МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Тепловозы и тепловые двигатели»

В. В. СКРЕЖЕНДЕВСКИЙ

ПРОГРАММИРУЕМЫЕ ЦИФРОВЫЕ УСТРОЙСТВА

Учебно-методическое пособие

Смотрите другие материалы по дисциплине на
http://eot-ttd.blog.tut.by

Гомель 2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Тепловозы и тепловые двигатели»

В. В. СКРЕЖЕНДЕВСКИЙ

ПРОГРАММИРУЕМЫЕ ЦИФРОВЫЕ УСТРОЙСТВА

Одобрено советом механического факультета
и методической комиссией заочного факультета
в качестве учебно-методического пособия

Гомель 2013

УДК 004.4:629.4 (075.8)
ББК 32.973+39
С45

Р е ц е н з е н т – зав. кафедрой «Микропроцессорная техника и информационно-управляющие системы» канд. физ.-мат. наук, доцент *Н. В. Рязанцева* (УО «БелГУТ»)

Скрежендевский, В. В.

С45 Программируемые цифровые устройства: учеб.-метод. пособие / В. В. Скрежендевский; М-во образования Респ. Беларусь, Белорус. гос. ун-т трансп. – Гомель: БелГУТ, 2013. – 64 с.
ISBN 978-985-554-220-0

Представлены сведения об архитектуре микроконтроллеров Microchip, справочные материалы о системе команд и карты памяти программ и данных микроконтроллера PIC16F877A.

Предназначено для студентов III курса механического факультета специальности 1-37 02 01-01 «Тяговый состав железнодорожного транспорта (тепловозы)» всех форм обучения.

УДК 004.4:629.4 (075.8)
ББК 32.973+39

ISBN 978-985-554-220-0

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина «Программируемые цифровые устройства» в программе обучения для направления специальности 1-37 02 01-01 «Тяговый состав железнодорожного транспорта (тепловозы)» появилась в связи с тем, что в настоящее время на железнодорожном транспорте широко внедряются микропроцессорные системы управления, регулирования, защиты технических систем от аварийных режимов. Не составляет исключение и тепловоз, который является одним из наиболее сложных и высокоавтоматизированных объектов, эксплуатируемых на железнодорожном транспорте. В настоящее время на Белорусской железной дороге интенсивно идет процесс внедрения:

– унифицированной системы тепловозной автоматики (УСТА);

– комплексного устройства локомотивной безопасности (КЛУБ-У);

– регистратора параметров работы тепловоза (РПРТ);

– унифицированной системы автоведения пригородного электропоезда (УСАВП) и др.

Парк тягового подвижного состава пополняется современными локомотивами, оборудованными микропроцессорными системами управления и регулирования основным и вспомогательным оборудованием.

В связи с этим представляется важным приобретение элементарных знаний и практических навыков в области микроконтроллеров для студентов направления специальности 1-37 02 01-01 «Тяговый состав железнодорожного транспорта (тепловозы)», так как это позволит им впоследствии обеспечивать квалифицированное техническое обслуживание и эксплуатацию современного тягового подвижного состава.

Данное учебно-методическое пособие имеет сугубо практическую направленность и построено таким образом, чтобы студенты могли в рамках небольшого курса дисциплины «Программируемые цифровые устройства» научиться самостоятельно разрабатывать несложные программы для микроконтроллеров фирмы Microchip и использовать интегрированную среду разработки MPLAB. В связи с ограниченным объемом курса дисциплины пособие построено как самоучитель и содержит в себе необходимые, минимальные сведения из теории и справочные материалы по микроконтроллерам среднего семейства.

Сложность изложения материала будет нарастать постепенно, поэтому многие важные и относительно сложные моменты вначале пособия умышленно не рассматриваются, чтобы не «перегрузить» читателя.

У вдумчивого и внимательного читателя при работе с данным пособием обязательно возникнут вопросы, ответы на которые можно найти в [1] и [2], а также в другой литературе, посвященной данной тематике.

1 ПРОГРАММИРУЕМЫЕ ЦИФРОВЫЕ УСТРОЙСТВА

1.1 Что такое микроконтроллер и как он работает

Ознакомившись с содержанием этого подраздела, вы будете иметь представление о назначении микроконтроллеров, структуре ядра микроконтроллеров компании Microchip, как «работает» программа микроконтроллера и для чего нужен тактовый генератор, как микроконтроллер «общается» с внешним миром.

Всем хорошо знакомы персональные компьютеры, которые позволяют нам решать разнообразные задачи для научных исследований, работы, учебы, коммуникаций и отдыха. Эти компьютеры отличаются мощным интерфейсом человек – машина, так как являются непосредственными помощниками человека в его разнообразной деятельности. В то же время существует целый класс «подпольных» компьютеров, которые спрятаны в различных устройствах. От такого компьютера не требуется обслуживания ресурсоемкого интерфейса человек – машина, как на персональном компьютере. Его задача куда более скромная – управлять каким-либо устройством или процессом: стиральной машиной, подачей топлива в двигатель внутреннего сгорания, регулировать ток возбуждения тягового генератора тепловоза (УСТА) и т. п. Поэтому не нужны высокая тактовая частота и большой объем памяти. Например, УСТА создана на базе компьютера с тактовой частотой 10 МГц, объемом памяти для программы 20 Кбайт (Flash ПЗУ) и 256 байтами для хранения данных (ОЗУ). Такой компьютер представляет собой микросхему с 40 выводами (рисунок 1). Микросхема содержит все необходимые устройства для работы процессора и дополнительные устройства для «общения» с «внешним миром». Этот компьютер принято называть микроконтроллером.

Как было упомянуто ранее, изучать микроконтроллеры мы будем на базе микроконтроллеров фирмы Microchip. Этот выбор основан с одной стороны на сложившейся традиции в нашем вузе, а с другой – относительной простотой системы команд микроконтроллеров этой фирмы, что существенно облегчает начальное обучение. Кроме этого, фирма Microchip хорошо представлена русскоязычной документацией на микроконтроллеры и средства разработки.

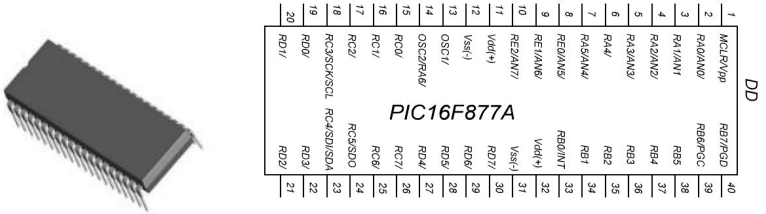


Рисунок 1 – Вид 40-выводного корпуса (DIP40) и условное обозначение микроконтроллера PIC16F877A

Микроконтроллер состоит из **ядра** и **периферийных модулей**. **Ядро** представляет собой собственно сам компьютер или интеллектуальную часть микроконтроллера, которая состоит из **памяти программ, памяти данных, центрального процессорного устройства (ЦПУ или в английском варианте – CPU)**, **тактового генератора** и ряда других устройств. **Периферийные модули** – это порты ввода-вывода, таймеры, аналого-цифровой преобразователь и ряд других устройств. **Тактовый генератор** может быть встроен в корпус микроконтроллера, а может быть выполнен как отдельное устройство. На рисунке 2 показана упрощенная структура ядра микроконтроллера среднего семейства Microchip.

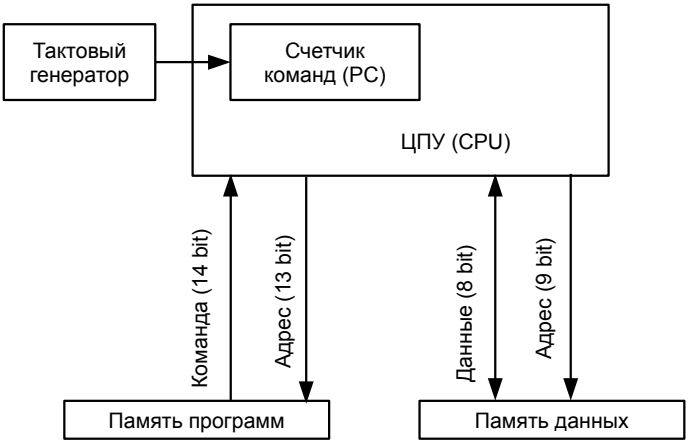


Рисунок 2 – Упрощенная структурная схема ядра микроконтроллера

Импульсы, которые вырабатывает тактовый генератор, используются для приращения значения в специальном регистре – **счетчике команд**. При работе тактового генератора в счетчик команд последовательно прибавляется единица. Значение (число) в счетчике команд является адресом регистра памяти программ микроконтроллера, из которого арифметико-логическое устройство читает команду и выполняет ее. Таким образом, последователь-

ность импульсов тактового генератора обеспечивает выполнение команд, загруженных в память программ микроконтроллера. Чем выше частота тактового генератора, тем больше команд выполняется в единицу времени и наоборот. **Если тактовый генератор по каким-то причинам не работает – программа микроконтроллера не выполняется!**

Сброс микроконтроллера – состояние, при котором в счетчик команд микроконтроллера загружается нулевое значение. При включении питания, до запуска тактового генератора, микроконтроллер находится в состоянии сброса. То есть при включении питания выполнение программы начинается с команды, записанной в нулевой адрес памяти программ. Другие события, при которых микроконтроллер может находится в состоянии сброса, здесь не рассматриваются.

Большинство команд, которые «понимает» арифметико-логическое устройство могут только изменять значения двоичных чисел в регистрах памяти данных микроконтроллера и некоторые команды – анализировать значения в регистрах памяти данных на равенство нулю. Как тогда микроконтроллер может решать довольно сложные задачи по управлению различными устройствами? Что является связующим звеном между программной и аппаратной частью микроконтроллера, которая физически соединена с «внешним миром»? Для этого используются регистры специального назначения памяти данных микроконтроллера и простейший периферийный модуль – порт ввода-вывода.

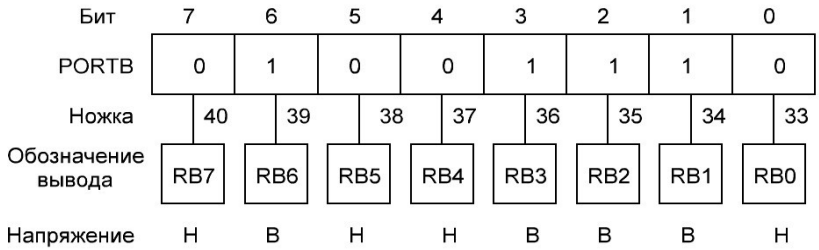
Память данных микроконтроллеров фирмы Microchip состоит из регистров **специального** и **общего** назначения. Назначение регистров специального назначения определено разработчиками данного микроконтроллера, например, для управления каким-нибудь периферийным модулем. Регистры общего назначения могут использоваться по усмотрению программиста-разработчика.

Поясню упоминавшиеся ранее термины: память, регистр, адрес, значение. **Память** – физическое устройство для хранения данных. Память данных в микроконтроллерах среднего семейства Microchip состоит из 8-битных регистров. **Регистр** – логическое устройство, используемое для хранения *n*-разрядных *двоичных* чисел, для микроконтроллеров Microchip $n = 8$. **Адрес** – символ или группа символов, которые идентифицируют регистр или другими словами являются именем регистра. **Значение** – число, записанное (сохраненное) в регистр.

Микроконтроллеры Microchip имеют **Гарвардскую архитектуру** памяти. Для хранения программы и данных выделены отдельные области памяти с разной разрядностью шины данных и адреса, как это видно на рисунке 2.

Теперь вернемся к вопросу связи микроконтроллера с «внешним миром» через **порт ввода-вывода**. Рассмотрим на примере микроконтроллера PIC16F877A периферийный модуль portb, который является 8-разрядным двунаправленным (может работать как вход и как выход) портом ввода-выво-

да. Не вдаваясь в подробности можно сказать, что с одной стороны этот модуль соединен с внешними выводами (33-40 ножки) микроконтроллера, а с другой «соединен» с регистрами специального назначения **PORTB и TRISB** (не путать portb и PORTB. В первом случае это периферийный модуль, во втором – адрес регистра). Регистр PORTB служит для обмена данными, а регистр TRISB для настройки направления передачи данных. Если при выполнении программы микроконтроллера в регистр TRISB записать единицы, то порт будет работать как вход. В этом случае, в зависимости от напряжения, которое будет приложено к 33-40 ножкам микроконтроллера от внешних цепей (датчиков, кнопок и т. п.), в регистр PORTB будут записаны «1» или «0». Если приложено высокое напряжение (близкое к напряжению питания) записывается единица, если низкое – записывается ноль. Записанные в регистр PORTB значения могут быть прочитаны программой. Причем каждому биту регистра PORTB соответствует своя ножка (рисунок 3).



Н – низкий уровень напряжения или **логический нуль** (близкий к нулю); В– высокий уровень напряжения или **логическая единица** (близкий к напряжению питания)

Рисунок 3 – Схема отображения уровней напряжения ножек микроконтроллера на регистр PORTB

Если при выполнении программы микроконтроллера в регистр TRISB записать нули, то порт portb будет работать как выход. В этом случае напряжение на 33-40 ножках микроконтроллера будет зависеть от значения, записанного в регистр PORTB. При записи командами программы в любой бит регистра PORTB «1» на соответствующей ножке микроконтроллер установит высокий уровень напряжения, при записи «0» – низкий.

Ножки PORTB можно настроить как входы и выходы в произвольной комбинации, задаваемой двоичным числом в регистре TRISB. Например, если TRISB = 01011100, то 40, 38, 34 и 33 ножки микроконтроллера будут работать как выходы, а 39, 37, 36 и 35 ножки – как входы.

Таким образом, с помощью загруженной в память программ программы, которая будет анализировать и изменять состояние битов и регистров PORTB и TRISB, микроконтроллер может определить, в каком состоянии

находятся внешние устройства (кнопки, контакты, датчики и т. п.) и в зависимости от этого выдавать сигналы для включения-отключения внешних исполнительных устройств (светодиодов, реле, полупроводниковых ключей и т. п.). Более сложные случаи управления и регулирования, которые можно реализовать с помощью микроконтроллера, мы пока рассматривать не будем.

Несколько слов о стадиях разработки устройства на базе микроконтроллера.

1 Разработка задания на проектирование.

2 Проектирование аппаратной части системы («железа»).

3 Разработка программного обеспечения для микроконтроллера.

4 Отладка программы для микроконтроллера в интегрированной среде разработки на персональном компьютере.

5 Изготовление макета готового устройства и программирование микроконтроллера.

6 Отладка программной и аппаратной частей устройства на макете.

7 Разработка конструкторской документации на серийное изделие.

В рамках данного курса мы будем выполнять в упрощенном виде 3-й и 4-й этапы разработки устройства на базе микроконтроллера.

Подведем итоги всего вышесказанного:

1 Микроконтроллер – микросхема, предназначенная для управления различными устройствами. Микроконтроллер сочетает на одном кристалле функции процессора и периферийных устройств, содержит оперативное запоминающее устройство (ОЗУ) и постоянное запоминающее устройство (ПЗУ). Это компьютер, способный выполнять задачи управления и регулирования, где не требуются большие вычислительные мощности и объем памяти.

2 Импульсы тактового генератора обеспечивают выполнение команд, загруженных в память программ микроконтроллера. Частотой тактового генератора определяется быстродействие микроконтроллера.

3 Команды микроконтроллера могут анализировать и изменять значения в регистрах специального и общего назначения памяти данных.

4 С помощью портов ввода-вывода микроконтроллер «общается» с «окружающим миром». Управление и отображение состояния портов ввода-вывода и других периферийных модулей осуществляется с помощью регистров специального назначения.

Повторю еще раз – многие важные и сложные вопросы пока мной не затронуты. Дальнейшее изложение материала составлено таким образом, что эти вопросы будут постепенно рассматриваться на конкретных примерах.

1.2 Назначение и основные компоненты MPLAB

Изучив этот подраздел, вы получите практические навыки работы с интегрированной средой разработки MPLAB. Научитесь создавать проект, изучите основные элементы интерфейса MPLAB.

MPLAB – это интегрированная среда разработки (ИСР), которая может быть загружена бесплатно с веб-сайта фирмы Microchip. MPLAB содержит все программные инструментальные средства, необходимые для того, чтобы написать программу на Ассемблере, ассемблировать ее, выполнить на имитаторе ее тестирование и, наконец, загрузить программу с помощью программатора в микроконтроллер.

MPLAB – это пакет, который непрерывно эволюционирует. К нему прилагаются отдельные руководства, он имеет встроенную интерактивную справочную систему. В этом пособии используются снимки экранов MPLAB версии 8.33.

MPLAB состоит из ряда отдельных компонентов, которые работают вместе, чтобы создать завершенную среду разработки:

Текстовый редактор. Обеспечивает возможность ввода исходного текста. Его работа до некоторой степени похожа на работу простого текстового редактора, например такого, как Notepad. Но вместе с тем он может распознавать основные элементы языка программирования, который используется. Так, при работе на Ассемблере он одним цветом раскрашивает команды, другим – метки, а третьим – комментарии. Благодаря этому программист может непосредственно видеть, имеется ли ошибка в представлении или использовании текста в строке программы на Ассемблере.

Менеджер проекта. Предпочтительный способ разработки программ в MPLAB – это создание проекта. Проект MPLAB группирует все файлы, относящиеся к одному проекту, вместе. Менеджер проекта обеспечивает правильное взаимодействие файлов проекта друг с другом и, в случае необходимости, обновление их надлежащим способом.

Ассемблер, который ассемблирует программу, т. е. переводит ее из мнемоники Ассемблера в машинные коды, готовые для выполнения микроконтроллером.

Компоновщик. В сложных проектах программный код может быть скомпонован из группы различных файлов. Роль компоновщика заключается в том, чтобы объединить их воедино, присваивая каждому фрагменту правильные адреса памяти, и при этом гарантировать, что переходы и обращения из одного файла к другому будут оформлены правильно.

Программный имитатор и отладчик. Программный имитатор позволяет тестировать программу, которая находится в стадии разработки. Это тестирование выполняется посредством ее запуска на ЦП, который имитируется хост-компьютером. Входные сигналы также могут имитироваться, выходные значения и состояния ячеек памяти могут наблюдаться. Отладчик включает средства, обеспечивающие полный контроль за выполнением программы, например пошаговое выполнение программы, запуск ее с пониженной скоростью выполнения либо останов по конкретному адресу памяти.

1.2.1 Создание проекта

1 В меню Project выбираем пункт New. В появившемся окне (рисунок 4) указываем имя проекта (Project Name) и размещение проекта («путь») на диске (Project Directory). Для указания размещения проекта на диске можно воспользоваться кнопкой Browse. При вводе имени проекта и «пути» следует использовать только латинские буквы и ограничить длину пути 255 символами.

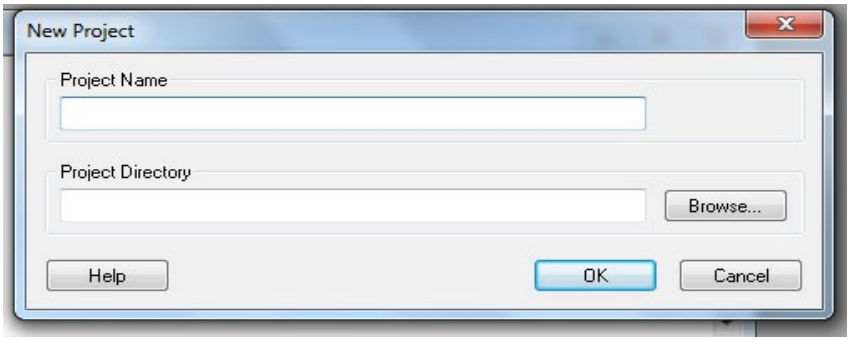


Рисунок 4 – Окно ввода имени и расположения проекта

2 В меню File выбираем пункт New. При этом на экране появится пустое окно исходного файла на Ассемблере, озаглавленное Untitled (рисунок 5).

3 Нужно проследить, чтобы мигающий курсор ввода текста находился в левом верхнем углу текстового поля окна Untitled. В меню File выбираем пункт Save As, в появившемся окне (рисунок 6) – «путь» к папке ранее сохраненного проекта и указываем в окне «Имя файла» имя файла с расширением .asm. Рекомендуется для версии 8.33 набирать имя файла и расширение вручную полностью, в противном случае проект может не работать, и его создание придется повторить с самого начала. Также рекомендуется задать имя файла одинаковое с именем проекта. После нажатия кнопки «Сохранить» в заголовке окна исходного файла на Ассемблере должно появиться введенное имя файла (вместо Untitled).

4 В меню Project выбираем пункт Add Files to Project. В открывшемся окне (рисунок 7) выбираем сохраненный на предыдущем шаге исходный файл на Ассемблере (расширение .asm). Нажимаем кнопку «Открыть». Если все правильно сделано, то в окне рабочего пространства («WorkSpace» рисунок 8) появится в папке исходные файлы («SourceFiles») имя исходного файла на Ассемблере.

Теперь в окне исходного файла (см. рисунок 5) можно набирать текст программы на Ассемблере или скопировать готовую программу (или ее фрагмент) из любого текстового файла.

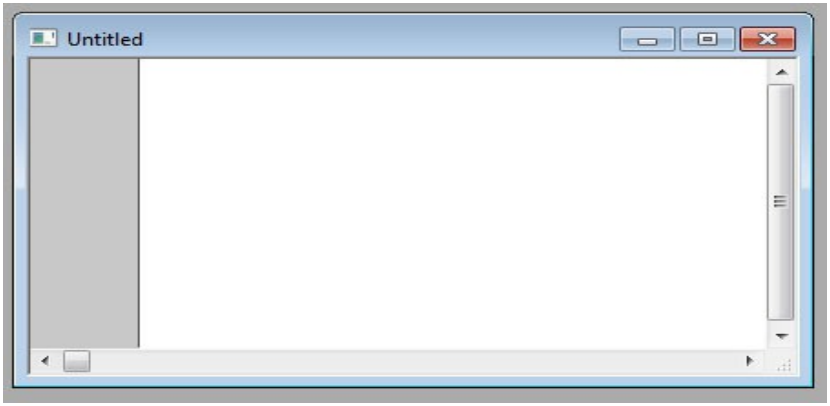


Рисунок 5 – Текстовое окно исходного файла на Ассемблере

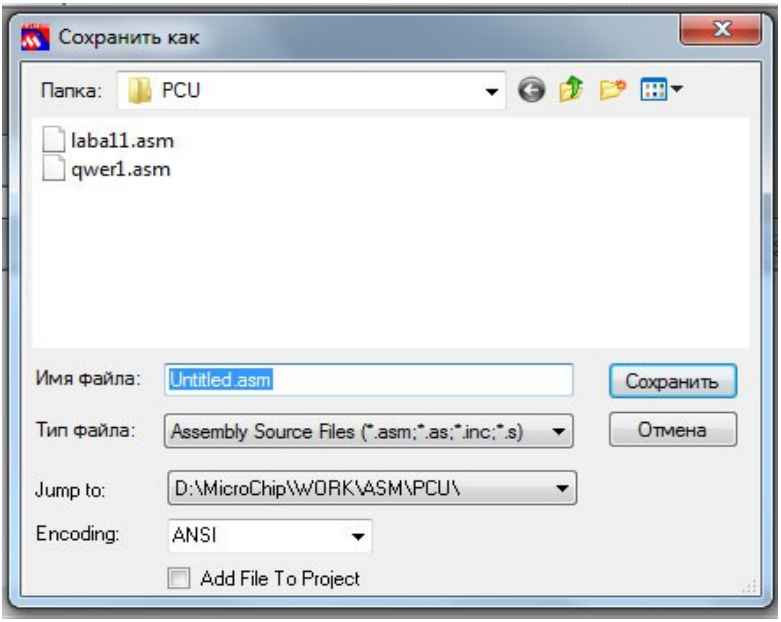


Рисунок 6 – Диалоговое окно «Сохранить как»

5 В меню «Configure» выбираем пункт «Select Device», а в открывшемся окне из выпадающего списка «Device» микроконтроллер, для которого пишется программа, например PIC16F877A, и нажимаем «OK» (рисунок 9).

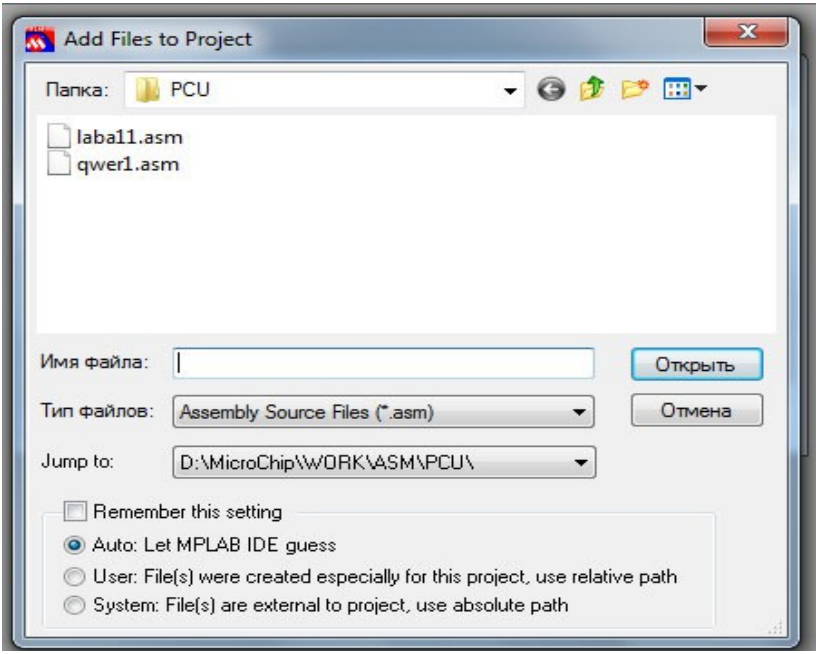


Рисунок 7 – Диалоговое окно «Добавить файлы в проект»

6 В меню «Configure» выбираем пункт «Configuration Bits», в открывшемся окне снимаем «птицу» «Configuration Bits Set in Code», после чего можно установить требуемые для данного проекта настройки битов конфигурации. Для сохранения выполненных изменений закрываем данное окно. Для учебной программы А.1, которая приведена в приложении А, следует установить биты конфигурации в соответствии с рисунком 10.

7 В меню «Debugger» выбираем пункт «Select Tool», в открывшемся списке выбираем пункт «MPLAB SIM». При этом на панели инструментов должна появиться панель инструментов отладчика программного имитатора MPSIM (рисунок 11).

8 В меню «Debugger» выбираем пункт «Settings», в открывшемся окне во вкладке «Osc/Trace» устанавливаем значение частоты **тактового генератора** и единицу ее измерения, например, 4 МГц.

9 Теперь скопируем в окно исходного файла (или наберем с клавиатуры) программу на Ассемблере. В качестве примера используем программу А.1.

Теперь проект создан и можно приступать к имитации выполнения программы.

В меню «Project» выбираем пункт «Build All», в результате откроется окно Output (Вывод), сообщающее о ходе компоновки. В окне Output вы либо получите сообщение "Build succeeded" (Компоновка успешна) (рисунок 12), либо со-

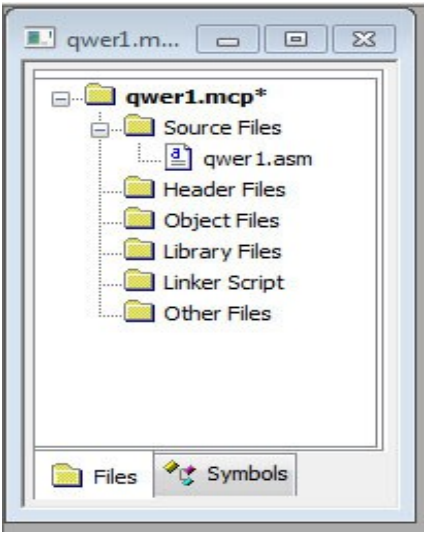


Рисунок 8 – Окно «Рабочее пространство» («WorkSpace»). Имя исходного файла на ассемблере: qwer1.asm

общение "Build failed" (Компоновка не удалась) (рисунок 13).

На рисунке 13 видно, что Ассемблером выявлена ошибка в тексте программы, связанная с тем, что в команде clrf была пропущена одна буква. Сообщение об ошибке Error[122] D:\MICROCHIP\111\ASDG.ASM 21: Illegal opcode (PORTD) содержит в себе код ошибки [122], номер строки, в которой она обнаружена (21), и словесное описание ошибки (Illegal opcode (PORTD). Следует отметить, что компоновщик обнаруживает только синтаксические ошибки. Ошибки в логике алгоритма программист должен выявлять сам.

Обратите внимание, теперь на левом поле, напротив команды, расположенной в **векторе сброса** микроконтроллера (по адресу 0x000 памяти программ), находится зеленая стрелка-указатель следующей выполняемой команды Ассемблера во время симуляции. Перемещение стрелки-указателя отображает изменение значения **счетчика команд** (PC) микроконтроллера.

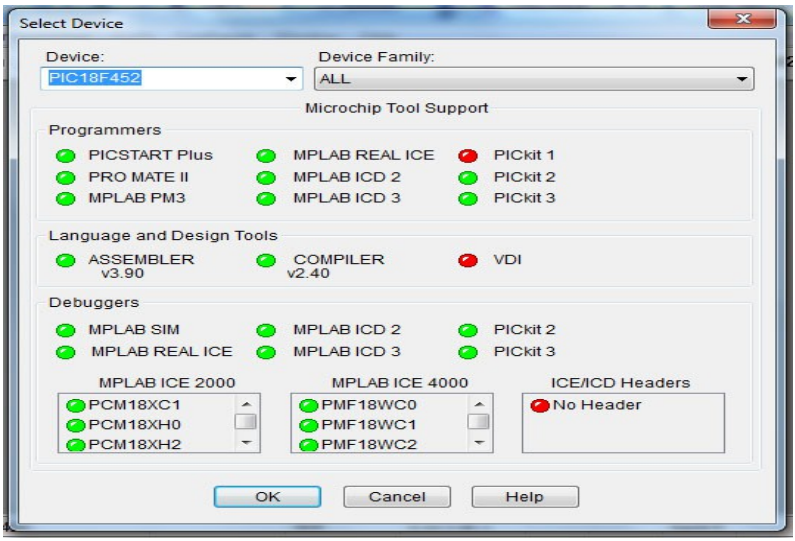


Рисунок 9 – Окно выбор устройства («Select Device»)

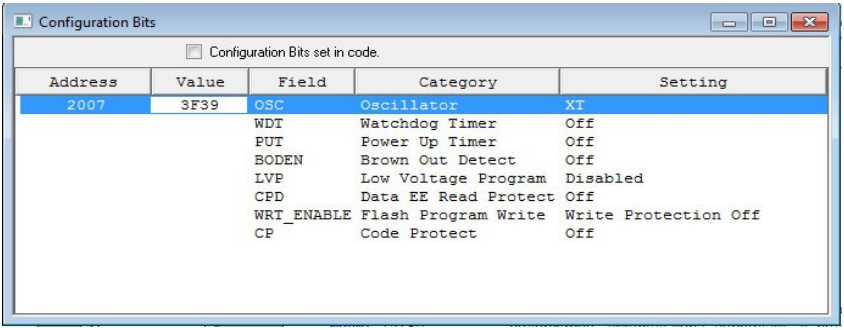


Рисунок 10 – Окно настройки битов конфигурации

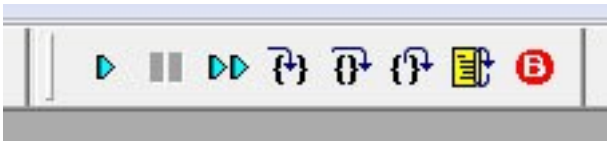


Рисунок 11 – Панель инструментов отладчика
программного имитатора MPSIM

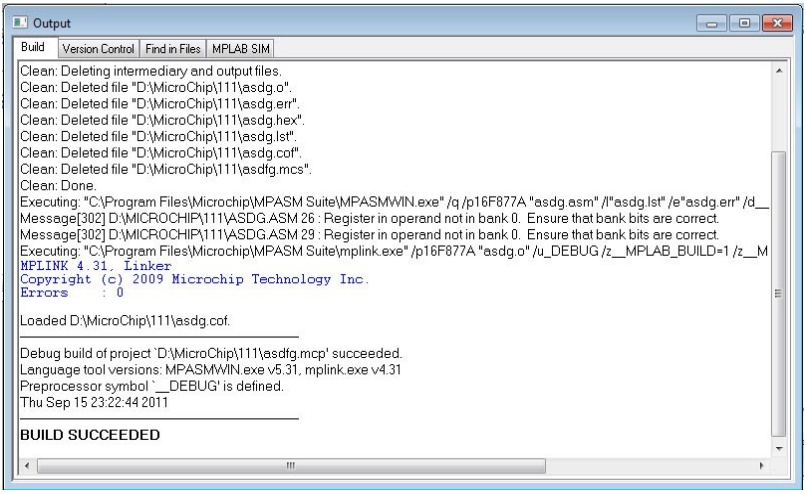


Рисунок 12 – Окно Output при успешной компоновке

В случае успешной компоновки откроется окно исходного файла на Ассемблере (рисунок 14).

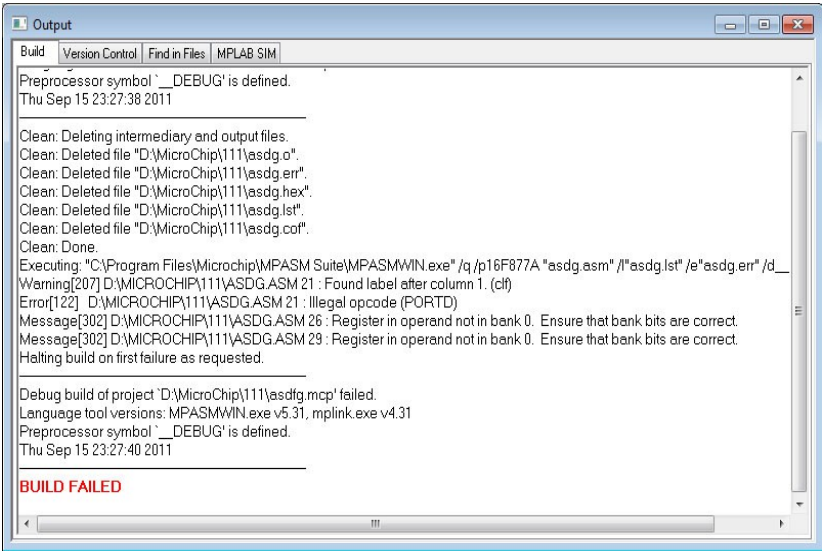


Рисунок 13 – Окно Output при не удачной компоновке

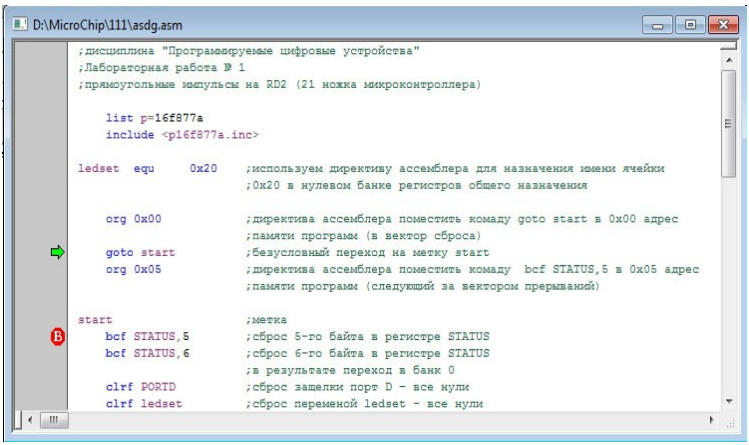


Рисунок 14 – Окно исходного файла на Ассемблере
после успешной компоновки

1.2.2 Моделирование (симуляция) выполнения программы микроконтроллера

В качестве примера возьмем программу А.1. На панели инструментов отладчика программного имитатора MPSIM (рисунок 15) имеются следующие кнопки:

– «Run» – пуск моделирования выполнения программы с текущей позиции стрелки-указателя (значения счетчика команд микроконтроллера);

– «Halt» – останов моделирования;

– «Animate» – автоматическое, замедленное, пошаговое выполнение программы с заданным шагом по времени. Шаг по времени можно задать в меню Debugger\Settings\Animation\RealtimeUpdates до 5 секунд на выполнение одной команды;

– «Step Into» – пошаговое выполнение, включая пошаговое выполнение подпрограмм;

– «Step Over» – пошаговое выполнение, исключая пошаговое выполнение подпрограмм. Подпрограммы в этом режиме выполняются не пошагово, т. е. быстро;

– «Reset» – сброс – установка в счетчик команд PC значения 0x000;

– «Breakpoints» – точка останова. Точки останова удобнее «расставлять» двойным щелчком левой клавиши мыши на левом поле окна исходного файла (см. рисунок 14). Таким же двойным щелчком левой клавиши мыши на символе точки останова его можно убрать.

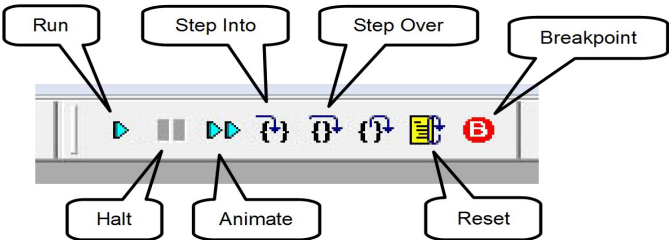


Рисунок 15 – Кнопки панели инструментов отладчика программного имитатора MPSIM

На команде, напротив которой установлен «Breakpoint», будет приостановлено моделирование. Возобновление моделирования осуществляется командами «Run», «Animate», «Step Into» и «Step Over». «Breakpoint» удобно использовать в тех случаях, когда необходимо большую часть программы выполнить быстро по команде «Run» и после «Breakpoint» пошагово, с помощью команд «Step Into» или «Step Over» внимательно проследить за выполнением небольшого фрагмента.

Окно «Watch». Кроме последовательности выполнения команд важно знать, как изменяются значения в регистрах специального и общего назначения виртуального микроконтроллера. Наблюдать за содержимым регистров специального и общего назначения можно с помощью окна «Watch». Это окно можно открыть из меню «View». На рисунке 16 показано окно «Watch», в котором отображаются значения регистра специального назначения PORTD, аккумулятора (рабочего регистра ядра микроконтроллера) WREG и определенной пользователем по адресу 0x20 в нулевом банке переменной ledset. Так как микроконтроллер оперирует только двоичными числами

адреса, значения (числа) и команды записываются двоичными числами. В приложении Б вы найдете необходимые пояснения по системам счисления и способам записи чисел при программировании на Ассемблере. Для удобства восприятия программы человеком адресам можно присвоить символьные имена, которые отображаются только в программе, записанной на Ассемблере. При ассемблировании программы все символьные имена заменяются на адреса, записанные двоичными цифрами. Добавление регистров в окне «Watch» осуществляется с помощью кнопок «Add SFR» и «Add Symbol» – «Добавить регистр специального назначения» и «Добавить регистр с использованием его символьного имени», соответственно. Если необходимо добавить регистр с известным адресом, можно выделить первую пустую строку в окне «Watch», установить курсор ввода текста в столбце «Address» и с клавиатуры набрать шестнадцатеричное значение адреса, после нажатия «Enter» появится новая строка с информацией о регистре.

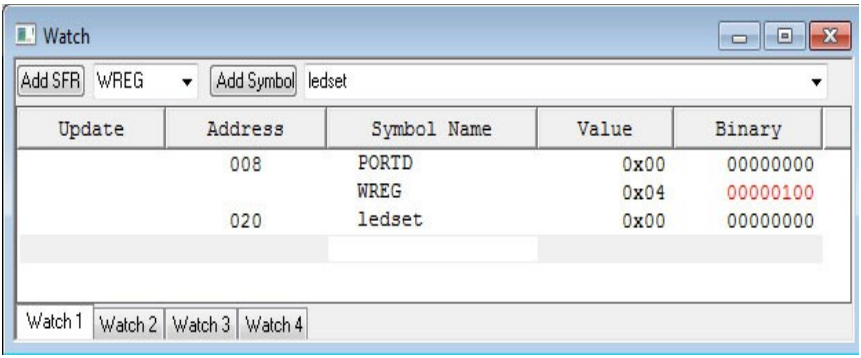


Рисунок 16 – Окно «Watch»

Информация о регистрах в окне «Watch» выведена в виде таблицы, в столбцах которой указаны:

– адрес регистра («Address») – имя регистра, «понятное» ядру микроконтроллера в виде числа;

– символьное имя регистра («Symbol Name») – имя регистра, заданное последовательностью букв, которое облегчает чтение программы программисту;

– значение («Value») – число, записанное в регистр в шестнадцатеричной форме;

– двоичное значение («Binary») – число, записанное в регистр в двоичной форме.

Если на заголовке таблицы окна «Watch» щелкнуть правой клавишей мыши, то выпадет список, с помощью которого можно добавить или удалить столбцы с информацией о регистре. Для того чтобы посмотреть изменение значений в регистрах, отображенных в окне «Watch», расположите окно ис-

ходного файла и окно «Watch» на мониторе таким образом, чтобы они не перекрывали друг друга. Нажимайте последовательно на кнопку «Step Into», и вы увидите, как стрелка-указатель последовательно перемещается по строкам программы. Понаблюдайте последовательность выполнения команд программы и изменение значений регистров в окне «Watch», сопоставьте значения в колонках «Value» и «Binary».

Для ускорения процесса анализа выполнения программы (это особенно важно для длинных программ) установите «Breakpoint» напротив последней команды программы goto change. Нажмите кнопку «Reset» при этом стрелка-указатель установится напротив команды, размещенной в **векторе сброса** микроконтроллера (0x000), что аналогично отключению питания на реальном микроконтроллере. Затем нажмите кнопку «Run», что аналогично включению питания на реальном микроконтроллере. Вы увидите, что симуляция выполнения программы остановится на точке останова «Breakpoint» после выполнения команды movwf PORTD, которая непосредственно изменяет значение в регистре PORTD. Снова нажимайте кнопку «Run», симуляция начнется с точки останова и продолжится до нее. Одновременно вы увидите, как изменяется значение в регистре PORTD. Когда во втором бите регистра PORTD установлена единица, на 21-й ножке микроконтроллера присутствует высокий уровень близкий к напряжению питания (уровень логической единицы), когда в этом бите установлен нуль, на 21-й ножке микроконтроллера присутствует низкий уровень близкий к напряжению общей шины (уровень логического нуля).

Окно «Stop Watch». Во многих случаях необходимо знать временные характеристики выполнения программы, иными словами время выполнения программы в целом и (или) ее отдельных частей. Для решения этой задачи можно использовать окно «Stop Watch». Это окно открывается из меню «Debugger», вид окна показан на рисунке 17. Для корректной симуляции временных характеристик необходимо установить частоту тактового генератора (меню «Debugger»/«Settings»/вкладка «Osc/Trace»).

В окне отображаются:

– «время» в количестве выполненных команд программы («Instruction Cycles»);

– время в микросекундах («Time (uSecs)»).

В колонке «Stopwatch» значение можно сбросить кнопкой «Zero». Сброс значений в колонке «Total Simulated» происходит при сбросе виртуального микроконтроллера кнопкой «Reset».

Разместите на мониторе окна исходного файла «Watch» и «Stop Watch» так, чтобы они не перекрывались. Нажимайте кнопку «Run» и по остановке симуляции на «Breakpoint» следите за изменением значений в окне «Stop Watch». Определите с помощью окон «Watch» и «Stop Watch» время присутствия на 21-й ножке микроконтроллера логического нуля и единицы.

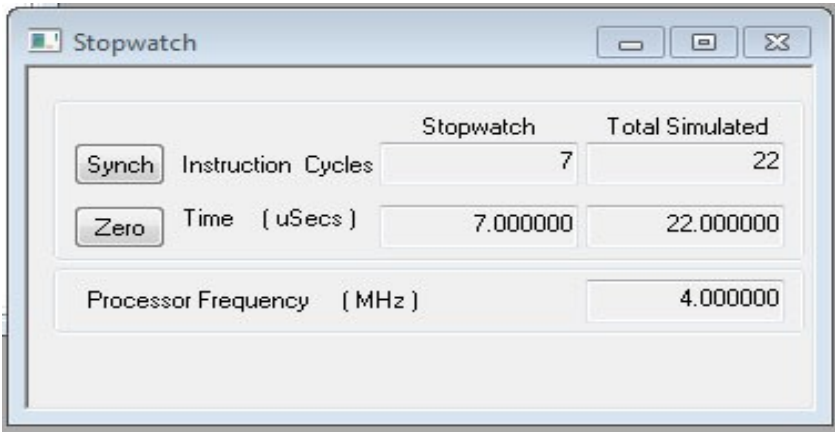


Рисунок 17 – Окно «Stop Watch»

При закрытии MPLAB должно появиться окно с запросом «Do you wish to save the workspase before closing?» («Вы хотите сохранить перед закрытием рабочую обстановку?»). Для того чтобы в следующем сеансе работы начать с того же места, необходимо нажать «Yes» («Да»).

1.3 Размещение программы в памяти микроконтроллера, символьные имена регистров ОЗУ

В этом подразделе вы можете найти сведения о директивах Ассемблера equ, org и командах Ассемблера goto, bcf, bsf, clrf, movlw и movwf.

Загрузите проект, с которым работали при изучении предыдущего подраздела. Для этого запустите MPLAB, в меню «Project» выберите пункт «Open». При этом откроется окно «Open Project» (рисунок 18). Используя традиционные для «Windows» элементы управления файлами откройте папку с сохраненным проектом и выберите ваш проект (файл с расширением mcp).

Директива equ. С помощью этой директивы можно назначить **регистрам** общего назначения символьные имена, которые существенно увеличивают наглядность программы на Ассемблере. В рассматриваемой программе A.1 адресу 0x20 в ОЗУ (первый регистр общего назначения в нулевом банке) присвоено имя ledset.

Строка кода выглядит так: ledset equ 0x20.

Просмотреть ОЗУ, содержащее регистры специального и общего назначения, можно с помощью меню «View» пункт «File Registers». При этом откроется окно «File Registers» (рисунок 19), в котором в виде таблицы отображается: шестнадцатеричное значение адреса регистра («Address»), шестнадцатеричное значение данных, хранимых в этом регистре («Hex»), десятичное значение

данных, хранимых в этом регистре («Decimal»), и символьное имя регистра («Symbol Name»). Для добавления или удаления столбцов таблицы необходимо щелкнуть правой кнопкой мыши на заголовке таблицы в окне «File Registers».

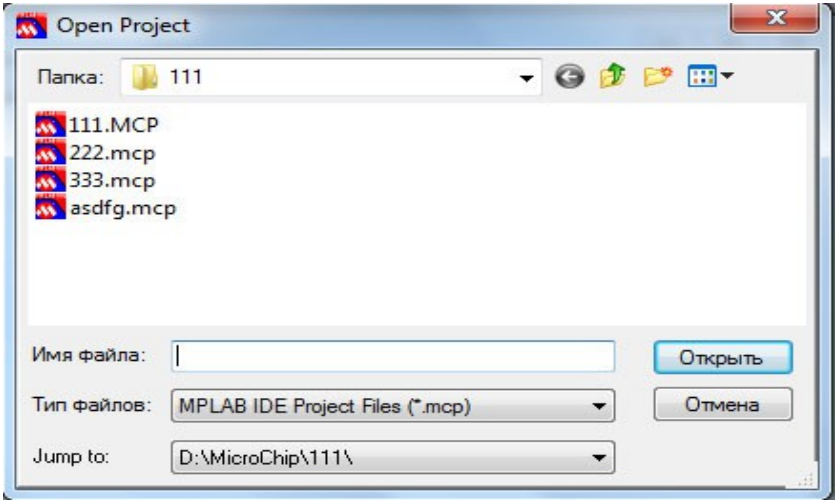


Рисунок 18 – Окно «Open Project»

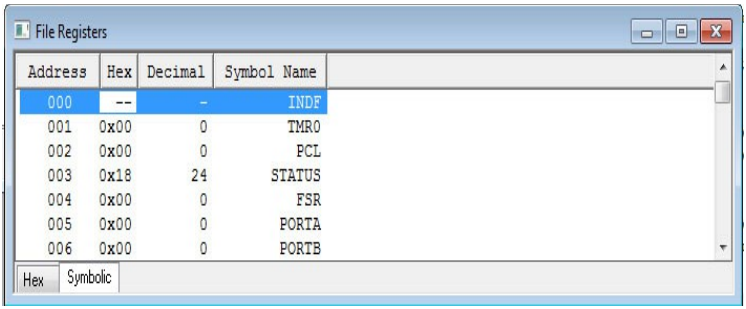


Рисунок 19 – Окно «File Registers»

С помощью вертикальной прокрутки найдите строку с символьным именем ledset. Посмотрите по какому адресу она находится. Измените значение адреса в строке ledset equ 0x20, например на 0x23. Выполните сборку проекта («Build All») и посмотрите какие изменения произошли в окне «File Registers». Выясните, повлияло ли изменение расположения переменной ledset в ОЗУ на выполнение программы в целом.

Директива org. С помощью этой директивы программист может определить расположение команд программы в памяти программ микроконтроллера. Внимательно почитайте комментарии к строкам программы, содержащим

директиву org. Просмотреть память программ можно с помощью меню «View» пункт «Program Memory». При этом откроется окно «Program Memory» (рисунок 20), в котором имеются три вкладки: «Opcode Hex» – шестнадцатеричные коды, «Machine» – машинные коды, «Symbolic» – символьные коды. Выберите вкладку «Machine». В данной вкладке окна «Program Memory» в виде таблицы отображается: десятичный номер строки («Line»), шестнадцатеричное значение адреса регистра памяти программ («Address»), шестнадцатеричное значение данных кода команды («Opcode»), мнемоническая запись команды с числовым отображением аргументов («Disassembly»).

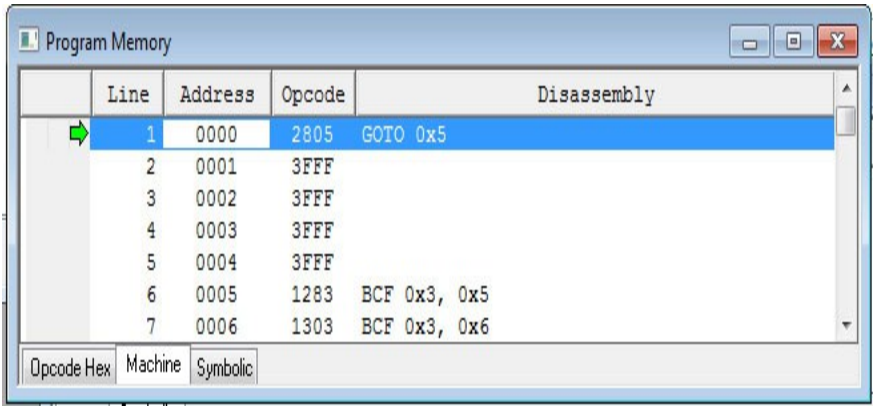


Рисунок 20 – Окно «Program Memory» вкладка «Machine»

Обратите внимание, что в **векторе сброса** (0x0000) директивой org 0x00 размещена команда goto start, при этом Ассемблером метка start заменена на адрес памяти программ (0x0005), на который должен быть осуществлен безусловный переход. С помощью директивы org 0x05 команда bcf STATUS,5 размещена по адресу памяти программ (0x0005). Как упоминалось ранее, при включении питания программа начинает выполняться с адреса 0x0000 памяти программ, в котором должна быть записана первая команда программы. В рассматриваемом примере по этому адресу размещена команда безусловного перехода goto. Само тело программы начинается с адреса 0x0005. Для чего выполнен «обход» адреса **0x0004 (вектор прерывания)** можно почитать в литературе [1] и [2].

На рисунке 20 символьное имя регистра специального назначения STATUS заменено на адрес в ОЗУ. С помощью окна «File Registers» сопоставьте значения адреса для регистра STATUS со значением адреса ОЗУ в команде bcf по адресу памяти программ 0x5.

Измените значение адреса в строке кода org 0x00 на 0x02, выполните сборку проекта («Build All») и посмотрите, какие изменения произошли в

окне «Program Memory». Восстановите значение адреса (0x00) в строке кода с директивой org, выполните сборку проекта («Build All»).

Команда goto. Команда безусловного перехода goto позволяет изменять порядок выполнения команд программы путем изменения значения в счетчике команд PC микроконтроллера. Произведите сброс («Reset») микроконтроллера и с помощью кнопки «Step Into» выполните команду goto start. В окне «Stop Watch» вы увидите, что данная команда выполнилась за два машинных цикла. Все команды, выполнение которых связанно с изменением значения в счетчике команд PC, выполняются за два машинных цикла. При дальнейшем пошаговом выполнении программы (кнопка «Step Into»), наблюдая за значениями в окне «Stop Watch», можно убедиться, что такие команды как bcf, bsf, clrf, movlw и movwf выполняются за один машинный цикл.

Команды bcf и bsf. Команды сброса/установки выбранного бита в регистре f. Например, команда bcf STATUS,5 сбрасывает пятый бит (записывает нуль в 5-й бит) регистра специального назначения STATUS, а команда bsf STATUS,5 устанавливает пятый бит (записывает единицу в 5-й бит) этого же регистра. Проследите с помощью окна «Watch» изменения 5-го и 6-го битов регистра STATUS при пошаговой симуляции.

Команда clrf. Команда сброса значения в регистре f. Например, команда clrf ledset сбрасывает (записывает нуль) в регистр, которому присвоено имя ledset.

Команды movlw и movwf. Команда movlw записывает константу, указанную в качестве аргумента, в рабочий регистр W (аккумулятор). Команда movwf записывает текущее значение из рабочего регистра W в регистр, указанный в качестве аргумента. Для того чтобы проследить логику работы команд movlw и movwf, добавьте в окно «Watch» регистры WREG (аккумулятор) и TRISD. При пошаговой симуляции проследите за изменением содержимого регистров W и TRISD.

1.4 Организация памяти данных и программ

В этом подразделе изложены сведения об организации памяти данных и программ микроконтроллера PIC16F877A, командах микроконтроллера movf f, d; movwf f; call.

В связи с ограничением разрядности регистров памяти программ микроконтроллера память данных и программ разделена на банки и страницы соответственно.

Банки памяти данных. Для адресации всего пространства памяти данных микроконтроллера PIC16F877A используется 9-битное двоичное число. Таким образом, можно адресовать максимум 512 регистров (от 0x0 до 0x1FF). Так как в теле команды отведено только 7 битов для записи адреса f

(например команда movwf f), память данных разделена на 4 банка: 0, 1, 2 и 3. Выбор банка осуществляется с помощью 5-го и 6-го битов регистра специального назначения STATUS, которые являются 7-м и 8-м битами адреса, «не поместившимися» в тело команды микроконтроллера (таблица 1). Регистр STATUS отображается на все четыре банка, поэтому биты RP1 и RP0 управления банками доступны из любого банка.

Обычно для управления банками используют бит-ориентированные команды bsf f, b и bcf f, b. Например, текущий банк 0, для того чтобы установить 3-й банк, необходимо выполнить последовательность команд:

```
bsf STATUS, 5
bsf STATUS, 6
```

Таблица 1 – **Управление банками (регистр STATUS)**

Бит	6	5
Символьное имя бита	RP1	RP0
Банк 0	0	0
Банк 1	0	1
Банк 2	1	0
Банк 3	1	1

Выполните задание по пересылке данных из одного регистра памяти данных в другой, причем эти регистры находятся в разных банках. Алгоритм:

– выбрать банк регистра-источника;

– переслать данные из регистра-источника в аккумулятор W;

– выбрать банк регистра-назначения;

– переслать данные из аккумулятора W в регистр-назначения.

Для того чтобы определить, в каком банке находится тот или иной регистр, необходимо воспользоваться картой памяти микроконтроллера (приложение В). Посмотрите внимательно текст программы А.1 и комментарии к нему в приложении А, найдите команды управления банками регистров памяти данных микроконтроллера.

Команда movf f, d выполняет перенос данных из регистра f в зависимости от значения параметра d. Если d=1, то данные из регистра f переносятся в этот же регистр f (используется для анализа данных в регистре f на нуль), если d=0 – данные из регистра f переносятся в аккумулятор W. **Команда movwf f** выполняет выполняет перенос данных из аккумулятора w в регистр f.

Сведения для углубленного изучения

Страницы памяти программ. Так как в теле команд goto и call на адрес памяти программ отведено 11 битов с помощью этих команд непосредственно не может быть адресовано все пространство памяти программ, так как адрес памяти программ содержит 13 битов. Аналогично памяти данных, которая разделена на банки, память

программ разделена на четыре страницы (приложение Г). Переходы в пределах одной страницы возможны с помощью команд goto и call. Если необходимо выйти за пределы текущей страницы перед выполнением команд goto и call, нужно соответствующим образом изменить 3-й и 4-й биты в регистре специального назначения PCLATH. Фактически в 3-м и 4-м битах регистра PCLATH содержатся два старших бита адреса памяти программ, «не поместившихся» в тело команд goto или call.

Создайте проект с текстом А.2 исходного файла. Выполните команду «Build All». Откройте, воспользовавшись меню «View», окно «Program Memory». Выберите в открывшемся окне вкладку «Machine» (см. рисунок 20). В этом окне вы увидите отображение памяти программ микроконтроллера с размещенными в ней командами.

В окне «Program Memory» видно, что по адресу 0x0005 размещена команда goto 0x0010, в то время как в исходном файле записана команда goto 0x0810. Если перевести указанные значения адреса в двоичную систему счисления:

0x10=b'0000000010000'

0x0810=b'0100000010000'

нетрудно заметить, что данные числа отличаются только 12-м битом, а 11 младших битов совершенно идентичны. Несмотря на то, что в исходном коде программы был указан полный адрес 0x0810, в тело команды записаны только 11 младших битов. Как нетрудно убедиться, при пошаговом выполнении программы безусловный переход выполняется на адрес 0x0010, на котором программой не предусмотрено никаких команд. То есть, произошло нарушение логики работы алгоритма. Для того чтобы перейти на адрес 0x0810, предусмотренный программистом, необходимо непосредственно перед командой goto 0x0810 выполнить команду bsf PCLATH,3 (установить 12-й бит адреса памяти программ). Внесите необходимые изменения в текст исходного файла и убедитесь, при пошаговом выполнении, что переход выполняется корректно из 0-й в 1-ю страницу памяти программ.

Особенности команды call. Данная команда используется для вызова подпрограммы. Выполнение команды call аналогично выполнению команды goto с той разницей, что при выполнении команды call 13-битный адрес команды, следующей за командой вызова подпрограммы, записывается в специальную область памяти микроконтроллера – **стек**. Адрес команды, следующей за командой вызова подпрограммы call, является адресом возврата из подпрограммы. В микроконтроллере PIC16F877A предусмотрен 8-уровневый стек. Работа стека аналогична работе магазина пистолета, только в стек загружаются не патроны, а 13-битные числа. Числа, загруженные в стек, выгружаются из него по принципу «первым зашел – последним вышел», точно так же, как и патроны из магазина пистолета. Работа стека проиллюстрирована на рисунке 21.

Выход из подпрограммы выполняется с помощью любой из команд return, retlw или retfie. При этом из вершины стека в счетчик команд PC загружается адрес возврата из подпрограммы – адрес команды, следующей за командой call. Таким образом, подпрограмма должна начинаться с адреса, указанного в команде вызова подпрограммы call, а заканчиваться командой return, retlw или retfie. Обратите внимание, что в теле команды call отведено только 11 битов для адреса перехода на подпрограмму. При обращении к

подпрограмме, расположенной на другой странице памяти программ, необходимо соответствующим образом, перед выполнением команды call, изменить 3-й и 4-й биты в регистре специального назначения PCLATH. При выходе из подпрограммы никаких дополнительных манипуляций не требуется, так как регистры стека, откуда загружается адрес возврата из подпрограммы, являются 13-битными.

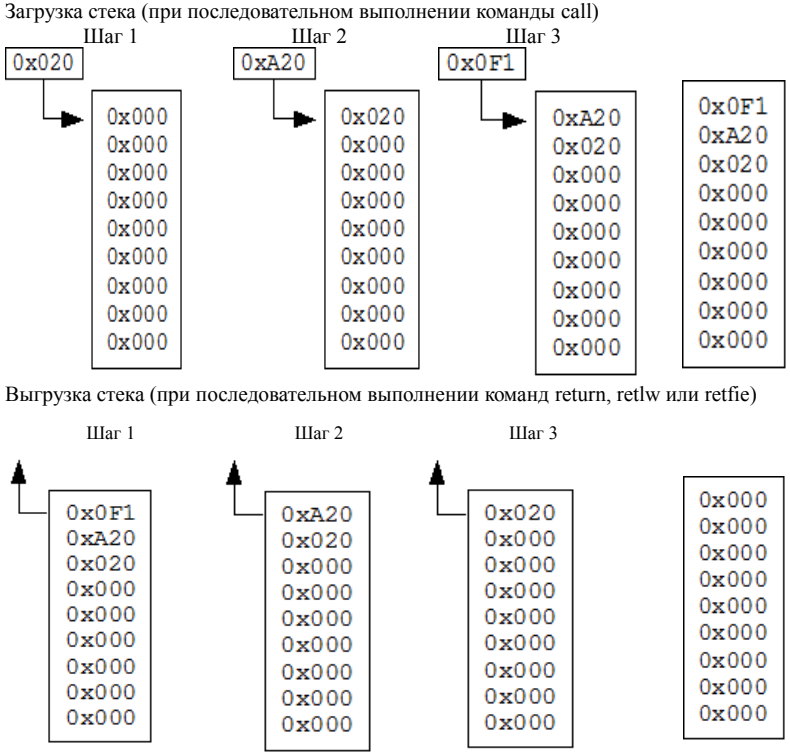


Рисунок 21 – Схема работы стека

1.5 Команды ветвления

В этом подразделе изложены сведения о командах ветвления btfsc и btfss. Изучив этот подраздел, вы научитесь писать простейшие программы с ветвлением.

Для реализации в программе блоков «принятия решения», связанных с анализом каких-либо данных и выполнения одного или другого (альтернативного) действия, используются команды btfsc и btfss. Краткое описание

этих и других команд можно найти в приложении Д. Рассмотрим выполнение этих команд на примере фрагмента программы, представленного в таблице 2. На рисунке 22 показана блок-схема этого фрагмента.

Таблица 2 – **Фрагмент программы, реализующей ветвление**

Метка	Мнемокод	Операнды	Комментарий
	btfsc	0x20,0	Проверка значения 0-го бита в регистре общего назначения по адресу 0x20. Если этот бит равен нулю, команда goto Label1 будет пропущена, и выполнится команда movlw 0xAA и далее. Если 0-й бит в регистре 0x20 не равен нулю, будет выполнена команда goto Label1
	goto	Label1	Безусловный переход на метку Label1, если 0-й бит в регистре 0x20 не равен нулю
	movlw	0xAA	Загрузка константы 0xAA в аккумулятор (W)
	movwf	0x21	Загрузка значения, помещенного в аккумулятор, в регистре общего назначения по адресу 0x21
	goto	Label2	Безусловный переход на метку Label2. Обход альтернативного блока команд
Label1	movlw	0xCC	Начало альтернативного блока команд. Загрузка константы 0xCC в аккумулятор (W)
	movwf	0x21	Загрузка значения, помещенного в аккумулятор (W), в регистре общего назначения по адресу 0x21
Label2	bsf	STATUS,5	
Примечание – В программе на Ассемблере комментарий должен быть отделен знаком «;», как это сделано в программах А.1 и А.2.			

В результате выполнения фрагмента программы, приведенного в таблице 2, в зависимости от состояния 0-го бита регистра 0x20, в регистр 0x21 будет загружено значение 0xAA или 0xCC. Отличие команды btfss заключается в том, что следующая за ней команда пропускается, если проверяемый бит равен единице.

Для тех, кто изучал английский язык, мнемонику команд btfsc и btfss легко запомнить, если обратиться к английскому варианту описания этих команд. Так, например, btfsc – Bit Test f, Skip if Clear (проверить бит в f, пропустить, если сброшен) или btfss – Bit Test f, Skip if Set (проверить бит в f, пропустить, если установлен).

На базе программ А.1 или А.2 и фрагмента из таблицы 2 создайте проект и убедитесь в его работоспособности. Не забудьте о правильном управлении банками при обращении к регистрам общего и специального назначения.

Воспользуйтесь пошаговой симуляцией выполнения программы и окном «Watch» для изменения значения бита, подвергающегося проверке, и отслеживания значения, загружаемого в регистр 0x21.

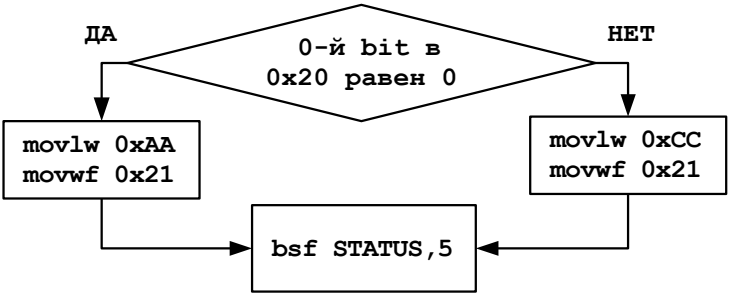


Рисунок 22 – Блок-схема фрагмента программы, приведенной в таблице 2

1.6 Организация циклов

В этом подразделе изложены сведения о командах decfsz и incfsz. Изучив этот подраздел, вы научитесь писать простейшие программы с циклами.

Часто для сокращения кода программы используют последовательности команд, повторяющихся заданное количество раз. Например, для организации временной задержки между выполнением двух фрагментов программы, генерирования последовательности заданного числа импульсов, выполнения операций умножения и деления путем последовательного сложения или вычитания чисел заданное количество раз, а также в других случаях.

Рассмотрим выполнение команды decfsz на примере фрагмента программы, представленного в таблице 3. На рисунке 23 показана блок-схема этого фрагмента.

Таблица 3 – **Фрагмент программы реализующей цикл**

Метка	Мнемокод	Операнды	Комментарий
	clrf	0x22	Сброс регистра 0x22 (загружаем в него нуль)
	movlw	0x0A	Загрузка константы 0x0A в аккумулятор (W)
	movwf	0x21	Загрузка значения, помещенного в аккумулятор, в регистре общего назначения по адресу 0x21. Регистр 0x21 – это счетчик
Label1	movlw	0x03	Загрузка константы 0x03 в аккумулятор (W). Цикл, повторяющийся 0x0A раз (начало цикла)

Окончание таблицы 3

Метка	Мнемокод	Операнды	Комментарий
	addwf	0x22,1	Сложить значение в аккумуляторе (W) с значением в регистре по адресу 0x22. Цикл, повторяющийся 0x0A раз
	decfsz	0x21,1	Из числа в регистре по адресу 0x21 вычитается 1, полученное значение сохраняется по адресу 0x21, затем это значение проверяется на 0. Если равно 0, тогда следующая команда пропускается – выход из цикла
	goto	Label1	Безусловный переход на метку Label1, если число в регистре 0x21 не равно нулю
	bsf	STATUS,5	Команда, следующая за циклом. Выполняется, когда значение в регистре 0x21 равно нулю
Примечание – В программе на Ассемблере комментарий должен быть отделен знаком «;», как это сделано в программах А.1 и А.2.			

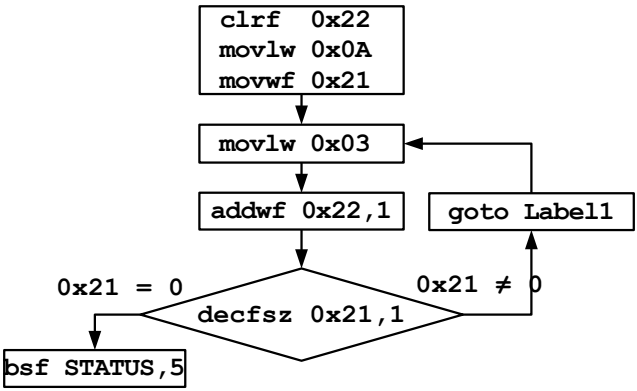


Рисунок 23 – Блок-схема фрагмента программы, приведенной в таблице 3

В результате выполнения фрагмента, представленного в таблице 3, в регистре 0x22 будет записано значение 0x1E (0x03 · 0x0A = 0x1E или 3·10 = 30). Команда incfsz f,d отличается от команды decfsz f,d тем, что при каждом выполнении к значению в регистре f прибавляется 1. Как при использовании команды incfsz f,d, может быть получено нулевое значение? При прибавлении единицы к 8-разрядному двоичному числу 0xFF (b'11111111') в результате получится 0x00 (b'00000000'). Единица, которая должна «появиться» в 9-м бите, просто «не поместилась» в 8-разрядном регистре микроконтроллера.

Если отнимать единицу от 8-разрядного двоичного числа 0x00 (b'00000000'), в результате будет получено число 0xFF (b'11111111').

На базе программ А.1 или А.2 и фрагмента из таблицы 3 создайте проект и убедитесь в его работоспособности. Не забудьте о правильном управлении банками при обращении к регистрам общего и специального назначения. Воспользуйтесь пошаговой симуляцией выполнения программы и окном «Watch» для отслеживания изменения значения в регистрах 0x21 и 0x22. Попробуйте изменить значения «сомножителей» на другие, посмотрите как это влияет на результат выполнения программы.

1.7 Ввод-вывод дискретных данных. Обработка нажатия кнопки

Изучив этот подраздел, вы получите представление об аппаратной реализации устройства на микроконтроллере, ознакомитесь со средствами моделирования «Simulator Logic Analyzer» и «Stimulus».

Рассмотрим принципиальную схему простейшего микроконтроллерного устройства, представленную на рисунке 24. Устройство состоит из микроконтроллера DD1, стабилизатора напряжения питания микроконтроллера DA1, индикатора напряжения питания HL1, цепи питания входа сброса микроконтроллера (MCLR) R1 и VD1, времязадающей цепи тактового генератора микроконтроллера C1, C2 и ZQ1, светодиода, управляемого микроконтроллером HL2, кнопки управления уровнем напряжения на входе микроконтроллера SB1, балластных резисторов R2, R3 и R4, разъема для подключения программатора XP2, перемычки, удаляемой при загрузке программы в микроконтроллер, XS1.

Как уже упоминалось, в микроконтроллерах числа представлены в **двоичной системе** счисления. Эта система счисления выбрана по причине простой физической реализации двоичных цифр (нуля и единицы) в виде двух хорошо различимых состояний аппаратной части цифровых устройств. В рассматриваемом нами микроконтроллере нуль и единица кодируются уровнем напряжения. **Логическая единица** кодируется напряжением в интервале от 2 до 5 В, а **логический нуль** – от 0 до 0,8 В [2]. Существуют и другие способы представления двоичных чисел, но их в данном пособии рассматривать не будем.

Кварц ZQ1 определяет частоту колебаний **тактового генератора** микроконтроллера. **Сброс** микроконтроллера происходит при низком уровне напряжения (близком к нулю или логическом нуле) на входе MCLR. В данной схеме R1 и VD1 подают напряжение питания (высокий уровень или логическая единица) на этот вход, что обеспечивает нормальную работу микроконтроллера при включенном питании (+5 В). Напряжение питания данного микроконтроллера должно лежать в пределах от 4,5 до 5,5 В [2]. Это

напряжение обеспечивает микросхема стабилизатора напряжения DA1, на вход которой подается не стабилизированное напряжение от 7 до 9 В. Резисторы R3 и R4 необходимы для ограничения тока светодиодов HL1 и HL2. Когда кнопка SB1 не нажата – на 38-ю ножку микроконтроллера через резистор R2 подается уровень **логической единицы**. При нажатии кнопки SB1 38-я ножка соединяется с общим проводом, потенциал которого соответствует уровню **логического нуля**.

Рассмотрим формирование **логических уровней (нуль или единица)** напряжения на входе в микроконтроллер более подробно. На рисунке 25 показана эквивалентная схема ножки микроконтроллера, настроенной как вход. В режиме входа сопротивление внутренних цепей микроконтроллера Rport очень велико (на рисунке условно показано 1000 кОм). При разомкнутом состоянии контактов кнопки SB1 сопротивление Rport и резистор R1 образуют делитель напряжения. Напряжение на 38-й ножке Uport определяется по формуле

$$Uport = Rport \frac{Unum}{(Rport + R1)} \quad . \qquad (1)$$

Подставим номиналы, тогда

$$Uport = 1000 \cdot \frac{5}{(1000 + 5)} = 4{,}98 \text{ В}.$$

Напряжение *Uport* = 4,98 В соответствует уровню логической единицы. Если кнопка SB1 нажата (сопротивление замкнутых контактов от единиц до десятков ом), малое сопротивление кнопки шунтирует внутренне сопротивление входного порта микроконтроллера. Аналогично при нажатой кнопке напряжение на 38-й ножке микроконтроллера можно рассчитать по формуле

$$Uport = Rsb1 \frac{Unum}{(Rsb1 + R1)} \quad , \qquad (2)$$

где *Rsb*1 – сопротивление замкнутых контактов кнопки SB1, примем

$$Rsb1 = 100 \text{ Ом}.$$

Напряжение *Uport* при замкнутых контактах

$$Uport = 0{,}1 \cdot \frac{5}{(0{,}1 + 5)} = 0{,}098 \text{ В}.$$

Напряжение *Uport* = 0,098 В соответствует уровню логического нуля.

Представленная на рисунке 24 схема позволяет реализовать ряд простых программ, например, если кнопка SB1 не нажата – светодиод HL2 горит, если SB1 нажата – светодиод HL2 не горит. Составим словесное описание алгоритма:

1 Настроить 38-ю ножку микроконтроллера (RB5) как вход, а 37-ю (RB4) – как выход (регистр TRISB).

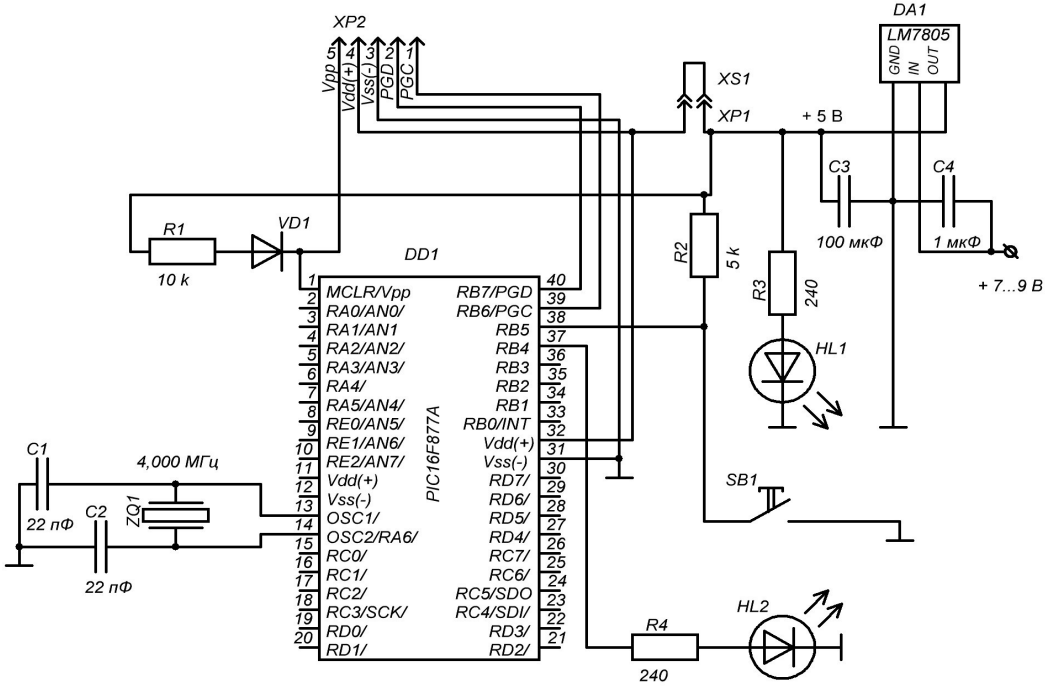


Рисунок 24 – Принципиальная схема микроконтроллерного устройства на базе PIC16F877A

2 Проверить состояние 5-го бита (RB5) регистра PORTB. Если RB5 = 1, установить в 4-м бите (RB4) регистра PORTB единицу, если RB5 = 0, установить в 4-м бите (RB4) регистра PORTB нуль (сбросить этот бит).

3 Повторить действие 2.

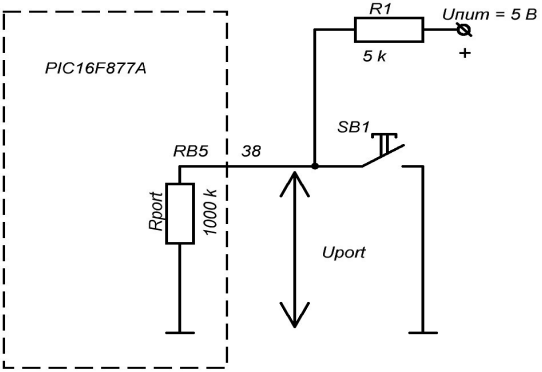


Рисунок 25 – Эквивалентная схема
входного порта микроконтроллера

Светодиод HL2 будет светиться в том случае, если на 37-й ножке микроконтроллера будет установлено напряжение, соответствующее логической единице (в рассматриваемом примере около 4,8 В). Для этого в 4-м бите (RB4) регистра PORTB надо установить единицу, о чем и было написано выше. Анализ и управление логическими уровнями на ножках микроконтроллера мы реализуем с помощью периферийного модуля portb.

Для управления **периферийным модулем** portb используются регистры специального назначения PORTB и TRISB, а также 7-й бит (-RBPU) регистра OPTION_REG. Не вдаваясь в подробности, которые можно изучить руководствуясь [1] и [2] (подтягивающие резисторы на входах portb), отметим только то, что для схемы на рисунке 24 7-й бит регистра OPTION_REG должен быть установлен (равен единице). Работу с регистрами PORTB и TRISB, упоминавшихся выше, рассмотрим более подробно.

Напомню о том, что запись portb обозначает периферийный модуль, а запись PORTB – адрес регистра специального назначения. Периферийный модуль portb микроконтроллера PIC16F877A является 8-разрядным двунаправленным (может работать как вход и как выход) **портом ввода-вывода** (33-40-я ножки микроконтроллера). Регистр PORTB служит для обмена данными, а регистр TRISB для настройки направления передачи данных. В рассматриваемом примере в регистр TRISB необходимо загрузить константу b'11101111', при этом 37-я ножка (RB4) будет настроена как выход, а остальные (в том числе и 38-я ножка), как входы. При нажатой кнопке SB1 в 5-м бите (RB5) регистра PORTB будет аппаратно (без участия программы) за-

гружаться нуль, а при ненажатой кнопке – загружаться единица. Значение в этом бите (RB5) может быть прочитано командами btfsc и btfss, которые позволяют анализировать значение по указанному биту. В зависимости от считанного с 38-й ножки значения (состояние бита RB5) можно с помощью команд bsf и bcf установить или сбросить (1 или 0) 4-й бит (RB4) регистра PORTB, что приведет к изменению уровня напряжения на 37-й ножке микроконтроллера. Соответствие битов регистра PORTB и ножек микроконтроллера PIC16F877A показано на рисунке 3. В приложении А приведена программа А.3, реализующая описанный выше алгоритм. Внимательно изучите текст программы, прочитайте все комментарии.

Создайте проект, в текстовое окно исходного файла скопируйте программу А.3. Соберите проект (меню «Project»/«Build All»). Если сборка удалась приступите к симуляции. Для отображения уровней напряжения на RB4 и RB5 воспользуйтесь инструментом «Simulator Logic Analyzer». В меню «View» пункт «Simulator Logic Analyzer», в открывшемся окне «Logic Analyzer» (рисунок 26) нажмите кнопку «Channels», при этом откроется окно «Configure Channels». В окне «Configure Channels», в списке «Available Signals» выберите RB4 и нажмите кнопку «Add», то же проделайте для RB5. Если все правильно сделано, в списке «Selected Signal(s)» появятся каналы RB4 и RB5. Нажмите «OK». В окне «Logic Analyzer» должны появиться выбранные каналы RB4 и RB5.

Для имитации внешнего напряжения, проложенного к ножкам микроконтроллера, используется инструмент «Stimulus». В меню «Debugger» пункт «Stimulus», а затем «New Workbook», откроется окно «Stimulus» (рисунок 27). Щелкните левой кнопкой мыши на заголовке колонки «Pin/SFR», откройте список и выберите RB5. Щелкните левой кнопкой мыши на заголовке колонки «Action», откройте список и выберите пункт «Toggle» («Переключатель»). Если все сделано правильно, окно «Stimulus» должно иметь вид, как на рисунке 27. Разместите окна «Stimulus», «Logic Analyzer» и текстовое окно исходного файла так, чтобы они не перекрывались.

Убедитесь, что виртуальный микроконтроллер находится в состоянии сброса (стрелка-указатель должна находиться на команде, размещенной в 0x0000 адресе памяти программ) и нажмите кнопку «Animate» (см. рисунок 15). В окне «Logic Analyzer» «побегут» линии графиков логических уровней на ножках RB4 и RB5 микроконтроллера. Щелкните левой кнопкой мыши в окне «Stimulus» на знаке «>» в колонке «Fire», рядом с RB5. Если все правильно работает, в окне «Logic Analyzer» будут видны соответствующие изменения логических уровней на ножках RB4 и RB5.

Попробуйте изменить логику работы программы, например, светодиод HL1 при нажатии кнопки SB1 загорается, или при нажатии кнопки SB1 светодиод HL1 начинает мигать.

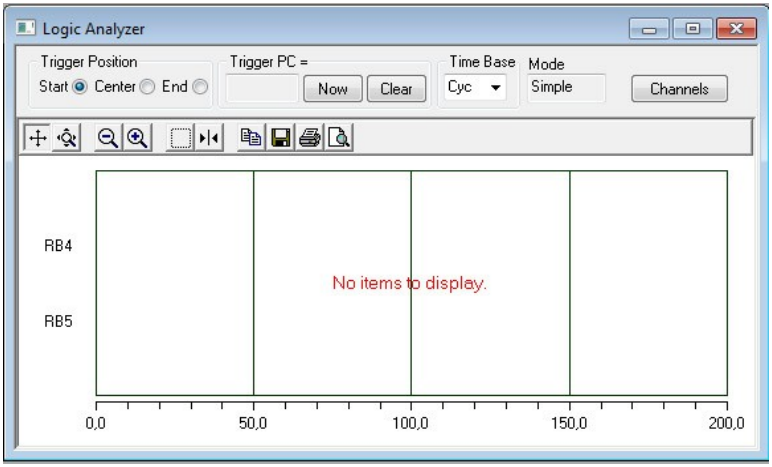


Рисунок 26 – Вид окна виртуального логического анализатора «Logic Analyzer»

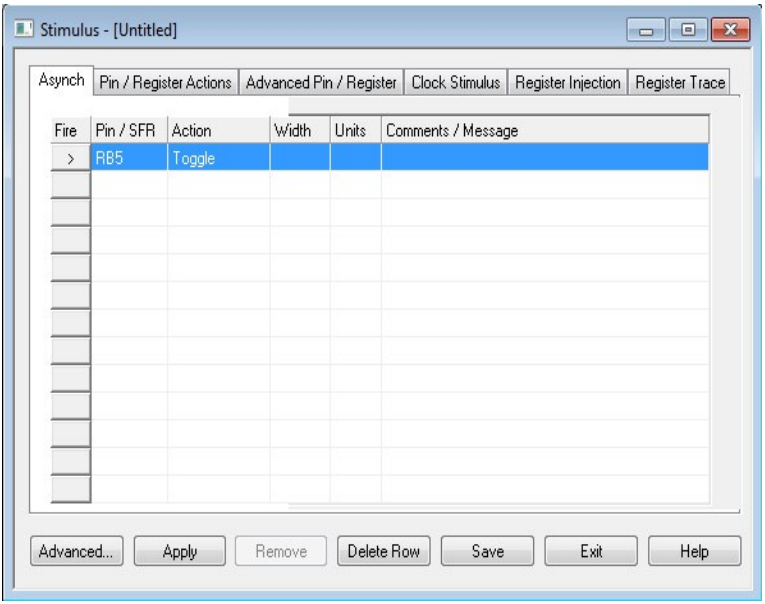


Рисунок 27 – Вид окна установки входных сигналов «Stimulus»

2 ПРАКТИКУМ

В данном практикуме приведены шесть лабораторных работ, выполнение которых позволит вам освоить основные приемы работы в интегрированной среде разработки MPLAB, изучить организацию памяти микроконтроллеров среднего семейства компании Microchip, систему их команд и научиться писать простейшие программы на Ассемблере. Для успешной работы вам необходимо установить на своем компьютере MPLAB IDE v8, который можно скачать на сайте www.microchip.com.

Добросовестное выполнение лабораторных работ по дисциплине «Программируемые цифровые устройства» (равно как и по другим дисциплинам) позволит вам понять и усвоить материал курса. Это будет способствовать применению полученных знаний при изучении дисциплин: «Электрооборудование тепловозов», «Автоматика и автоматизация тепловозов», и, естественно, в дальнейшем успешной профессиональной деятельности. Для подкрепления своих слов приведу цитату из [3] «… невозможно научиться играть на музыкальном инструменте, плавать, управлять автомобилем только прочитав учебник и(или) прослушав лекцию – необходима практика». Точно также для освоения инженерной специальности необходима практика, которая заключается в САМОСТОЯТЕЛЬНОМ выполнении учебных заданий всех видов, в том числе и лабораторных работ. Не следует обманывать себя, не выполняя учебных заданий – это обернется, в конечном итоге, невозможностью эффективно работать по специальности.

При выполнении лабораторных работ вам постоянно придется обращаться к материалу в первом разделе данного пособия, который содержит все необходимые сведения и справочные данные.

Отчет по лабораторной работе должен быть оформлен с соблюдением требований ГОСТ 2.105-95 (приложение Ж). Блок-схема алгоритма оформляется как рисунок. Программа на Ассемблере к каждой строке должна иметь комментарий. Допускается оформление лабораторных работ в тонкой ученической тетради.

Если лабораторной работой предусмотрено индивидуальное задание, то его полный текст необходимо привести в отчете. При написании вывода не следует переписывать цель лабораторной работы. В выводе необходимо отразить результаты анализа результатов, полученных в ходе выполнения работы.

Лабораторная работа № 1

СОЗДАНИЕ ПРОЕКТА В СРЕДЕ MPLAB

Цель. Приобрести практические навыки работы с интегрированной средой разработки MPLAB. Научиться создавать проект, изучить основные элементы интерфейса MPLAB.

Порядок выполнения работы

1 Создайте проект.

2 Запустите моделирование (симуляцию) выполнения программы микроконтроллера.

3 Просмотрите данные в окне «Watch».

4 Проверьте временные характеристики выполнения программы в окне «Stop Watch».

Содержание отчета

1 Краткое описание назначения интегрированной среды разработки MPLAB.

2 Перечень основных пунктов меню и команд, используемых для создания проекта.

3 Вывод.

Контрольные вопросы

1 Из каких основных этапов состоит процесс создания проекта в интегрированной среде разработки MPLAB?

2 Где находится пункт «Configuration Bits»?

3 Где устанавливается частота виртуального тактового генератора?

4 С помощью каких кнопок (команд) можно управлять процессом моделирования (симуляции) выполнения программы микроконтроллера?

5 Поясните назначение окна «Watch».

6 Поясните назначение окна «Stop Watch».

Лабораторная работа № 2

РАЗМЕЩЕНИЕ ПРОГРАММЫ В ПАМЯТИ МИКРОКОНТРОЛЛЕРА

Цель. Изучить директивы Ассемблера equ, org и команды Ассемблера goto, bcf, bsf, clrf, movlw и movwf.

Порядок выполнения работы

1 Загрузите проект, созданный при выполнении лабораторной работы № 1.

2 Через меню «View» (пункт «File Registers») просмотрите содержимое ОЗУ.

3 Поменяйте адрес в директиве equ и посмотрите какие изменение произошли в ОЗУ микроконтроллера.

4 Воспользуйтесь директивой org, чтобы изменить расположение программы в памяти микроконтроллера. Просмотрите изменения в окне «Program Memory».

5 С помощью окна «Watch» пронаблюдайте за выполнением команд bsf, bcf, clrf, movlw и movwf.

6 В пошаговом режиме выполнения программы пронаблюдайте за выполнением команды goto. Обратите внимание, за какое число машинных циклов выполняется эта команда.

Содержание отчета

1 Текст программы и ее блок-схема.

2 Краткое описание назначения директив макроассемблера org и equ.

3 Краткое описание команд goto, bsf, bcf, clrf, movlw и movwf.

4 Вывод.

Контрольные вопросы

1 Как можно указать расположение программы в памяти микроконтроллера?

2 Каким образом можно задать символьные имена регистрам, битам и константам?

3 Для чего предназначена команда goto? Какие особенности выполнения этой команды?

4 Для чего предназначены команды bsf и bcf?

5 Для чего предназначены команды movlw и movwf?

6 Для чего предназначена команда clrf?

7 Каким образом можно просмотреть содержимое памяти данных и программ микроконтроллера?

8 Как выполнить в режиме симуляции сброс и пошаговое выполнение программы?

9 Как и для чего применяют инструмент «Breakpoint»?

Лабораторная работа № 3

ПЕРЕСЫЛКА ДАННЫХ

Цель. Изучить организацию памяти данных и программ микроконтроллера PIC16F877A, команд микроконтроллера movf f, d и call.

Порядок выполнения работы

1 Создайте проект.

2 Напишите программу, выполняющую пересылку данных из одного регистра памяти данных в другой согласно варианту (таблица 1). Во время эмуляции работы программы в среде MPLAB воспользуйтесь окном Watch для того, чтобы убедиться в правильной работе написанной программы.

Таблица 1 – **Варианты задания для лабораторной работы № 3**

Вариант	Адреса	Вариант	Адреса	Вариант	Адреса
1	0x27→0x1C3	11	0x1BE→0xA3	21	0x1AA→0xAA
2	0x72→0x12D	12	0x51→0x1DA	22	0x15E→0x6B
3	0x127→0x1AF	13	0x66→0xC4	23	0x1E8→0x3F
4	0xDD→0x1E3	14	0x3F→0x14A	24	0x1CF→0x2A
5	0x1E2→0xE1	15	0x1A9→0x133	25	0x155→0xB0
6	0x15D→0x13C	16	0x12E→0x1A9	26	0x1E1→0x151
7	0x121→0x6A	17	0xA7→0x23	27	0x1CC→0xC2
8	0x77→0xEE	18	0x4F→0x1BB	28	0x1DE→0xE4
9	0xEA→0x1E5	19	0x128→0x5C	29	0x15F→0x7B
10	0xB2→0x3A	20	0x70→0xDE	30	0x122→0x9F

3 В режиме симуляции убедитесь в том, что написанная программа работает правильно.

Содержание отчета

1 Текст программы (с комментариями к каждой строке) и ее блок-схема.

2 Краткое описание команд movf f, d и call.

3 Вывод.

Контрольные вопросы

1 Опишите процедуру переноса данных из одного регистра памяти данных микроконтроллера в другой. Какую роль в этом играет рабочий регистр (аккумулятор)?

2 Банки памяти данных микроконтроллеров Microchip. Управление банками.

3 Для чего предназначена команда movf f, d?

4 Дайте определение понятиям данные (значение) и адрес (имя регистра). Приведите примеры использования данных и адреса в качестве параметров команд.

5 Напишите фрагмент кода перехода из одного заданного банка в другой.

6 Напишите фрагмент кода для переноса данных из одного регистра памяти данных в другой.

Лабораторная работа № 4

РАЗРАБОТКА АЛГОРИТМА С ВЕТВЛЕНИЕМ

Цель. Изучить команды ветвления btfsc и btfss.

Порядок выполнения работы

1 Создайте проект.

2 Напишите программу на Ассемблере, реализующую следующий алгоритм: если BIT бит по адресу ADR0 памяти данных равен N, то в ADR1 загружается значение Z0, если BIT бит по адресу ADR0 не равен N, то в ADR1 загружается значение Z1. Значения BIT, ADR0, ADR1, N, Z0 и Z1 выбираются в соответствии с вариантом из таблицы 1.

3 В режиме симуляции убедитесь в том, что написанная программа работает правильно.

Содержание отчета

1 Текст программы (с комментариями к каждой строке) и ее блок-схема.

2 Краткое описание команд btfsc и btfss.

3 Вывод.

Контрольные вопросы

1 В чем отличие команд btfsc и btfss?

2 Приведите блок-схему ветвления в общем виде с применением команд btfsc и btfss.

3 Напишите фрагмент кода с использованием команды btfsc.

4 Напишите фрагмент кода с использованием команды btfss.

Таблица 1 – **Варианты задания для лабораторной работы № 4**

Вариант	BIT	ADR0	ADR1	N	Z0	Z1	Вариант	BIT	ADR0	ADR1	N	Z0	Z1
1	5	0x045	0x1A6	0	224	31	8	1	0x067	0x0A8	0	213	42
2	2	0x069	0x0A1	1	71	184	9	4	0x053	0x0A5	0	146	109
3	1	0x120	0x0AD	1	156	99	10	2	0x140	0x0AE	0	225	30
4	1	0x057	0x1D0	1	178	77	11	4	0x05E	0x1C7	1	152	103
5	6	0x16D	0x1B7	0	124	131	12	5	0x065	0x1B8	0	34	221
6	5	0x028	0x1BE	0	129	126	13	3	0x056	0x0C7	0	40	215
7	6	0x056	0x1C0	1	97	158	14	7	0x14C	0x1DB	0	202	53

Окончание таблицы 1

Вариант	BIT	ADR0	ADR1	N	Z0	Z1	Вариант	BIT	ADR0	ADR1	N	Z0	Z1
15	4	0x036	0x1E6	0	166	89	26	4	0x16F	0x1C1	0	199	56
16	1	0x043	0x1B1	1	135	120	27	6	0x042	0x1B2	0	117	138
17	7	0x034	0x0C9	0	122	133	28	2	0x028	0x1CD	1	111	144
18	3	0x16D	0x1A0	0	85	170	29	1	0x039	0x1C8	0	179	76
19	1	0x160	0x1D3	1	190	65	30	6	0x160	0x1BB	0	233	22
20	6	0x16F	0x0B1	1	34	221	31	4	0x166	0x1D7	0	231	24
21	3	0x13E	0x0D7	0	171	84	32	5	0x16B	0x0AC	1	55	200
22	4	0x037	0x1B4	0	85	170	33	4	0x120	0x1C3	1	171	84
23	7	0x042	0x0BF	1	93	162	34	2	0x04C	0x1C5	0	65	190
24	5	0x12A	0x0E5	0	230	25	35	5	0x16A	0x0D1	0	217	38
25	1	0x044	0x1BA	1	136	119	36	2	0x051	0x0D3	1	48	207

Лабораторная работа № 5

ЦИКЛИЧЕСКИЙ АЛГОРИТМ

Цель. Изучить команды Ассемблера decfsz, incfsz и addwf.

Порядок выполнения работы

1 Создайте проект.

2 Выполните индивидуальное задание. Напишите программу на Ассемблере, реализующую следующий алгоритм: счетчик разместить по адресу ADRI памяти данных, число повторений в цикле N, метка начала цикла LABEL, команды COM, повторяющиеся N раз. Значения ADRI, N, LABEL и COM выбираются в соответствии с вариантом из таблицы 1.

Таблица 1 – **Варианты задания для лабораторной работы № 5**

Вариант	ADR1	N	LABEL	COM	Вариант	ADR1	N	LABEL	COM
1	0x045	36	REPEAT	movlw 0x19 movwf 0x0BC	3	0x14F	48	LOC1	movlw 0x6B movwf 0x0C7
2	0x1ED	46	REPEAT	movlw 0x1A movwf 0x13D	4	0x0C2	29	POINT	movlw 0xD6 movwf 0x160

Окончание таблицы 1

Вариант	ADR1	N	LABEL	COM	Вариант	ADR1	N	LABEL	COM
5	0x060	14	REPEAT	movlw 0xE1 movwf 0x0B2	21	0x137	48	METKA	movlw 0x5C movwf 0x0E8
6	0x1C7	59	REPEAT	movlw 0x82 movwf 0x121	22	0x1C6	21	POINT	movlw 0xC1 movwf 0x16F
7	0x049	72	METKA	movlw 0x21 movwf 0x1C3	23	0x033	37	LOC1	movlw 0x3D movwf 0x1A6
8	0x1B7	59	POINT	movlw 0x30 movwf 0x160	24	0x1D1	15	POINT	movlw 0xE5 movwf 0x14E
9	0x16D	49	POINT	movlw 0x73 movwf 0x0C0	25	0x052	70	METKA	movlw 0x39 movwf 0x0C5
10	0x0AD	36	LOC1	movlw 0xE1 movwf 0x03E	26	0x1CA	31	LOC1	movlw 0xF movwf 0x162
11	0x134	55	METKA	movlw 0x35 movwf 0x1B4	27	0x035	31	REPEAT	movlw 0x53 movwf 0x0C8
12	0x1E5	42	REPEAT	movlw 0x74 movwf 0x158	28	0x1EF	28	LOC1	movlw 0xEA movwf 0x16B
13	0x149	20	METKA	movlw 0x19 movwf 0x1DD	29	0x059	30	LOC1	movlw 0xEE movwf 0x0DD
14	0x1E6	65	LOC1	movlw 0xD1 movwf 0x13D	30	0x1E7	51	LOC1	movlw 0xDB movwf 0x063
15	0x030	38	LOC1	movlw 0x47 movwf 0x0E3	31	0x020	74	LOC1	movlw 0x70 movwf 0x0D7
16	0x0EF	10	LOC1	movlw 0xA5 movwf 0x14E	32	0x1D8	67	REPEAT	movlw 0xC0 movwf 0x051
17	0x127	20	POINT	movlw 0x19 movwf 0x1A7	33	0x061	36	LOC1	movlw 0xEC movwf 0x0EE
18	0x1E4	29	METKA	movlw 0x66 movwf 0x047	34	0x1C6	20	METKA	movlw 0x7F movwf 0x14F
19	0x02A	26	REPEAT	movlw 0x40 movwf 0x1B8	35	0x124	65	LOC1	movlw 0x13 movwf 0x1D8
20	0x1D9	35	METKA	movlw 0xF7 movwf 0x060	36	0x0D6	24	METKA	movlw 0x61 movwf 0x02C

3 В режиме симуляции убедитесь в том, что написанная программа работает правильно.

Содержание отчета

1 Текст программы (с комментариями к каждой строке) и ее блок-схема.
2 Краткое описание команд decfsz, incfsz и addwf.
3 Вывод.

Контрольные вопросы

1 В чем отличие команд decfsz и incfsz?
2 Для чего предназначена команда addwf?
3 Напишите фрагмент кода с использованием команды addwf.
4 Напишите фрагмент кода с использованием команды decfsz.
5 Напишите фрагмент кода с использованием команды incfsz.

Лабораторная работа № 6

ОБРАБОТКА НАЖАТИЯ КНОПКИ

Цель. Изучить азы аппаратной реализации устройства на микроконтроллере. Знакомство со средствами моделирования «Simulator Logic Analyzer» и «Stimulus».

Порядок выполнения работы

1 Ознакомьтесь с принципиальной схемой демонстрационной платы на базе микроконтроллера PIC16F877A.
2 Создайте проект.
3 Напишите программу, реализующую обработку нажатия на кнопку, в соответствии с индивидуальным заданием.
4 В режиме симуляции убедитесь в том, что написанная программа работает правильно.

Содержание отчета

1 Текст программы (с комментариями к каждой строке) и ее блок-схема.
2 Вывод.

Контрольные вопросы

1 Поясните, каким образом аппаратно формируется уровень логического нуля и единицы?
2 Из каких элементов состоит времязадающая цепь тактового генератора микроконтроллера?
3 Из каких элементов состоит цепь аппаратного сброса микроконтроллера?
4 Какие регистры специального назначения используются для управления дискретным (цифровым) портом ввода-вывода portb микроконтроллера PIC16F877A?
5 Напишите фрагмент кода настройки заданных ножек микроконтроллера PIC16F877A как входы и выходы.
6 Напишите фрагмент кода, устанавливающего логический нуль или единицу на заданной ножке микроконтроллера PIC16F877A.

Список литературы

1 **Уилмсхерст, Т.** Разработка встроенных систем с помощью микроконтроллеров PIC. Принципы и практические примеры / Т. Уилмсхерст; пер. англ. – Киев: МК-Пресс, СПб.: Корона-ВЕК, 2008. – 544 с.

2 Документация. PIC – полные переводы даташитов на некоторые серии микроконтроллеров. [Электронный ресурс]. – Режим доступа: http://www.microchip.ru/files/d-sheets-rus/pic16f87x.pdf. – Дата доступа: 19.08.2013 Загл. с экрана.

3 **Страуструп, Б.** Программирование. Принципы и практика использования C++ / Б. Страуструп; пер. англ. – М.: ООО «И. Д. Вильямс», 2011. – 1235 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ПРИЛОЖЕНИЕ А
(*обязательное*)

ПРИМЕРЫ ПРОГРАММ НА АССЕМБЛЕРЕ

в электронном виде на http://eot-ttd.blog.tut.by

Программа А.1

```
;дисциплина "Программируемые цифровые устройства"
;Лабораторная работа № 2
;прямоугольные импульсы на RD2 (21-я ножка микроконтроллера)

       list p=16f877a
       include <p16f877a.inc>

ledset equ  0x20     ;используем директиву Ассемблера для назначения имени ячейки
                     0x20 в нулевом банке регистров общего назначения

       org 0x00      ;директива Ассемблера «поместить команду goto start»
                     ; в 0x00 адрес памяти программ (в вектор сброса)
       goto start    ;безусловный переход на метку start
       org 0x05      ;директива Ассемблера «поместить команду bcf STATUS,5 в 0x05»
                     ;адрес памяти программ (следующий за вектором прерываний)

start                ;метка
      bcf STATUS,5   ;сброс 5-го бита в регистре STATUS
      bcf STATUS,6   ;сброс 6-го бита в регистре STATUS
                     ;в результате переход в банк 0
      clrf PORTD     ;сброс защелки порт D - все нули
      clrf ledset    ;сброс переменой ledset - все нули
      bsf STATUS,5   ;установить 5-й бит в регистре STATUS (5-й бит = 1)
                     ;в результате переход в банк 1
      movlw b'11111011'  ;константу b'11111011' переслать в аккумулятор (рабочий
                     ;регистр W)
      movwf TRISD    ;содержимое аккумулятора переслать в регистр TRISD
                     ;в результате настройка направления передачи сигналов на
                     ;выводах порт D. вывод RD2 (21 ножка) настроен как выход,
                     ;остальные - вход
      bcf TRISE,4    ;настройка порт D в режим цифровых каналов ввода-вывода
change               ;метка
      bcf STATUS,5   ;переход в банк 0
      movlw b'00000100' константу b'00000100' переслать в аккумулятор
                     ;(рабочий регистр W)
     xorwf ledset,1  ;выполнить побитное исключающее ИЛИ между W и ledset,
                     ;результат сохранить в ledset
                     ;в результате изменение 2-го бита на противоположное в
                     ;переменной ledset
      movfw ledset   ;запись ledset в "аккумулятор"
      movwf PORTD    ;запись ledset в защелку порт D
      goto change    ;зацикливаем - безусловный переход на метку change
      end            ;конец программы
```

Программа А.2

```
;дисциплина "Программируемые цифровые устройства"
;Лабораторная работа № 4
;Изучаем организацию памяти программ и памяти данных

       list p=16f877a
       include <p16f877a.inc>

ledset   equ  0x20  ;используем директиву Ассемблера для назначения имени ячейки
                    ;0x20 в нулевом банке регистров общего назначения

      org 0x00      ;директива Ассемблера «поместить команду goto start в 0x00»
                    ;адрес памяти программ (в вектор сброса)
      goto start    ;безусловный переход на метку start
      org 0x05      ;директива Ассемблера «поместить команду bcf STATUS,5 в 0x05»
                    ;адрес памяти программ (следующий за вектором прерываний)

start               ;метка
    goto 0x0810     ;безусловный переход на 0x0810 адрес памяти программ
    org 0x0810      ;директива Ассемблера поместить команду bcf STATUS,5 в 0x0810
                    ;адрес памяти программ
    bcf STATUS,5    ;сброс 5-го байта в регистре STATUS
    bcf STATUS,6    ;сброс 6-го байта в регистре STATUS
                    ;в результате переход в банк 0
    clrf PORTD      ;сброс защелки порт D - все нули
    clrf ledset     ;сброс переменой ledset - все нули
    bsf STATUS,5    ;установить 5-й байт в регистре STATUS (5-й байт = 1)
                    ;в результате переход в банк 1
    movlw b'11111011' ;константу b'11111011' переслать в аккумулятор (рабочий
                    ;регистр W)
    movwf TRISD     ;содержимое аккумулятора переслать в регистр TRISD
                    ;в результате настройка направления передачи сигналов на
                    ;выводах порт D. вывод RD2 (21 ножка) настроен как выход,
                    ;остальные - вход
     bcf TRISE,4    ;настройка порт D в режим цифровых каналов ввода-вывода
change              ;метка
    bcf STATUS,5    ;переход в банк 0
    movlw b'00000100' ;константу b'00000100' переслать в аккумулятор (рабочий
                    ;регистр W)
    xorwf ledset,1  ;выполнить побитное исключающее ИЛИ между W и ledset,
                    ;результат сохранить в ledset
                    ;в результате изменение 2 бита на противоположное в пере-
менной ledset
      movfw ledset  ;запись ledset в "аккумулятор"
      movwf PORTD   ;запись ledset в защелку порт D
      goto change   ;зацикливаем - безусловный переход на метку change
         end        ;конец программы
```

Программа А.3

```
;дисциплина "Программируемые цифровые устройства"
;Лабораторная работа № 7
;Изучаем обработку нажатия кнопки
;Обработка дребезга контактов не предусмотрена

    list p=16f877a
    include <p16f877a.inc>

    org  0x00
    goto start
    org  0x05
start
;*****************************************************
;настройка микроконтроллера (инициализация)
;*****************************************************
    bcf STATUS,5      ;переход в банк 0
    bcf STATUS,6      ;переход в банк 0
    clrf PORTB        ;Сбрасываем регистр PORTB (загружаем в него нуль)
    clrfINTCON        ;Сбрасываем регистр INTCON. Запрет всех прерываний
    bsf STATUS,5      ;переход в банк 1
    bsf  OPTION_REG,7   ;устанавливаем (загружаем единицу) в 7-й бит регистра
                      ;OPTION_REG. Отключение подтягивающих резисторов
    movlw b'11101111'  ;Загрузка константы b'11101111' в аккумулятор(W)
    movwf TRISB        ;Загрузка значения, помещенного в аккумулятор, в
                        ;регистре TRISB. RB4 - выход, остальные - входы.
    bcf STATUS,5        ;переход в банк 0

;*****************************************************
;текст программы обработки нажатия кнопки на RB5.
;Кнопка нажата - лог. 0, не нажата - лог. 1.
;*****************************************************
Label1                  ;метка
     btfsc PORTB,5      ;проверяем на нуль 5-й бит регистра PORTB
     goto Label2        ;если не равно нулю (кнопка не нажата), переход на
метку                   ;Label2
     bcf PORTB,4        ;если равно нулю (кнопка нажата), сбрасываем 4-й бит
                        ;регистра PORTB (отключаем светодиод)
     goto Label3        ;обход альтернативного блока команд "кнопка не нажата"
Label2                  ;метка
     bsf PORTB,4        ;альтернативный блок команд "кнопка не нажата".
                        ;Устанавливаем 4-й бит регитра PORTB (включаем
                        ;светодиод)
Label3                  ;метка
      goto Label1       ;Безусловный переход на метку Label1, бесконечный цикл
                        ;проверки состояния кнопки на RB5.
end                     ;конец программы
```

ПРИЛОЖЕНИЕ Б
(*справочное*)

СИСТЕМЫ СЧИСЛЕНИЯ

Так как естественным «алфавитом» микроконтроллера являются **двоичные числа**, то при их программировании используют запись чисел в двоичной и **шестнадцатеричной** системах счисления. Связь между двоичной и шестнадцатеричной системами счисления будет показана ниже по тексту. Чтобы избежать путаницы будем использовать обозначения чисел в различных системах счисления, принятые в Ассемблер Microchip:

– двоичные – b'0101011' (двоичное число заключено в апострофы);

– шестнадцатеричные – 0x1A или 1Ah (число 1A).

Если число записано без дополнительных обозначений – оно по умолчанию считается десятичным.

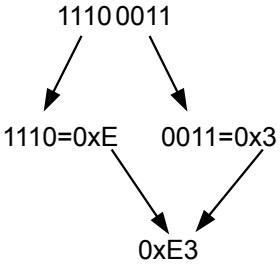
Вспомним правила позиционной записи чисел на примере десятичных. Число 209 можно представить в виде $2\cdot10^2+0\cdot10^1+9\cdot10^0$. Нетрудно заметить, что цифры умножаются на основание системы счисления 10, возведенное в степень, соответствующую позиции (или разряда): единицы – нулевая степень, десятки – первая степень, сотни – вторая степень и т. д. Аналогично, в двоичной системе счисления, когда доступно только две цифры, запись b'0101011' можно представить в виде $0\cdot2^7+0\cdot2^6+1\cdot2^5+0\cdot2^4+1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0=43$.

В шестнадцатеричной системе счисления основанием является число 16, числа записываются следующими цифрами (значками для записи чисел): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (таблица Б.1). Число 0x2C для перевода в десятичное можно представить в виде $2\cdot16^1+12\cdot16^0=44$.

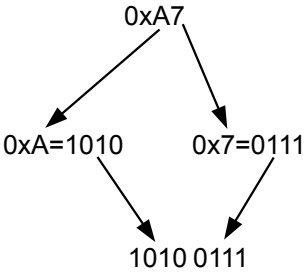
Удобство шестнадцатеричной системы счисления заключается в простоте перевода чисел из нее в двоичную систему счисления и обратно. Если разделить байт (8 бит) на два полубайта по четыре бита (на старший и младший полубайты), то можно заметить, что каждый полубайт представляет собой число от 0 до 15 и, соответственно, может быть записан в виде шестнадцатеричной цифры. Другими словами, байт может быть записан двухзначным шестнадцатеричным числом, поэтому для перевода двоичного числа достаточно перевести отдельно старший и младший полубайты в шестнадцатеричные цифры (рисунок Б.1).

Таблица Б.1 – **Примеры записи чисел в различных системах счисления**

Десятичное	Двоичное	Шестнадцатеричное
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
…	…	…
255	11111111	FF



Перевод двоичного числа
в шестнадцатеричное



Перевод шестнадцатеричного
числа в двоичное

Рисунок Б.1 – Перевод чисел

ПРИЛОЖЕНИЕ В
(*справочное*)

КАРТА ПАМЯТИ ДАННЫХ МИКРОКОНТРОЛЛЕРА PIC16F877A [2]

Символьное имя	Адрес	Символьное имя	Адрес	Символьное имя	Адрес	Символьное имя	Адрес
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Резерв(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Резерв(2)	18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h	Регистры общего назначения	116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h	16 байт	118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh

Окончание приложения В

Символьное имя	Адрес	Символьное имя	Адрес	Символьное имя	Адрес	Символьное имя	Адрес
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
Регистры общего назначения 96 байт	20h 7Fh	Регистры общего назначения 80 байт	A0h EFh	Регистры общего назначения 80 байт	120h 16Fh	Регистры общего назначения 80 байт	1A0h 1EFh
		Доступ к 70h-7Fh	F0h FFh	Доступ к 70h-7Fh	170h 17Fh	Доступ к 70h-7Fh	170h 17Fh
Банк 0		Банк 1		Банк 2		Банк 3	

ПРИЛОЖЕНИЕ Г
(*справочное*)

КАРТА ПАМЯТИ ПРОГРАММ МИКРОКОНТРОЛЛЕРА PIC16F877A [2]

Описание области памяти	Адрес
Вектор сброса	0000h
	0001h
	0002h
	0003h
Вектор прерываний	0004h
Страница 0	0005h
	…
	07FFh
Страница 1	0800h
	…
	0FFFh
Страница 2	1000h
	…
	17FFh
Страница 3	1800h
	…
	1FFFh

ПРИЛОЖЕНИЕ Д
(*обязательное*)

КРАТКОЕ ОПИСАНИЕ КОМАНД МИКРОКОНТРОЛЛЕРА СРЕДНЕГО СЕМЕЙСТВА КОМПАНИИ MICROCHIP

Каждая команда микроконтроллеров PIC16F87X состоит из одного 14-разрядного слова, разделенного на код операции (OPCODE), определяющий тип команды, и один или несколько операндов, определяющих операцию команды. Команды разделены на следующие группы (таблица Д.1):

– байт-ориентированные команды;

– бит-ориентированные команды;

– команды управления и операций с константами.

Для байт-ориентированных команд f является указателем регистра, а d – указателем адресата результата. Указатель адресата определяет, где будет сохранен результат. Если d = 0, результат сохраняется в регистре W. Если d = 1, результат сохраняется в регистре f, который используется в команде.

В бит-ориентированных командах b определяет номер бита, участвующего в операции, а f – указатель регистра, который содержит этот бит.

В командах управления или операциях с константами k представляет 8 битов значения константы или 11 битов значения адреса памяти программ (для команд call и goto) [2]. Краткое описание команд приведено в таблице Д.2.

Полное описание команд микроконтроллера можно найти в [2].

Таблица Д.1 – **Описание обозначений команд микроконтроллера [2]**

Поле	Описание
f	Адрес регистра (от 0x00 до 0x7F)
w	Рабочий регистр (аккумулятор)
b	Номер бита в 8-разрядном регистре
k	Константа (данные или метка)
x	Не имеет значения (0 или 1). Ассемблер генерирует x=0 для совместимости программы микроконтроллера с инструментальными средствами
d	Указатель адресата результата операции: d=0 – результат сохраняется в регистре w; d=1 — результат сохраняется в регистре f; по умолчанию d=1
label	Имя метки
TOS	Вершина стека
PC	Счетчик команд
PCLATH	Буфер старшего байта счетчика команд
GIE	Бит глобального разрешения прерываний
WDT	Сторожевой таймер
-TO	Флаг переполнения WDT

Окончание таблицы Д.1

Поле	Описание
-PD	Флаг сброса по включению питания
dest	Приемник, регистр w или регистр памяти
[ ]	Дополнительные параметры
( )	Содержимое
→	Присвоение
< >	Битовое поле
є	Из набора
курсив	Термин, определяемый пользователем

Формат команд микроконтроллеров среднего семейства компании Microchip приведен в таблицах Д.2–Д.5.

Таблица Д.2 – **Байт-ориентированные команды (операции с регистрами)**

Бит слова команды	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Назначение битов	OPCODE						d	f						

Примечания
OPCODE – код команды (указание что нужно сделать).
d – указатель адресата выполнения команды (OPCODE): d = 0 – результат сохраняется в аккумулятор (рабочий регистр W), d = 1 – результат сохраняется в регистр f.
f – 7-битный адрес регистра с данными, в отношении которых выполняется команда (OPCODE).

Таблица Д.3 – **Бит-ориентированные команды (операции с регистрами)**

Бит слова команды	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Назначение битов	OPCODE				b			f						

Примечания
OPCODE – код команды (указание что нужно сделать).
b – 3-разрядный номер бита в регистре f, в отношении которого выполняется команда (OPCODE).
f – 7-битный адрес регистра с данными.

Таблица Д.4 – **Команды операций с константами**

Бит слова команды	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Назначение битов	OPCODE						k							

Примечания
OPCODE – код команды (указание что нужно сделать с константой k).
k – 8-разрядное значение (константа).

Таблица Д.5 – **Команды управления (только call и goto)**

Бит слова команды	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Назначение битов	OPCODE			k										

Примечания
OPCODE – код команды (указание что нужно сделать).
k – 11-разрядное значение (константа-адрес памяти программ).

Таблица Д.6 – **Краткое описание команд микроконтроллеров среднего семейства Microchip [2]**

Мнемоника команды		Описание	Цик-лов	14-разрядный код		Флаги	Приме-чание
				Бит 13	Бит 0		
Байт-ориентированные команды							
ADDWF	f,d	Сложение W и f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f,d	Побитное «И» W и f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Очистить f	1	00 0001 1fff ffff		Z	2
CLRW	-	Очистить W	1	00 0001 0xxx xxxx		Z	
COMF	f,d	Инвертировать f	1	00 1001 dfff ffff		Z	1,2
DECF	f,d	Вычесть 1 из f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f,d	Вычесть 1 из f и пропу-стить, если 0	1 (2)	00 1011 dfff ffff			1, 2, 3
INCF	f,d	Прибавить 1 к f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f,d	Прибавить 1 к f и пропу-стить, если 0	1 (2)	00 1111 dfff ffff			1, 2, 3
IORWF	f,d	Побитное «ИЛИ»W и f	1	00 0100 dfff ffff		Z	1,2
MOVF	f,d	Переслать f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Переслать W в f	1	00 0000 1fff ffff			
NOP	-	Нет операции	1	00 0000 0xx0 0000			
RLF	f,d	Циклический сдвиг f влево через перенос	1	00 1101 dfff ffff		C	1,2
RRF	f,d	Циклический сдвиг f вправо через перенос	1	00 1100 dfff ffff		C	1,2
SUBWF	f,d	Вычесть из	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f,d	Поменять местами полубайты в регистре f	1	00 1110 dfff ffff			1,2
XORWF	f,d	Побитное исключающее «ИЛИ» W и f	1	00 0110 dfff ffff		Z	1,2
Бит-ориентированные команды							
BCF	f,b	Очистить бит b в регистре f	1	01 00bb bfff ffff			1,2

Окончание таблицы Д.6

Мнемоника команды		Описание	Циклов	14-разрядный код	Флаги	Примечание
DSF	f,b	Установить бит b в регистре f	1	01 01bb bfff ffff		1,2
BTFSC	f,b	Проверить бит b в регистре f, пропустить, если 0	1 (2)	01 10bb bfff ffff		3
BTFSS	f,b	Проверить бит b в регистре f, пропустить, если 1	1 (2)	01 11bb bfff ffff		3
Команды управления и операций с константами						
ADDLW	k	Сложить константу с W	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW	k	Побитное «И» константы и W	1	11 1001 kkkk kkkk	Z	
CALL	k	Вызов подпрограммы	2	10 0kkk kkkk kkkk		
CLRWDT	-	Очистить WDT	1	00 0000 0110 0100	-TO, -PD	
GOTO	k	Безусловный переход	2	10 1kkk kkkk kkkk		
IORLW	k	Побитное «ИЛИ» константы и W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Переслать константу в W	1	11 00xx kkkk kkkk		
RETFIE	-	Возврат из подпрограммы с разрешением прерываний	2	00 0000 0000 1001		
RETLW	k	Возврат из подпрограммы с загрузкой константы в W	2	11 01xx kkkk kkkk		
RETURN	-	Возврат из подпрограммы	2	00 0000 0000 1000		
SLEEP	-	Перейти в режим SLEEP	1	00 0000 0110 0011	-TO, -PD	
SUBLW	k	Вычесть W из константы	1	11 110x kkkk kkkk	C, DC, Z	
XORLW	k	Побитное «исключающее ИЛИ» константы и W	1	11 1010 kkkk kkkk	Z	

ПРИЛОЖЕНИЕ Е
(*справочное*)

РАБОЧАЯ ПРОГРАММА ПО ДИСЦИПЛИНЕ «ПРОГРАММИРУЕМЫЕ ЦИФРОВЫЕ УСТРОЙСТВА»

Цель преподавания дисциплины

Цель дисциплины "Программируемые цифровые устройства" – подготовка студентов к квалифицированному обслуживанию современных систем автоматического регулирования, управления, диагностики и отображения информации, использующихся на тепловозах.

Задачи изучения дисциплины

Основными задачами дисциплины являются: изучение основных характеристик и архитектуры микроконтроллеров, изучение программирования на языке Ассемблер, основных схемных решений устройств сопряжения микроконтроллера с объектом управления и интерфейсом человек – машина.

В результате изучения дисциплины студент должен

знать:

– основные характеристики и архитектуру микроконтроллеров;

– основные схемные решения устройств сопряжения микроконтроллера с объектом управления и интерфейсом человек–машина;

уметь:

– читать электрические схемы цифровых и аналоговых устройств;

– писать простейшие программы для микроконтроллеров на языке Ассемблер;

– разрабатывать структурные схемы систем автоматического регулирования, управления, диагностики и отображения информации;

– разрабатывать укрупненные блок-схемы алгоритмов управляющих программ.

Содержание дисциплины

Раздел 1 Введение

Т е м а 1 *Введение*

Термины и определения. Однокристальные микропроцессоры; однокристальные микроконтроллеры. Роль цифровых технологий в современном обществе. Понятие о встраиваемых системах. Применение микропроцессорных систем на железнодорожном транспорте.

Т е м а 2 *Элементы цифровых устройств*

Тристабильная логика. Логические элементы И, ИЛИ, И-НЕ, триггеры, их разновидности, аналого-цифровой преобразователь, двойное интегрирование, разрядность преобразования. Методы управления аналоговыми устройствами с помощью микропроцессорных устройств: цифроаналоговый преобразователь и широтно-импульсная модуляция.

Т е м а 3 *Средства разработки и отладки*

Изучение интерфейса MPLAB и основных настроек программной оболочки для эмуляции микроконтроллера и отладки программы. Программаторы и внутрисхемные отладчики. Создание проекта на базе готовой закомментированной программы на языке Ассемблер для микроконтроллера PIC16F877A.

Раздел 2 Организация памяти программируемых устройств

Т е м а 4 *Организация памяти программируемых устройств*

Организация памяти программ и данных микроконтроллеров. Неймановская и Гарвардская архитектура памяти. Преимущества и недостатки Неймановской и Гарвардской архитектура памяти. Адресация, шина адреса и шина данных. Аппаратная реализация запоминающего устройства, триггер, Flash-технология.

Раздел 3 Микроконтроллеры

Т е м а 5 *Структура микроконтроллера*

Электрические параметры микроконтроллеров фирмы Microchip. Организация памяти программ и данных микроконтроллеров среднего семейства фирмы Microchip. Банки памяти данных и страницы памяти программ. Тактовый генератор микроконтроллера и режимы его работы, машинный цикл (4T) и машинный такт (T). Настройка битов конфигурации. Арифметико-логическое устройство, рабочий регистр (аккумулятор).

Т е м а 6 *Регистры специального и общего назначения*

Особенности обращения к регистрам специального и общего назначения микроконтроллера. Регистр STATUS – информация о текущем состоянии микроконтроллера. Назначение регистров INTCON, OPTION_REG. Периферийные модули микроконтроллеров фирмы Microchip и управление ими.

Т е м а 7 *Система команд микроконтроллера*

Особенности языка программирования assembler. Бит-ориентированные команды bcf, bsf, байт-ориентированные команды movf, movwf, команды управления goto, call, return и операций с константами movlw, addlw, andlw. Формат команд названных групп, номер бита, указатель адреса. Директивы макроассемблера org, include, equ.

Т е м а 8 *Прерывания, порты ввода-вывода*

Понятие прерывания, вектор прерываний микроконтроллеров фирмы Microchip. Программная и аппаратная реализация прерываний в микроконтроллерах фирмы Microchip. Примеры использования прерываний (RB0/INT, TMR0). Конфигурирование портов ввода-вывода, управление логическими уровнями на выходах, обработка логических уровней на входах. Электрические схемы подключения внешних устройств к дискретным портам ввода-вывода микроконтроллеров фирмы Microchip (клавиатура, индикаторы, исполнительные устройства).

Т е м а 9 *Косвенная адресация и циклы*

Регистры косвенной адресации FSR, INDF. Организация циклов с помощью команд условия DECFSZ, INFSZ, BTFSC, BTFSS и безусловного перехода. Примеры использования косвенной адресации работы с массивами и циклических алгоритмов для обеспечения временных задержек.

Раздел 4 Встроенные аппаратные средства микроконтроллеров

Т е м а 10 *Таймеры*

Назначение и работа аппаратных таймеров микроконтроллера TMR0, TMR1. Регистры управления таймерами TMR0, INTCON, OPTION_REG, TMR1H, TMR1L. Примеры использования таймеров в практических конструкциях.

Т е м а 11 *Энергонезависимая память микроконтроллера*

Встроенная энергонезависимая память (EEPROM память) микроконтроллера фирмы Microchip. Регистры управления EEPROM памятью: EEDATA, EEADR,

EECON1, EECON2. Использование EEPROM памяти микроконтроллера фирмы Microchip в практических конструкциях.

Т е м а 12 *Аналогово-цифровой преобразователь*

Технические характеристики встроенного многоканального аналогово-цифрового преобразователя (АЦП) микроконтроллера фирмы Microchip. Регистры управления АЦП: ADRESH, ADRESL, ADCON0, ADCON1. Подключение внешних устройств к аналоговым входам микроконтроллера. Примеры использования АЦП микроконтроллера фирмы Microchip в практических конструкциях.

Т е м а 13 *Широтно-импульсный модулятор*

Встроенный широтно-импульсный модулятор (ШИМ) микроконтроллера фирмы Microchip. Регистры управления ШИМ: PR2, CCPR1L, CCPR1CON, T2CON. Примеры использования ШИМ микроконтроллера фирмы Microchip. Схема подключения к микроконтроллеру силового ключа, коммутация индуктивной нагрузки.

Раздел 5 Программирование микроконтроллеров на Си

Т е м а 14 *Программирование микроконтроллеров на Си*

Преимущества и недостатки программирования на языке высокого уровня, категория задач, которые целесообразно решать с помощью языков высокого уровня. Особенности языка Си для микроконтроллеров фирмы Microchip. Компиляторы Си MPLAB C18 и MicroC.

Примеры заданий на зачет

Задание 1 Какие логические уровни необходимо подать на входы *X*1 и *X*2 схемы (рисунок 1), чтобы на выходе *Y*1 была логическая единица?

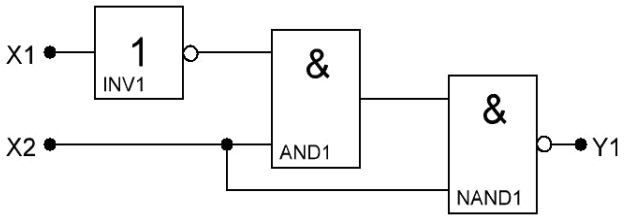


Рисунок 1 – Принципиальная схема соединения логических элементов

Задание 2 Какие логические уровни необходимо подать на входы *X*1 и *X*2 схемы (рисунок 2), чтобы на выходе *Y*1 был логический нуль?

Задание 3 Написать фрагмент программы для микроконтроллера фирмы Microchip, пересылающей данные из одного регистра общего назначения памяти данных в другой регистр, находящийся в другом банке.

Задание 4 Написать фрагмент программы для микроконтроллера фирмы Microchip, пересылающей данные из одного регистра общего назначения памяти данных в другой регистр, находящийся в другом банке при помощи косвенной адресации.

Задание 5 Написать фрагмент программы для микроконтроллера фирмы Microchip, конфигурирующий заданную ножку заданного порта ввода/вывода как вход и выполняющий анализ логического сигнала, поданного на эту ножку.

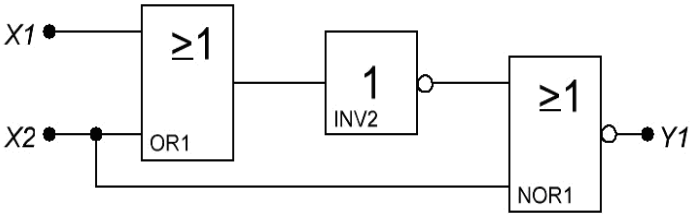


Рисунок 2 – Принципиальная схема к заданию 2

Задание 6 Написать фрагмент программы для микроконтроллера фирмы Microchip, конфигурирующий заданную ножку заданного порта ввода/вывода как выход и устанавливающий логический нуль (или единицу) на этой ножке.

Задание 7 Написать фрагмент программы для микроконтроллера фирмы Microchip, анализирующий значение в заданном регистре общего назначения памяти данных на нуль.

Задание 8 Написать фрагмент программы для микроконтроллера фирмы Microchip, реализующий повтор заданной команды (команд) *n* раз (до 255 раз).

Задание 9 Написать фрагмент программы для микроконтроллера фирмы Microchip настройки аппаратного таймера *TMR*0 для генерации прерывания от него через заданное время при заданной частоте тактового генератора.

Задание 10 Написать фрагмент программы, записывающий данные в энергонезависимую память (по заданному адресу) микроконтроллера фирмы Microchip.

Задание 11 Написать фрагмент программы, считывающий данные из заданного адреса энергонезависимой памяти микроконтроллера фирмы Microchip.

Задание 12 Написать фрагмент программы для микроконтроллера фирмы Microchip настройки аппаратного модуля ШИМ (PWM) с заданными частотой и коэффициентом заполнения при заданной частоте тактового генератора.

ПРИЛОЖЕНИЕ Ж
(*справочное*)

НЕКОТОРЫЕ ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ТЕКСТОВЫХ ДОКУМЕНТОВ ПО ГОСТ 2.105-95

Заголовки разделов должны иметь порядковые номера в пределах всего документа (части, книги), обозначенные арабскими цифрами без точки и записанные с абзацного отступа. Подразделы должны иметь нумерацию в пределах каждого раздела. Номер подраздела состоит из номеров раздела и подраздела, разделенных точкой. В конце номера подраздела точка не ставится. В конце заголовка точка также не ставится.

Пояснения символов и числовых коэффициентов, входящих в формулу, если они не пояснены ранее в тексте, должны быть приведены непосредственно под формулой. Пояснения каждого символа следует давать с новой строки в той последовательности, в которой символы приведены в формуле. Первая строка пояснения должна начинаться со слова «где» без двоеточия после него. Формулы, следующие одна за другой и не разделенные текстом, разделяют запятой. Переносить формулы на следующую строку допускается только на знаках выполняемых операций, причем знак в начале следующей строки повторяют. При переносе формулы на знаке умножения применяют знак «х».

Нумерация формул, за исключением формул, помещаемых в приложении, выполняется сквозной нумерацией арабскими цифрами, которые записывают на уровне формулы справа в круглых скобках. Одну формулу обозначают – (1). Допускается нумерация формул в пределах раздела. В этом случае номер формулы состоит из номера раздела и порядкового номера формулы, разделенных точкой, например (1.1). Пример оформления формул можно посмотреть на странице 31 пособия.

Рисунки, за исключением рисунков приложений, следует нумеровать арабскими цифрами сквозной нумерацией. Если рисунок один, то он обозначается «Рисунок 1». Допускается нумеровать рисунки в пределах раздела. В этом случае номер рисунка состоит из номера раздела и порядкового номера рисунка, разделенных точкой, например «Рисунок 1.1».

Слово "Рисунок" и его наименование помещают после пояснительных данных и располагают следующим образом: «Рисунок 1 – Структурная схема ядра микроконтроллера». Пример оформления рисунков можно посмотреть на страницах 13–15 пособия.

Рисунки располагают после первого упоминания его в тексте. В случае недостатка места допускается размещать рисунок на следующей странице за первой ссылкой на него. На все рисунки документа должны быть приведены ссылки в тексте документа, при ссылке следует писать слово "рисунок" с указанием его номера.

Таблицы применяют для лучшей наглядности и удобства сравнения однотипных показателей. Таблицы обозначаются сверху словом «Таблица», номером и через тире названием, например: «Таблица 1.1 – Карта памяти микроконтроллера». Таблицы, за исключением таблиц приложений, следует нумеровать арабскими цифрами сквозной нумерацией. Допускается нумеровать таблицы в пределах раздела. В этом случае номер таблицы состоит из номера раздела и порядкового номера таблицы, разделенных точкой.

Название таблицы должно отражать ее содержание, быть точным, кратким. При переносе части таблицы на ту же или другие страницы название помещают только над первой частью таблицы, на следующей странице пишут, например, «Продолжение таблицы 1.1». Если в конце страницы таблица прерывается и ее продолжение будет на следующей странице, в первой части таблицы нижнюю горизонтальную линию, ограничивающую таблицу, не проводят.

На все таблицы документа должны быть приведены ссылки в тексте документа, при ссылке следует писать слово "таблица" с указанием ее номера.

Пример оформления таблиц можно посмотреть на страницах 39 – 42 пособия.

Приложения. Материал, дополняющий текст документа, допускается помещать в приложениях. Приложениями могут быть, например, графический материал, таблицы большого формата, расчеты и т. п. Приложение оформляют как продолжение данного документа на последующих его листах или выпускают к виде самостоятельного документа. Приложения могут быть обязательными или информационными. Информационные могут быть рекомендуемого или справочного характера. В тексте документа на все приложения должны быть даны ссылки.

Каждое приложение следует начинать с новой страницы с указанием наверху посередине страницы слова "Приложение" и его обозначения, а под ним в скобках для обязательного приложения пишут слово "обязательное", а для информационного – "рекомендуемое" или "справочное". Приложение должно иметь заголовок, который записывают симметрично относительно текста с прописной буквы отдельной строкой.

Приложения обозначают заглавными буквами русского алфавита, начиная с А, за исключением букв Ё, З, И, О, Ч, Ь, Ы, Ъ. После слова "Приложение" следует буква, обозначающая его последовательность. Если в документе одно приложение, оно обозначается "Приложение А".

Учебное издание

СКРЕЖЕНДЕВСКИЙ Виктор Владимирович

Программируемые цифровые устройства

Учебно-методическое пособие

Редактор *Т. М. Маруняк*
Технический редактор *В. Н. Кучерова*

Подписано в печать 15.11.2013 г. Формат 60x84 $^{1}/_{16}$.
Бумага офсетная. Гарнитура Times. Печать на ризографе.
Усл. печ. л. 3,72. Уч.-изд. л. 3,75. Тираж 150 экз.
Зак. № . Изд. № 73

Издатель и полиграфическое исполнение
Белорусский государственный университет транспорта:
ЛИ № 02330/0552508 от 09.07.2009 г.
ЛП № 02330/ 0494150 от 03.04.2009 г.
246653, г. Гомель, ул. Кирова, 34.

