hex = new hex\_code("0e", "94", "f5", "00");

\_codes.push\_back(new\_hex);

cout << "0e 94 f5 00\n";

}

if (adress\_call == 316) //digitalRead

{

new\_hex = new hex\_code("0e", "94", "9e", "00");

\_codes.push\_back(new\_hex);

}

if (adress\_call == 588) //shiftOut

{

new\_hex = new hex\_code("0e", "94", "26", "01");

\_codes.push\_back(new\_hex);

cout << "0e 94 26 01\n";

}

if (adress\_call == 634) //shiftIn

{

new\_hex = new hex\_code("0e", "94", "3d", "01");

\_codes.push\_back(new\_hex);

}

}

if (operation == "nop")

{

new\_hex = new hex\_code("00", "00", "", "");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 148 |
| Изм. | Лист | № докум | Подпись | Дата |

\_codes.push\_back(new\_hex);

}

if (operation == "cpi")

{

cout << "cpi: ";

int operand1 = triad->get\_num\_reg();

int command1 = operand1 - 16;

string command = to\_string(command1);

int operand2 = triad->get\_num\_or\_reg();

int command2 = operand2;

command += to\_string(command2);

//cout << command << " 30\n";

new\_hex = new hex\_code(command, "30");

\_codes.push\_back(new\_hex);

}

if (operation == "mov")

{

cout << "mov: ";

int operand1 = triad->get\_num\_reg();

int command1 = operand1 - 16;

string command = to\_string(command1);

int operand2 = triad->get\_num\_or\_reg();

int command2 = operand2 - 16;

command += to\_string(command2);

cout << command << " 2f\n";

new\_hex = new hex\_code(command, "2f");

\_codes.push\_back(new\_hex);

}

if (operation == "breq")

{

int operand = triad->get\_num\_reg();

if (operand == 4)

{

new\_hex = new hex\_code("11", "f0", "", "");

\_codes.push\_back(new\_hex);

}

}

}

new\_hex = new hex\_code("b5", "cf", "", "");

\_codes.push\_back(new\_hex);

}

if (name->get\_name() == "выход")

{

new\_hex = new hex\_code("f8", "94", "", "");

\_codes.push\_back(new\_hex);

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 149 |
| Изм. | Лист | № докум | Подпись | Дата |

if (name->get\_name() == "стоп\_прогр")

{

new\_hex = new hex\_code("ff", "cf", "", "");

\_codes.push\_back(new\_hex);

}

}

}

void print\_hex\_code()

{

ofstream outf ("code.hex");

string result;

int number = 0;

int sum = 0;

int shift = 0;

int shift1;

int shift2;

for (auto& hexs : \_codes)

{

result = result + hexs->get\_hc1();

sum = sum + stoi(hexs->get\_hc1(), 0, 16);

number += 1;

if (number >= 16)

{

ofstream shift\_temp("shift.txt");

shift\_temp << to\_hex(shift) << '\n';

shift\_temp.close();

if (shift >= 256)

{

shift1 = shift / 256;

shift2 = shift - 256;

}

else

{

shift1 = 0;

shift2 = shift;

}

int check\_sum = ~(sum + 16 + shift1 + shift2) + 1;

ofstream temp("kontr\_sum.txt");

temp << hex << check\_sum << '\n';

temp.close();

string k\_sum;

string hex\_str;

ifstream inf("kontr\_sum.txt");

if (inf.is\_open())

{

while (getline(inf, k\_sum))

{

//cout << to\_hex(shift) << '\n';

hex\_str = ":10" + to\_hex(shift) + "00" + result + k\_sum[k\_sum.length() - 2] + k\_sum[k\_sum.length() - 1] + '\n';

}

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 150 |
| Изм. | Лист | № докум | Подпись | Дата |

inf.close();

outf << hex\_str;

result = "";

shift += 16;

number = 0;

sum = 0;

}

result = result + hexs->get\_hc2();

sum = sum + stoi(hexs->get\_hc2(), 0, 16);

number += 1;

if (number >= 16)

{

ofstream shift\_temp("shift.txt");

shift\_temp << to\_hex(shift) << '\n';

shift\_temp.close();

if (shift >= 256)

{

shift1 = shift / 256;

shift2 = shift - 256;

}

else

{

shift1 = 0;

shift2 = shift;

}

int check\_sum = ~(sum + 16 + shift1 + shift2) + 1;

ofstream temp("kontr\_sum.txt");

temp << hex << check\_sum << '\n';

temp.close();

string k\_sum;

string hex\_str;

ifstream inf("kontr\_sum.txt");

if (inf.is\_open())

{

while (getline(inf, k\_sum))

{

//cout << to\_hex(shift) << '\n';

hex\_str = ":10" + to\_hex(shift) + "00" + result + k\_sum[k\_sum.length() - 2] + k\_sum[k\_sum.length() - 1] + '\n';

}

}

inf.close();

outf << hex\_str;

result = "";

shift += 16;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 151 |
| Изм. | Лист | № докум | Подпись | Дата |

number = 0;

sum = 0;

}

if (hexs->get\_hc3() != "")

{

result = result + hexs->get\_hc3();

sum = sum + stoi(hexs->get\_hc3(), 0, 16);

result = result + hexs->get\_hc4();

sum = sum + stoi(hexs->get\_hc4(), 0, 16);

number += 2;

}

if (number >= 16)

{

ofstream shift\_temp("shift.txt");

shift\_temp << to\_hex(shift) << '\n';

shift\_temp.close();

if (shift >= 256)

{

shift1 = shift / 256;

shift2 = shift - 256;

}

else

{

shift1 = 0;

shift2 = shift;

}

int check\_sum = ~(sum + 16 + shift1 + shift2) + 1;

ofstream temp("kontr\_sum.txt");

temp << hex << check\_sum << '\n';

temp.close();

string k\_sum;

string hex\_str;

ifstream inf("kontr\_sum.txt");

if (inf.is\_open())

{

while (getline(inf, k\_sum))

{

//cout << to\_hex(shift) << '\n';

hex\_str = ":10" + to\_hex(shift) + "00" + result + k\_sum[k\_sum.length() - 2] + k\_sum[k\_sum.length() - 1] + '\n';

}

}

inf.close();

outf << hex\_str;

result = "";

shift += 16;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 152 |
| Изм. | Лист | № докум | Подпись | Дата |

number = 0;

sum = 0;

}

}

if (shift >= 256)

{

shift1 = shift / 256;

shift2 = shift - 256;

}

else

{

shift1 = 0;

shift2 = shift;

}

int check\_sum = ~(sum + number + shift1+shift2) + 1;

ofstream temp("kontr\_sum.txt");

temp << hex << check\_sum << '\n';

temp.close();

string k\_sum;

string hex\_str;

ifstream inf("kontr\_sum.txt");

if (inf.is\_open())

{

while (getline(inf, k\_sum))

{

//cout << number << '\n';

hex\_str =":0" +to\_hex\_2(number) + to\_hex(shift) + "00" + result + k\_sum[k\_sum.length() - 2] + k\_sum[k\_sum.length() - 1] + '\n';

}

}

inf.close();

outf << hex\_str;

outf.close();

}

};

# Приложение Ж

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 153 |
| Изм. | Лист | № докум | Подпись | Дата |

**(обязательное)**

**Руководство пользователя**

1. Введение

1.1. Область применения

Требования настоящего документа применяются при:

- опытной эксплуатации;

- приемочных испытаниях;

- промышленной эксплуатации.

1.2. Краткое описание возможностей

Программный продукт предназначен для создания тестов проверки блоков релейной логики.

При выполнении написания теста проверки используются инструменты пользователя, которые представляют собой следующие возможности:

- написание программы;

- получение информации о результатах выполнения компиляции;

- запись программы в микроконтроллер;

2. Подготовка к работе

Для подготовки к работе пользователю необходимо:

- включить ПК;

- подключить устройство проверки к ПК через USB-порт;

- найти exe-файл с компилятором;

В случае, если необходимо разработать программу, то пользователю дополнительно к предыдущим пунктам необходимо открыть любой текстовый редактор с возможностью сохранения и открытия файла.

3. Описание функций

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 154 |
| Изм. | Лист | № докум | Подпись | Дата |

3.1. Написание программы

Структура программы выглядит следующим образом:

определить <имя\_переменной> н 8 к

главная

н

<код>

к

Для удобства дальнейшей разработки в самом начале программы необходимо указать переменную и номер порта, с которым она в последующем будет связана. Для этого описывается директива, состоящая из кодового слова «определить», названия переменной и номера порта, записанного внутри скобок(н к). Дополнительно стоит отметить, что в программе возможно определение нескольких директив.

Далее идет функция «главная», внутри которой пользователь может написать тест проверки, используя синтаксис разработанного языка и переменные, которые были определены выше.

Синтаксис языка подразумевает использование следующих функций:

- пинРежим(<имя\_переменной>, <режим>);

- чтЦифр(<имя\_переменной>);

- запЦифр(<имя\_переменной>, <сигнал>);

- РегистрВых(<имя1>, <имя2>, <направление>, <данные>);

- РегистрВх(<имя1>, <имя2>,<направление>).

В самом начале написания теста проверки необходимо определить режим для каждого из используемого порта. Это необходимо для того, чтобы остальные функции корректно работали. Чтобы выполнить определения режима работы порта необходима написать функцию пинРежим, которой в качестве параметров необходимо указать имя переменной для соответствующего порта и режим – ВХ или ВЫХ.

Функция чтЦирф выполняет чтение сигнала с порта, номер которого указан в качестве параметра. Так как функция получает какое-то значение, то для его дальнейшего использования необходимо присвоить полученное значение какой-нибудь переменной. Об операции присваивания будет рассказано дальше.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 155 |
| Изм. | Лист | № докум | Подпись | Дата |

Функция запЦифр выполняет подачу сигнала на указанный номер порта. В качестве сигнала можно подать только ВЫСОК или НИЗК, 1 или 0 соответственно.

Функция РегитсрВых выводит байт информации на порт вход/выхода последовательно (побитно).

<имя1>: номер порта вход/выхода, на который выводятся биты (int)

<имя2>: номер порта по которому производится синхронизация (int)

<направл>: используемая последовательность ЛЕВ — слева или  ПРАВ  — справа. Рекомендуемо, использовать ЛЕВ.

<данные>: значение (байт) для вывода

Функция РегитсрВх вводит байт информации на порт вход/выхода последовательно (побитно).

<имя1>: номер порта вход/выхода, на который выводятся биты (int)

<имя2>: номер порта по которому производится синхронизация (int)

<направл>: используемая последовательность ЛЕВ — слева или  ПРАВ  — справа. Рекомендуемо, использовать ЛЕВ.

Кроме того, синтаксис языка подразумевает выполнение следующих операций: присваивания, арифметических, логических, сдвиги, сравнения. Для написания какой-либо операции необходимо указать имя переменной, в которую запишется результат, символ операции и вторую переменную. Причем второй переменной может выступать не только переменная, но и число, а также функция, которая возвращает значение.

Например, операция присваивания переменной «в» значения функции чтЦифр будет иметь следующий вид:

в = чтЦифр(пинДиод);

Стоит отметить, что все операции обязательно заканчиваются символом «;». Это необходимо для определения конца операции.

Помимо операций, возможно использование конструкций для проверки условий. Выглядят они так:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 156 |
| Изм. | Лист | № докум | Подпись | Дата |

если (<условие>)

н <код> к

иначе

н <код> к

В скобках указывается условие, которое необходимо проверить. Записывается оно так же, как и операция, только в качестве символа операции могут использовать только операторы логики или сравнения.

После условия располагается код, который будет выполняться в случае выполнения условия.

Код, который выполняется в случае невыполнения условия, помешается внутри блока «иначе» и отделяется скобками (н к).

Возможна ситуация, когда нет необходимости в блоке «иначе». В это случае данный блок не пишется и тогда конструкция примет следующий вид:

если (<условие>)

н <код> к

В случае возникновения ошибок при неправильном написании программы будет выдано сообщение об ошибке, с указанием места, где она, предположительно допущена.

3.2. Запись программы в микроконтроллер

Для осуществления записи в микроконтроллер необходимо сохранить файл программы, если он был создан. Файл с написанной программой необходимо перенести в папку с компилятором (exe – файл), и запустить прошивку, запустив exe.

Во время выполнения прошивки микроконтроллера могут возникнуть следующие ошибки:

- вывод сообщения об ошибке программы

Необходимо обратиться к пункту 3.1 Руководства пользователя

- вывод сообщения об отсутствии подключения устройства проверки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ТПЖА.09.03.01.487 ПЗ | Лист |
|  |  |  |  |  | 157 |
| Изм. | Лист | № докум | Подпись | Дата |

Необходимо проверить правильность подключения устройства через USB-порт к ПК. В случае, если устройство подключено правильно, но продолжает появляться ошибка – попробуйте заменить провод, плату.

- не работает устройство проверки

Необходимо убедиться, что подключение происходит именно к тому порту, с которым взаимодействует программа. Если это не помогает, то необходимо убедиться в правильности подключения элементов и наличия питания на каждом из них. Помимо этого, стоит убедиться в работоспособности элемента. Для этого стоит попробовать заменить элемент на другой такой же элемент. Если все сделано правильно, но устройство все еще не работает или работает неправильно, ищите ошибку в логике разработанной программы.