

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Карякин Андрей Виссарионович
Должность: Руководитель НТИ НИЯУ МИФИ
Дата подписания: 16.01.2025 10:27:12
Уникальный программный ключ:
2e905c9a64921ebc9b6e02a1d35ea14517838874

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Новоуральский технологический институт—
филиал федерального государственного автономного образовательного учреждения высшего образования
«Национальный исследовательский ядерный университет «МИФИ»
(НТИ НИЯУ МИФИ)

Колледж НТИ

**Цикловая методическая комиссия общетехнических дисциплин
энергетики и электроники**

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ
ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКОЙ
(ЛАБОРАТОРНОЙ) РАБОТЫ УЧЕБНОЙ
ДИСЦИПЛИНЫ
ОП.07 МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ**

для студентов колледжа НТИ НИЯУ МИФИ,
обучающихся по программе среднего профессионального образования

специальность 11.02.16

«Монтаж, техническое обслуживание и ремонт электронных приборов
и устройств»

очная форма обучения

на базе основного общего образования

квалификация

специалист по электронным приборам и устройствам

Новоуральск 2021

СОДЕРЖАНИЕ

Введение	4
1. Практическая работа 1.	7
2. Лабораторная работа 1.	13
3. Лабораторная работа 2.	16
4. Лабораторная работа 3.....	23
5. Лабораторная работа 4.....	31
6. Лабораторная работа 5.....	38
7. Лабораторная работа 6.....	53
8. Лабораторная работа 7.....	64
9. Лабораторная работа 8.	75
10. Лабораторная работа 9.	81
11.Лабораторная работа 10.	87
12. Лабораторная работа 11.	92

Введение

Лабораторные и практические занятия по учебной дисциплине ОП.07 «Микропроцессорные системы» составляют важную часть теоретической и профессиональной практической подготовки и направлены на подтверждение теоретических положений и формирование практических умений и практического опыта. Обучающимися осваиваются следующие умения

- производить выбор элементной базы для проектирования цифровых схем;
- производить синтез и анализ цифровых схем;
- проводить исследование типовых схем цифровой электроники;
- выполнять упрощение логических схем
- использовать универсальные базисы для построения схем на логических элементах
- выполнять сравнительную характеристику ЦИМС различных типов логик, пользуясь справочной литературой
- выполнять построение счетчиков с произвольным основанием
- использовать комбинационные устройства для реализации логических функции

Лабораторные и практические занятия относятся к основным видам учебных занятий.

Выполнение студентами лабораторных и практических работ направлено:

- на обобщение, систематизацию, углубление, закрепление полученных теоретических знаний по конкретным темам дисциплин;
- формирование умений применять полученные знания на практике;
- реализацию единства интеллектуальной и практической деятельности;
- развитие интеллектуальных умений (аналитических, проектировочных, конструкторских и др.) у будущих специалистов;
- выработку при решении поставленных задач таких профессионально значимых качеств, как самостоятельность, ответственность, точность, творческая инициатива.

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений.

Ведущей дидактической целью практических занятий является формирование практических умений – профессиональных или учебных, необходимых в последующей учебной деятельности.

Содержанием лабораторных работ по дисциплине наблюдение развития явлений, и процессов. В ходе выполнения заданий у студентов формируются практические умения и навыки обращения с приборами, установками, лабораторным оборудованием, аппаратурой, которые могут составлять часть профессиональной практической подготовки, а также исследовательские

умения (наблюдать, сравнивать, анализировать, устанавливать зависимости, делать выводы и обобщения, самостоятельно вести исследование, оформлять результаты).

Содержанием практических занятий по дисциплине являются решение разного задач, в том числе профессиональных выполнение вычислений, расчетов.

Содержание практических, лабораторных занятий охватывают весь круг профессиональных умений, на подготовку к которым ориентирована данная дисциплина, которые в дальнейшем закрепляются и совершенствуются в процессе курсового проектирования, практикой по профилю специальности и преддипломной практикой.

Лабораторные занятия проводятся в специально оборудованных учебных лабораториях. Практическое занятие должно проводиться в учебных кабинетах или специально оборудованных помещениях (площадках). Продолжительность занятия – не менее 2-х академических часов. Необходимыми структурными элементами занятия, помимо самостоятельной деятельности студентов, являются инструктаж, проводимый преподавателем, а также организация обсуждения итогов выполнения работы.

Все студенты, связанные с работой в лаборатории, обязаны пройти инструктаж по безопасному выполнению работ, о чем расписываются в журнале инструктажа по технике безопасности.

Выполнению лабораторных и практических работ предшествует проверка знаний студентов, их теоретической готовности к выполнению задания.

Лабораторные и практические работы студенты выполняют под руководством преподавателя. При проведении лабораторных и практических занятий учебная группа может делиться на подгруппы численностью не менее 8 человек. Объем заданий для лабораторных и практических занятий спланирован с расчетом, чтобы за отведенное время они могли быть выполнены качественно большинством студентов.

Формы организации работы обучающихся на лабораторных работах и практических занятиях: фронтальная, групповая и индивидуальная.

При фронтальной форме организации занятий все студенты выполняют одновременно одну и ту же работу. При групповой форме организации занятий одна и та же работа выполняется бригадами по 2 - 5 человек. При индивидуальной форме организации занятий каждый студент выполняет индивидуальное задание.

Отчет по практической и лабораторной работе представляется в печатном виде в формате, предусмотренном шаблоном отчета по практической, лабораторной работе. Защита отчета проходит в форме доклада обучающегося по выполненной работе и ответов на вопросы преподавателя.

Оценки за выполнение лабораторных работ и практических занятий могут выставляться по пятибалльной системе или в форме зачета и учитываться как показатели текущей успеваемости студентов.

Критерии оценки лабораторных, практических работ.

Оценка «5» ставится, если учащийся выполняет работу в полном объеме с соблюдением необходимой последовательности проведения опытов и измерений; самостоятельно и рационально монтирует необходимое оборудование; все опыты проводит в условиях и режимах, обеспечивающих получение правильных результатов и выводов; соблюдает требования правил безопасности труда; в отчете правильно и аккуратно выполняет все записи, таблицы, рисунки, чертежи, графики, вычисления; правильно выполняет анализ полученных результатов.

Оценка «4» ставится, если выполнены требования к оценке «5», но было допущено два - три недочета, не более одной негрубой ошибки и одного недочёта.

Оценка «3» ставится, если работа выполнена в ходе проведения опыта и измерений были допущены грубые ошибки.

Оценка «2» ставится, если работа выполнена не полностью и объем выполненной части работы не позволяет сделать правильных выводов: если опыты, измерения, вычисления, наблюдения производились неправильно.

ВЫПОЛНЕНИЕ СРАВНИТЕЛЬНОГО АНАЛИЗА МИКРОСХЕМ
МИКРОКОНТРОЛЛЕРОВ СЕРИИ AVR

1. Цель работы: Приобретение практических навыков сравнительного анализа микросхем микроконтроллеров серии AVR.

2. Время выполнения работы-2 час

3. Краткие теоретические сведения

Микроконтроллер (МК) характеризуется:

1) тактовой частотой, определяющей максимальное время выполнения переключения элементов микропроцессорной системы;

2) разрядностью, т.е. максимальным числом одновременно обрабатываемых двоичных разрядов, существенно влияет на быстродействие.

Разрядность МК обозначается $m/n/k/$ и включает:

m - разрядность внутренних регистров, определяет принадлежность к тому или иному классу процессоров;

n - разрядность шины данных, определяет скорость передачи информации;

k - разрядность шины адреса, определяет размер адресного пространства.

Например, МП i8088 характеризуется значениями $m/n/k=16/8/20$;

3) архитектурой. Понятие архитектуры МК включает в себя систему команд и способы адресации, возможность совмещения выполнения команд во времени, наличие дополнительных устройств в составе микропроцессора, принципы и режимы его работы.

В процессе развития архитектура МК претерпела существенные изменения. Первые МК строились по так называемой принстонской архитектуре (архитектуре фон Неймана) в которой память для команд и данных является общей. Эта архитектура имеет свои достоинства – простоту, возможность оперативного перераспределения памяти между областями хранения команд и данных и др. Недостаток – последовательная во времени выборка из памяти команд и данных, передаваемых по одной и той же системной шине, что ограничивает производительность МК. Тем не менее, в силу своих достоинств, принстонская архитектура не только длительное время доминировала в микропроцессорной технике, но и сохранила свои позиции до настоящего времени.

В системах гарвардской архитектуры память команд и память данных разделены, причем каждая из них имеет собственную шину для обмена с процессором. При этом во время передач данных для выполнения текущей команды можно производить выборку и расшифровку следующей, что повышает производительность МК системы. Реализация системы по сравнению с Принстонской усложняется (в системе больше шин), ниже коэффициент использования памяти. Но в системах высокой производительности и

внутренних структурах высокопроизводительных МК гарвардская архитектура находит широкое применение.

По другому архитектурному признаку, связанному с характером системы команд, МК делятся на:

- с CISC процессорами
- с RISC процессорами
- с VLIW процессорами
- с ARM процессорами

МК CISC имеют так называемую полную систему команд (ComplexInstructionSetComputer), т. е. большой набор разноформатных команд при использовании многих способов адресации. Архитектура CISC силу многообразия команд (общее число команд составляет 100...200) позволяет применять эффективные методы решения задач, но, в тоже время, усложняет схему процессора и увеличивает его стоимость и в общем случае не обеспечивает максимального быстродействия. Они характеризуются следующими особенностями:

- нефиксированным значением длины команды.
- исполнение операций, таких как, загрузка в память, арифметические действия кодируется в одной инструкции.
- небольшим числом регистров, каждый из которых выполняет строго определенную функцию.

МК RISC имеют сокращенную систему команд (ReduseInstructionSetComputer), из которой исключены редко применяемые команды. Общее число команд находится в пределах 50...100.. Простая архитектура позволяет, как удешевить МК, так и увеличить тактовую частоту.

Характерные особенности МК RISC:

- фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды.
- одна инструкция выполняет только одну операцию с памятью — чтение или запись. Операции вида «прочитать-изменить-записать» отсутствуют.
- большое количество регистров общего назначения (32 и более).

Форматы команд, в большинстве своем, идентичны (например, все команды имеют длину 4 байта), резко уменьшено число используемых способов адресации. Данные, как правило, обрабатываются только с регистровой или непосредственной адресацией. Значительно увеличено число регистров процессора, что позволяет редко обращаться к внешней памяти, что повышает быстродействие. Идентичность временных циклов выполнения команд позволяет проще организовывать конвейерные методы обработки информации

МК VLIW- МК с системой команд сверхбольшой разрядности. Команда сверхбольшой разрядности состоит из группы команд, которые могут выполняться параллельно. Команды сверхбольшой разрядности формируются специальным компилятором планирования перед выполнением прикладной программы

ARM-(Advanced RISC machine) усовершенствованная RISC архитектура, 32-битная микропроцессорная архитектура с сокращённым набором команд (RISC), разрабатываемая британской корпорацией ARM Limited.

AVR — семейство восьмибитных микроконтроллеров фирмы Atmel. Микроконтроллеры AVR имеют **гарвардскую** архитектуру и систему команд, близкую к идеологии **RISC**. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл. Система команд микроконтроллеров AVR весьма развита и насчитывает в различных моделях от 90 до 133 различных инструкций. Большинство команд занимает только 1 ячейку памяти (16 бит) и выполняется за 1 такт. Управление периферийными устройствами осуществляется через адресное пространство данных. Для удобства существуют «сокращённые команды» IN/OUT.

Маркировка микроконтроллера - это сгруппированный набор буквы цифр, обозначающих расшифровку производителя чипа, его модели и дополнительных параметров

AT - с данного префикса начинается название микроконтроллера и оно обозначает изготовителя ATMEL.

Слова "tiny", "mega", "xmega" - следует после AT и обозначает семейство микроконтроллеров. Например, маркировка "ATmega" - означает чип от фирмы ATMEL семейства "mega".



В большинстве случаев после AT(tiny, mega, xmega) следуют цифры 1, 2, 4, 8, 16, 32, которые указывают на размер встроенной Flash-памяти в кБ, необходимой для сохранения рабочей программы.

Следующие цифры после размера Flash-памяти обозначают версию(модификацию).

После цифр может присутствовать буква, обозначающая режим питания микроконтроллера:

L - работа при пониженном (Low) напряжении питания 2,7 вольта (2,7-5,5В), но при этом максимальная частота может достигать 8 МГц;

V - возможна работа при напряжении питания 1,8 вольта (1,8-5,5В);

U - возможна работа при напряжении питания 0,7 вольта (0,7-5,5В);

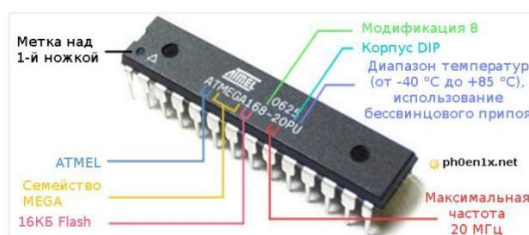
P - энергоэкономичная версия с потреблением тока до 100 нА в режиме Power-down, изготовлена по технологии picoPower;

А - специальная версия с заниженным энергопотреблением, напряжение питания 1,8-5,5В. Если же буквы после цифр нет, то это означает что стандартное питание чипа - 5В, а точнее диапазон 4,5-5,5В.

Цифры после дефиса обозначают максимальную рабочую частоту кристалла (1, 2, 4, 8, 16 - степень двойки).

Далее следуют буквы, обозначающие корпус микроконтроллера:

- А - TQFP;
- С - CBGA;
- СК - LGA;
- Ј - PLCC;
- М - MFL;
- МА - UDFN/USON;
- Р - DIP (PDIP);
- S - EIAJ SOIC;
- SS - JEDEC SOIC;
- Т - TSOP;
- TS - SOT-23;
- Х - TSSOP.



Следующая буква обозначает температурные характеристики микросхемы, а также дополнительные свойства:

- А - диапазон температур (от -20 °C до +85 °C),
- С - коммерческий (Commercial) диапазон температур (от 0 °C до 70 °C)
- Д - расширенный автомобильный диапазон температур (от -40 °C до +150 °C)
- F - расширенный диапазон температур (от -40 °C до +125 °C)
- Н - промышленный диапазон температур (от -40 °C до +85 °C), с использованием NiPdAu (Nickel-Palladium-Gold)
- І - промышленный (Industrial) (от -40 °C до +85 °C)
- Н - расширенный диапазон температур (от -40 °C до +105 °C), использование бессвинцового припоя
- Р - упаковка в ленты для автоматизированных систем сборки.
- U - промышленный (industrial) диапазон температур (от -40 °C до +85 °C), использование бессвинцового припоя
- Z - автомобильный диапазон температур (от -40 °C до +125 °C)

4 Перечень используемого оборудования

4.1 Программа PICSimulatorIDE.

4.2 ПК.

Задание:

Выполнить сравнительный анализ микросхем микроконтроллеров серии AVR.

5. Порядок выполнения

5.1. Изучите краткие теоретические сведения

5.2. Получите вариант задания у преподавателя

5.3. Пользуясь справочной литературой, составьте таблицу справочных данных для заданных ИМС

5.4. Сделать выводы

	ИМС:	ИМС:	ИМС:
Fmax, МГц			
Ucc., В			
размер Flash-памяти в кБ,			
Температурный диапазон			
Тип корпуса			

6. Указания к выполнению отчета

Отчет должен содержать

- тему и цель работы,
- таблицу справочных данных для заданных ИМС
- выводы по проведенным исследованиям
- ответы на контрольные вопросы

7. Контрольные вопросы

1. Что такое архитектура микроконтроллера?
2. Основные характеристики микроконтроллера
3. Преимущества и недостатки прынстонской архитектуры
4. Преимущества и недостатки гарвардской архитектуры
5. Характерные особенности МК RISC:

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб. пособие для студ. СПО.— М.: Издательский центр «Академия», 2019г..

2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021

3. Алиев, М. Т. Микропроцессоры и микропроцессорные системы управления. 8-разрядные процессоры семейства AVR [Электронный ресурс]. – Йошкар-Ола: ПГТУ, 2016

Приложение

Таблица справочных данных для микроконтроллеров

Маркировка	V _{cc} (В)	F _{max} (МГц)	Корпус
ATtiny11L-2	2,7—5,5	2	8P, 8S
ATtiny11-6	4,0—5,5	6	
ATtiny12V-1	1,8—5,5	1	8P, 8S
ATtiny12L-4	2,7—5,5	4	
ATtiny12-8	4,0—5,5	8	
ATtiny15L-1	2,7—5,5	1,6	8A, 8S
AT90LS2323-4	2,7—6,0	4	8P, 8S
AT90S2323-10	4,0—6,0	10	
AT90LS2343-4	2,7—6,0	4	8P, 8S
AT90S2343-10	4,0—6,0	10	
AT90S1200-4	2,7—6,0	4	20P, 20S, 20Y
AT90S1200-12	4,0—6,0	12	
AT90S2313-4	2,7—6,0	4	20P, 20S
AT90S2313-10	4,0—6,0	10	
ATtiny28L-4	2,7—5,5	4	32A, 28P, 32M
ATtiny28V-1	1,8—5,5	1,2	
AT90LS4433-4	2,7—6,0	4	32A, 28P
AT90S4433-8	4,0—6,0	8	
AT90S8515-4	2,7—6,0	4	44A, 44J, 40P
AT90S8515-8	4,0—6,0	8	
AT90LS8535-4	2,7—6,0	4	44A, 44J, 40P
AT90S8535-8	4,0—6,0	8	
ATmega163L-4	2,7—5,5	4	44A, 40P
ATmega163-8	4,0—5,5	8	
ATmega103L-4	2,7—3,6	4	64A
ATmega103-6	4,0—5,5	6	

V_{cc}- диапазон допустимых значений напряжения питания

F_{та}- максимальное значение тактовой частоты

ЛАБОРАТОРНАЯ РАБОТА №1
ЗНАКОМСТВО С ПРОГРАММОЙ PIC SIMULATOR IDE

1. Цель работы: получение практических навыков работы с эмулятором PIC Simulator IDE.

2. Время выполнения работы-2час

3. Краткие теоретические сведения

PIC Simulator IDE – это универсальная программа для просмотра и эмулирования результатов выполнения ассемблерной программы для PIC микроконтроллеров и содержит встроенные эмулятор, компилятор BASIC, ассемблер, дизассемблер и отладчик. Основное окно программы отображает состояние всех внутренних регистров микроконтроллера, мнемонику последней выполненной и следующей команд, тактовый цикл и указатель команд, реальное время выполнения команды при заданной тактовой частоте и типе микроконтроллера.

Программа также включает следующие инструменты разработчика:

- просмотр программной памяти
- редактор EEPROM данных
- просмотр аппаратного стека
- внешняя разводка контроллера с отображением состояния выводов
- менеджер точек останова
- виртуальный LCD дисплей
- эмулируемый аппаратный интерфейс UART
- эмулируемый последовательный порт ПК
- эмулируемый программный интерфейс UART
- осциллограф
- 7-сегментный LED дисплей

4. Перечень используемого оборудования:

4.1 Программа PIC Simulator IDE.

4.2 ПК.

Задание:

Составить и выполнить программу на языке ассемблера для вычисления арифметического выражения по заданному варианту и определить, какие из регистров общего назначения (GPR) используются для хранения слагаемых и результата

5. Порядок выполнения работы

5.1. Запустите программу PIC Simulator IDE из меню Пуск.

- откройте меню Options\Select Microcontroller, выберите 'PIC16F84'

- откройте Tools\BASIC Compiler

5.2. Введите в поле для набора следующую программу:

Dim a As Word '1-е число

Dim b As Word '2-е число

Dim x As Word 'результат

a = 1 'задание 1-го числа

b = 2 'задание 2-го числа

x = a + b 'результат

5.3. Сохраните файл File\Save As. Нажмите Tools\Compile. Компилятор сгенерирует файл с расширением *.asm.

5.4. Закройте компилятор BASIC.

5.5. Откройте Tools\Assembler. Нажмите File\Open

5.6. Выберите сохраненный ранее файл *.asm. В окне редактора отобразится исходный код программы на языке ассемблера.

5.7. Изучите программу, сравните ее с написанной на BASIC. Выделите участки кода для каждой из операций, напишите комментарии для каждой команды.

5.8. Закройте окно Ассемблера.

5.9. Откройте меню File\Load Program. Выберите *.hex файл. При этом программа загрузится в программную память эмулятора.

5.10 Откройте Tools\Breakpoints Manager. Нажмите Yes, чтобы использовать существующий файл листинга программы *.lst.

5.11 Щелчком левой кнопки мыши задайте 3 точки останова, соответствующие заданию чисел a, b, и результату x. Выставьте опцию Hold PC In Focus.

5.12 Выберите Rate\Extremely Fast simulation rate. Нажмите Simulation\Start. Эмулятор будет переходить в режим пошагового выполнения программы после каждой точки останова. Для дальнейшего запуска программы используйте клавишу F1. Выясните, какие из регистров общего назначения (GPR) используются для хранения слагаемых и результата

5.13. Составьте программу на языке ассемблера для вычисления арифметического выражения по одному из вариантов в таблице. Результат поместите в ячейку с адресом X.

Вариант	1	2	3	4	5
Выражение	$(A-C)2+3D$	$A+3(2B-D)$	$2A-3B+D$	$3A+2(B-D)$	$2(3A-B+D)$
X	30h	34h	28h	25h	31h

5.14. Запишите выводы по проведенным исследованиям и ответьте на контрольные вопросы.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- перечень используемого оборудования
- исходные тексты программ
- таблицу результатов
- выводы по проведенным исследованиям
- ответы на контрольные вопросы.

7. Контрольные вопросы

1. Каково назначение программы PIC Simulator IDE?
2. На каких языках программирования возможно написание отлаживаемой программы?
3. Зачем нужны средства BASIC Compiler, Assembler, Breakpoints Manager?

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб. пособие для студ. СПО.— М.: Издательский центр «Академия», 2019г..
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021
3. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ЛАБОРАТОРНАЯ РАБОТА №2

СИНХРОНИЗАЦИЯ ПРОЦЕССОВ УПРАВЛЕНИЯ В СРЕДЕ PIC SIMULATOR IDE.

1. **Цель работы:** Исследование возможностей МК при выполнении команд переходов и организации подпрограмм.

2. **Время выполнения работы-4час**

3. **Краткие теоретические сведения**

Для согласования работы программы МК с внутренними и внешними событиями, а также для отсчета необходимых интервалов времени при взаимодействии с внешними устройствами (ВУ) в микропроцессорной системе на базе МК необходима синхронизация работы МК и ВУ с выработкой тактовых сигналов. Для формирования тактовых интервалов можно использовать подпрограммы задержки, которые реализуются при помощи пустого цикла или с использованием встроенного в МК таймера. В первом случае требуется знать количество тактов, занимаемых командами, входящими в цикл (см. таблицу 1).

Таблица 1 — Команды МК PIC

Мнемоника	Описание	Кол. тактов	Изменяет флаги	Мнемоника	Описание	Кол. тактов	Изменяет флаги
ADDWf,d	Сложение W и f	1	C, DC, Z	XORWf,d	Логическая операция исключающего ИЛИ с W и f	1	Z
ANDWf,d	Поразрядная операция "И" с W и f	1	Z	BCFf,d	Очистка бита в f	1	-
CLRF f	Очистка регистра f	1	Z	BSFf,b	Установка бита в f	1	-
CLRW	Очистка регистра W	1	Z	BTFSCf,b	Проверка на равенство бита нулю, пропускаем след.команду если да.	1(2)	-
COMFf,d	Инвертирование битов регистра f	1	Z	BTFSSf,b	Проверка на равенство бита 1, пропускаем след.команду если да.	1(2)	-
DECFf,d	Уменьшение значения регистра f	1	Z	ADDLW k	Сложение константы и W	1	C, DC, Z
DECFSZf,d	Уменьшение значения регистра f, пропуск следующей инструкции если результат равен нулю.	1(2)	-	ANDLW k	Логическая операция "И" с константой и W	1	Z
INCFf,d	Увеличение значения регистра f на 1	1	Z	CALL k	Вызов подпрограммы	2	-
INCFSZf,d	Увеличение значения регистра f, пропуск следующей инструкции если результат равен нулю.	1(2)	-	GOTO k	Безусловный переход	2	-

IORWf,d	Логическая операция исключающего ИЛИ W и f	1	Z	IORLW k	Логическая операция исключающего ИЛИ с константой и W	1	Z
MOVf,d	Пересылка содержимого регистра f	1	Z	MOVLW k	Пересылка константы в регистр W	1	-
MOVWF f	Пересылка содержимого регистра W в регистр f	1	-	RETFIE	Возврат управления после прерывания	2	-
RLFf,d	Циклический сдвиг влево через флаг переноса	1	C	RETLW k	Возврат с записью константы в W	2	-
RRFf,d	Циклический сдвига вправо через флаг переноса	1	C	RETURN	Возврат из подпрограммы	2	-
SUBWFf,d	Вычитание W из f	1	C, DC, Z	SUBLW k	Вычитание W из константы	1	C, DC, Z
SWAPFf,d	Обмен местами полу-байт регистра f	1	-	XORLW k	Логическая операция исключающего ИЛИ с константой и W	1	Z

4. Перечень используемого оборудования:

4.1 Программа PIC Simulator IDE.

4.2 ПК.

Задание:1. Используя команды условного перехода и регистры МК, напишите программу задержки длительностью 1 сек (тактовая частота МК — 4 МГц).

2. Напишите программу задержки длительностью 1 сек., используя таймер МК (тактовая частота МК — 4 МГц).

3. Написать программу «бегущие огни» для порта В МК, к которому подключена линейка светодиодов, в соответствии с заданием.

№	Задание	№	Задание
1	С верхнего светодиода вниз.	16	С нижнего светодиода вверх.
2	Сверху по два светодиода вниз.	17	Снизу по два светодиода вверх.
3	Начиная с 3-го светодиода вниз.	18	Начиная с 6-го светодиода вверх.
4	Начиная с середины, два огня движутся в разные стороны.	19	Начиная с краев, два огня движутся на встречу друг другу.
5	Горят все светодиоды, а потухший начинает двигаться с верхнего вниз.	20	Горят все светодиоды, а потухший начинает двигаться нижнего вверх.
6	Движение начинается с крайнего светодиода вверх.	21	Движение начинается с 3-го светодиода вверх.

7	Бегущий огонь с верхнего светодиода через один вниз.	22	Бегущий огонь с нижнего светодиода через один вверх.
8	Начиная с 4-го нижнего вверх.	23	Начиная с 4-го верхнего вниз.
9	Начиная со 2-го верхнего, по два зажженных светодиода вверх.	24	Начиная со 2-го нижнего, по два зажженных светодиода вниз.
10	С 6-го через один вверх.	25	Со 2-го через один вниз.
11	Начиная с 5-го, светодиоды зажигаются по одному вниз.	26	Начиная с 5-го, светодиоды зажигаются по одному вверх.
12	С первого верхнего, по одному, зажигаются все светодиоды. Дойдя до последнего, гаснут в обратном направлении.	27	С первого нижнего, по одному, зажигаются все светодиоды. Дойдя до последнего, гаснут в обратном направлении.
13	Начиная с верхнего, по одному зажигаются все светодиоды. Следом гаснут по два в том же направлении.	28	Начиная с нижнего, по одному зажигаются все светодиоды. Следом гаснут по два в том же направлении.
14	Начиная с верхнего, через два, зажигаются светодиоды. Потом, со второго через два. Далее с третьего, через два.	29	Начиная с нижнего, через два, зажигаются светодиоды. Потом, со второго через два. Далее с третьего, через два.
15	С нижнего светодиода вверх и обратно.	30	С верхнего светодиода вниз и обратно.

В МК может существовать несколько таймеров и функции у них могут быть различными: от простого счета тактовых импульсов до счета внешних импульсов и выработке сигнала ШИМ. Работу таймеров в контроллерах PIC рассмотрим на примере самого простого таймера/счетчика TMR0 который имеет следующие особенности:

- 8-разрядный таймер/счетчик;
- возможность чтения и записи текущего значения счетчика;
- 8-разрядный программируемый предделитель;
- внутренний и внешний источник тактового сигнала;
- выбор активного фронта внешнего тактового сигнала;
- прерывания при переполнении (переход от FFh к 00h).

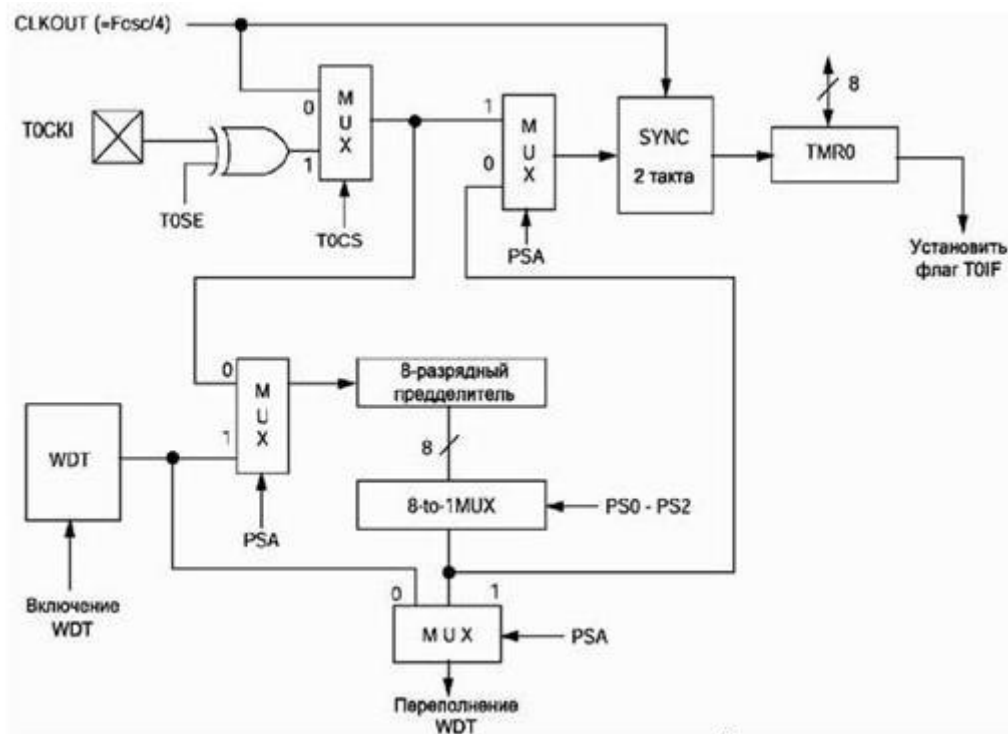


Рисунок 1 - Схема таймера/счетчика TMR0

Настройка TMR0 осуществляется с помощью регистра OPTION

RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1
-RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Бит 7							Бит 0

Рисунок 2 — Регистр OPTION

Регистр OPTION_REG доступен для чтения и для записи. Он содержит биты: конфигурации делителя(PSC) для TMR0/WDT, внешнего прерывания INT и состояния выходов порта В. Назначение битов регистра OPTION:

Бит 7: RBPU: включение подтягивающих резисторов на порте В (0 = подтягивающий резистор включен, 1 = подтягивающий резистор выключен).

Бит 6: INTEDG: выбор активного фронта сигнала прерывания на входе RB0/INT (0 -прерывание по заднему фронту, 1 -прерывания по переднему фронту.)

Бит 5: T0CS: выбор источника тактового сигнала для TIMER0 (1 = тактовый сигнал с входа RA4/T0CKI, 0 = внутренний источник тактового сигнала (CLKOUT)).

Бит 4: T0SE: выбор фронта приращения TMR0 при внешнем тактовом сигнале(1 = приращение по заднему фронту, сигнала на T0CKI, 0 = приращение по переднему фронту сигнала на T0CKI).

Бит 3: PSA: выбор включения делителя (1 = делитель включен перед таймером WDT, 0 = делитель включен перед таймером TMR0)

Биты 2-0: PS2 - PS0; выбор коэффициента предделителя. Коэффициенты деления предделителя показаны в таблице 2

Таблица 2 - Коэффициенты деления в зависимости от значений PS2- PS0

PS2	PS1	PS0	TMR0 (PSA = 0)	WDT (PSA = 1)
0	0	0	1:2	01:01:00
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Установка коэффициента деления предделителя 1:1 для TMR0 соответствует переключению предделителя на сторожевой таймер.

В самом простом случае таймер/счетчик считает тактовые импульсы контроллера до момента переполнения (8-битный счетчик считает от 0 до 255), после чего он сбрасывается в 0, вызывает прерывание, устанавливая бит T0IF, начинает новый отсчет. Все остальные действия производятся в подпрограмме обработки прерываний.

Для начала необходимо рассчитать сколько раз переполнится таймер для отсчета 1 секунды (при тактовой частоте 4МГц). В PIC-контроллерах входная частота делится на 4! Таким образом рабочая частота контроллера будет в 4 раза меньше. 1000000 — рабочая частота контроллера ($F_{osc}/4$). Для того чтобы произошло переполнение таймера он должен отсчитать 256 тактов. $1000000 / 256 = 3906,25$ — именно столько раз должен переполниться таймер чтобы отсчитать 1 секунду. 3906 слишком большое число (максимальное 8-битное число — 255), хотя можно отсчитать и его (как 16-битное или как 2 счетчика). Но гораздо удобнее воспользоваться предделителем таймера. Например, используя предделитель с коэффициентом 16 необходимо считать не до 3906,25, а до 244,14. Так как контроллер считает целые числа, то получается число 244 — именно столько раз должен переполниться таймер TMR0 (вызвать прерывание) чтобы отсчитать 1 секунду. Так как для работы таймера используется не точное число, то отсчет будет неправильным. Проведем обратные вычисления. $244 * 16 * 256 * 4 = 3997696$, $4000000 - 3997696 = 2304$ такта кварца $1 / 4000000 * 2304 = 0,000576 \text{ с} = 0,576 \text{ мс} = 576 \text{ мкс}$. Именно на 576 микросекунд таймер будет спешить каждую секунду. За 1736 секунды (29 минут) таймер убежит на 1 секунду вперед. Получается, что например часы с таким таймером убегут вперед на 5 часов за 1 год. Тут есть несколько выходов:

1) программный — каким-то алгоритмом учитывать накопление ошибки и производить корректировку.

2) можно использовать кварц с рабочей частотой, которая будет хорошо делиться. Например, частота кварц 4,096 МГц делится без дробных частей $4096000 / 4 / 16 / 256 = 250$.

3) также можно отсчитывать время от внешнего источника тактовых импульсов, который будет делиться без дробных частей. В качестве источника таких импульсов может служить кварц с частотой 32768 Гц ($32768 / 256 = 32$).

4. Порядок выполнения.

4.1. Используя команды условного перехода и регистры МК, напишите программу задержки длительностью 1 сек (тактовая частота МК — 4 МГц).

4.2. Напишите программу задержки длительностью 1 сек., используя таймер МК (тактовая частота МК — 4 МГц). Алгоритм работы программы следующий:

- задать начальное значение счетчика переполнений таймера равным 244 (ввести соответствующую переменную программы);

- разрешить прерывания от таймера/счетчика (установить бит T0IE в регистре INTCON);

- установить таймер/счетчик на работу от внутреннего источника тактовых импульсов (сбросить бит T0CS в регистре OPTION);

- установить значение коэффициента деления передделителя равным 16 и выбрать включение передделителя перед таймером (заданием соответствующих бит регистра OPTION);

- в цикле при возникновении прерывания (установлен бит T0IF в регистре INTCON) декрементировать значение счетчика переполнений и сбросить бит T0IF и так до тех пор пока счетчик переполнений не станет равен 0.

Оформить код в виде подпрограммы.

4.3. Написать программу «бегущие огни» для порта В МК, к котором подключена линейка светодиодов, в соответствии с заданием.

4.4. Запишите выводы по проведенным исследованиям и ответьте на контрольные вопросы.

5. Указания к выполнению отчета

Отчет должен содержать:

- исходные тексты программ по пунктам 4.1-4.3 с комментариями.
- ответы на контрольные вопросы.
- выводы по проведенным исследованиям

6. Контрольные вопросы

1. В каких режимах может работать таймер МК? В каких случаях удобно использовать предделитель?
2. Что собой представляет стек МК? Можно ли помещать в стек произвольные данные?
3. Какие методы создания подпрограмм задержки вы знаете? Назовите их достоинства и недостатки.
4. Укажите назначение регистров TRISA и TRISB.

7. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.– М.: Издательский центр «Академия», 2019г..
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021
3. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ИЗУЧЕНИЕ РАБОТЫ МОДУЛЯ АЦП МК В СРЕДЕ PIC SIMULATOR IDE.

1. Цель работы: Исследование возможностей исследования возможностей МК при работе с модулем АЦП, а также при формировании управляющих последовательностей импульсов.

2. Время выполнения работы-4час

3.Краткие теоретические сведения

Отличительная особенность современных МК — интегрированный на кристалле модуль многоканального аналого-цифрового преобразователя (АЦП). Модуль АЦП предназначен для ввода в МК аналоговых сигналов с датчиков физических величин и преобразования этих сигналов в двоичный код с целью последующей программной обработки и выдачи управляющих сигналов. Структурная схема типового модуля АЦП представлена на рис. 1.

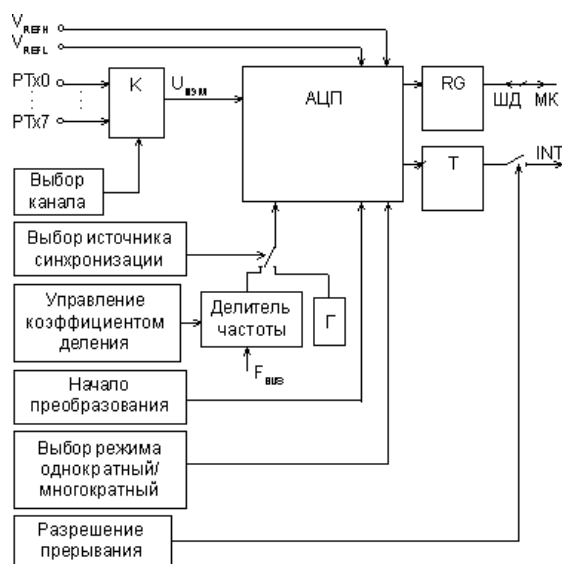


Рисунок 1 - Схема модуля АЦП МК

Многоканальный аналоговый коммутатор служит для подключения одного из источников аналоговых сигналов (PTx0... PTx7) к входу АЦП. Выбор источника сигнала для измерения осуществляется посредством записи номера канала коммутатора в соответствующие разряды регистра управления АЦП ADCON0. Диапазон измеряемых значений напряжения аналоговых входов определяется опорным напряжением $U_{оп}$. Разрешающая способность АЦП составляет $U_{оп} / 2^n$, где n — число двоичных разрядов в слове результата. Для 10 разрядного АЦП с опорным напряжением 5В разрешающая способность составит около 5 мВ.

Максимальное значение опорного напряжения, как правило, равно напряжению питания МК. Два вывода модуля АЦП используются для задания опорного напряжения: VREFH — верхний предел $U_{оп}$, VREFL - нижний предел. Разность потенциалов на этих входах и составляет $U_{оп}$. Если измеряемое напряжение $U_{изм} \Rightarrow VREFH$, то результат преобразования будет равен 3FF, код 00 соответствует напряжениям $U_{изм} \leq VREFL$. Для достижения максимальной точности измерения следует выбрать максимально допустимое значение $U_{оп}$. В этом случае напряжение смещения нуля входного буфера и нелинейность передаточной характеристики АЦП будут вносить относительно малые погрешности.

В данной лабораторной работе изучается модуль АЦП микроконтроллера PIC16F877. Модуль аналого-цифрового преобразователя у данного МК имеет восемь аналоговых вводов. АЦП обеспечивает преобразование аналогового входного сигнала в соответствующий 10-разрядный цифровой код. Преобразование осуществляется методом последовательного приближения, на время преобразования уровень входного сигнала удерживается устройством выборки и хранения. Источник опорного напряжения задается программно. Внутренним источником является положительное напряжение питания устройства (VDD). АЦП может работать, когда микроконтроллер находится в режиме SLEEP. В этом случае АЦП должен тактироваться от внутреннего генератора, т.к. основной генератор контроллера в SLEEP режиме отключается.

АЦП настраивается регистрами ADCON0, ADCON1. Результат преобразования помещается в регистры ADRESH, ADRESL.

Регистр ADCON0 (адрес 1Fh)

№ бита	7	6	5	4	3	2	1	0
Обозначение	ADSC1	ADSC0	CHS2	CHS1	CHS0	GO/DONE	-	ADONE

Назначение битов регистра ADCON0:

Биты 7-6: ADSC1:ADSC0 – выбор частоты преобразования

00= FOSC/2

01= FOSC/8

10= FOSC/32

11= FRC (синхронизация от внутреннего RC генератора)

Бит 5-3: CHS2:CHS0 – выбор аналогового канала

000 = канал 0, (RA0/AN0)

001 = канал 1, (RA1/AN1)

010 = канал 2, (RA2/AN2)

011 = канал 3, (RA3/AN3)

100 = канал 4, (RA5/AN4)

101 = канал 5, (RE0/AN5)

110 = канал 6, (RE1/AN6)

111 = канал 7, (RE2/AN7)

Бит 2: GO/DONE –состояние преобразования

Если GO/DONE = 1 выполняется преобразование (установка бита запускает преобразование)

Если GO/DONE = 0 преобразование окончено (автоматически сбрасывается аппаратным путем при окончании преобразования)

Бит 1: зарезервированный бит, читается как “0”

Бит 0: ADON –включение модуля АЦП 1 = модуль включен; 0 = модуль выключен и выходы преобразователя закрыты для снижения потребления.

Регистр **ADCON1**(адрес **9Fh**)

№ бита	7	6	5	4	3	2	1	0
Обозначение	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

Назначение битов регистра **ADCON1**:

Бит 7: ADFM –формат записи результата преобразования: 1 = правостороннее выравнивание, 6 старших значащих битов регистра **ADRESH** читается как “0”, 0 = левостороннее выравнивание, 6 младших значащих битов регистра **ADRESH** читается как “0”.

Бит 6-4: зарезервированные биты, читаются как “0”.

Бит 3-0: PCFG3:PCFG0 –управление конфигурацией входов АЦП (см. Таблицу 1).

Регистр **ADCON0** используется для настройки АЦП, а с помощью регистра **ADCON1** выбирается одна из предложенных комбинаций настройки соответствующих портов для работы цифровых и аналоговых каналов, внешних или внутренних источников опорного напряжения АЦП (см. табл. 1). Бит **ADFM** в регистре **ADCON1** дает возможность выбора, как представить 10-ти разрядный результат преобразования в двух 8-ми разрядных регистрах. Когда преобразование закончилось, результат загружается в регистры **ADRESH** (старшие биты результата) и **ADRESL** (младшие биты результата). Вместе с этим сбрасывается бит **GO/DONE** в регистре **ADCON0** и устанавливается флаг прерывания **ADIF** в регистре **PIR1**.

Регистр PIR1 (адрес **0Ch**) доступен для чтения и для записи. Он содержит флаги прерываний периферийных устройств. Флаги должны быть сброшены программно.

№ бита	7	6	5	4	3	2	1	0
Обозначение	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

Назначение битов регистра **PIR1**:

Бит 7: PSPIF – флаг прерывания параллельного ведомого порта. 1 = если выполнена операция чтения/записи параллельного порта. 0 = если нет операции чтения/записи параллельного порта.

Бит 6: ADIF – флаг прерываний преобразования АЦП. 1 = если преобразование завершено. 0 = если преобразование не завершено.

Бит 5: RCIF- флаг прерываний от приемника USART. 1 = буфер приемника USART заполнен. 0 = буфер приемника USART пуст.

Бит 4: TXIF- флаг прерываний передатчика USART. 0=буфер передатчика USART заполнен. 1 = буфер передатчика USART пуст.

Бит 3: SSPIF- флаг прерывания синхронного последовательного порта. 1 = если произошло прерывание SSP, то он должен быть программно очищен перед возвращением из подпрограммы обработки прерываний. Условия установки этого бита: В SPI режиме – если произошла передача/прием данных. В режиме I2C ведомого – если произошла передача/прием данных. I2C ведущего – если произошла передача/прием данных. 0 = если прерываний SSP не было.

Бит 2: CCP1IF- флаг прерывания CCP1. Режим захвата: 1 = если значение TMR1 зафиксировано. 0 = если фиксация не произошла. Режим сравнения: 1 = если значение TMR1 равно заданному. 0 = если не равно. Режим PWM: не используется.

Бит 1: TMR2IF- флаг прерывания при равенстве TMR2 и регистра PR2 (регистр периода TMR2). 1 = при равенстве TMR2 и регистра PR2. 0 = если не равно.

Бит 0: TMR1IF- флаг прерывания по переполнению TMR1. 1 = если произошло переполнение TMR1. 0 = если нет переполнения TMR1.

Таблица 1 — Конфигурация битов PCFG3:PCFG0

PCFG3-PCFG0	AN7/RE2	AN6/RE1	AN5/RE0	AN4/RA5	AN3/RA3	AN2	AN1	AN0	V _{REF} ⁺	V _{REF} ⁻
0000	A	A	A	A	A	A	A	A	V _{DD}	V _{SS}
0001	A	A	A	A	V _{REF+}	A	A	A	RA3	V _{SS}
0010	D	D	D	A	A	A	A	A	V _{DD}	V _{SS}
0011	D	D	D	A	V _{REF+}	A	A	A	RA3	V _{SS}
0100	D	D	D	D	A	D	A	A	V _{DD}	V _{SS}
0101	D	D	D	D	V _{REF+}	D	A	A	RA3	V _{SS}
011x	D	D	D	D	D	D	D	D	V _{DD}	V _{SS}
1000	A	A	A	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	V _{DD}	V _{SS}
1010	D	D	A	A	V _{REF+}	A	A	A	RA3	V _{SS}
1011	D	D	A	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2
1100	D	D	D	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2
1101	D	D	D	D	V _{REF+}	V _{REF-}	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	V _{DD}	V _{SS}
1111	D	D	D	D	V _{REF+}	V _{REF-}	D	A	RA3	RA2

A = аналоговый вход, D = цифровой канал ввода/вывода.

Регистр **PIE1** (адрес **8Ch**) содержит маски прерываний периферийных устройств. Для того, что бы разрешить прерывания от периферийных устройств, необходимо установить бит PEIE (бит 6 регистра INTCON).

№ бита	7	6	5	4	3	2	1	0
Обозначение	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

Назначение битов регистра **PIE1**:

Бит 7: PSPIE – маска прерывания параллельного ведомого порта. 1 = прерывание при чтении/записи порта разрешено 0 = прерывание при чтении/записи порта запрещено.

Бит 6: ADIE – маска прерываний преобразования АЦП. 1 = прерывание разрешено. 0 = прерывание запрещено.

Бит 5: RCIE – маска прерываний приемника USART.

Бит 4: TXIE - маска прерываний передатчика USART.

Бит 3: SSPIE – маска прерывания синхронного последовательного порта.

Бит 2: CCP1IE – маска прерывания CCP1

Бит 1: TMR2IE – маска прерывания равенства TMR2 и регистра PR2.

Бит 0: TMR1IE – маска прерывания по переполнению TMR1.

Широтно-Импульсная Модуляция (ШИМ) - это способ задания аналогового сигнала путём изменения ширины (длительности) прямоугольных импульсов. Выходной сигнал преобразуется в последовательность управляющих им-пульсов с длительностью $D = Y*(T/100\%)$, где D – длительность импульса, сек; T – период следования импульсов, сек. программируется для изменения длительности импульсов); Y – выходной сигнал регулятора.

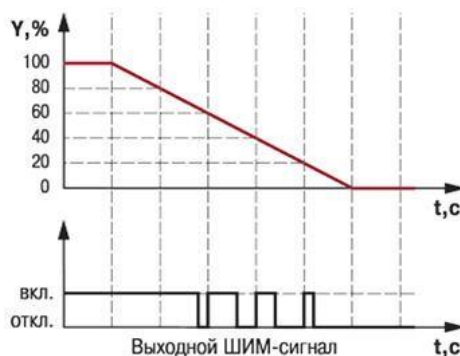


Рисунок 2 Принцип ШИМ

Для формирования импульсов ШИМ удобно использовать таймер,

позволяющий осуществлять деление частоты тактовых импульсов, а также позволяющий изменять полный коэффициент пересчета. Такими возможностями обладает таймер TMR2. Модуль таймера 2 – это 8-разрядный счетчик с предделителем и постделителем. Тактирование таймера осуществляется внутренней частотой (FOSC/4) через предделитель, который может иметь коэффициент деления 1:1, 1:4 или 1:16. Таймера имеет 8-разрядный регистр периода PR2. Значение таймера может инкрементироваться от 00h до значения, записанного в регистр PR2, после чего сбрасывается в 00h, генерируя сигнал прерывания.

ВНИМАНИЕ! Прерывание генерируется с учетом коэффициента постделителя, т.е. если коэффициент пересчета постделителя 1:2, то сигнал о прерывании будет сгенерирован только после двух переполнений таймера 2.

В таблице 2 приведен регистр управления таймером 2.

Регистр T2CON: регистр управления таймера 2 (адрес 12h).

№ бита	7	6	5	4	3	2	1	0
Обозначение	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Назначение битов регистра **T2CON:**

Биты 6-3: TOUTPS3: TOUTPS0 – коэффициент деления постделителя

0000=1:1

0001=1:2

0010=1:3

.....

1111=1:16

Бит 2: TMR2ON – управление таймером 2. 1 = таймер2 включен. 0 = таймер2 выключен.

Биты 1-0: T2CKPS1:T2CKPS0 – коэффициент деления предделителя

00=1:1

01=1:4

1x=1:16.

4.Перечень используемого оборудования:

4.1 Программа PIC Simulator IDE.

4.2 ПК.

Задание.

1. Написать программу выводящую значение канала AN0 АЦП на выходы порта В

2. Написать программу выдающую последовательность импульсов с фиксированной скважностью и частотой 1 КГц на вывод RB.0 порта В с использованием таймера TMR2.

5.Порядок выполнения.

5.1. Для чтения значения АЦП использовать следующий алгоритм:

5.1.1. Установить конфигурацию модуля АЦП, выбрав аналоговые входы и источник опорного напряжения (ADCON1), канал АЦП (ADCON0) и источник синхронизации АЦП (ADCON0).

5.1.2. Включить модуль АЦП (бит ADON в ADCON0).

5.1.3. Разрешить прерывания, установив биты GIE, PEIE (INTCON) и ADIE (PIE1).

5.1.4. Сбросить бит ADIF (PIR1).

5.1.5. Выдержать время, требуемое для устройства выборки и хранения. После того как был выбран один из аналоговых каналов, перед преобразованием должно пройти определенное время, которое складывается из времени задержки усилителя аналогового сигнала (около 2 мкс) и времени заряда конденсатора защелки аналогового входа. При температуре 25°C и сопротивлении источника 10 кОм общая временная задержка составит около 17 мкс.

5.1.6. Начать преобразование, установив бит GO/DONE (ADCON0).

5.1.7. Ожидать конца преобразования АЦП: опрашивая бит GO/DONE (ADCON0) и ожидая прерывания АЦП (опрашивая бит ADIF (PIR1)).

5.1.8. Считать регистр результата преобразования АЦП (ADRES) и очистить бит ADIF.

5.2. Написать программу выдающую последовательность импульсов с фиксированной скважностью и частотой 1 КГц на вывод RB.0 порта В с использованием таймера TMR2.

6 Содержание отчета

Отчет должен содержать:

- исходные тексты программ по пунктам 4.1-4.2 с комментариями.
- ответы на контрольные вопросы.
- выводы по проведенным исследованиям

7. Контрольные вопросы

1. Какие регистры, работающие с АЦП, вы знаете? Каково их назначение?
2. Установка какого бита вызывает начало преобразования АЦП? Какие возможные способы для определения окончания преобразования АЦП?
3. Может ли АЦП работать, когда микроконтроллер находится в «спящем режиме»?
4. Как получается, что 10-разрядный результат АЦП преобразования, помещается в 8-разрядный регистр?

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.– М.: Издательский центр «Академия», 2019г..

2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021
3. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ЛАБОРАТОРНАЯ РАБОТА №4
ИЗУЧЕНИЕ РАБОТЫ МОДУЛЯ АЦП МК В СРЕДЕ PIC SIMULATOR
IDE.

1. **Цель работы:** исследование возможностей программной и аппаратной реализации UART МК

2. **Время выполнения работы-4час**

3. **Краткие теоретические сведения**

UART (UniversalAsynchronousReceive/Transmit – универсальный асинхронный прием/передача) это реализованный программно или аппаратно механизм, который позволяет МК получать и отправлять данные по последовательному каналу связи, как правило, через интерфейс RS232.

RS232 (RevisedStandard 232) - это интерфейс, определяющий назначение сигналов и уровни напряжения, используемые аппаратурой приема и передачи данных (такими как ПК и модем). Стандартный разъем RS232 позволяет передавать набор из 9 сигналов. Передаваемые по линии данных уровни напряжения обозначаются как 'MARK' (логическая единица) и 'SPACE' (логический ноль). Напряжение для MARK находится в диапазоне: -3...-15 В. SPACE: +3...+15 В.

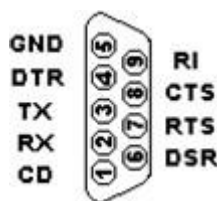


Рисунок 1 - Разъем DB9 интерфейса RS232

GND (SignalGround) — сигнальная (схемная) земля, относительно которой действуют уровни сигналов

TX (Transmit) — последовательные данные — выход передатчика

RX (Receive) — последовательные данные — вход приемника

RTS (RequestToSend) — выход запроса передачи данных: состояние «включено» уведомляет модем о наличии у терминала данных для передачи.

CTS (ClearToSend) — вход разрешения терминалу передавать данные. Состояние «выключено» запрещает передачу данных.

DSR (DataSetReady) — вход сигнала готовности от аппаратуры передачи данных (модем).

DTR (DataTerminalReady) — выход сигнала готовности терминала (ПК) к обмену данными.

CD (CarrierDetected) — вход сигнала обнаружения несущей удаленного модема

RI (RingIndicator) — вход индикатора вызова (звонка).

Микросхемы приемопередатчиков RS232 используют последовательную побитовую передачу сообщений. Передаваемые данные, как правило, содержат: старт - бит, 7 (или менее) бит данных в кодах ASCII, бит четности и один или несколько стоп-бит. ASCII (American Standard Code for Information Interchange) — это Американский стандартный код для обмена информацией. Сообщения при передаче могут идти одно за другим или отсылаться с необходимой задержкой. В примере на рис. 2 передается символ 'A' (41h, 01000001b).

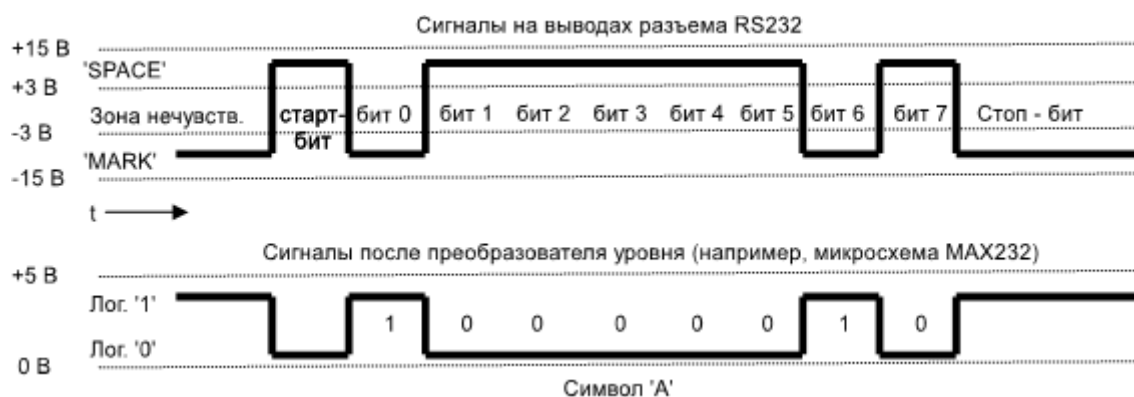


Рисунок 2 — Пример передачи символа по последовательному интерфейсу

Бит четности используется как самое простое средство контроля ошибок при передаче данных. Для более точного контроля ошибок при передаче используется такой метод как циклический контроль четности или подсчет контрольной суммы CRC (Cyclic Redundancy Character) для нескольких последовательных передач.

Программная реализация последовательного интерфейса МК PIC (для МК в которых нет аппаратного модуля UART) должна включать:

- 1) Настройка скорости обмена и формата передаваемых данных.
- 2) Ввод или вывод данных через порт контроля непосредственно на выводы интерфейса RS232.

При вводе данных:

- ожидание прихода старт-бита: запись текущего значения 8 бит данных во временную переменную/регистр (TEMP), очистка переменной/регистра (SERBUF), хранящего принимаемый символ, задержка времени на половину периода тактовой частоты работы интерфейса, побитовый прием данных (с задержкой между приемом битов в один период тактовой частоты интерфейса), запись результата в регистр-аккумулятор.

При выводе данных:

- запись передаваемого символа в переменную/регистр (SERBUF), сброс старт бита, задержка времени на период тактовой частоты работы интерфейса,

побитовая передача данных (с задержкой между приемом битов в один период тактовой частоты интерфейса), выдача стоп-бита.

Аппаратный модуль USART может работать как в синхронном, так и в асинхронном режиме. В настоящее время синхронный режим используется очень редко и для обмена данными с персональным компьютером следует использовать асинхронный режим. Для работы с USART используются следующие регистры: TXSTA, RCSTA, SPBRG, TXREG, RCREG, PIR1, PIE1.

TXSTA – регистр состояния и управления передачей

№ бита	7	6	5	4	3	2	1	0
Обозначение	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D

CSRC – бит выбора источника синхронизации. В асинхронном режиме не используется. В синхронном режиме: 0 – Slavemode (внешний источник синхронизации). 1 – Mastermode (внутренний источник синхронизации – BRG).

TX9 – бит разрешения 9-битной передачи. 0 – используется 8-битная передача. 1 – используется 9-битная передача.

TXEN – бит разрешения передачи. 0 – передача запрещена. 1 – передача разрешена.

SYNC – бит выбора режима работы USART. 0 – асинхронный режим. 1 – синхронный режим.

BRGH – бит выбора скорости BRG (BaudRateGenerator). В асинхронном режиме: 0 – низкая скорость. 1 – высокая скорость. В синхронном режиме не используется.

TRMT – бит состояния TSR (TransmitShiftRegister – Сдвигающий Регистр Передачи). 0 – TSR полон. 1 – TSR пуст.

TX9D – 9-й бит (в режиме 9-битной передачи).

RCSTA – регистр состояния и управления приемом.

№ бита	7	6	5	4	3	2	1	0
Обозначение	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

SPEN – бит разрешения работы последовательного порта. 0 – последовательный порт выключен. 1 – последовательный порт включен.

RX9 – бит разрешения 9-битного приема. 0 – используется 8-битный прием. 1 – используется 9-битный прием.

SREN – бит разрешения разового приема. В асинхронном режиме не используется. В синхронном режиме (Mastermode): 0 – запрещение разового приема (обнуляется после приема данных). 1 – разрешение разового приема. В синхронном режиме (Slavemode) не используется.

CREN – бит разрешения непрерывного приема. В асинхронном режиме: 0 – запрещение непрерывного приема. 1 – разрешение непрерывного приема. В синхронном режиме: 0 – запрещение непрерывного приема. 1 – разрешение непрерывного приема (CREN обнуляет SREN).

ADDEN – бит разрешения обнаружения адреса. В асинхронном режиме (в режиме 9-битного приема): 0 – запрещение обнаружения адреса. 1 – разрешение обнаружения адреса. **FERR** – FramingErrorbit (ошибка передачи кода). 0 – ошибки нет. 1 – ошибка.

OERR – OverrunErrorbit (переполнение буфера). 0 – ошибки нет. 1 – ошибка.

RX9D – 9-й бит (в режиме 9-битного приема).

SPBRG – регистр задающий скорость передачи данных по последовательному каналу.

TXREG – в этот регистр загружаются данные для передачи по последовательному каналу.

RCREG – из этого регистра считываются данные принятые из последовательного канала.

PIR1 – содержит флаги запросов прерываний от периферийных устройств. К USART имеют отношение биты RCIF и TXIF, которые сигнализируют о завершении приема и передачи данных (0 – идет прием/передача; 1 – прием/передача завершены).

PIE1 – регистр разрешения прерываний от периферийных устройств. К USART относятся биты RCIE и TXIE, разрешающие/запрещающие прерывания при завершении приема и передачи данных (0 – прерывания запрещены; 1 – прерывания разрешены).

Кроме настройки перечисленных регистров нужно правильно задать направление передачи данных по линиям RX (прием) и TX (передача). Линии RX и TX подключены соответственно к 7 и 6 разрядам порта C микроконтроллера. Для задания направления передачи данных через порт используются регистры TRISx, где x – имя порта. Значение каждого бита этих регистров указывает направление передачи сигналов по линиям порта (0 – линия передает данные, 1 – линия принимает данные). Для настройки линий TX и RX используется 6 и 7 биты регистра TRISC.

Для инициализации USART нужно:

1. Установить асинхронный режим работы (бит SYNC регистра TXSTA).
2. Установить низкую скорость генератора (бит BRGH регистра TXSTA).
3. Установить скорость соединения 9600 (записать число 25 в регистр SPBRG).
4. Настроить направление линий RX и TX (регистр TRISC).
5. Установить 8-битные прием и передачу (биты TX9, RX9 регистров TXSTA, RCSTA).
6. Включить последовательный порт (бит SPEN регистра RCSTA).
7. Отключить прерывания приема и передачи (биты RCIE и TXIE регистра PIE1).
8. Включить прием (бит CREN регистра RCSTA).
9. Включить передачу (бит TXEN регистра TXSTA).

Для передачи данных по последовательному каналу нужно:

1. Проверить состояние буфера передачи (бит TXIF регистра PIR1).
2. Если буфер пуст записать данные в регистр TXREG (передача начнется автоматически), если же буфер не пуст – ждать освобождения буфера.

Для приема данных из последовательного канала нужно:

1. Проверить наличие ошибок приема (Framingerror и Overrunerror). Если ошибки обнаружены их нужно устранить, и начать прием заново.
2. Проверить состояние буфера приема (бит RCIF регистра PIR1).
3. Если буфер полон, считать данные из регистра RCREG, если же пуст – ждать поступления данных

Для устранения ошибки Framingerror нужно считать неверные данные из регистра RCREG и начать прием заново. Для устранения ошибки Overrunerror нужно сначала сбросить, а затем установить бит CREN регистра RCSTA. После этого начать новый прием.

4. Перечень используемого оборудования:

4.1 Программа PIC Simulator IDE.

4.2 ПК.

Задание:

1. Написать программу выводящую символ «?» при помощи программной реализации UART на вывод RC.0 МК. Осуществить прием символа по выводу RC.1

2. Написать программу, которая считывает строку из последовательного канала и выполняет заданную в ней операцию (чтение данных из EEPROM или запись данных в EEPROM). Формат строки может быть таким: 1-й символ – номер строки в EEPROM (например, от 0 до 9). 2-й символ – код операции (W –

запись, R – чтение). Остальные символы – записываемая в EEPROM строка (в случае записи, а в случае чтения, символы игнорируются).

Примеры: 3WTestWrite – Записывает в 3 строку EEPROM текст “TestWrite”. 5R – Чтение 5 строки EEPROM.

5. Порядок выполнения.

5.1. Записываемая в EEPROM строка данных имеет ограниченную длину (например, 15 символов). Прочитанные из памяти строки выводятся в последовательный канал. Запись символа в последовательный порт и чтение символа из последовательного порта оформить в виде подпрограмм

5.2. Для чтения из EEPROM

5.2.1. Поместить в регистр EEADR адрес ячейки памяти.

5.2.2. Обнулить биты EEPGD и CFGS регистра EECON1.

5.2.3. Установить бит RD регистра EECON1.

5.2.4. Считать данные из регистра EEDATA

5.3. Для записи в EEPROM

5.3.1. Поместить в регистр EEADR адрес ячейки памяти.

5.3.2. Поместить в регистр EEDATA записываемые данные.

5.3.3. Обнулить биты EEPGD и CFGS регистра EECON1.

5.3.4. Установить бит WREN регистра EECON1.

5.3.5. Запретить все прерывания (сбросить бит GIE регистра INTCON).

5.3.6. Выполнить специальную последовательность записи (Writesquence). Последовательно записать в регистр EECON2 числа 55h и 0AAh

```
MOVLW 55h
```

```
MOVWFEECON2
```

```
MOVLW 0AAh
```

```
MOVWFEECON2
```

5.3.7. Установить бит WR регистра EECON1 для начала записи.

5.3.8. После этого можно либо продолжить выполнение программы (если повторной записи в EEPROM не требуется), либо организовать цикл ожидания окончания записи (ждать сброса бита WR).

5.3.9. Разрешить прерывания (установить бит GIE регистра INTCON).

5.3.10. Обнулить бит WREN регистра EECON1 (для того чтобы случайно не испортить содержимое EEPROM).

6. Указания к выполнению отчета

Отчет должен содержать:

- исходные тексты программ по пунктам 4.1-4.2 с комментариями.

- ответы на контрольные вопросы.
- выводы по проведенным исследованиям

7. Контрольные вопросы

1. В чем преимущества и недостатки программной реализации UART?
2. Для чего при передаче данных от МК по интерфейсу RS232 необходимо использовать преобразователи уровня сигнала?
3. Какие методы контроля ошибок при передаче данных по последовательному интерфейсу аппаратным и программным способом вы знаете

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.– М.: Издательский центр «Академия», 2019г..
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021
3. Компиляторы для PIC-контроллеров. – 2017. – Режим доступа: <http://www.microchip.ru>.
4. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ДЛЯ PIC-МИКРОКОНТРОЛЛЕРОВ

1. Цель работы: Изучить и практически исследовать методы разработки и отладки программ на языке Си для PIC-микроконтроллеров с помощью интегрированной среды

2. Время выполнения работы-4час

3.Краткие теоретические сведения

Для создания программного обеспечения микроконтроллерных систем широко используются средства вычислительной техники, в том числе персональные компьютеры, позволяющие разработчику выполнить весь цикл проектирования, включая отладку целевой программы. В настоящее время самым мощным средством разработки программного обеспечения для микроконтроллеров являются интегрированные среды разработки IDE (IntegratedDevelopmentEnvironment).

Одним из таких программных инструментов является MPLAB IDE – интегрированная среда разработки для микроконтроллеров PICmicro компании Microchip. MPLAB IDE содержит все инструментальные средства, необходимые для того, чтобы написать программу на языке Ассемблер, получить ее машинные коды, выполнить на симуляторе ее тестирование и загрузить машинные коды в программатор. IDE имеет встроенные и сменные модули, позволяющие сконфигурировать среду с различными инструментальными средствами, программным обеспечением и аппаратными средствами. Для создания программ на языке Си в MPLAB IDE могут быть добавлены компиляторы C18 и C30 разработки компании Microchip. Могут также использоваться компиляторы фирм HI-TECH, IAR, CSS, ByteCraft. Пакет программ MPLAB IDE можно бесплатно загрузить с сайта www.microchip.com. С сайта www.microchip.ru можно также бесплатно загрузить демоверсии некоторых компиляторов.

Сербская компания MikroElektronika разработала IDE, которая позволяет быстро создавать эффективные программы на языке Си. Среда имеет удобный интерфейс пользователя со встроенным редактором текста и мощным отладчиком программ. Встроенный мастер проектов позволяет в считанные минуты создать заготовку рабочей программы для любого микроконтроллера из целого семейства. Библиотека готовых функций обеспечивает программиста поддержкой для быстрого и безошибочного создания программы. Компания MikroElektronika создала среду разработки для таких популярных и известных микроконтроллеров, как семейство PIC компании Microchip, AVR компании

Atmel и семейство MCS-51. С сайта www.mikroe.com компании MikroElektronika можно бесплатно скачать демонстрационную версию среды для PIC-контроллеров, которая позволяет создавать программы с объемом исполняемого кода до 2 Кбайт.

Разработка программного обеспечения при использовании среды разработки IDE состоит из следующих основных этапов:

1. Создание проекта.
2. Создание исходных файлов на языке Си.
3. Построение проекта.
4. Тестирование программы и ее отладка.

Среда IDE организует программное обеспечение в виде проектов, состоящих из одного файла проекта (файл с расширением *.mcpri) и одного или нескольких исходных файлов на языке Си (файлов с расширением *.c). Исходные файлы могут компилироваться только в том случае, если они включены в проект.

Файл проекта содержит:

- имя проекта;
- тип микроконтроллера;
- тактовую частоту его работы;
- слово конфигурации микроконтроллера;
- список исходных файлов для проекта;
- другие (вспомогательные) файлы.

Построение проекта в mikroC PRO – это процесс компиляции исходных файлов, их компоновки и создание hex-файла, предназначенного для загрузки в программную память микроконтроллера. Для построения проекта надо выполнить команду меню **Build > Build**. После процесса компиляции и компоновки в окне Messages (в нижней части экрана) выдается сообщение об итогах построения. Сообщение содержит информацию о количестве затраченных ресурсов памяти, наличии ошибок (Errors) и предупреждений (Warnings).

Возникающие ошибки (обычно синтаксические) не позволяют компилятору сгенерировать машинные коды программы на основе исходного текста, поэтому такие ошибки обязательно должны быть исправлены. При наличии ошибок выводится сообщение “**Finished** (with errors)”.

Предупреждения выдаются компилятором в том случае, когда он может сгенерировать машинные коды, но в процессе их формирования возникли обстоятельства, требующие внимания разработчика. К таким обстоятельствам относятся, например, использование переменной, которой не присвоено

начальное значение, потеря точности при преобразовании типов переменных и т. п. Как правило, предупреждения выдаются в потенциально опасных ситуациях и позволяют избежать трудно выявляемых логических ошибок в процессе работы программы, вследствие этого игнорировать их не следует.

Если в результате построения проекта возникли сообщения об ошибках или предупреждения, то их перечень с указанием номера строки и типа ошибки отобразится (красным цветом) в окне Messages. Место ошибки в исходном тексте можно легко найти, дважды щелкнув мышью на строке соответствующего сообщения.

Среда IDE имеет средства **отладки**, позволяющие проверить работоспособность программы и исправить при необходимости ошибки в исходном тексте. **Отладку** можно выполнять как с помощью внутреннего симулятора работы микроконтроллера, так и с помощью аппаратного отладчика, подключив к нему целевую микроконтроллерную систему. По умолчанию отладчик IDE настроен для работы в режиме симулятора.

Для запуска отладчика надо выполнить команду из меню

Run > Start Debugger. В результате IDE перейдет в режим отладки (в нем текущая строка исходного кода обозначается зеленой стрелкой и по умолчанию выделена синим цветом), и откроется окно наблюдения Watch Values, позволяющее отслеживать в ходе выполнения программы содержимое различных переменных и регистров. Значения обновляются в процессе симуляции работы программы в отладчике. Последние измененные элементы в окне наблюдения выделяются красным цветом.

Примечание. Если после запуска отладчика на экране будет отсутствовать окно наблюдения Watch Values, то его можно открыть с помощью команды меню View > Debug Window > Watch Window или нажатием комбинации клавиш shift + F5. Формат отображения данных в окне наблюдения будет по умолчанию десятичный.

Для внесения в список окна наблюдения Watch Values той или иной переменной программы, значение которой необходимо отслеживать, ее следует выбрать в раскрывающемся списке строки Select variable from list и нажать кнопку [Add].

После того как переменная добавлена в список окна Watch Values, можно настроить параметры отображения ее в столбце Value. Для этого можно воспользоваться одним из способов:

- дважды щелкнуть мышью на соответствующей строке списка Watch Values в столбце Name или Address;
- щелкнуть в столбце Value и затем нажать расположенную справа

кнопку [...].

В любом случае на экране появится диалоговое окно Edit Value, позволяющее отредактировать значение переменной прямо в ходе выполнения программы или же изменить формат ее отображения.

Для выбора формата отображения необходимо установить флажок в соответствующем окошке:

- Dec – десятичный;
- Hex – шестнадцатеричный;
- Bin – двоичный;
- Float – с плавающей точкой; – Char – символьный.
- Установка флажка Signed означает активизацию отображения знака.

Для того чтобы подтвердить изменения, внесенные в значение или формат отображения переменной, в диалоговом окне Edit Value следует нажать кнопку [OK]. Нажатие кнопки [Cancel] отменяет все внесенные изменения.

Для удаления текущего элемента в списке Watch Values служит кнопка [Remove], а для полной очистки этого списка – кнопка [Remove All].

В режиме отладки открывается также окно хронометража Watch Clock. В нем отображается текущий счетчик Current Count командных (машинных) циклов и времени в микросекундах (us) от момента запуска отладчика. Секундомер (Stopwatch) измеряет время исполнения в количестве командных циклов и микросекундах от момента запуска отладчика и может быть обнулен в любой момент времени. Разность (Delta) представляет фактическое время выполнения участка программы от предыдущей точки останова до текущей, отображаемое в количестве командных циклов и микросекундах (при пошаговом исполнении отображается время выполнения одной строки кода программы на языке Си).

Также в окне Watch Clock отображается текущая тактовая частота микроконтроллера (Clock). Тактовую частоту в окне хронометража можно изменять, что приведет к пересчету времени в микросекундах. Однако это изменение не влияет на текущие установки проекта, где также задана тактовая частота микроконтроллера, а влияет только на расчет времени симуляции.

Управлять процессом отладки можно тремя способами:

- 1) с помощью пунктов меню Run;
- 2) при помощи «горячих клавиш» клавиатуры;
- 3) с использованием кнопок (значков) в окне Watch Values.

Практика показывает, что наиболее удобно для отладки использовать именно кнопки окна Watch Values.

Примечание. Чтобы узнать функцию кнопки (значка), надо поместить

курсор мыши на эту кнопку. На экране появится описание этой функции.

Название и функция основных кнопок управления отладкой (а также соответствующих клавиш клавиатуры) следующие:

- Start Debugger (F9) – запуск отладчика.
- Run/Pause Debugger (F6) – запуск/останов программы в непрерывном (автоматическом) режиме.
- Stop Debugger (Ctrl + F2) – прекращение работы отладчика.
- Step Into (F7) – шаг на одну строку программы.
- Step Over (F8) – шаг через одну строку программы.
- Step Out (Ctrl + F8) – шаг из функции.
- Run To Cursor (F4) – выполнять до курсора.
- Toggle Breakpoint (F5) – поставить/удалить точку останова.
- Команда Step Into позволяет «шагать» по каждой строке исходного текста. При помощи ее можно войти в вызываемую функцию.
- Команда Step Over позволяет «перешагнуть» через вызываемую функцию, не входя внутрь, а выполнив ее как единое целое.

Для завершения работы с отладчиком в любой момент времени следует выбрать из меню команду Run > Stop Debugger. Произойдет возврат в режим редактирования.

Симулятор среды IDE имеет **возможность останавливать выполнение программы в любом месте. Это делается с помощью точек останова (breakpoints).**

4. Перечень используемого оборудования:

4.1 Программа PIC Simulator IDE.

4.2 ПК.

Задание.

1. Создать папку для работы в среде IDE
2. Выполнить построение проекта программы вывода данных в порт В
3. Выполнить тестирование и отладку в IDE первой программы вывода в порт out.c.
4. Выполнить тестирование и отладку второй программы out2.c , в которой состояния выходов порта циклически переключаются, в пошаговом и автоматическом режимах
5. Выполнить тестирование и отладку программы вывода данных в порт В out3.c, в которой между переключениями выходов порта В имеется временная задержка (с временной задержкой)
6. Выполнить тестирование и отладку программы count.c, которая в бесконечном цикле увеличивает число и выводит его в порт В(программы счета с выводом в порт)

7. Выполнить тестирование и отладку программы суммирования чисел `sum.c` в пошаговом и автоматическом режимах с использованием точек останова.

8. Модернизируйте программу `count.c` таким образом, чтобы вывод в порт после каждого увеличения переменной `counter` выполнялся с временной задержкой длительностью 1 с.

9. Модернизируйте программу `count.c` таким образом, чтобы цикл вывода в порт `B` выполнялся только 5 раз.

5. Порядок выполнения.

5.1. Для выполнения лабораторных работ Вам необходимо создать рабочую папку. Затем выберите папку `MPinCS` (сокращение от `Microprocessors in Control Systems`) и раскройте ее. Внутри нее создайте новую папку (с помощью клавиши `F7`) с именем, соответствующим Вашей фамилии (буквы обязательно латинские), например: Для выполнения каждой лабораторной работы в папке `Ivanov` создавайте папку с именем, например, `Lab1` или `Lab5`.

5.2. Запуск среды разработки. Интегрированная среда разработки IDE запускается из стартового меню `Windows`. После запуска программы на экране компьютера Вы увидите рабочий стол среды IDE. В верхней части экрана находится строка заголовка, в которой выводится имя проекта. Под ней расположена строка главного меню. Главное меню содержит обширный набор команд для доступа к функциям среды.

Если по умолчанию установлена опция `Show Page On Startup`, которая открывает **Start Page** (стартовую страницу), то на рабочем столе среды разработки будут открыты следующие окна.

В центре экрана располагается окно `Get Started` с различными сервисными функциями для помощи начинающим пользователям. В левой части экрана расположены команды для создания нового проекта (**New Project...**), открытия существующего проекта (**Open Project...**) и открытия папки примеров (**Open Examples Folder...**). В нижней части экрана находится окно сообщений. Стартовую страницу можно открыть также с помощью команды меню **View > Start Page**.

Рассмотрим методику разработки программного обеспечения в интегрированной среде на примерах простейших программ для микроконтроллера `PIC16F84A`.

5.3. Для построения проекта программы вывода данных в порт `B` иницилируйте процесс создания нового проекта в среде IDE. Для этого подведите курсор мыши к строке команды для создания нового проекта **New**

Project ... и щелкните левой кнопкой.

На экране появится окно мастера создания нового проекта – New Project Wizard. Начальное окно содержит список действий, которые должны быть выполнены для создания нового проекта. Этот процесс разбит на несколько шагов (Steps), которые Вы должны последовательно проделать.

Для продолжения щелкните по кнопке [Next].

5.4. Установка настроек проекта

На этом шаге необходимо определить общую информацию для проекта: выбрать микроконтроллер, его тактовую частоту и, конечно, дать проекту название.

5.4. 1. Очистите строку Project Name и введите новое имя проекта. При выборе имени проекта желательно, чтобы оно имело какую-то смысловую нагрузку. В нашей первой программе будет выполняться вывод данных в порт МК, поэтому можно дать имя этому проекту как out (вывод). Напоминаем, что среда использует только латинские буквы. Использование кириллицы недопустимо! Таким образом, имя проекта – out(расширение .mcpri набирать не нужно!). В строке ProjectName должно быть: **ProjectName: out**

5.4. 2. Для выбора папки хранения проекта щелкните по кнопке [Browse] – обзор. Откроется окно обзора папок. Последовательно выберите и раскройте папку, созданную в п.4.1

5.4. 3. Введите новое имя для микроконтроллера – PIC16F84A. С этой целью щелкните по стрелке в правой части строки Device Name. В раскрывшемся списке PIC-микроконтроллеров выберите PIC16F84A и щелкните по этой строке. В строке Device Name должно появиться:

Device Name: PIC16F84A

5.4. 4. В строке DeviceClock установите тактовую частоту работы микроконтроллера 8 МГц.

5.4. 5. Щелкните по кнопке [Next] для продолжения

В ответ на предложение *Добавление файлов и Подключение библиотек*

5.4. 6. Чтобы разрешить автоматическое открытие окна редактирования проекта выберите команду меню Edit > Project.

5.4. 7. Для выполнения задания для микроконтроллера PIC16F84A должны быть установлены следующие биты конфигурации:

HS – высокочастотный генератор (тактовая частота от 4 до 20 МГц);

WDToff – сторожевой таймер WDT отключен;

CodeProtectionoff – отключена защита кода от чтения, т. е. можно читать и записывать код в микроконтроллер.

5.4. 8. Для окончания построения проекта щелкните по кнопке [Finish].

Результатом работы мастера проектов будет новый проект **out.mcprj**, имя которого вместе с указанием полного пути к нему появится в строке заголовка в верхней части рабочего стола среды. В центре рабочего стола откроется окно редактора кода с пустым исходным файлом для текста программы на языке Си. Исходный файл по умолчанию имеет то же имя, что и проект, т. е. **out.c**.

5.5. Для создания исходного файла наберите в окне редактора кода текст программы вывода данных в порт В микроконтроллера. Текст комментария для экономии времени можно не набирать

Текст исходной программы:

```
/******  
out.c – программа вывода данных в порт В  
*****/  
voidmain( )  
{  
TRISB = 0;           // настроить все линии порта В на вывод  
PORTB = 0x00;       // вывод данных (все нули) в порт В  
PORTB = 0xFF;       // вывод данных (все единицы) в порт В  
}
```

После набора сохраните файл **out.c**, выполнив команду меню **File > Save**.

5.6. Выполните построение проекта **out.mcprj** с помощью команды меню **Build > Build**. В случае успешной компиляции выведете на экран ассемблерный файл **out.asm**, используя команду меню **View > Assembly**. В этом файле в правой части находятся строки исходной программы на языке Си. В средней части размещены команды Ассемблера, сформированные компилятором. Обратите внимание, что при окончании функции **main()** – (правая фигурная скобка **}**) компилятор сгенерировал команду **GOTO \$**. Эта команда формирует бесконечный цикл повторения самой себя, выход из которого возможен только путем сброса микроконтроллера. Таким образом, после окончания главной функции **main()** дальнейшее выполнение программы микроконтроллером фактически прекращается (программа закичивается).

Для закрытия окна ассемблерного файла надо щелкнуть мышью по значку “х” в правой части строки **out.asm**.

5.7. Выполните тестирование и отладку в IDE первой программы вывода в порт **out.c**. С этой целью перейдите в режим отладки с помощью команды меню **Run > Start Debugger**.

Чтобы проверить работу программы **out.c**, нужно следить за состоянием выводов порта В, которые отображаются переменной **PORTB**. Для занесения в

окно наблюдения переменной подведите курсор мыши к строке **Select variable from list** (выбрать переменную из списка) и щелкните по стрелке в правой части. В раскрывшемся списке выберите строку PORTB и щелкните по ней. Для добавления выбранной переменной PORTB в список окна наблюдения щелкните по кнопке [Add]. Строка с переменной PORTB (желтого цвета) появится в окне наблюдения.

5.8. Выполните программу out.c в пошаговом режиме Step Over, используя соответствующую кнопку управления отладкой или клавишу F8. При каждом шаге синяя полоса будет перемещаться по тексту программы. Наблюдайте за значением переменной PORTB на каждом шаге. Выполнение программы out.c прекращается при достижении правой фигурной скобки функции main(). Программа закичивается с помощью команды GOTO \$, которую сгенерировал компилятор (смотреть файл out.asm).

5.9. Выполните сброс микроконтроллера путем повторного запуска отладчика командой **Run > Start Debugger** или щелчком по соответствующей кнопке управления отладкой в окне Watch Values. Это нужно для возможности нового выполнения программы

Значения переменных в окне наблюдения по умолчанию отображаются в десятичном формате. В программе out.c значения PORTB записаны в шестнадцатеричном формате. Для изменения формата отображения щелкните два раза по строке PORTB. В раскрывшемся окне редактирования **Edit Value PORTB** поставьте флажок в окошке HEX. Затем щелкните по кнопке [OK] для подтверждения выбора и закрытия окна редактирования.

5.10. Выполните вновь программу out.c в пошаговом режиме **Step Over**. Убедитесь, что формат отображения PORTB стал шестнадцатеричным.

5.11. Завершите работу с отладчиком командой меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования.

5.12. Закройте проект с помощью команды **Project > Close Project**.

5.13. Выполните разработку второй программы out2.c с циклическим переключением состояния выходов порта программы. Для этого:

5.14.1. С помощью команды **NewProject...** запустите **NewProjectWizard**. Создайте новый проект с именем out2 в папке Lab1, микроконтроллер PIC16F84A, тактовая частота 8 МГц.

5.14.2. Наберите текст исходной программы out2.c и сохраните его в папке.

Текст исходной программы:

```
/*  
out2.c – вторая программа вывода данных в порт В  
*/
```

```

voidmain( )
{
TRISB = 0;           // настроить линии порта В на вывод
while(1)            // бесконечный цикл повторения
{
PORTB = 0x00;       // вывод нулей в порт В
PORTB = 0xFF;       // вывод единиц в порт В
}
}

```

5.14.3. Выполните построение проекта. При отсутствии ошибок построения проекта переключите mikroC PRO в режим отладки программы с помощью симулятора. Для тестирования программы занесите в окно наблюдения Watch Values имя порта PORTB из исходной программы. Установите HEX-формат отображения PORTB.

5.14.4. Выполните программу в пошаговом режиме Step Over. После нескольких шагов убедитесь, что состояния выходов порта В циклически переключаются.

Можно узнать время, затраченное на выполнение каждого оператора в строке программы по секундомеру, который находится в окне хронометража Watch Clock. В строке Delta показывается количество командных циклов и время в микросекундах (us), затраченных на выполнение одной строки исходной программы. Продолжайте выполнять программу в пошаговом режиме Step Over и наблюдайте за показаниями в окне Watch Clock. При каждом шаге показания секундомера Stopwatch будут увеличиваться. По значениям в строке Delta следует, что оператор вывода в порт (строка программы) выполняется за время – 1 мкс.

5.14.5. Выполните сброс МК с помощью команды **Run > Start Debugger** или щелчком по соответствующей кнопке управления в окне Watch Values.

5.14.6. Запустите программу в автоматическом режиме с помощью кнопки управления **Run/Pause Debugger** или нажатием на клавишу F6. При работе программы в автоматическом режиме значения переменной PORTB не обновляются, а в окне хронометража идет отсчет времени от момента запуска.

5.14.7. Остановите выполнение программы повторным щелчком по кнопке управления **Run/Pause Debugger** или нажатием на клавишу F6. Зеленая стрелка слева от текста программы покажет номер строки, на которой произошел останов выполнения. Значение переменной PORTB (красного цвета) обновится. В окне Watch Clock будет отображаться общее время выполнения программы в автоматическом режиме от момента запуска.

5.14.7. Завершите работу с отладчиком, для этого выполните команду

меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования.

5.14.8. Закройте проект с помощью команды **Project > Close Project**.

5.15. Выполнить тестирование и отладку программы вывода данных в порт с временной задержкой out3.c.

Из результатов исследования программы out2.c следует, что вывод в порт (одна строка программы) выполняется за 1 мкс. Очевидно, что, если бы к порту были присоединены светодиоды, то их переключение было бы незаметно. Следовательно, необходимо ввести временную задержку между переключениями состояний выводов порта.

Создайте новый проект с именем out3 и поместите его в папку Lab1, микроконтроллер PIC16F84A, тактовая частота 8 МГц. Наберите текст программы в файле out3.c и сохраните его в папке.

Текст исходной программы:

```
/******  
out3.c – программа вывода данных в порт В с временной задержкой  
*****/
```

```
voidmain( )  
{  
  TRISB = 0;  
  while(1)  
  {  
    PORTB = 0x00;  
    Delay_ms(500);    // временная задержка на 500 мс  
    PORTB = 0xFF;  
    Delay_ms(500);    // временная задержка на 500 мс  
  }  
}
```

В программе out3.c для реализации временной задержки используется встроенная функция компилятора:

Delay_ms (time_in_ms),

Выполните построение проекта, а затем перейдите в режим отладки.

Для тестирования программы занесите в окно наблюдения Watch Values имя порта PORTB из исходной программы.

Выполните программу в пошаговом режиме Step Over. По показаниям в строке Delta окна хронометража убедитесь, что время задержки равно заданному в программе и реализовано компилятором с высокой точностью.

Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования.

В заключение закройте проект с помощью команды **Project > Close Project**.

5.16. Выполнить тестирование и отладку программы count.c, которая в бесконечном цикле увеличивает число и выводит его в порт В микроконтроллера. Для этого

создайте новый проект с именем count и поместите его в папку Lab1, микроконтроллер PIC16F84A, тактовая частота 8 МГц. Наберите текст программы в файле count.c и сохраните его.

Текст исходной программы:

```
/******  
count.c – программа счета с выводом результата в порт В  
*****/  
char counter; // объявление однобайтной переменной-счетчика  
void main( )  
{  
  TRISB = 0;  
  counter = 0;  
  while(1) // бесконечный цикл счета и вывода в порт  
  {  
    PORTB = counter;  
    counter = counter + 1;  
  }  
}
```

Выполните построение проекта, а затем перейдите в режим отладки.

Для тестирования программы занесите в окно наблюдения имена переменных counter и порта PORTB из исходной программы.

Выполните программу в пошаговом режиме Step Over, наблюдая за изменением значений переменных counter и порта PORTB. Убедитесь в правильности работы программы, сделав примерно 10 шагов.

Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования.

Закройте проект с помощью команды **Project > Close Project**.

5.17 Для исследования программы суммирования чисел sum.c. создайте новый проект с именем sum и поместите его в папку Lab1, микроконтроллер PIC16F84A, тактовая частота 8 МГц. Наберите текст программы в файле sum.c и сохраните его.

Текст исходной программы:

```
/******  
sum.c – программа вычисления суммы целых чисел от 1 до 10 и вывода  
результата в порт B  
*****/  
charsum, i; // объявление однобайтных переменных  
void main()  
{  
    TRISB = 0;  
    sum = 0;  
    for (i = 1; i <= 10; i++) // цикл суммирования чисел  
    {  
        sum = sum + i;  
    }  
    PORTB = sum; // вывод результата суммирования в порт B }
```

Выполните построение проекта, а затем перейдите в режим отладки.

Для тестирования программы занесите в окно наблюдения имена переменных `i`, `sum`, `PORTB` из исходной программы.

Выполните программу `sum.c` до конца в пошаговом режиме `Step Over`. Наблюдайте за изменением значений переменных `i`, `sum`, `PORTB`. Убедитесь, что цикл `for` выполняется 10 раз.

5.15. Установите, а затем удалите точки останова в произвольных местах программы с использованием следующих способов

5.15.1. Подвести курсор мыши к нужной строке программы и щелкнуть левой кнопкой. Появится мигающая линия, означающая, что данная строка выбрана. Затем подведите курсор к кнопке управления отладкой `Toggle Breakpoint` в окне `Watch Values` и щелкните левой кнопкой (можно вместо этого нажать клавишу `F5`). Выбранная строка программы окрасится красным цветом, что означает – на установку точки останова. Для отмены точки останова подвести опять курсор к нужной строке программы и щелкнуть левой кнопкой. Затем надо подвести курсор к кнопке управления `Toggle Breakpoint` и щелкнуть левой кнопкой (можно вместо этого нажать клавишу `F5`).

5.15.2. Щелкнуть левой кнопкой мыши по маркеру (синему кружку) в левой части строки программы. Выбранная строка окрасится красным цветом, а маркер поменяет свой цвет на красный. Для отмены точки останова щелкнуть левой кнопкой мыши по маркеру красного цвета выделенной строки текста программы

Предположим, что в программе `sum.c` нас интересует результат суммирования, выполненный оператором цикла `for`. Кроме того, нужно узнать

время, затраченное на выполнение программы от момента запуска до конца цикла `for`. С этой целью установите точку останова на строке текста программы:

```
PORTB = sum;
```

Выполните сброс микроконтроллера командой **Run > Start Debugger** (клавиша **F9**). Затем запустите программу на выполнение в автоматическом режиме при помощи кнопки управления **Run/Pause Debugger** или клавиши **F6**.

Выполнение программы остановится на строке `PORTB = sum`, которая примет синий цвет. Результатом суммирования в цикле `for` будет значение `sum = 55`. Время выполнения программы до точки останова будет 50 мкс.

Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования.

В заключение закройте проект с помощью команды **Project > Close Project**.

5.16. Для разработки программы, в которой вывод в порт после каждого увеличения переменной `counter` выполнялся с временной задержкой длительностью 1 с.

5.16.1. Создайте новый проект с именем `count2` и поместите его в папку `Lab1`, выберите МК `PIC16F84A`, частота 8 МГц.

5.16.2. Запишите текст программы в файл `count2.c` и сохраните его

5.16.3. Выполните построение проекта (команда **Build > Build.**)

5.16.4. Перейдите в режим отладки (команда **Run > Start Debugger**).

5.16.5. Для тестирования программы занесите в окно наблюдения имена переменных `counter` и порта `PORTB` из исходной программы

5.16.6. Выполните программу в пошаговом режиме `StepOver`. Наблюдайте за изменением содержимого переменных `counter` и `PORTB`. При каких значениях `counter` и `PORTB` прекращается цикл вывода?

5.16.7. Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования.

5.16.8. Закройте проект с помощью команды **Project > Close Project**.

5.17. Для разработки программы, `count.c`, в которой цикл вывода в порт будет выполняться только 5 раз.

5.17.1. Создайте новый проект с именем `count3` в папке `Lab1`. Выберите МК `PIC16F84A`, тактовая частота 8 МГц

5.17.2. Запишите текст программы в файл `count3.c` и сохраните его

5.17.3. Выполните построение проекта (команда **Build > Build.**)

5.17.4. Перейдите в режим отладки (команда **Run > Start Debugger**).

5.17.5. Для тестирования программы занесите в окно наблюдения имена

переменных counter и порта PORTB из исходной программы

5.17.6. Выполните программу в пошаговом режиме Step Over. Наблюдайте за изменением содержимого переменных counter и PORTB. При каких значениях counter и PORTB прекращается цикл вывода?

5.17.7 Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования.

5.17.8. Закройте проект с помощью команды **Project > Close Project**.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- тексты программ к заданиям для самостоятельной работы (комментарии в программах обязательны!)
- выводы по проведенным исследованиям
- ответы на контрольные вопросы

7. Контрольные вопросы

1. Что такое IDE?
2. Какие функции выполняет IDE для PIC?
3. Поясните процесс разработки программного обеспечения в среде для PIC-микроконтроллеров.
4. Что такое проект в IDE?
5. На каком этапе осуществляется выбор микроконтроллера?
6. Какие окна используются в симуляторе для наблюдения за ходом выполнения программы?
7. Как можно осуществить пошаговое выполнение программы?
8. Что такое точка останова?

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.– М.: Издательский центр «Академия», 2019г..
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021
3. Компиляторы для PIC-контроллеров. – 2017. – Режим доступа: <http://www.microchip.ru>.
4. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ЛАБОРАТОРНАЯ РАБОТА №6
ИССЛЕДОВАНИЕ ОСНОВНЫХ ОПЕРАЦИЙ ЯЗЫКА
ПРОГРАММИРОВАНИЯ СИ

Цель работы: Изучить операции языка программирования Си: арифметические; присваивания; логические; отношения и поразрядные. Практически исследовать выполнение этих операций в программах для микроконтроллеров семейства PIC16.

2. Время выполнения работы-4час

3. Краткие теоретические сведения

Программа на языке Си состоит из выражений. Выражение – это последовательность операндов, операций и символов-разделителей. Основные операции в Си: арифметические, присваивания, отношения, логические, поразрядные. Для обозначения этих операций используются специальные символы (значки).

Арифметические операции

Язык Си для PIC-микроконтроллеров включает стандартный набор арифметических операций, обозначаемых значками: сложение “ + ”, вычитание “ – “, умножение “ * ”, деление “ / “, которые не требуют особого пояснения. Специфичными являются операции определения остатка от деления, а также инкремента и декремента.

Операцию определения остатка от деления, обозначаемую значком “ % “, поясняет следующий пример:

```
inta = 5, b = 2, d;  
d = a % b;          // d = 1 – остаток от деления 5 / 2  
d = b % a;          // d = 2
```

Следует отметить, что операция определения остатка от деления применима только к целым числам.

Операции инкремента (обозначается как “ ++ “) и декремента (обозначается как “ -- “) могут применяться только к переменным. Существуют две формы их записи: префиксная, когда операнд располагается справа от знака операции (например, ++i, --j), и постфиксная, когда операнд располагается слева от знака операции (i++, j--).

В префиксной форме (для инкремента) сначала выполняется увеличение операнда на 1, и увеличенное значение используется в выражении. В постфиксной форме (для инкремента) сначала используется в выражении значение операнда и только после этого его значение увеличивается на 1. Например:

```
int a = 0, b = 1, d;
d = a++;    // d = 0, a = 1
d = ++a;    // d = 2, a = 2
d = ++b;    // d = 2, b = 2
```

Операции присваивания

Операция присваивания в языке Си обозначается как “=”.

Обычно она используется в виде **переменная = выражение**;

При выполнении операции присваивания переменная получает значение выражения. Выражение может быть одиночной константой или сложной комбинацией переменных, операторов и констант. Например:

```
x = 10;
x = x + y;
x = x + y + 50;
```

При программировании часто используются операторы вида

```
x = x + 4
x = x - y;
и т. п.
```

В этих операторах переменная, которой присваивается результат выражения, является также первым операндом. Для выполнения подобных операций язык `microC` предлагает составные операторы присваивания, которые объединяют простые арифметические операции с присваиванием. В таблице 1 приведены арифметические составные операторы присваивания.

Таблица 1 – Арифметические составные операторы присваивания

Оператор присваивания	Длинная форма	Пример
<code>x += y;</code>	<code>x = x + y;</code>	<code>x += 12;</code>
<code>x -= y;</code>	<code>x = x - y;</code>	<code>x -= 34 + y;</code>
<code>x *= y;</code>	<code>x = x * y;</code>	<code>x *= 10;</code>
<code>x /= y;</code>	<code>x = x / y;</code>	<code>x /= 5;</code>
<code>x %= y;</code>	<code>x = x % y;</code>	<code>x %=2;</code>

*Примечание. В составном операторе присваивания между знаком арифметической операции (+, -, *, /, %) и знаком присваивания “=” пробел не допускается!*

В данной работе будет исследоваться программа `arifm2.c`, в которой используются арифметические составные операторы присваивания для целых чисел и чисел с плавающей точкой:

Операции отношения и логические

Операции отношения применяются для вычисления соотношений между

операндами. Логические операции, используя правила логики, также возвращают соотношения между операндами.

В формальной логике ключевым понятием являются ЛОЖЬ и ИСТИНА. В языке Си лжи соответствует 0, а истине – любое значение, отличное от 0. Выражения, использующие операции отношения или логические, возвращают 0 для лжи и 1 для истины. В таблице 2 приведены обозначения логических и операций отношения, применяемых в языке mikroC.

Таблица 2– Логические операции и операции отношения

Знак операции	Выполняемое действие	Пример
&&	Логическое И	if (k > 1 && k < 10)
	Логическое ИЛИ	if (c == 0 c == 9)
!	Логическое НЕ	if (!(c > 1 && c < 9))
<	Меньше	if (j < 0)
<=	Меньше или равно	if (j <= 0)
>	Больше	if (j > 10)
>=	Больше или равно	if (x >= 8.2)
==	Равно	if (c == b)
!=	Не равно	if (c != b)

В данной работе будет исследоваться программа logic.c, в которой используются операции отношения и логические для целых чисел:

Поразрядные операции

В языке Си широко используются поразрядные (побитовые) операции, с помощью которых можно выполнять тестирование, установку, сброс, инвертирование и сдвиг отдельных битов операндов. Поразрядные операции могут выполняться только с целыми типами данных, т. е. char, int, long. В таблице 3 приведены поразрядные операции языка mikroC.

Таблица.3 – Поразрядные операции в языке Си

Знак операции	Выполняемое действие	Пример
&	Поразрядное И	i & 0x25
	Поразрядное ИЛИ	j 64
^	Поразрядное исключающее ИЛИ	k ^ 0x0F
~	Поразрядное НЕ (инверсия)	~ m
<<	Поразрядный сдвиг влево	i << 2
>>	Поразрядный сдвиг вправо	j >> 3

В данной работе будет исследоваться программа bit_op.c, в которой используются различные поразрядные операции:

Поразрядные составные операторы присваивания

Для сокращения текста программ в Си широко используются поразрядные составные операторы присваивания, которые являются комбинацией поразрядных операций и присваивания. В таблице 4 приведены составные поразрядные операции присваивания

Таблица.4 – Составные поразрядные операторы присваивания

Оператор	Длинная форма	Пример
$x \&= y;$	$x = x \& y;$	$i \&= 0x0F;$
$x = y;$	$x = x y;$	$j = 64;$
$x \wedge= y;$	$x = x \wedge y;$	$k \wedge= 0x80;$
$x \ll= n;$	$x = x \ll n;$	$j \ll= 2;$
$x \gg= n;$	$x = x \gg n;$	$k \gg= 3;$

Примечание. В составных операторах присваивания между знаком поразрядной операции (&, |, ^, >>, <<) и знаком присваивания “=” пробел не допускается!

В данной работе будет исследоваться программа bit_or2.c, в которой используются различные составные поразрядные операторы присваивания:

4. Перечень используемого оборудования:

4.1 Программа MPLAB IDE

4.2 ПК.

4.3. Платформа Freeduino MaxSerial

Задание.

1. Создать папку для работы в среде IDE
2. Выполнить построение проекта программы arifm.c, первой программы для исследования простейших арифметических операций,
3. Выполнить тестирование и отладку в IDE первой программы arifm.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных
4. Выполнить построение проекта второй программы arifm2.c, для исследования арифметических операций в которой используются арифметические составные операторы присваивания для целых чисел и чисел с плавающей точкой.
5. Выполнить тестирование и отладку программы arifm2.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных.
6. Выполнить построение проекта logic.c первой программы для исследования операций отношения и логических
7. Выполнить тестирование и отладку программы logic.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных .
8. Выполнить построение проекта программы bit_or.c, в которой

используются различные поразрядные операции:

9. Выполнить тестирование и отладку программы bit_or.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных .

10. Выполнить построение проекта программы bit_or2.c, в которой используются составные поразрядные операторы присваивания

11. Выполнить тестирование и отладку программы bit_or2.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных

5. Порядок выполнения.

Исследование арифметических операций

5.1. Создайте новый проект с именем arifm в папке Lab2. Выберите МК PIC16F84A, тактовая частота 8 МГц.

5.2. Наберите текст программы в файле arifm.c и сохраните его.

/*
arifm.c – первая программа для исследования арифметических операций

```
*****/  
inti, k = 6, n, m; // объявления переменных  
void main( )  
{  
i = 10 * (k++); // i = , k =  
  
k--; // k =  
i = 10 * (++k); // i = , k =  
k--; // k =  
n = i / k; // n =  
m = i % k; // m = }
```

5.3. Выполните построение проекта и затем перейдите в режим отладки. Для тестирования программы и занесите в окно наблюдения Watch Values имена переменных i, j, k, n, m из исходной программы.

Примечание. 1. Если после запуска отладчика на экране будет отсутствовать окно наблюдения Watch Values, то его можно открыть с помощью команды меню **View > Debug Window > Watch Window** или нажатием комбинации клавиш **Shift + F5**. 2. Формат отображения данных в окне наблюдения будет по умолчанию десятичный.

5.4. Исследуйте работу программы arifm.c в пошаговом режиме Step Over. После каждого шага наблюдайте за содержимым переменных в окне Watch Values и записывайте эти значения в текст комментария программы. Эти

значения необходимо будет привести в отчете по данной работе. Объясните полученные результаты выполнения операторов программы.

5.5. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования.

5.6. Сохраните файл arifm.c с комментариями, полученными в результате исследования программы, с помощью команды меню **File > Save**.

5.7. Закройте проект с помощью команды **Project > Close Project**.

5.8. Создайте новый проект с именем arifm2 в папке Lab2, МК PIC16F84A, частота 8 МГц.

5.9. Наберите текст программы в файле arifm2.c и сохраните его.

/******

arifm2.c – вторая программа для исследования арифметических операций

```
.....  
/ int i = 55, j = 66;  
float x = 2.5, y = 4.6;  
void main( )  
{  
    i += j;           // i =  
    j -= 6;          // j =  
    i *= 4;          // i =  
    j /= 3;          // j =  
    i %= 2;          // i =  
    x += y;          // x =  
    y -= 4.0;        // y =  
    x *= 4.0;        // x =  
    y /= 3.0;        // y =  
}
```

5.10. Выполните построение проекта, а затем перейдите в режим отладки. Для тестирования программы занесите в окно наблюдения Watch Values имена переменных i, j, x, y из исходной программы

5.11. Исследуйте работу программы arifm2.c в пошаговом режиме Step Over. После каждого шага наблюдайте за содержимым переменных в окне Watch Values и записывайте эти значения в текст комментария программы. Эти значения необходимо будет привести в отчете по данной работе. Объясните полученные результаты выполнения операторов программы.

5.12. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования. Сохраните файл arifm2.c с комментариями, полученными в

результате исследования программы, используя команду

File > Save.

5.13. Закройте проект с помощью команды **Project > Close Project.**

Исследование операций отношения и логических

5.14. Создайте новый проект с именем logic в папке Lab2, МК PIC16F84A, частота 8 МГц.

5.15. Наберите текст программы в файле logic.c и сохраните его
/*****
logic.c – первая программа для исследования операций отношения и логических

```
.....  
/ char var1 = 10, var2 = 20;  
char res1, res2, res3, res4, res5, res6, res7, res8;  
void main( )  
{  
    res1 = var1 > var2;           // res1 =  
    res2 = var1 < var2;           // res2 =  
    res3 = var1 == var2;          // res3 =  
    res4 = var1 != var2;          // res4 =  
    res5 = var1 && var2;           // res5 =  
    res6 = var1 || var2;          // res6 =  
    res7 = ! var1;                // res7 =  
res8 = !var2;                    // res8 =  
}
```

5.16. Выполните построение проекта, а затем перейдите в режим отладки. Для тестирования программы занесите в окно наблюдения Watch Values имена переменных var1, var2, res1, res2, res3, res4, res5, res6, res7, res8 из исходной программы.

5.17. Исследуйте работу программы logic.c в пошаговом режиме Step Over. После каждого шага наблюдайте за содержимым переменных в окне Watch Values и записывайте эти значения в текст комментария программы. Эти значения необходимо будет привести в отчете по данной работе. Объясните полученные результаты выполнения операторов программы.

5.18. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования. Сохраните файл logic.c с комментариями, полученными в результате исследования программы.

5.19. Закройте проект с помощью команды **Project > Close Project.**

Исследование поразрядных операций

5.20. Создайте новый проект с именем bit_or в папке Lab2, МК

PIC16F84A, частота 8 МГц.

5.21. Наберите текст программы в файле bit_or.c и сохраните его.

```
/******  
bit_or.c – первая программа для исследования поразрядных операций  
*****/  
char i = 0x0F, j = 0x1A, k1, k2, k3, k4, k5, k6, k7;  
void main( )  
{  
k1 = i & j;          // k1 =  
k2 = i | j;          // k2 =  
k3 = i ^ j;          // k3 =  
k4 = ~ i;            // k4 =  
k5 = ~ j;            // k5 =  
k6 = i << 2;         // k6 =  
k7 = j >> 3;         // k7 =  
}
```

5.22. Выполните построение проекта, а затем перейдите в режим отладки. Для тестирования программы занесите в окно наблюдения Watch Values имена переменных k1, k2, k3, k4, k5, k6, k7 из исходной программы. Установите двоичный формат отображения значений переменных.

5.23. Исследуйте работу программы bit_or.c в пошаговом режиме Step Over. После каждого шага наблюдайте за содержимым переменных в окне Watch Values и записывайте эти значения в текст комментария программы. Эти значения необходимо будет привести в отчете по данной работе. Объясните полученные результаты выполнения операторов программы

5.24. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования. Сохраните файл bit_or.c с комментариями, полученными в результате исследования программы.

5.25. Закройте проект с помощью команды **Project > Close Project**.

5.26. создайте новый проект с именем bit_or2 папке Lab2, МК PIC16F84A, частота 8 МГц.

5.27. Наберите текст программы в файле bit_or2.c и сохраните его.

```
/******  
bit_or2.c – вторая программа для исследования поразрядных составных  
операций присваивания  
.....  
/ char i, j, k;  
void main( )
```

```

{
i = 0xFF;           // i =
i &= 0x0F;         // i =
i |= 0x31;         // i =
i ^= 0x94;         // i =
j = 0xFF;          // j =
    /* установить код 0110 в старшей тетраде переменной j */
j &= 0x0F;         // j =
j |= 0x60;         // j =
    /* инвертировать 7-й разряд в переменной j */
j ^= 0x80;         // j =
    /* инвертировать 3-й разряд в переменной j */
j ^= 0x04;         // j =

k = 0xF5;          // k =
    /* сдвиги беззнаковой переменной k */
k >>= 1;          // k =
k >>= 2;          // k =
k <<= 1;          // k =
k <<= 2;          // k =

}

```

5.28. Выполните построение проекта, а затем перейдите в режим отладки. Для тестирования программы занесите в окно наблюдения Watch Values имена переменных *i*, *j*, *k* из исходной программы. Установите двоичный формат отображения значений переменных

5.29. Исследуйте работу программы `bit_op2.c` в пошаговом режиме Step Over. После каждого шага наблюдайте за содержимым переменных в окне Watch Values и записывайте эти значения в текст комментария программы. Эти значения необходимо будет привести в отчете по данной работе. Объясните полученные результаты выполнения операторов программы.

5.30. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом произойдет возврат в режим редактирования. Сохраните файл `bit_op2.c` с комментариями, полученными в результате исследования программы.

5.31. Закройте проект с помощью команды **Project > Close Project**.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.

- перечень используемого оборудования
- тексты всех исследуемых программ, включая задания для самостоятельной работы (комментарии в программах обязательны!).
- выводы по проведенным исследованиям
- ответы на контрольные вопросы.

7. Контрольные вопросы

1. Какие арифметические операции имеются в языке Си?
2. Как выполняются операции преинкремента и постинкремента?

Приведите примеры.

3. Как выполняется операция определения остатка от деления? Приведите пример.

4. Как обозначаются и выполняются составные арифметические операции присваивания?

5. Как кодируются логические понятия ЛОЖЬ и ИСТИНА в Си?

6. Какие логические операции имеются в языке Си?

7. Какие операции отношения имеются в языке Си?

8. Какие имеются в языке Си поразрядные операции?

9. Как обозначаются и выполняются составные поразрядные операции присваивания?

7. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.– М.: Издательский центр «Академия», 2019г..

2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021

3. Компиляторы для PIC-контроллеров. – 2017. – Режим доступа: <http://www.microchip.ru>.

4. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ИССЛЕДОВАНИЕ ОПЕРАТОРОВ ВЫБОРА ДЛЯ УПРАВЛЕНИЯ ПРОГРАММОЙ В ЯЗЫКЕ СИ

1. Цель работы: Изучить операторы выбора для управления программой в языке Си. Практически исследовать выполнение этих операторов в программах для микроконтроллеров семейства PIC16

2. Время выполнения работы-4час

3.Краткие теоретические сведения

Операторы управления программой – это средства, с помощью которых можно изменять порядок выполнения программы. Они позволяют выполнять ветвление, циклическое повторение одного или нескольких операторов, передачу управления в нужное место кода программы.

Язык Си предоставляет три категории операторов управления программой:

- 1) операторы выбора –if и switch;
- 2) операторы цикла –while, do ... while и for;
- 3) операторы перехода –break, continue и goto.

В данной лабораторной работе Вы будете изучать и исследовать операторы выбора.

Язык Си поддерживает два типа операторов выбора: if (если) и switch (переключатель). Эти операторы позволяют проверять выполнение определенных условий и выбирать возможное продолжение вычислительного процесса. Возможны следующие конструкции этих операторов:

- 1) оператор if единственного выбора;
- 2) оператор if ... else двойного выбора;
- 3) оператор if ... else if ... else множественного выбора
- 4) оператор switch множественного выбора

Оператор if единственного выбора для одного выполняемого оператора имеет вид

```
if (условие)
    оператор;
```

для блока выполняемых операторов:

```
if (условие)
{   оператор1;
    .....
    операторN;
}
```

Программа вычисляет условие, заключенное в круглых скобках. Если оно истинно, то выполняется оператор (или блок операторов, заключенный в фигурных скобках). Если условие ложно, то оператор (или блок) не выполняется.

Оператор `if ... else` (если ... то) двойного выбора обеспечивает две альтернативы продолжения выполнения программы. Выбор осуществляется, исходя из проверяемого условия.

Общий вид оператора `if ... else` двойного выбора для одного выполняемого оператора:

```
if (условие)
    оператор1;
else
    оператор2;
```

Вместо одиночных операторов 1 и 2 могут быть блоки операторов, заключенные в фигурные скобки.

В случае истинности условия выполняется оператор1, в противном случае – оператор2.

Язык `microC` позволяет использовать вложенные операторы `if ... else` для реализации множественного выбора.

Общий вид оператора `if ... elseif ... else` множественного выбора:

```
if (условие1)
    оператор1;
elseif (условие2)
    оператор2;
.....
elseif (условиеN)
    операторN;
else
    оператор(N+1);
```

Последняя `else`-часть в операторе `if ... else` может и отсутствовать.

Оператор `if ... else if ... else` множественного выбора выполняет серию последовательных проверок до тех пор, пока не будет установлено следующее:

Одно из условий в `if`-части или в частях `else if` является истиной. В этом случае выполняются соответствующие ему операторы.

Ни одно из вложенных условий не является истиной. Программа выполнит операторы в последней else-части, если она имеется.

Оператор switch (можно перевести как переключатель) используется для выбора одного варианта из многих. Он проверяет, совпадает ли значение выражения с одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы.

Общий вид оператора switch:

```
switch (выражение)
{
  case константа1:
    оператор1;
  break;
  case константа2:
    оператор2;
  break;
  .....
  case константаN:
    операторN;
  break;
  default:
    оператор(N+1);
}
```

Оператор switch выполняется так. Сначала вычисляется выражение, стоящее в скобках после ключевого слова switch. Вычисленное значение сравнивается со значением констант: константа1, константа2, ..., константаN. При совпадении вычисленного значения с некоторой константой выполняется соответствующий ей оператор. Затем управление передается оператору break (прервать), который производит немедленный выход из оператора switch. Если вычисленное значение не совпадает ни с одной из констант, выполняется оператор в ветви, помеченной default (по умолчанию).

Правила использования оператора switch следующие.

Switch требует выражения целого типа. Это значение может быть константой, переменной или выражением. Оператор switch не работает с типами данных с плавающей точкой.

Значение после каждой метки case должно быть константой.

Язык Си не поддерживает метки case, содержащие диапазон значений. Каждое значение должно указываться с отдельной меткой case.

Необходимо использовать оператор `break` после каждого набора выполняемых операторов. Оператор `break` вызывает продолжение выполнения программы после завершения текущего оператора `switch`. Если не использовать оператор `break`, то выполнение программы продолжится на последующих метках `case`.

Ветвь, помеченная словом `default` (по умолчанию), выполняется тогда, когда ни одна из констант ветвей `case` не подходит. Ветвь `default` не является обязательной и, если она отсутствует, оператор `switch` ничего не выполнит.

Набор операторов в каждой ветви `case` не нужно заключать в фигурные скобки.

Примечание. В связи с тем, что в ветвях, помеченных словом `case`, можно указывать только константу, для анализа принадлежности к диапазону значений следует использовать оператор `if ... else if ... else` множественного выбора.

4. Перечень используемого оборудования:

4.1 Программа MPLAB IDE

4.2 ПК.

4.3. Платформа Freeduino MaxSerial

Задание.

1. Создать папку для работы в среде IDE
2. Выполнить построение проекта программы `if1.c`, программы в которой используется оператор `if` единственного выбора:
3. Выполнить тестирование и отладку в IDE первой программы вывода в порт `if1.c`, исследуйте работу программы в пошаговом режиме, фиксируя полученные результаты выполнения операторов программы при различных значениях данных на входах порта В, наблюдая работу соответствующих ветвей программы
4. Выполнить построение проекта второй программы `if2.c`, в которой используется оператор `if ... elseif ... else` множественного выбора
5. Выполнить тестирование и отладку программы `if2.c`, исследуйте работу программы в пошаговом режиме, фиксируя полученные результаты выполнения операторов программы при различных значениях данных на входах порта В, наблюдая работу соответствующих ветвей программы
6. Выполнить построение проекта `switch.c` программы, в которой используется оператор `switch`.
7. Выполнить тестирование и отладку программы `switch.c` исследуйте работу программы в пошаговом режиме, фиксируя полученные результаты выполнения операторов программы при различных значениях данных на входах порта В, наблюдая работу соответствующих ветвей программы

8. Разработайте программутаx.c, которая определяет большее из трех введенных чисел. В программе числа вводятся последовательно из порта В микроконтроллера PIC16F877: сначала – первое, затем – второе, а потом – третье. Для хранения каждого введенного числа надо предусмотреть отдельную переменную. Найденное большее число выводится в порт С.

9. Выполнить тестирование и отладку программытаx.c, исследуйте работу программы в пошаговом режиме, фиксируя значения переменных.

5. Порядок выполнения.

Исследование оператора if единственного выбора.

5.1. Создайте новый проект с именем if1 в папке f:\...\Lab3. Выберите микроконтроллер PIC16F877, тактовая частота 8 МГц.

5.2. Наберите текст исходной программы if1.c и сохраните его в папке (с помощью команды меню File > Save).

```
/******  
if1.c – программа для исследования оператора if единственного выбора.  
Микроконтроллер PIC16F877  
*****/
```

```
charin_port; // переменная для хранения вводимых данных  
              // из порта В  
  
voidmain( )  
{  
TRISB = 0xFF; // настроить порт В на ввод  
TRISC = 0; // настроить порт С на вывод PORTC  
= 0x00; // вывести нули в порт С  
in_port = PORTB; // вводизпортаВ  
if (in_port >= 0x0F)  
PORTC = in_port; // выводвпорт С  
}
```

В программе вводится число из порта В и записывается в переменную in_port. Если это значение больше или равно значению 0x0F = 00001111, то программа выводит его в порт С. В противном случае программа никаких действий не производит.

5.3. Выполните построение проекта (с помощью команды меню **Build > Build**).

5.4. При отсутствии ошибок построения проекта переключите mikroC PRO в режим отладки (с помощью команды меню **Run > Start Debugger**).

5.5. Проведите исследование работы программы if1.c в следующей последовательности.

5.5.1. Для тестирования программы занесите в окно наблюдения Watch

Values последовательно имена портов PORTB и PORTC, а затем имя переменной in_port.

5.5.2. Для исследования работы программы измените формат отображения всех переменных в окне наблюдения на шестнадцатеричный.

5.5.3. Выполните программу if1.c в пошаговом режиме Step Over путем нажатия на клавишу F8, или щелкая мышью по соответствующему значку управления отладкой. Когда синяя полоса окажется на строке с номером 6 (in_port = PORTB;), поставьте курсор мыши на колонку Value в строке PORTB и дважды щелкните левой кнопкой. Наберите новое число, например, 0x7F, а затем нажмите клавишу Enter. Продолжайте выполнять программу в пошаговом режиме, наблюдая за перемещением синей полосы по ветвям программы. Когда синяя полоса окажется на фигурной скобке (конец функции main), дальнейшее выполнение программы прекратится.

5.5.4. Произведите сброс микроконтроллера с помощью команды меню **Run > Start Debugger**, или щелкнув по соответствующему значку управления отладкой.

5.5.5. Введите новое значение на входах порта В, например, 0x03. Выполните программу до конца в пошаговом режиме, наблюдая за перемещением синей полосы по ветвям программы. Объясните полученные результаты выполнения операторов программы при различных значениях данных на входах порта В.

5.6. Для завершения работы с отладчиком нужно выполнить команду меню **Run > Stop Debugger**, при этом происходит возврат в режим редактирования.

5.7. Закройте проект с помощью команды **Project > Close Project**.

Исследование оператора if ... else if ... else множественного выбора.

5.8. Создайте новый проект с именем if3 в папке f:\...\Lab3. Выберите микроконтроллер PIC16F877, тактовая частота 8 МГц.

5.9. Наберите текст исходной программы if3.c и сохраните его в папке (с помощью команды меню File > Save).

```
/******
```

```
if3.c – программа исследования оператора if ... elseif ... else  
множественного выбора. Микроконтроллер PIC16F877
```

```
*****/
```

```
char in_port; // переменная для хранения вводимых данных
```

```
// из порта В
```

```
void main( )
```

```
{
```

```

TRISB = 0xFF;           // настроить порт В на ввод
TRISC = 0;              // настроить порт С на вывод   PORTC
=0x00;                 // вывод нулей в порт С
in_port = PORTB;       // ввод данных из порта В
if (in_port > 0 && in_port < 0x0F)
    PORTC = 0x01;       // вывод в порт С значения 0x01
else if (in_port >= 0x0F && in_port < 0x3F)
    PORTC = 0x02;       // вывод в порт С значения 0x02
else if (in_port >= 0x3F && in_port < 0x7F)
    PORTC = 0x03;       // вывод в порт С значения 0x03
else
    PORTC = 0x04;       // вывод в порт С значения 0x04 }
}

```

Программа вводит данные с входов порта В и записывает их в переменную `in_port`. Затем используется оператор `if ... else if ... else` множественного выбора для определения нахождения значения `in_port` в определенном диапазоне:

```

0 < in_port < 0x0F      – диапазон 1
0x0F <= in_port < 0x3F – диапазон 2
0x3F <= in_port < 0x7F – диапазон 3
0x7F <= in_port <= 0xFF – диапазон 4

```

Полученное значение номера диапазона выводится в порт С.

5.10. Выполните построение проекта (с помощью команды меню **Build > Build**).

5.11. При отсутствии ошибок построения проекта переключите `mikroc PRO` в режим отладки (с помощью команды меню **Run > Start Debugger**).

5.12. Проведите исследование работы программы `if3.c` в следующей последовательности.

5.12.1. Для тестирования программы занесите в окно наблюдения `Watch Values` последовательно имена портов `PORTB` и `PORTC`, а затем имя переменной `in_port`.

5.12.2. Для исследования работы программы измените формат отображения всех переменных в окне наблюдения на шестнадцатеричный.

5.12.3. Выполните программу `if3.c` в пошаговом режиме `Step Over` путем нажатия на клавишу `F8`, или щелкая мышью по соответствующему значку управления отладкой. Когда синяя полоса окажется на строке с номером 6

(in_port = PORTB;), поставьте курсор мыши на колонку Value в строке PORTB и дважды щелкните левой кнопкой. Наберите новое число, например, 0x07 (соответствующее диапазону 1), а затем нажмите клавишу Enter. Продолжайте выполнять программу в пошаговом режиме, наблюдая за перемещением синей полосы по ветвям программы. Убедитесь в правильности выполнения программы по значению, выведенному в порт C. Когда синяя полоса окажется на фигурной скобке (конец функции main), дальнейшее выполнение программы прекратится.

5.12.4. Произведите сброс микроконтроллера с помощью команды меню Run > Start Debugger, или щелкнув по соответствующему значку управления отладкой.

5.12.5. Введите новое значение на входах порта B, например, 0x1F (соответствующее диапазону). Выполните программу до конца в пошаговом режиме, наблюдая за перемещением синей полосы по ветвям программы. Убедитесь в правильности выполнения программы по значению, выведенному в порт C.

5.12.6. Далее исследуйте работу программы при значениях 0x5F (диапазон 3) и 0x8F (диапазон 4) на входах порта B. Объясните полученные результаты выполнения операторов программы при различных значениях данных на входах порта B.

5.13. Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**.

5.14 Закройте проект с помощью команды **Project > Close Project**.

Исследование оператора выбора switch.

5.15. Создайте новый проект с именем switch в папке f:\ \Lab3. Выберите микроконтроллер PIC16F877, тактовая частота 8 МГц.

5-16. Наберите текст исходной программы switch.c и сохраните его в папке (с помощью команды меню File > Save)

```
/******  
switch.c – программа исследования оператора выбора switch Микроконтроллер  
PIC16F877  
*****/  
char in_port;           // переменная для хранения введенного кода  
void main( )  
{  
    TRISB = 0xFF;       // настроить порт B на ввод  
    TRISC = 0;          // настроить порт C на вывод  
    PORTC = 0x00;      // вывод нулей в порт C  
    in_port = PORTB;    // ввод данных из порта B
```

```

switch (in_port)
{
case 0xFE:
PORTC = 0x01;           // вывод «вариант 1» в порт Cbreak;
case 0xFD:
PORTC = 0x02;           // вывод «вариант 2» в порт Cbreak;
case 0xFB:
PORTC = 0x03;           // вывод «вариант 3» в порт Cbreak;
case 0xF7:
PORTC = 0x04;           // вывод «вариант 4» в порт Cbreak;
default:
PORTC = 0xFF;          // вывод «неизвестный код» в порт C

}
}

```

В программе `switch.c` предполагается ввод определенного кода из порта В. С помощью оператора `switch` проверяется один из четырех вариантов входного кода. Для каждого варианта кода выполняется вывод соответствующего «сообщения» в порт С. В качестве такого «сообщения» выбрано определенное число

Входной код из порта В	Вариант кода	Код для вывода в порт С
0xFE = 11111110	Вариант 1	0x01 = 00000001
0xFD = 11111101	Вариант 2	0x02 = 00000010
0xFB = 11111011	Вариант 3	0x03 = 00000011
0xF7 = 11110111	Вариант 4	0x04 = 00000100

Если входной код не соответствует ни одному из указанных четырех вариантов, то в порт С выводится код `0xFF = 11111111`.

Входной код представляет собой двоичное число, в котором только один 0. Такой код получается при опросе контактов переключателей (клавиш), присоединенных к выводам порта В. Наличие 0 в определенном разряде кода будет соответствовать замкнутому контакту.

5.17. Выполните построение проекта (с помощью команды меню **Build > Build**).

5.18. При отсутствии ошибок построения проекта переключите mikroC PRO в режим отладки (с помощью команды меню **Run > Start Debugger**).

5.19. Проведите исследование работы программы `switch.c` в следующей последовательности.

5.19.1. Для тестирования программы занесите в окно наблюдения Watch

Values последовательно имена портов PORTB и PORTC, а затем имя переменной in_port.

5.19.2. Для исследования работы программы измените формат отображения всех переменных в окне наблюдения на шестнадцатеричный.

5.19.3. Выполните программу switch.c в пошаговом режиме Step Over путем нажатия на клавишу F8, или щелкая мышью по соответствующему значку управления отладкой. Проверьте работу программы для всех четырех вариантов кода на входах порта В, указанных в таблице. В заключение установите на входах порта В код 0xFF. Объясните полученные результаты выполнения операторов программы.

5.20. Для завершения работы с отладчиком выполните команду меню **Run > Stop Debugger**.

5.21. Закройте проект с помощью команды **Project > Close Project**.

5.22. Разработайте программу, которая определяет большее из трех введенных чисел. В программе числа вводятся последовательно из порта В микроконтроллера PIC16F877: сначала – первое, затем – второе, а потом – третье. Для хранения каждого введенного числа надо предусмотреть отдельную переменную. Найденное большее число выводится в порт С. Разработанную программу назовите max.c.

5.23. Для исследования программы создайте проект с именем max в папке f:\...\Lab3. Выберите PIC16F877, частота 8 МГц. Выполните построение проекта, а затем перейдите в режим отладки.

5.24. Для тестирования разработанной программы занесите в окно наблюдения Watch Values имена портов PORTB и PORTC, а также имена переменных для хранения введенных чисел.

5.25. Для исследования работы программы max.c выполните ее в пошаговом режиме Step Over. Убедитесь в правильности выполнения алгоритма программы.

5.26. Для завершения работы с отладчиком выполните команду меню **Run>StopDebugger**.

5.27. Закройте проект с помощью команды **Project > Close Project**.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- перечень используемого оборудования
- тексты всех исследуемых программ, включая задания для самостоятельной работы (комментарии в программах обязательны!).
- выводы по проведенным исследованиям
- ответы на контрольные вопросы.

7. Контрольные вопросы

1. Какие операторы выбора имеются в языке Си?
2. В каких случаях оператор `switch` не может использоваться вместо оператора `if ... elseif ... else`?
3. Какие функции выполняет оператор `break` в конструкции с оператором `switch`?
4. Может ли в операторе `switch` отсутствовать ветвь `default`? Как в этом случае выполняется программа?

8. Список используемой литературы

1. Келим Ю. М. Вычислительная техника/ Учеб.пособие для студ. СПО.— М.: Издательский центр «Академия», 2019г..
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд.перераб. и доп. —Москва: Издательство Юрайт, 2021
3. Компиляторы для PIC-контроллеров. — 2017. — Режим доступа: <http://www.microchip.ru>.
4. MikroC PRO for PIC. User's manual. — 2017. <http://www.mikroe.com>

ЛАБОРАТОРНАЯ РАБОТА №8

ИССЛЕДОВАНИЕ ОПЕРАТОРОВ ЦИКЛА И ПЕРЕХОДА ДЛЯ УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМ ПРОЦЕССОМ В ЯЗЫКЕ СИ

Цель работы: Изучить операторы цикла и перехода для управления вычислительным процессом в языке программирования Си. Практически исследовать выполнение этих операторов в программах для микроконтроллеров семейства PIC16.

2. Время выполнения работы-4час

3. Краткие теоретические сведения

Операторы управления вычислительным процессом позволяют выполнять ветвление, циклическое повторение одного или нескольких операторов, передачу управления в нужное место кода программы.

Язык Си предоставляет три категории операторов управления программой:

- 1) операторы выбора – это if и switch;
- 2) операторы цикла – это while, do ... while и for;
- 3) операторы перехода – это break, continue и goto.

В данной лабораторной работе Вы будете изучать и исследовать операторы цикла и перехода.

Оператор цикла for

Цикл for является универсальным, поскольку компоненты цикла могут быть произвольными выражениями. Общая форма записи оператора цикла for имеет вид

```
for (инициализация; условие; увеличение/уменьшение переменной
цикла) {
    тело цикла;
}
```

Оператор for имеет три компонента:

Инициализация – это место, где обычно находится оператор присваивания, используемый для установки начального значения переменной цикла.

Условие – это место, где находится выражение, определяющее условие работы цикла.

Увеличение/уменьшение переменной цикла – это место, где определяется характер изменения переменной цикла на каждой итерации, т. е. повторения цикла.

Тело цикла – это оператор или группа операторов, которые будут выполняться в цикле. Цикл `for` работает до тех пор, пока условие истинно. Когда условие становится ложным, выполнение программы продолжается с оператора, следующего за циклом `for`.

Например, в следующей программе осуществляется вывод чисел от 1 до 100 включительно в порт C микроконтроллера:

```
void main( )
{   char x;
  for (x = 1; x <= 100; x++)
  PORTC = x;
}
```

Вариации цикла for.

Важной особенностью цикла `for` является то, что все три компонента цикла являются необязательными.

Например, если оставить все три компонента пустыми, то получим бесконечный цикл:

```
for (; );
```

Кроме того, тело цикла может быть пустым, т. е. не содержать операторов. Такие циклы могут использоваться для получения временных задержек при выполнении программы:

```
for (i = 0; i < 10000; i++);
```

Оператор цикла while

Цикл `while` является разновидностью условного цикла, повторяющегося до тех пор, пока условие выполнено. Таким образом, цикл `while` может не выполняться ни разу, если условие проверки изначально ложно. Форма записи оператора цикла `while` имеет вид

```
while (условие)
{
  блок операторов;
}
```

Пример. Следующий цикл выводит в порт C числа от 1 до 100:

```
int x = 1;
while (x <= 100)
```

```

{
    PORTC = x;
    x++;
}

```

Оператор цикла do...while

В цикле do... while условие повторения проверяется после каждого прохождения тела цикла. Следовательно, цикл do... while выполняется, по крайней мере, один раз.

Форма записи цикла do... while имеет вид

```

do{    блок операторов;
}
while (условие);

```

Пример. Следующий цикл выводит в порт С значения квадратов чисел от 2 до 10:

```

int x = 2;
do
{
    PORTC = x * x;
    x++;
}
while (x <= 10);

```

Оператор перехода break

Оператор break имеет два назначения. Первое – это окончание работы оператора switch. Второе – это принудительное окончание цикла, минуя стандартную проверку условия. Когда оператор break встречается в теле цикла, цикл немедленно заканчивается и выполнение программы переходит на строку, следующую за циклом.

Рассмотрим пример использования оператора break для досрочного прекращения работы цикла for:

```

char in_port, x;
void main( )
{
    TRISB = 0xFF;           // настроить порт В на ввод
    TRISC = 0;             // настроить порт С на вывод
    PORTC = 0;

```

```

in_port = PORTB;           // ввод числа из порта В
for (x = 1; x < 100; x++)
{
if (x == in_port)
break;                    // прекращение цикла вывода
PORTC = x;                // вывод в порт С
} }

```

В этой программе оператор цикла `for` выполняет вывод в порт С переменной `x`, изменяющейся от 0 до 100. Однако, если значение переменной `x` будет равно переменной `in_port`, введенной из порта В, причем это значение меньше 100, то цикл вывода немедленно прекращается.

Оператор перехода `continue`

Работа оператора `continue` чем-то похожа на работу оператора `break`. Но вместо форсированного окончания цикла оператор `continue` переходит к следующей итерации цикла, пропуская оставшийся код тела цикла.

Рассмотрим пример применения оператора `continue` для вывода в порт С чисел от 1 до 10 за исключением числа 5:

```

void main()
{
chark;                    // объявление однобайтной переменной
TRISC = 0;                // настроить порт С на вывод
PORTC = 0;                // вывод нулей в порт С
for (k = 1; k <= 10; k++)
{
if (k == 5)               // если k равно 5, то
continue;                // пропустить вывод в порт С
PORTC = k;                // вывод в порт С
}
}

```

Оператор перехода `goto`

Оператор `goto` (идти к ...) – это безусловный переход на метку. Метка – это идентификатор Си, завершающийся двоеточием. Пример записи оператора `goto`:

```

gotolabel;
.....
.....
label: .....

```

Рассмотрим простейшую программу, где используется оператор перехода `goto`:

```

void main()
{
char x = 1;
loop:
x++;
if (x <= 10)
goto loop;
}

```

В программе реализуется цикл инкремента переменной *x* от 1 до 10 с помощью оператора *goto* и метки *loop*. Очевидно, что такой цикл можно было бы реализовать и с использованием операторов *while* или *for*.

4. Перечень используемого оборудования:

4.1 Программа MPLAB IDE

4.2 ПК.

4.3. Платформа Freeduino MaxSerial

Задание.

1. Ознакомьтесь с краткими теоретическими сведениями
2. Разработайте все варианты программы, которая находит сумму целых чисел от 1 до 15 включительно и выводит полученное значение в порт С микроконтроллера PIC16F877.

Вариант 1. Используйте в программе для организации цикла оператор *for*. Программу назовите *for.c*. Создайте проект с именем *for* в папке *f:\...\Lab4*, выполните его построение и произведите тестирование программы.

Примечание. Для того чтобы при отладке программы можно было занести переменные в окно наблюдения, их нужно объявлять в исходной программе как глобальные, т. е. до функции *main()*.

Вариант 2. Используйте для организации цикла суммирования оператор *while*. Программу назовите *while.c*. Создайте проект с именем *while* в папке *f:\...\Lab4*, выполните его построение и произведите тестирование программы.

Вариант 3. Используйте для организации цикла суммирования оператор *do...while*. Программу назовите *dowhile.c*. Создайте проект с именем *dowhile* в папке *f:\...\Lab4*, выполните его построение и произведите тестирование программы.

Вариант 4. Используйте для организации цикла суммирования оператор *for* (как в варианте 1). Однако при достижении значения числа, равному 10, цикл суммирования должен немедленно прекратиться. Для этой цели примените оператор перехода *break*. Программу назовите *break.c*. Создайте проект с именем *break* в папке *f:\...\Lab4*, выполните его построение и произведите тестирование программы.

Вариант 5. Используйте для организации цикла суммирования оператор *for* (как в варианте 1). Однако для суммирования должны использоваться только четные значения чисел, т. е. 2, 4, 6 и т. п. Для пропуска суммирования

нечетных значений чисел используйте оператор перехода `continue`. Для проверки целого числа, например, `x`, на четность можно использовать арифметическую операцию `x % 2` – определение остатка от деления на 2. Если результат операции будет равен 0, то число `x` – четное, если же результат будет 1, то `x` – нечетное. Программу назовите `continue.c`. Создайте проект с именем `continue` в папке `f:\...\Lab4`, выполните его построение и произведите тестирование программы.

Вариант 6. Используйте для организации цикла суммирования чисел от 1 до 15 оператор перехода `goto`. Программу назовите `goto.c`. Создайте проект с именем `goto` в папке `f:\...\Lab4`, выполните его построение и произведите тестирование программы.

5. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- тексты программ к заданиям всех вариантов программ к заданию для самостоятельной работы (комментарии в программах обязательны!).
- выводы по проведенным исследованиям
- ответы на контрольные вопросы

6. Контрольные вопросы

1. Какие операторы цикла имеются в языке Си?
2. Какие операторы перехода имеются в языке Си?
3. Чем отличаются операторы цикла `while` и `do ... while`?
4. Какими способами можно реализовать бесконечный цикл?
5. В каких случаях удобно использовать оператор `break`?
6. Чем отличается действие оператора `continue` от действия оператора `break`?
7. Как можно применять оператор `goto`?

7. Список используемой литературы

1. Пигарев, Л. А. Микропроцессорные системы автоматического управления [Электронный ресурс]. – Санкт-Петербург: СПбГАУ, 2017.
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021
3. Компиляторы для PIC-контроллеров. – 2017. – Режим доступа: <http://www.microchip.ru>.
4. MikroC PRO for PIC. User's manual. – 2017. <http://www.mikroe.com>

ЛАБОРАТОРНАЯ РАБОТА №9
РАЗРАБОТКА ПРОГРАММЫ УПРАВЛЕНИЯ МИГАЮЩИМ
СВЕТОДИОДОМ НА ЯЗЫКЕ СИ

Цель работы: Получить практические навыки по использованию возможностей среды разработки Arduino IDE для составления программ;

2. Время выполнения работы-4 час

3. Краткие теоретические сведения

Аппаратная платформа Freeduino

Arduino - это микроконтроллерная система, состоящая из платы ввода/вывода с микроконтроллером фирмы Atmel и среды разработки с открытым исходным кодом на основе упрощенного языка программирования C. Freeduino - совместимая с Arduino открытая платформа



Рисунок 1. Модуль SB-Freduino на базе микроконтроллера ATmega328

Микроконтроллер программируется с помощью компьютера и может работать самостоятельно или в сочетании с ПК. К микроконтроллеру могут присоединяться различные аналоговые и цифровые датчики, которые регистрируют состояние окружающей среды и передают данные в микроконтроллер. После обработки данных программа микроконтроллера может вывести информацию на монитор или, например, осуществить управление приводом. Поддержка проектов разработчиков осуществляется за счет готовых программ и библиотек функций среды программирования Arduino.

Микроконтроллер ATmega328 имеет 32 Кбайт программной памяти (из которых 2 Кбайт используются для хранения загрузчика). Кроме того, микросхема имеет 2 Кбайт ОЗУ (SRAM), 1 Кбайт долговременной памяти данных (EEPROM).

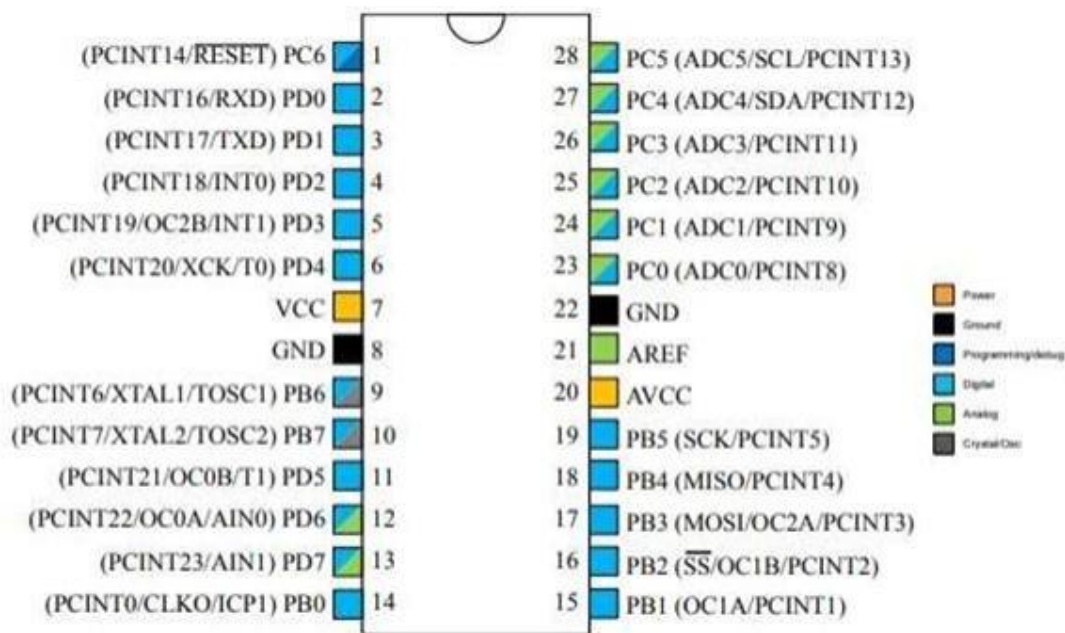


Рисунок 2. Вводы/выводы микроконтроллера ATmega328

Модуль имеет 14 контактов цифрового ввода/вывода, шесть из которых могут быть использованы для вывода сигналов ШИМ, и шесть аналоговых входов АЦП с дискретностью в 1024 значений. Модуль также содержит: кварцевый резонатор на 16 МГц, разъем USB (либо COM в варианте MaxSerial), разъем питания типа Mini-Jack, разъем для подключения внешнего программатора ICSP и кнопку сброса. Каждый из 14 выводов модуля можно использовать как вход или как выход данных. Эти выводы работают с сигналами уровнем 0...5В. Каждый из выводов рассчитан на входной (или выходной) ток до 20 мА и имеет внутренний программно отключаемый подтягивающий резистор сопротивлением 20-50 кОм, который по умолчанию отключен. Выводы 3, 5, 6, 9, 10 и 11 могут работать в режиме выходов сигналов широтно-импульсного модулирования. Если вы хотите подключить внешние устройства к выводам 1 или 2, то вы должны помнить, что эти выводы совмещены с цепями, работающими в режиме обмена по USB/COM каналу. Поэтому в момент записи программы и при других операциях обмена данными с компьютером внешние цепи нужно будет отключать.

Модуль также имеет 6 аналоговых входов (АЦП), каждый из которых обеспечивает 10 разрядное аналого-цифровое преобразование (т.е. различает 1024 значений уровня сигнала). По умолчанию входы настроены на диапазон входного напряжения от 0 до 5В. Можно снижать верхнюю границу этого диапазона при использовании внешнего источника опорного напряжения и соответствующим образом составленной программы.

Микроконтроллер ATmega328 имеет встроенный последовательный

интерфейс UART работающий с сигналами TTL уровней (0...+5В), сигналы которого выведены на контакты 1 (RxD) и 2 (TxD) модуля.

USB версии модулей имеют в своем составе USB конвертор FT232RL фирмы FTDI, который обеспечивает работу через создаваемый автоматически виртуальный COM-порт при подключении модуля к компьютеру. Платформа Freeduino MaxSerial оснащена преобразователем MAX232 для преобразования TTL уровней к стандарту RS232.

Микроконтроллер ATmega328 также поддерживает I2C (TWI) шину и шину SPI. Библиотека Freeduino содержит функции для работы с I2C. Для реализации работы с SPI можно обратиться к фирменной документации на микроконтроллер.

Среда разработки Arduino IDE

Программное обеспечение Arduino с открытым исходным кодом (IDE) позволяет легко писать код и загружать его на борт. Он работает в Windows, Mac OS X и Linux. Оболочка приложения написана на Java и основана на обработке и другом программном обеспечении с открытым исходным кодом. Программирование микроконтроллеров Arduino осуществляется на языке программирования C++, упрощенная версия.

Среда разработки Arduino IDE состоит из следующих компонентов:

- Текстовый редактор кода программы (1);
- Область сообщений (2);
- Консоль (3);
- Панель инструментов (4);
- Панель с часто используемыми командами (5)

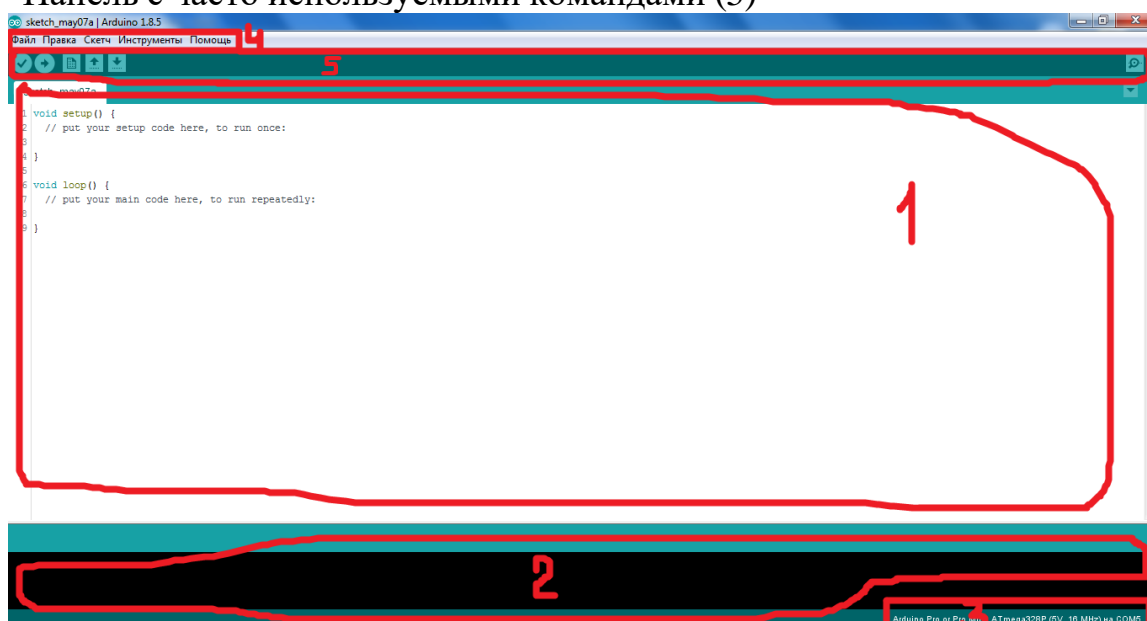












Рисунок3.Основные компоненты среды разработки Arduino IDE



Рисунок 4. Основное окно редактора кода программы

В основном окне (рисунок 4.) пишется код программы. После написания программы, нажатием на кнопку , производится проверка кода на наличие ошибок и её компиляция. Если компиляция прошла успешно, то необходимо загрузить программу на микроконтроллер, нажав на кнопку . Также на панели часто используемыми командами      Сохранить есть кнопки  для создания нового файла программы, загрузки и её сохранения  .

4. Перечень используемого оборудования:

- 4.1 Программа MPLAB IDE
- 4.2 ПК.
- 4.3. Платформа Freeduino MaxSerial

Задание

Составьте программу управления включением и выключением светодиода, подключенного к 13-му ПИН микроконтроллера с параметрами согласно варианту, указанному в таблице 1.

Таблица 1 – Варианты для практической работы

№	ПИН	Время горения светодиода, с	Время паузы, с
1	2	1	3
2	4	3	5
3	6	8	7
4	3	9,3	9,5
5	5	4	2,2
6	9	1,2	4
7	12	2	6
8	10	11,5	5
9	13	5	3,5
10	11	1	2
11	15	15	9
12	16	3,5	8
13	24	4,8	7,5
14	27	6	5,5
15	28	8,8	6,4
16	25	4	4
17	18	6	5
18	19	2,5	2

5. Порядок выполнения.

5.1. Изучить описание примера программы

Текст программы:

```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(2000);
}
```

Первая строка объявляет обязательную функцию **setup()**.

Функция **setup()** вызывается, когда стартует скетч (программа для микроконтроллера). Используется для инициализации переменных, определения режимов работы выводов, запуска используемых библиотек и т.д. Она производит вызов стандартной функции **pinMode**, которая устанавливает режим работы заданного вход/выхода (ПИН) как входа или как выхода. С

помощью данной функции 13-й ПИН микроконтроллера переводится в режим **вывода**, чтобы затем можно было управлять напряжением на этом контакте.

Затем объявляется функция **loop()**—она будет постоянно вызываться в бесконечном цикле в процессе работы.

Функция **digitalWrite** подает HIGH или LOW значение на цифровой вход/выход (pin). Выставляем на 13-ом ПИН «высокое» значение (HIGH)—это 5 В (3.3 В для плат, питающихся от 3.3 В).

Функцию **delay()**, обеспечивает паузу в тысячу миллисекунд (одну секунду).

Далее с помощью **digitalWrite** выставляется «низкое» значение (LOW)—это 0 В, и делаем паузу в две секунды. После завершения функции **loop()** она будет вызвана снова.

Таким образом, под управлением данной программы светодиод, подключенный к 13-му ПИН микроконтроллера, будет работать с периодом включения в одну секунду и периодом выключения в две секунды.

5.2. Написать и выполнить программу для управления работой светодиодом с периодом включения и выключения, указанным в варианте.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- код программы, написанной в соответствии с вариантом.
- выводы по проделанной работе

7. Контрольные вопросы

1. Перечислите основные компоненты среды разработки Arduino IDE
2. Назначение функции **setup()**.
3. С помощью какой функции можно обеспечить подачу HIGH или LOW на выход микроконтроллера?
4. С помощью какой функции задается пауза?

8. Список используемой литературы

1. Пигарев, Л. А. Микропроцессорные системы автоматического управления [Электронный ресурс]. – Санкт-Петербург: СПбГАУ, 2017.
2. Шпак, Ю. А. Программирование на языке C для AVR и PIC микроконтроллеров / Ю. А. Шпак. – К. : МК-Пресс ; СПб. : КОРОНАВЕК, 2011.
3. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г. Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021

ЛАБОРАТОРНАЯ РАБОТА №10
РАЗРАБОТКА АВТОМАТА «БЕГУЩИЕ ОГНИ»

1. Цель работы: Получить практическое представление о реализации конечных автоматов с использованием возможностей среды разработки Arduino IDE

2. Время выполнения работы-2 час

3. Краткие теоретические сведения

Конечные автоматы

В задачах, обрабатывающих ввод по символно, или череду каких-нибудь событий, часто возникают похожие решения, которые включают в себя некоторое текущее состояние-переменную и правила перехода между переменными после обработки нового ввода или события. Такие конструкции были обобщены теоретиками под названием конечного автомата.

Конечный автомат состоит из конечного множества состояний S , начального состояния S_0 и конечного множества переходов. Переходы имеют вид (d, X, t, a) , где

$d \in S$ – начальное состояние, из которого возможно выполнить переход;

X – множество символов, которые активируют переход;

$t \in S$ – состояние, в которое будет выполнен переход

a – действие, которое конечный автомат выполняет при переходе из состояния

d в t . Часто действие может отсутствовать.

Конечный автомат, это некоторое обобщение системы, которая в зависимости от внешних событий может изменять своё состояние, и в процессе таких изменений как-то реагировать на эти события. Частым примером конечного автомата является кофейный автомат. У автомата есть различные события, на которые он может реагировать (нажатие кнопки автомата), состояние, которое он может изменять при нажатии кнопок, действия, которые происходят при изменениях состояний.

Рассмотрим конечный автомат на примере программы, которая подсчитывает во вводимом тексте количество слов, состоящих только из заглавных букв. Основная идея в том, чтобы следить, были ли все уже введённые буквы текущего слова заглавными, и сохранять эту информацию в переменной, которая называется “состояние”. Состояний будет три:

- после ввода одного или нескольких пробелов,
- все введённые буквы текущего слова заглавные, и
- не все введённые буквы слова заглавные.

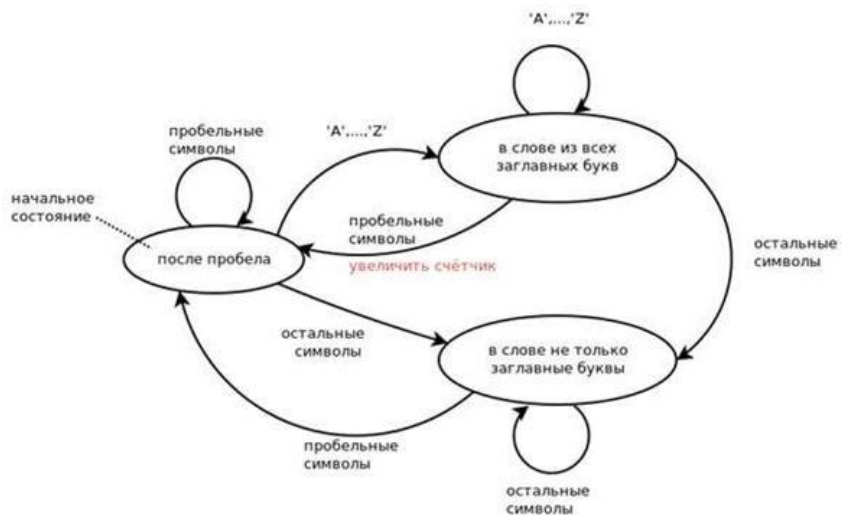


Рисунок 1. Схема, поясняющая принцип работы конечного автомата

Запишем эту схему в виде кода на Си++, применяя оператор `switch`.
 После считывания символа со стандартного ввода сначала проверяем, в каком состоянии мы находимся, затем, в зависимости от состояния и поступившего символа, совершаем переход в другое состояние или остаёмся в текущем.

```

int x;
x = get_some_value();
switch(x) {
case 0:
// Действие 0.
break;
case 1:
// Действие 1.
break;
case 2:
// Действие 2.
break;
case 3:
// Действие 3.
break;
default:
// Действие для всех остальных x.
break;
}

```

Оператор **switch...case** управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. В частности, оператор **switch** сравнивает

значение переменной со значением, определенном в операторах **case**. Когда найден оператор **case**, значение которого равно значению переменной, выполняется программный код в этом операторе.

Ключевое слово **break** является командой выхода из оператора **case** и обычно используется в конце каждого **case**. Без оператора **break** оператор **switch** будет продолжать вычислять следующие выражения, пока не достигнет **break** или конец оператора **switch**. Обратите внимание на **break** в конце каждого действия в программе. Чтобы не забывать **break**, рекомендуется писать его сразу же, как только написан **case**, а уже потом добавлять в **case** действия.

Bswitch(x) **x** должно быть переменной целого типа или выражением с результатом целого типа;

В **case** могут стоять только целые числа, и они должны быть константами. Иначе при компиляции не получится сформировать таблицу переходов.

4. Перечень используемого оборудования:

4.1 Программа MPLABIDE

4.2 ПК.

4.3. Платформа Freeduino MaxSerial

Задание

Составьте программу с различными вариантами управления светодиодами.

5. Порядок выполнения.

5.1. Изучить описание примера программы

Программа предполагает три варианта горения четырех светодиодов. Вариант будет выбираться пользователем командой с ПК. При отправке «1» светодиоды будут последовательно загораться через каждые четверть секунды, при отправке «2» светодиоды будут последовательно загораться в обратном порядке через каждые четверть секунды, при отправке «3» светодиоды будут загораться в порядке 1 – 4 – 3 – 2 с интервалом времени в 0.5 секунды. Для ввода команд необходимо открыть **Serial Monitor (Инструменты – Монитор порта)** и выставить скорость передачи данных, как в написанной программе.

```
int value = 0;
int led1 = 2;
int led2 = 4; int led3 = 6;
int led4 = 3;
void setup()
{
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
```



```

pinMode(led4, OUTPUT); }
void loop()
{
if (Serial.available())
{
value = Serial.read();
}
switch(value)
{
case(1): {
digitalWrite(led1, HIGH);
delay(250);
digitalWrite(led2, HIGH);
delay(250);
digitalWrite(led3, HIGH);
delay(250);
digitalWrite(led4, HIGH);
delay(250);
break;
}
case(2):
{
digitalWrite(led4, HIGH);
delay(250);
digitalWrite(led3, HIGH);
delay(250);
digitalWrite(led2, HIGH);
delay(250);
digitalWrite(led1, HIGH);
delay(250);
break;
}
}case(3):
{
digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led2, HIGH);
delay(500);
digitalWrite(led3, HIGH);
delay(500);
digitalWrite(led4, HIGH);
delay(500);
break;
}
}

```

```

    }
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
}

```

Переменная **value**, будет содержать значение варианта горения светодиодов, и переменные ПИНов светодиодов. Далее инициализируем UART со скоростью 9600 и настраиваем ПИНЫ светодиодов на выход. В бесконечном цикле мы считываем данные из UART и записываем их в переменную **value**. Далее, в зависимости от значения **value**, выполняется определенный вариант горения светодиодов.

5.2. Написать и выполнить программу для управления работой четырех светодиодов, варианты горения светодиодов, выдержки времени и номера ПИН (см. Лабораторное занятие №9, рисунок 2) выбрать самостоятельно.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- код программы, написанной в соответствии с вариантом.
- выводы по проделанной работе

7. Контрольные вопросы

1. В каких случаях целесообразно применение конечного автомата?
2. Назначение переменной **value**
3. Назначение переменной **case**

8. Список используемой литературы

1. Пигарев, Л. А. Микропроцессорные системы автоматического управления [Электронный ресурс]. – Санкт-Петербург: СПбГАУ, 2017.

2. Шпак, Ю. А. Программирование на языке C для AVR и PIC микроконтроллеров / Ю. А. Шпак. – К. : МК-Пресс ; СПб. : КОРОНАВЕК, 2011.

3. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г.Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021

ЛАБОРАТОРНАЯ РАБОТА №11
РАЗРАБОТКА ПРОГРАММЫ «БЕГУЩИЕ ОГНИ»
С ИСПОЛЬЗОВАНИЕМ ПРЕРЫВАНИЙ ПО ТАЙМЕРУ

1. Цель работы: Получить практическое представление о реализации прерываний по таймеру в микроконтроллерах ATmega328 автоматов с использованием возможностей среды разработки Arduino IDE

2. Время выполнения работы-2 час

3. Краткие теоретические сведения

Прерывание это событие, при котором происходит приостановка основной программы и переход на выполнение подпрограммы прерывания. При этом в стек записывается содержимое счетчика команд, а в сам счетчик записывается вектор прерывания, по которому находится команда безусловного перехода к подпрограмме обработки прерывания. После выполнения подпрограммы обработки прерывания, в счетчик команд загружается сохраненное значение из стека и выполнение основной программы продолжается с того места, где оно остановилось.

Таблица векторов прерывания располагается в памяти программ, начиная с адреса 0x0002, либо в области загрузчика. Приоритет прерывания зависит также от его расположения в таблице векторов прерывания: чем меньше адрес, тем выше приоритет прерывания.

Для микроконтроллеров ATmega328 таблица векторов

0x0000	RESET - сброс
0x0002	INT0 - внешнее прерывание 0
0x0004	INT1 - внешнее прерывание 0
0x0006	PCINT0 - прерывание по изменению состояния нулевой группы выводов
0x0008	PCINT1 - прерывание по изменению состояния первой группы выводов
0x000A	PCINT2 - прерывание по изменению состояния второй группы выводов
0x000C	WDT - прерывание от сторожевого таймера
0x000E	TIMER2 COMPA - прерывание от таймера/счетчика T2 при совпадении с A
0x0010	TIMER2 COMPB - прерывание от таймера/счетчика T2 при совпадении с B
0x0012	TIMER2 OVF - прерывание по переполнению таймера/счетчика T2
0x0014	TIMER1 CAPT - прерывание от таймера/счетчика T1 по записи
0x0016	TIMER1 COMPA - прерывание от таймера/счетчика T1 при совпадении с A
0x0018	TIMER1 COMPB - прерывание от таймера/счетчика T2 при совпадении с B
0x001A	TIMER1 OVF - прерывание по переполнению таймера/счетчика T1
0x001C	TIMER0 COMPA - прерывание от таймера/счетчика T0 при совпадении с A
0x001E	TIMER0 COMPB - прерывание от таймера/счетчика T0 при совпадении с B
0x0020	TIMER0 OVF - прерывание по переполнению таймера/счетчика T0
0x0022	SPI, STC - прерывание по окончанию передачи модуля SPI
0x0024	USART, RX - прерыванию по окончанию приема модуля USART
0x0026	USART, UDRE - прерывание по опустошению регистра данных модуля USART
0x0028	USART, TX - прерывание по окончанию приема модуля USART
0x002A	ADC - прерывание по завершению преобразования АЦП
0x002C	EE READY - прерывание по готовности памяти EEPROM
0x002E	ANALOG COMP - прерывание от аналогового компаратора
0x0030	TWI - прерывание от модуля I2C (TWI)
0x0032	SPM READY - прерывание по готовности flash памяти

Инициализация таймера

Для начала работы таймером необходимо его инициализировать путем настройки необходимых регистров. Разберем регистры таймера T0 – TCNT0 и OCR0A и OCR0B (Регистры сравнения А и В), TCCR0A и TCCR0B (Конфигурационные регистры А и В), TIMSK0 (Регистр по управлению прерываниями от таймера) и TIFR0 (Регистр флагов прерываний таймера).

1) Регистр TCNT0 (Счетный регистр)

TCNT0								
Bit	7	6	5	4	3	2	1	0
	TCNT0[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Это 8-ми разрядный счетный регистр. Когда таймер работает, по каждому импульсу тактового сигнала значение TCNT0 изменяется на единицу, в зависимости от режима работы таймера в сторону увеличения или уменьшения.

Когда таймер работает, изменять его содержимое TCNT0 не рекомендуется, так как это блокирует схему сравнения на один такт.

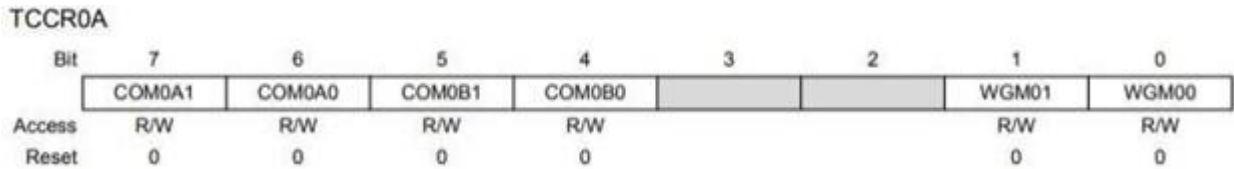
2) Регистры OCR0A и OCR0B (Регистры сравнения)

OCR0A								
Bit	7	6	5	4	3	2	1	0
	OCR0A[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

OCR0B								
Bit	7	6	5	4	3	2	1	0
	OCR0B[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Это 8-разрядные регистры сравнения. Его значение постоянно сравнивается со счетным регистром TCNT0, и в случае совпадения таймер может выполнять какие-то действия - вызывать прерывание, менять состояние вывода OC0 и т.д. в зависимости от режима работы.

3) TCCR0A (Конфигурационный регистр А)



Биты 7:6 – COM1A 1:0: контролируют поведение выхода OC1A в зависимости от режима работы таймера (см. Таблицы 1, 2, 3, 4)

Таблица 1– Нормальный режим

COM0A1	COM0A0	Описание состояния выхода
0	0	вывод OC0A не функционирует
0	1	изменение состояния вывода OC0A на противоположное при совпадении с A
1	0	сброс вывода OC0A в 0 при совпадении с A
1	1	установка вывода OC0A в 1 при совпадении с A

Таблица 2– Режим коррекции фазы ШИМ

COM0A1	COM0A0	Описание состояния выхода
0	0	вывод OC0A не функционирует
0	1	если бит WGM02 регистра TCCR0B установлен в 0, вывод OC0A не функционирует если бит WGM02 регистра TCCR0B установлен в 0, изменение состояния вывода OC0A на противоположное
1	0	сброс вывода OC0A в 0 при совпадении с A во время увеличения значения счетчика, установка вывода OC0A в 1 при совпадении с A во время уменьшения значения счетчика
1	1	установка вывода OC0A в 1 при совпадении с A во время увеличения значения счетчика, сброс вывода OC0A в 0 при совпадении с A во время уменьшения значения счетчика

Таблица 3 – CTC режим работы таймера

COM0A1	COM0A0	Описание состояния выхода
0	0	вывод OC0A не функционирует
0	1	изменение состояния вывода OC0A на противоположное при совпадении с A
1	0	сброс вывода OC0A в 0 при совпадении с A
1	1	установка вывода OC0A в 1 при совпадении с A

Таблица 4– Работа таймера в режиме ШИМ

COM0A1	COM0A0	Описание состояния выхода
0	0	вывод OC0A не функционирует
0	1	если бит WGM02 регистра TCCR0B установлен в 0, вывод OC0A не функционирует если бит WGM02 регистра TCCR0B установлен в 1, изменение состояния вывода OC0A на противоположное при совпадении с A
1	0	сброс вывода OC0A в 0 при совпадении с A, установка вывода OC0A в 1 если регистр TCNT0 принимает значение 0x00 (неинверсный режим)
1	1	установка вывода OC0A в 1 при совпадении с A, установка вывода OC0A в 0 если регистр TCNT0 принимает значение 0x00 (инверсный режим)

Биты 5:4 – COM1B 1:0: контролируют поведение выхода OC1B в зависимости от режима работы таймера (см. Таблицы 5, 6, 7, 8)

Таблица 5 – Нормальный режим

COM0B1	COM0B0	Описание состояния выхода
0	0	вывод OC0B не функционирует
0	1	изменение состояния вывода OC0B на противоположное при совпадении с B
1	0	сброс вывода OC0B в 0 при совпадении с B
1	1	установка вывода OC0B в 1 при совпадении с B

Таблица 6 – Режим коррекции фазы ШИМ

COM0B1	COM0B0	Описание состояния выхода
0	0	вывод OC0B не функционирует
0	1	резерв
1	0	сброс вывода OC0B в 0 при совпадении с B во время увеличения значения счетчика, установка вывода OC0B в 1 при совпадении с B во время уменьшения значения счетчика
1	1	установка вывода OC0B в 1 при совпадении с B во время увеличения значения счетчика, сброс вывода OC0B в 0 при совпадении с B во время уменьшения значения счетчика

Таблица 7 – CTC режим работы таймера

COM0B1	COM0B0	Описание состояния выхода
0	0	вывод OC0B не функционирует
0	1	изменение состояния вывода OC0B на противоположное при совпадении с В
1	0	сброс вывода OC0B в 0 при совпадении с В
1	1	установка вывода OC0B в 1 при совпадении с В

Таблица 8 – Работа таймера в режиме ШИМ

COM0B1	COM0B0	Описание состояния выхода
0	0	вывод OC0B не функционирует
0	1	резерв
1	0	сброс вывода OC0B в 0 при совпадении с В, установка вывода OC0B в 1 если регистр TCNT0 принимает значение 0x00 (неинверсный режим)
1	1	установка вывода OC0B в 1 при совпадении с В, установка вывода OC0B в 0 если регистр TCNT0 принимает значение 0x00 (инверсный режим) сброс вывода OC0B в 0 если регистр TCNT0 принимает значение 0x00 (инверсный режим)

Биты 3:2 – Не используются.

Биты 1:0 – WGM01, WGM00 служат для настройки режима работы. Всего их может быть четыре - нормальный режим (normal), сброс таймера при совпадении (CTC), и два режима широтно-импульсной модуляции (FastPWM(Режим ШИМ) и Phase Correct PWM (Режим коррекции фазы ШИМ)). Все возможные значения описаны в таблице 9.

Таблица 9– Настройка режима работы

WGM02	WGM01	WGM00	Режим работы таймера
0	0	0	Normal
0	0	1	Phase Correct PWM
0	1	0	CTC
0	1	1	Fast PWM
1	0	0	Резерв
1	0	1	Phase Correct PWM
1	1	0	Резерв
1	1	1	Fast PWM

4) TCCR0B (Конфигурационный регистр А)



Биты 7:6 – FOC0A и FOC0B принудительно устанавливают значение на выводах OC0A и OC0B.

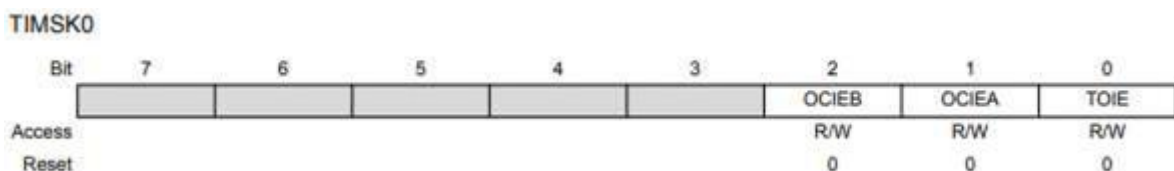
Бит 3 – WGM00 служит для настройки режима работы (см. Таблицу 9)

Биты 2:0 – CS02, CS01, CS00 устанавливают режим тактирования и пред делителя тактовой частоты таймера/счетчика T0:

Таблица 10–Режимы тактирования

CS02	CS01	CS00	Режимы тактирования
0	0	0	таймер/счетчик T0 остановлен
0	0	1	тактовый генератор CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	внешний источник на выводе T0 по спаду сигнала
1	1	1	внешний источник на выводе T0 по возрастанию сигнала

5) TIMSK0(Регистр управления прерываний таймеров)



Биты 2:1 – OCIEB и OCIEA разрешают прерывания при совпадении с A и B, а бит 0 – TOIE разрешает прерывание по переполнению при установке 1. Если в эти биты записать 0, прерывания от таймера/счетчика будут запрещены.

6) TIFR0 (Регистр флагов прерываний таймеров)



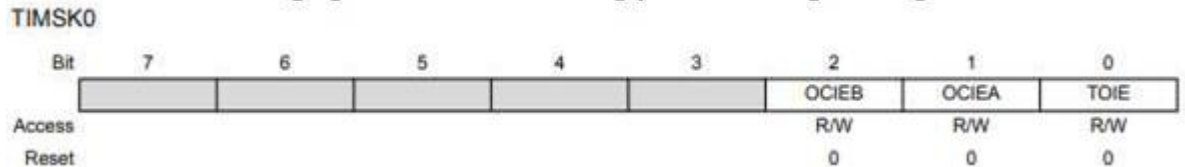
Биты 2:1 –OCFB и OCFA устанавливаются в 1 при совпадении с счетного регистра с регистром сравнения (A и B), а бит 0 – TOV устанавливается в 1 при переполнении счетного регистра.

Прерывания по таймеру

В микроконтроллере ATmega328 прерывания по таймеру можно производить двумя способами:

- по переполнению таймера;
- по совпадению значения с регистром сравнения OCR**.

За активацию прерывания по таймеру отвечает регистр TIMSK*.



Биты 2:1 – OCIEB и OCIEA разрешают прерывания при совпадении с каналом A и каналом B, а бит 0 – TOIE разрешает прерывание по переполнению при установке 1. Если в эти биты записать 0, прерывания от таймера/счетчика будут запрещены

4. Перечень используемого оборудования:

- 4.1 Программа MPLABIDE
- 4.2 ПК.
- 4.3. Платформа Freeduino MaxSerial

Задание

Составьте программу последовательного включения каждого из четырех светодиодов с циклическим повторением

5. Порядок выполнения.

5.1. Изучить описание примера программы

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
void InitTimer()
{
TCCR1A|=(1<<COM0A1)|(0<<COM0A0)|(0<<WGM00)|(0<<WGM01);
TCCR1B|= (0<<CS02) | (1<<CS01)| (0<<CS00);
TIMSK0|= (1<< OCIE1A);
TCNT1= 0;
OCR1A = 1999;
}
void setup() {
longcounter = 1;
intled1 =2;
```

```

intled2 =4;
intled3 =6;
intled4 =3;
  InitTimer();
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
}
ISR(TIMER0_COMPA_vect)
{
if(counter <= 4000)
{
  counter++; }
else {
  counter = 0;
}
}
void loop()
{
switch(counter)
{
case(1000):
{
digitalWrite(led1, HIGH);
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
break;
}
case(2000):
{
digitalWrite(led1, LOW);

```

```

digitalWrite(led2, HIGH);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW); break;
}
case(3000):
{
digitalWrite(led1, LOW);
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
digitalWrite(led4, LOW);
break;
}
case(4000):
{
digitalWrite(led1, LOW);
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, HIGH);
break;
}
case(0):
{
digitalWrite(led1, LOW);
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
    break;
}
}
}
}

```

Timer1-функция инициализации 16-битного таймера, который будет работать с прерыванием каждую 1 мс.

Функция **setup()** объявляет переменные выводов, к которым подключены светодиоды, вызывается инициализация таймера, а выводы светодиодов устанавливаются на выход. В обработчике прерываний переменная counter каждую 1 мс будет увеличиваться на единицу. В бесконечном цикле, при достижении значения 1000 (т.е. таймер вызвал 1000 прерываний, что означает прошествие одной секунды), будет загораться первый светодиод, при достижении двух секунд будет загораться второй светодиод, при достижении трех секунд – третий. На четвертой секунде, загорится четвертый светодиод, счетчик таймера обнулится и цикл начнется сначала.

5.2. Написать и выполнить программу для последовательного включения каждого из четырех светодиодов с циклическим повторением, варианты горения светодиодов, выдержки времени и номера ПИН (см. Лабораторное занятие №9, рисунок 2) выбрать самостоятельно.

6. Указания к выполнению отчета

Отчет должен содержать:

- наименование и цель работы.
- код программы, написанной в соответствии с вариантом.
- выводы по проделанной работе

7. Контрольные вопросы

1. Опишите процедуру обработки прерываний
2. От чего зависит приоритет прерывания?
3. Какими способами можно производить прерывания по таймеру в микроконтроллере ATmega328?

8. Список используемой литературы

1. Пигарев, Л. А. Микропроцессорные системы автоматического управления [Электронный ресурс]. – Санкт-Петербург: СПбГАУ, 2017.
2. Основы электроники: учебник для СПО/ О.В., Миловзоров, И.Г. Панков, — 6-е изд. перераб. и доп. — Москва: Издательство Юрайт, 2021
3. Шпак, Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров / Ю. А. Шпак. – К. : МК-Пресс ; СПб. : КОРОНАВЕК, 2011.