Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

Э. М. Виноградов

ПРОГРАММИРОВАНИЕ PIC-МИКРОКОНТРОЛЛЕРОВ НА ЯЗЫКЕ АССЕМБЛЕР

ПРАКТИКУМ
по дисциплине «Микропроцессорная техника»
для студентов специальности 1-36 04 02
«Промышленная электроника»
дневной и заочной форм обучения

Гомель 2016

УДК 621.58(075.8)
ББК 30.973.26-04я73
В49

Рекомендовано научно-методическим советом
факультета автоматизированных систем ГГТУ им. П. О. Сухого
(*протокол № 4 от 30.11.2015 г.*)

Рецензент: зав. каф. «Автоматизированный электропривод» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. С. Захаренко*

Виноградов, Э. М.

В49 Программирование PIC-микроконтроллеров на языке Ассемблер : практикум по дисциплине «Микропроцессорная техника» для студентов специальности 1-36 04 02 «Промышленная электроника» днев. и заоч. форм обучения / Э. М. Виноградов. – Гомель : ГГТУ им. П. О. Сухого, 2016. – 64 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

Содержит три лабораторные работы с основными теоретическими сведениями, порядком их выполнения, заданиями для самостоятельной работы и контрольными вопросами.

Для студентов специальности 1-36 04 02 «Промышленная электроника» дневной и заочной форм обучения.

УДК 621.58(075.8)
ББК 30.973.26-04я73

Лабораторная работа № 1

Интегрированная среда MPLAB IDE разработки программ для PIC-микроконтроллеров

1. Цель работы

Изучить и практически исследовать методы разработки и отладки программ для PIC-микроконтроллеров с помощью интегрированной среды разработки MPLAB IDE.

2. Основные теоретические сведения

MPLAB IDE – это интегрированная среда разработки (Integrated Development Environment – IDE) для микроконтроллеров PICmicro фирмы Microchip Technology Incorporated. MPLAB IDE содержит все инструментальные средства, необходимые для того, чтобы написать программу на языке Ассемблер, ассемблировать ее (получить машинные коды), выполнить на симуляторе ее тестирование и, наконец, загрузить машинные коды в программатор. MPLAB IDE имеет встроенные компоненты и сменные модули, позволяющие сконфигурировать среду с различными инструментальными средствами, программным обеспечением и аппаратными средствами.

Встроенные компоненты MPLAB IDE включают:

- **Менеджер проектов.** Используется для создания проектов и работы с группой файлов, входящих в проект. Выполняет интеграцию и взаимодействие между IDE и различными языковыми инструментальными средствами.
- **Редактор.** Предназначен для создания и редактирования текстовых файлов, таких как исходные и сценариев компоновки.
- **MPASM** – универсальный ассемблер. Ассемблер может использоваться автономно для трансляции единственного файла или может использоваться совместно с линкером (компоновщиком), чтобы строить проект из нескольких исходных и библиотечных файлов.
- **MPLINK** – компоновщик (линкер). Линкер отвечает за то, чтобы позиционировать оттранслированный код программы в области памяти целевого микроконтроллера.
- **MPLIB** – библиотекарь. Он управляет библиотечными файлами.

- **MPLAB SIM** – симулятор. Это программное средство использует персональный компьютер для имитации работы процессорного ядра и периферийных модулей PIC-микроконтроллеров. Симулятор позволяет моделировать выполнение команд и входные/выходные сигналы микроконтроллеров.

Дополнительные компоненты могут использоваться и быть добавлены в MPLAB IDE:

- **Компиляторы для языка Си.** MPLAB C18 и MPLAB C30 компиляторы разработки фирмы Microchip позволяют получить полностью интегрированный и оптимизированный программный код.
- **Программаторы.** MPLAB PM3, PICSTART Plus, PICkit 1 и 2 позволяют записать программные коды в целевые микроконтроллеры. MPLAB IDE выполняет полный контроль за процессом записи кода программы и данных, а также битов конфигурации для всевозможных вариантов режимов работы микроконтроллеров.
- **Внутрисхемные эмуляторы.** MPLAB REAL ICE и MPLAB ICE 2000 – это внутрисхемные эмуляторы для PIC-микроконтроллеров. Они присоединяются к персональному компьютеру через порты ввода/вывода и позволяют провести отладку программного обеспечения и аппаратной части микроконтроллерного устройства в реальном масштабе времени.
- **Внутрисхемные отладчики.** MPLAB ICD 2 и PICkit 2 являются экономичной альтернативой эмуляторам. Используя небольшие ресурсы целевого микроконтроллера, MPLAB ICD 2 позволяет загружать программный код в микроконтроллер, размещенный в разрабатываемой аппаратуре, устанавливать точки останова, выполнять пошаговый режим, отслеживать содержимое регистров и переменных.

Пакет программ MPLAB IDE можно бесплатно загрузить с сайтов www.microchip.com или www.microchip.ru.

3. Порядок выполнения работы

3.1. Создание папки для работы

3.1.1. Для выполнения лабораторных работ Вам необходимо создать на компьютере свою рабочую папку. Сделать это можно следующим образом.

На панели Total Commander выберите диск E. Затем выберите папку Users и раскройте ее. Далее выберите папку с именем MPT и раскройте ее. Затем выберите папку с именем своей группы (PM-31, PC-31, ZPE-31 …). Раскройте эту папку и внутри нее создайте новую папку (с помощью клавиши F7) с именем, соответствующим вашей фамилии (буквы обязательно латинские), например: Ivanov.

Примечание. При использовании в именах папок русских букв возможна неправильная работа исследуемых программ.

В дальнейшем Вы будете записывать и хранить в вашей папке все файлы в процессе выполнения лабораторных работ. Полный путь к ней:

E:\Users\MPT\PM-31\Ivanov

3.1.2. При выполнении лабораторных работ Вы будете создавать много различных файлов. Чтобы было легче ориентироваться в них, желательно для каждой лабораторной работы иметь свою отдельную папку. Допустим, что для выполнения лабораторной работы № 1 мы будем использовать папку с именем Lab1, которую необходимо предварительно создать.

С этой целью откройте вашу рабочую папку (с именем, соответствующим вашей фамилии) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab1.

В дальнейшем Вы будете записывать и хранить в этой папке все файлы в процессе выполнения лабораторной работы № 1. Полный путь к ней:

E:\Users\MPT\PM-31\Ivanov\Lab1

3.2. Настройка среды проектирования

Запустить MPLAB IDE можно нажатием на кнопку мыши, поместив курсор на ярлык MPLAB IDE на рабочем столе компьютера.

После запуска программы вы увидите на экране рабочий стол среды (рисунок 1), который имеет следующие компоненты:

1) Главное текстовое меню.
2) Графическое меню.
3) Рабочая область, в которой размещаются открытые окна с файлами, диалогами или другой информацией.
4) Линейка состояния, отображающая текущую настройку системы.

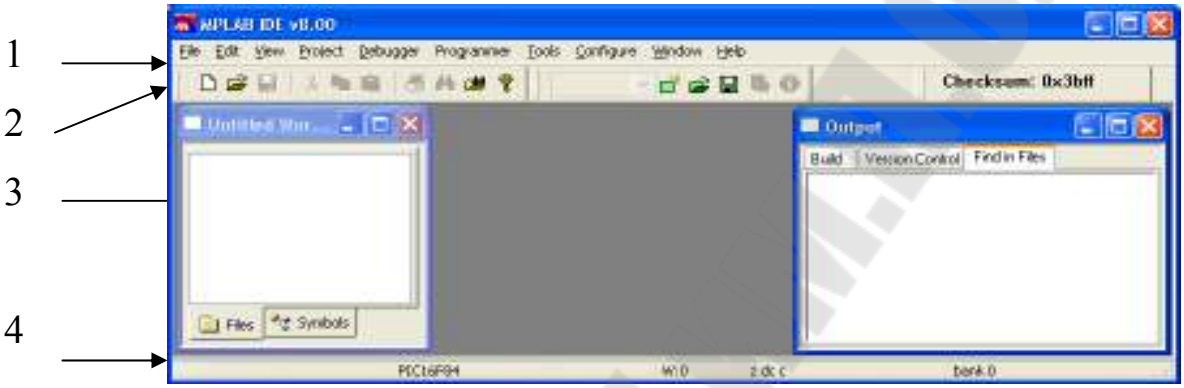

Рисунок 1. Рабочий стол среды MPLAB IDE

Первое, что необходимо сделать для работы в MPLAB IDE, это выбрать тип микроконтроллера, который будет использоваться в проекте. С этой целью выберите пункт меню **Configure > Select Device.** В диалоговом окне Select Device (см. рисунок 2), выберите микроконтроллер PIC16F84A из списка.

Цветные индикаторы указывают, какие компоненты MPLAB IDE поддерживают этот микропроцессор:

- Зеленый индикатор указывает полную поддержку.
- Желтый индикатор указывает на частичную поддержку и означает, что некоторые операции или функции, возможно, недоступны.
- Красный индикатор указывает на отсутствие поддержки этому микроконтроллеру.

Щелкните по кнопке [OK] для подтверждения выбора и закрытия окна.

3.3. Проверка правильности размещения языковых средств

С помощью пункта меню **Project > Set Language Tool Locations…** откройте диалоговое окно Set Language Tool Locations. Выберите из списка пункт Microchip MPASM Toolsuite, и дважды щелкните левой кнопкой мыши. В раскрывшемся списке дважды щелкните по строке Executables – исполняемые файлы.

Теперь надо последовательно щелкнуть по каждому файлу из нового списка, чтобы удостовериться по содержимому окна Location, что они правильно установлены.

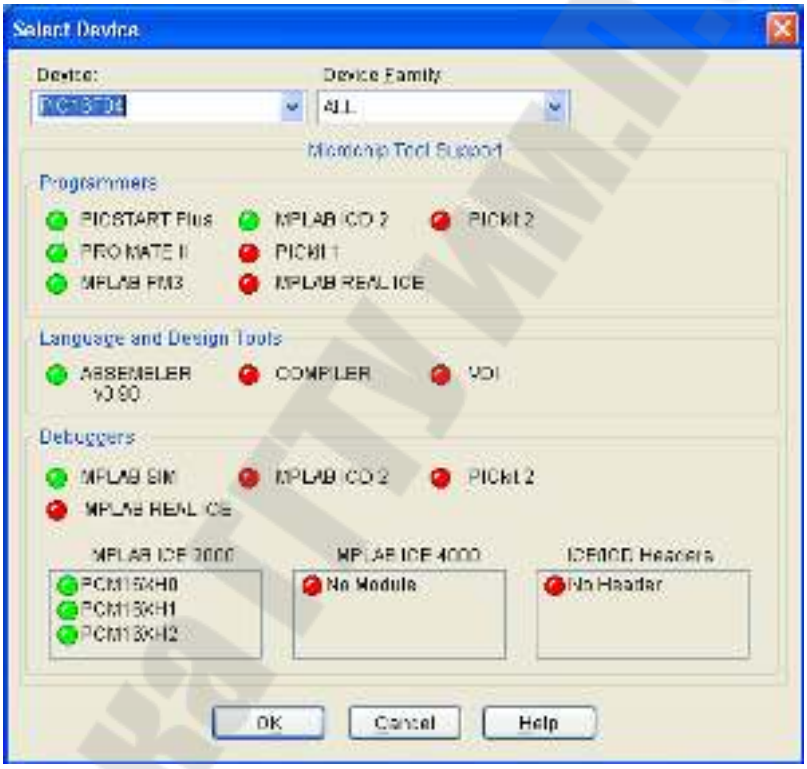

Рисунок 2. Диалоговое окно выбора микроконтроллера

Для ассемблера MPASM Assembler проверьте, что путь к нему установлен в виде: c:\Program Files\Microchip\MPASM Suite\MPASMWIN.exe.

Для библиотекаря MPLIB удостоверьтесь, что путь к нему установлен в виде: c:\Program Files\Microchip\MPASM Suite\mplib.exe.

Для линкера MPLINK удостоверьтесь, что путь к нему установлен в виде: c:\Program Files\Microchip\MPASM Suite\mplink.exe.

В случае правильности размещения этих средств щелкните по кнопке [OK] для сохранения установок и закрытия окна. В случае не-

соответствия расположения файлов по указанным путям нужно щелкнуть по кнопке [Browse…] (Обзор) и открыть нужный файл по указанному пути.

3.4. Ввод исходного текста программы

Для ввода исходного кода программы выберите пункт меню **File > New.** Откроется окно редактора MPLAB IDE.

Задание. Введите следующий текст программы с именем proba.asm в точности, как написано ниже. Текст комментария для экономии времени можно не набирать.

При наборе текста программы необходимо придерживаться определенных правил. Все метки и символические имена (в данном примере Count, reset, start, loop) должны начинаться с начала строки (колонка номер 1). Мнемоники команд микроконтроллера и директивы Ассемблера (в данном примере list, #include, org, end) должны размещаться со 2-й и далее колонок строки. В последней строке текста программы обязательно должна быть директива end. Хотя в языке Ассемблер символы, написанные заглавными и строчными буквами, транслируются одинаковыми кодами, современный стиль программирования рекомендует мнемокоды команд, директивы и метки записывать строчными буквами, а имена регистров специальных функций микроконтроллера и их отдельных битов – заглавными буквами. Имена регистров пользователя (в данном примере Count) и имена констант будем записывать строчными буквами, причем первая буква имени обязательно заглавная.

Обратите внимание, что перед директивой __CONFIG надо обязательно ввести два символа нижнего тире.

```
;*****************************************************
; proba.asm – пробная программа для работы в среде MPLAB
;*****************************************************
        list p=16f84a
        #include<p16f84a.inc>
        __CONFIG   _WDT_OFF   &   _HS_OSC
Count   equ     0x0C    ; переменная – счетчик циклов
        org     0x000   ; установка начального адреса по сбросу МК
reset:  goto    start   ; переход на начало программы
        org     0x005   ;  стартовый адрес размещения программы
start:  movlw   0x09    ; загрузить в рабочий регистр W число 0x09
```

```
        movwf  Count    ; переслать содержимое W в счетчик
loop:   incfsz  Count, F  ; инкрементировать счетчик, пропустить
                 ; следующую команду, если результат равен нулю
        goto    loop      ; цикл инкрементирования
        goto    bug       ; переинициализировать счетчик
        end               ; конец текста программы
```

В программе строка "__CONFIG ...." является директивой Ассемблера, под руководством которой в выходной hex-файл заносится информация о битах конфигурации микроконтроллера. Она означает:

_WDT_OFF – сторожевой таймер отключен;

_ HS_OSC – высокочастотный кварцевый резонатор.

Набранный текст программы необходимо сохранить в вашей папке под именем proba.asm. Для этого следует выбрать пункт меню **File > Save As...**_ В раскрывшемся диалоговом окне нужно найти папку e:\...\Lab1, раскрыть ее и ввести имя файла – proba.asm, затем необходимо нажать кнопку [Сохранить].

3.5. Ассемблирование исходного текста программы

Ассемблирование исходного текста программы можно выполнить без создания проекта. С этой целью выберите пункт меню **Project > Quickbuild proba.asm.** После выбора указанного пункта меню исходный текст программы сохраняется и запускается программа MPASM. Как только ассемблирование будет завершено, на экране появится окно результатов (рисунок 3).

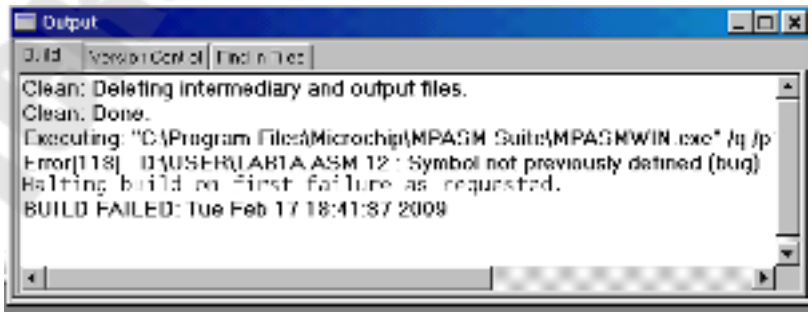

Рисунок 3. Окно Output результатов ассемблирования

В окне результатов появилось сообщение BUILD FAILED, информирующее о неудаче ассемблирования вследствие ошибки (Error) в тексте программы.

В последней строке текста программы преднамеренно была сделана ошибка. При выполнении ассемблирования MPASM выдаст ошибку о несуществующей метке bug. Двойной щелчок мышью на сообщении об ошибке перенесет курсор на строку в исходном тексте, где была сделана ошибка.

Исправьте последнюю строку программы, заменив слово bug на start.

Вновь дайте команду на ассемблирование, выбрав пункт меню **Project > Quickbuild proba.asm.** После исправления всех ошибок на экране появится окно результатов с сообщением об успешном ассемблировании BUILD SUCCEEDED. Теперь можно использовать симулятор для проверки работы программы.

3.6. Проверка работы программы с помощью симулятора

Выберите симулятор как инструмент для выполнения отладки программы. Это делается с помощью пункта меню **Debugger > Select Tool**. После выбора MPLAB SIM на рабочем столе среды MPLAB IDE должны произойти некоторые изменения (см. соответствующие числа на рисунке 4):

(1) В линейке состояния (внизу окна MPLAB IDE) должна появиться надпись MPLAB SIM.

(2) Дополнительные пункты должны появиться в меню отладчика Debugger.

(3) Дополнительные значки должны появиться в области графического меню отладчика.

(4) Вкладка MPLAB SIM будет добавлена в окне Output.

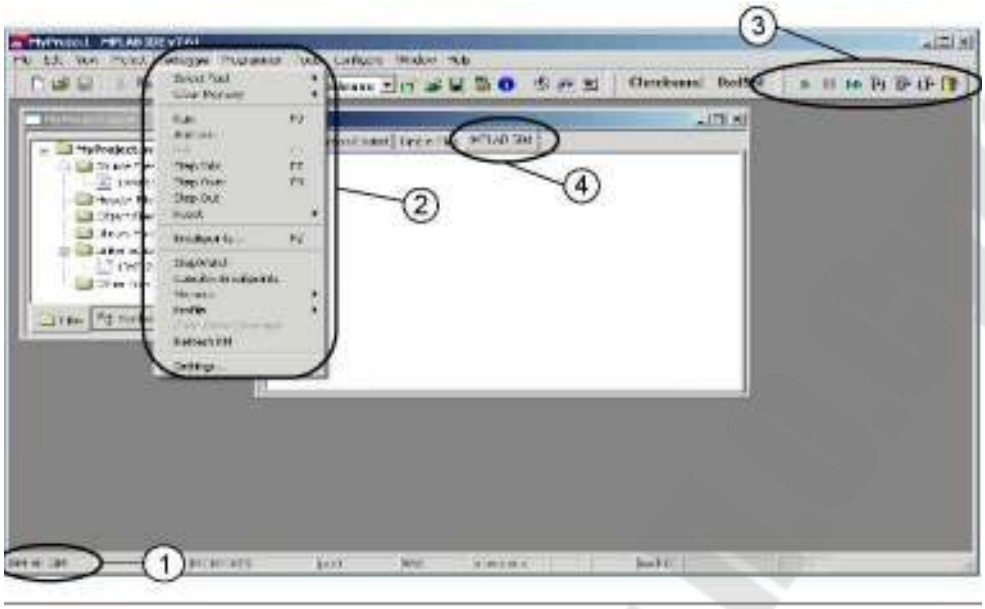

Рисунок 4. Рабочий стол MPLAB IDE с выбранным симулятором в качестве отладчика

3.7. Запуск программы и проверка ее работы в различных режимах

Выберите пункт меню **Debugger > Reset > Processor Reset.** В окне открытого файла proba.asm появится зеленая стрелка, которая показывает, с какого места будет начинаться выполнение программы. Это будет метка reset. Счетчик команд PC будет установлен в нуль, что является вектором сброса для микроконтроллера PIC16F84A. В линейке состояния содержимое PC будет равно нулю.

Для осуществления одного шага выполнения программы, выберите пункт меню **Debugger > Step Into**. При этом выполнится ранее отмеченная строка

reset goto start,

и зеленая стрелка укажет на следующую строку программы

start movlw 0x09.

Значение счетчика команд станет равным PC: 0x05.

Выполните еще несколько шагов, наблюдая за значениями счетчика команд PC и рабочего регистра W, а также положением зеленой стрелки.

Далее осуществите выполнение программы в автоматическом режиме. С этой целью выберите пункт меню **Debugger > Run.** В этом режиме программа выполняется с максимальной скоростью, возмож-

ной для симулятора. Программисты часто называют такой режим работы «прогоном программы». В линейке состояния появится надпись Running, а все параметры в ней исчезнут до останова выполнения программы. Зеленая стрелка также не будет изменять своего положения. Остановите выполнение программы, выбрав пункт меню **Debugger > Halt**. После этого появится информация в линейке состояния, а зеленая стрелка укажет на место в программе, где произошел останов.

Теперь осуществите выполнение программы в режиме Animate, выбрав пункт меню **Debugger > Animate.** В этом режиме происходит замедленное автоматическое выполнение программы. Можно следить за работой программы по перемещению зеленой стрелки и обновлению информации в линейке состояния. Остановить работу можно с помощью пункта меню **Debugger > Halt.**

Использование пунктов меню **Debugger** в процессе проверки работы программы не совсем удобно. Проще воспользоваться значками (кнопками) в области графического меню или «горячими клавишами». В таблице 1 приведены соответствия между пунктами меню **Debugger**, значками (кнопками) и «горячими клавишами».

Таблица 1

Способы управления отладкой

Debugger Menu	Toolbar Buttons	Hot Key
Run		F9
Halt		F5
Animate		
Step Into		F7
Step Over		F8
Step Out		
Reset		F6

Выполняемые функции при отладке поясняются в таблице 2.

Таблица 2

Выполняемые действия при отладке

Название режима	Выполняемые действия
Run	Автоматическое выполнение программы

Halt	Останов выполнения программы
Animate	Автоматическое выполнение программы в замедленном режиме
Step Into	Выполнение одной команды
Step Over	Выполнение одной команды с пропуском вызова подпрограммы
Step Out	Выполнение одной команды с выходом из подпрограммы
Reset	Сброс микроконтроллера

Примечания. 1. Чтобы узнать функцию кнопки (значка) графического меню, нужно поместить курсор на эту кнопку и на экране появится название функции. 2. Пошаговые режимы Step Over и Step Out действуют только при выполнении подпрограмм.

Задание 1. Выполните действия по проверке работы программы (сброс, пошаговое выполнение, прогон в автоматическом и замедленном режимах, останов) с использованием кнопок (значков) графического меню.

Задание 2. Выполните аналогичные действия с использованием «горячих клавиш».

Сделайте выводы об удобствах каждого способа управления выполнением программы. В дальнейшем вы можете использовать любой из этих способов.

3.8. Открытие дополнительных окон

В MPLAB IDE существует большое количество способов контролировать ход выполнения программы. Например, программа, которая используется в примере, увеличивает значение счетчика, но как можно убедиться в том, что это действительно происходит? Для просмотра текущего значения регистра воспользуйтесь пунктом меню **View > File Registers**. На экране появится окно со всеми регистрами памяти данных PIC16F84A.

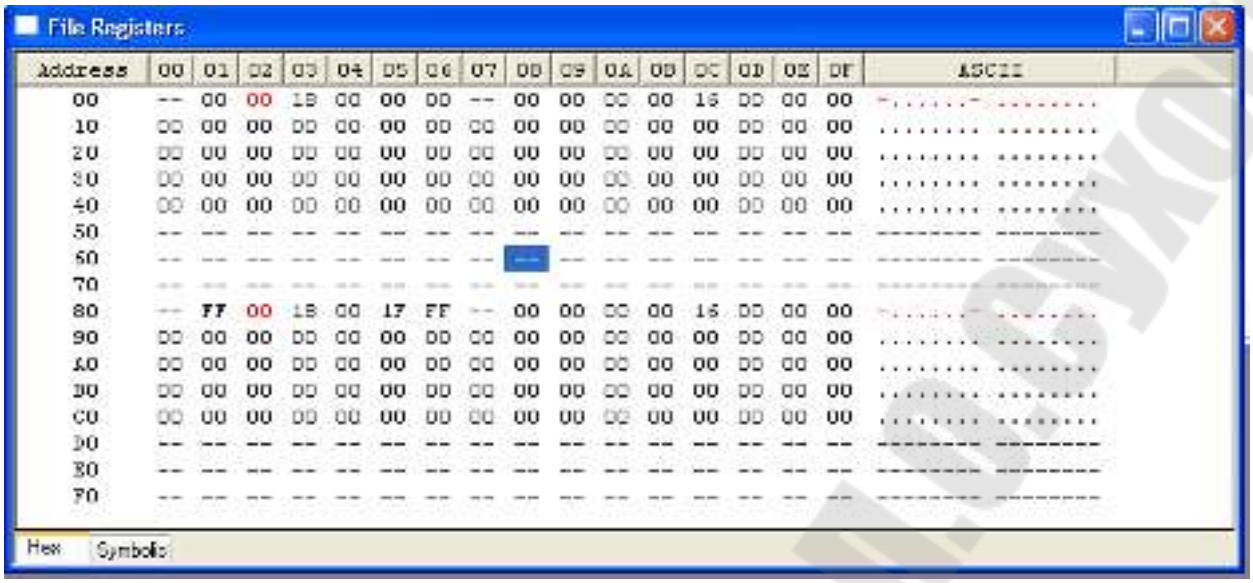

Рисунок 5. Окно с текущими значениями регистров

При каждом шаге в процессе выполнении программы значение регистров в окне будет обновляться. Сделайте несколько шагов. В нашем случае изменяется значение счетчика Count в регистре с адресом 0x0C. Изменение значения регистра отображается в окне памяти данных, выделяя его другим цветом.

Для «ручного» изменения содержимого регистра надо подвести курсор на нужный адрес и щелкнуть левой кнопкой мыши. Выделенный адрес изменит цвет на синий. Затем надо щелкнуть два раза левой кнопкой мыши и ввести с клавиатуры новое значение содержимого регистра.

Задание. Занесите в регистры (ячейки памяти данных) с адресом 11h (0x11) значение 55h (0x55), а по адресу 18h (0x18) значение AAh (0xAA).

Обратите внимание, что содержимое регистров общего назначения, расположенных в 0-м банке памяти данных по адресам 0x0C...0x4F дублирует содержимое регистров, расположенных в 1-м банке по адресам 0x8C...0xCF. Другими словами, регистры общего назначения, а их всего 68 в микроконтроллере PIC16F84A, отображаются в банке 0 на адреса 0x0C...0x4F, а в банке 1 – на адреса 0x8C...0xCF.

В сложных программах одновременно могут изменяться несколько регистров, что затрудняет контроль за ходом выполнения программы. Проблема может быть решена использованием окон наблюдения за переменными.

3.9. Использование окон наблюдения за переменными

Чтобы открыть окно, выберите пункт меню **View > Watch**. На экране появится пустое диалоговое окно наблюдения за переменными, показанное на рисунке 6.

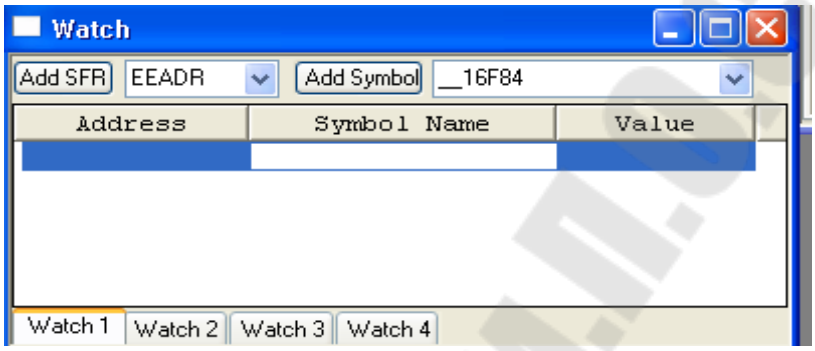

Рисунок 6. Окно наблюдения за переменными

С помощью кнопки [Add Symbol] и расположенного справа от нее раскрывающегося списка можно добавить в окно наблюдения символы (имена переменных), используемые в программе. Например, для помещения в окно переменной Count щелкните по стрелке в правой части этого поля и в раскрывшемся списке выберите Count. Затем щелкните мышью по кнопке [Add Symbol].

Кнопка [Add SFR] позволяет добавить в окно наблюдения регистры специальных функций. Пусть, для примера, нужно поместить в окно регистр состояния микроконтроллера STATUS. С этой целью щелкните по стрелке в поле Add SFR и в раскрывшемся списке выберите имя STATUS. Затем щелкните по кнопке [Add SFR].

Для удаления переменной из окна наблюдения надо ее выделить мышью и нажать клавишу [Delete] или же воспользоваться командой Delete контекстного меню.

Значения переменных в окне наблюдения по умолчанию отображаются в HEX-формате. В процессе отладки программы часто бывает удобно отображать значения переменных в других форматах, например, двоичном. С этой целью надо поместить курсор на строку с надписью Value и щелкнуть правой кнопкой мыши. В раскрывшемся списке нужно щелкнуть по строке Binary. Теперь окно наблюдения должно принять вид, показанный на рисунке 7.

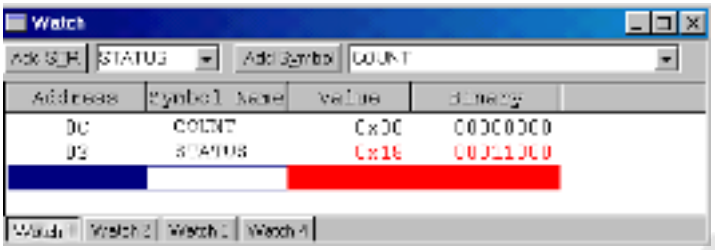

Рисунок 7. Окончательный вид окна наблюдения

Сделайте несколько шагов для выполнения программы, наблюдая за изменением содержимого регистров в окне. Обратите внимание на то, что если значение переменной изменяется при выполнении шага, т.е. выполнении команды, то оно выделяется красным цветом. При отсутствии изменения цвет остается черным.

3.10. Точки останова

Симулятор MPLAB SIM имеет возможность останавливать выполнение программы в любом месте. Это делается с помощью точек останова (breakpoints).

Допустим, нас интересует процесс выполнения программы от ее начала (сброса МК) до команды incfsz Count, F. Чтобы установить точку останова, подведите стрелку курсора на строку программы

incfsz Count, F

и щелкните левой кнопкой мыши. Появится мигающая линия, означающая, что данная строка выбрана. Затем щелкните правой кнопкой мыши. В появившемся контекстном меню выберите пункт Set Breakpoint (Установить точку останова) и щелкните левой кнопкой мыши. С левой стороны строки программы появится красный кружок с буквой “B”, означающий на установку точки останова.

Примечание. Можно также для установки точки останова просто дважды щелкнуть левой кнопкой мыши на нужной строке программы.

Для отмены точки останова следует выделить нужную строку программы и щелкнуть правой кнопкой мыши. В появившемся контекстном меню следует выбрать пункт Remove Breakpoint (Убрать точку останова) и щелкнуть левой кнопкой мыши.

Можно также для установки или удаления точки останова просто дважды щелкнуть левой кнопкой мыши на выбранной строке текста программы.

Задание. Установите, а затем удалите точки останова в произвольных местах программы с использованием обоих способов. Оставьте точку останова только на строке incfsz Count, F.

Для проверки работы программы сделайте сброс микроконтроллера, а затем выполните прогон программы в автоматическом режиме. Выполнение программы остановится на строке incfsz Count, F.

Окно наблюдения покажет значение переменной Count = 0x09, а в линейке состояния содержимое рабочего регистра будет W = 0x09. Это подтверждает правильность выполнения программы.

Теперь проверим работу цикла loop, в котором происходит инкрементирование содержимого регистра Count. С этой целью вновь запустите программу. В окне просмотра значение переменной Count станет равным 0x0A, т.е. выполнилось ее инкрементирование. Запустив программу несколько раз, убедитесь в правильности выполнения цикла loop.

Теперь проверим правильность выхода из цикла loop. Для этого поставим точку останова на строке goto start, а точку останова на строке incfsz Count, F удалим. Проверим работу программы в режиме Animate. По изменению содержимого регистра Count в окне просмотра убеждаемся в корректности работы программы. Для ускорения проверки переключимся в режим прогона Run. Выполнение программы остановится на строке goto start. Значение переменной Count будет равно нулю, что подтверждает правильность выполнения условия выхода из цикла loop. Содержимое регистра STATUS при этом не изменилось, т.е. флаг Z – признак нулевого результата остался сброшенным. Это тоже правильно, так как команда incfsz не влияет на флаги микроконтроллера.

3.11. Окно секундомера

Часто требуется знать время выполнения отдельных участков или всей программы. Симулятор MPLAB SIM позволяет измерить реальное время выполнения программы с помощью окна секундомера (Stopwatch).

Допустим, что нас интересует время выполнения программы из рассматриваемого примера от момента сброса микроконтроллера до момента окончания цикла loop. С этой целью произведите сброс МК и установите точку останова на строке goto start. Используя пункт

меню **Debugger > StopWatch**, откройте окно секундомера. Затем запустите программу в режиме Run. После останова программы окно секундомера примет вид, приведенный на рисунке 8.

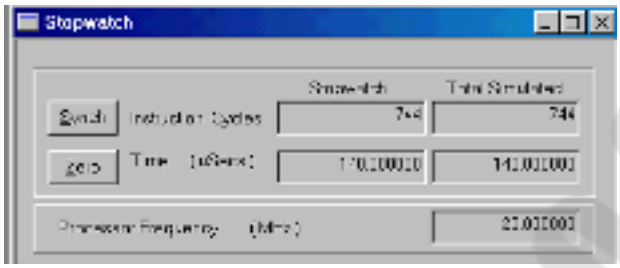

Рисунок 8. Окно секундомера после останова программы

При тактовой частоте работы МК, установленной 20 МГц по умолчанию, время выполнения данного участка программы будет равно 148,8 мкс, а количество машинных циклов - 744. Изменить тактовую частоту МК можно с помощью пункта меню **Debugger > Settings**, выбрав вкладку Osc/Trace.

Задание. Установите тактовую частоту работы МК равной 4 МГц и определите время выполнения программы.

3.12. Отладка подпрограмм

Отладку программ, содержащих подпрограммы, удобно выполнять с использованием трех вариантов пошагового режима:

1) режим Step Into, в котором исполняется одна команда программы, в том числе и команды подпрограммы. После каждого шага все окна обновляются;

2) режим Step Out, в котором производится ускоренное завершение выполнения подпрограммы и останов на команде, следующей за командой вызова call. В окнах будут результаты выполнения подпрограммы;

3) режим Step Over, в котором исполняются только команды основной программы без захода в подпрограммы. Если очередная команда будет вызовом call, то выполняется ускоренное выполнение подпрограммы и останов на команде, следующей за call. В окнах будут результаты выполнения подпрограммы.

Задание. В открытом окне MPLAB IDE создайте новый файл и занесите в него текст программы incr.asm, которая выполняет беско-

нечный цикл увеличения на 1 (инкремента) содержимого регистра Sum. Инкремент производится через определенное время, задаваемое подпрограммой задержки delay.

```
;*****************************************************
; incr.asm – программа инкрементирования содержимого
; регистра Sum
;*****************************************************
      list p=16f84a
      #include<p16f84a.inc>
      __CONFIG  _WDT_OFF  &  _HS_OSC

Count     equ  0x0C          ; регистр-счетчик циклов
Sum       equ  0x10          ; регистр-накопитель суммы
          org  0x000
begin:    clrf  Sum          ; очистить регистр Sum
again:    incf  Sum, F       ; инкремент содержимого регистра Sum
          call   delay       ; вызов подпрограммы delay
          goto  again    ; бесконечный цикл инкремента регистра Sum

delay:                       ; подпрограмма временной задержки
          movlw  .100
          movwf  Count       ; загрузить в регистр Count число
                             ; повторений 100
loop:     decfsz   Count, F  ; декремент Count и проверка его на нуль
          goto   loop        ; переход на метку loop, если не нуль
          return             ; возврат из подпрограммы

          end                ; конец текста программы
```

Сохраните набранный файл в папке e:\...\Lab1 под именем incr.asm. Выполните ассемблирование полученного файла с помощью пункта меню **Project > Quickbuild incr.asm_** и убедитесь в его правильности. Затем подключите симулятор MPLAB SIM с помощью пункта меню **Debugger > Select Tool > MPLAB SIM**.

С помощью пункта меню **View >Watch** откройте окно наблюдения и занесите в него переменные Count и Sum.

Выполните сброс микроконтроллера.

Сначала проверьте работу программы в режиме Step Into. Сделайте около 10 шагов с заходом в подпрограмму delay. По изменению

содержимого регистра Count, убедитесь в правильности работы.

Затем сделайте шаг в режиме Step Out. Убедитесь, что произошел останов выполнения программы на команде goto again, а содержимое регистра-счетчика Count стало равным нулю, т.е. подпрограмма delay полностью выполнена. Дальнейшее продвижение по программе в режиме Step Out невозможно, так как он выполняется только внутри подпрограммы.

Продолжите проверку программы в режиме Step Over. До команды вызова подпрограммы call delay он выполняется аналогично режиму Step Into. Однако после выполнения команды call delay программа остановится на команде goto again, а содержимое регистра Count вновь станет равным нулю. Таким образом, произошло ускоренное выполнение подпрограммы задержки delay в течение одного шага. Продолжите проверку работы программы в режиме Step Over, сделав несколько шагов и наблюдая за изменением содержимого регистра-накопителя Sum.

Теперь протестируйте программу в режиме Animate. По содержимому регистров Sum и Count убедитесь в правильности ее работы.

И, наконец, выполните программу в автоматическом режиме Run. Убедитесь, что в этом режиме содержимое окна наблюдения Watch не обновляется. Остановите выполнение программы. Зеленая стрелка укажет на точку останова работы программы, а содержимое окна Watch обновится.

Сделайте выводы об удобстве режимов Step Out и Step Over при отладке подпрограмм. Представьте себе, что подпрограмма содержала бы несколько сотен циклов!

4. Содержание отчета

Наименование и цель работы. Краткое описание способов задания режимов отладки программы с помощью пунктов меню Debugger, «горячих клавиш» и значков графического меню. Тексты исследуемых программ (комментарии обязательны!).

Контрольные вопросы

1. Что такое MPLAB IDE?
2. Перечислите состав встроенных компонентов MPLAB IDE.
3. Как выполняется создание нового исходного файла в

MPLAB IDE?

4. По каким признакам можно определить, что выбран MPLAB SIM в качестве отладчика?

5. Как можно просмотреть содержимое ячеек памяти данных и изменить их содержимое?

6. Как создается окно наблюдения за переменными? Как можно изменять формат их отображения?

7. Какими способами можно задать и убрать точки останова выполнения программы?

8. Какие виды пошагового режима отладки программы имеется в MPLAB SIM?

9. Что такое режим Animate и как он запускается?

Лабораторная работа № 2

Исследование команд микроконтроллеров семейства PIC16

1. Цель работы

Изучить и исследовать команды микроконтроллеров семейства PIC16 с помощью интегрированной среды программирования MPLAB IDE.

2. Основные теоретические сведения

При изучении команд микроконтроллеров обычно используется язык Ассемблер, в котором каждой команде присваивается мнемоническое имя. Для PIC-микроконтроллеров компания Microchip разработала специальный Ассемблер, который называется MPASM. Он

обычно используется в составе интегрированной среды разработки MPLAB IDE.

2.1. Правила записи программ на языке Ассемблера MPASM

Исходный текст программы на языке Ассемблера состоит из последовательности предложений определенного формата. Каждое предложение представляет собой строку из четырех полей:

МЕТКА МНЕМОНИКА ОПЕРАНДЫ КОММЕНТАРИЙ

Поля могут отделяться друг от друга произвольным числом пробелов. Порядок и позиция полей важны. Так, метки должны начинаться в первой колонке строки. Мнемоника может начинаться во второй колонке или далее. Операнды идут за мнемоникой. Комментарий может следовать за операндами, мнемоникой или меткой, и даже начинаться в любой колонке, если в качестве первого символа используется точка с запятой (;). Один или несколько пробелов должны отделять метку и мнемонику или мнемонику и операнды. Операнды должны отделяться запятой.

Метки. В поле метки размещается символическое имя операнда или символический адрес ячейки памяти, на которую имеется ссылка в программе. Все метки должны начинаться в колонке 1. Метка представляет собой буквенно-цифровую комбинацию, начинающуюся с буквы. Используются только буквы латинского алфавита. Ассемблер допускает использование в метках символа подчеркивания (_). Длина метки может быть до 32 символов. Метки могут сопровождаться двоеточием (:), пробелом, табуляцией или концом строки. Примеры оформления меток:

```
begin:   movlw  0x0A
         goto     next
           . . . . . .
begin    movlw   0x0A
         goto     next
           ……..
begin
         movlw  0x0A
         goto     next
```

В качестве символических имен и меток не могут быть использованы мнемокоды команд и директив Ассемблера, а также мнемонические обозначения регистров и других внутренних блоков МК.

Мнемоника. В этом поле записывается мнемоническое обозначение команды МК или директивы Ассемблера, которое является сокращением (аббревиатурой) полного английского наименования выполняемого действия. Если имеется метка на той же самой строке, то мнемоника команды или директивы ассемблера должны отделяться от этой метки двоеточием, или одним и большим количеством пробелов.

Операнды. В этом поле помещаются операнды (или операнд), участвующие в операции. Операнды должны отделяться от мнемоники одним или более пробелов. Списки операндов (операнды) должны отделяться запятыми.

Операнд может быть задан непосредственно или в виде его адреса (прямого или косвенного). Непосредственный операнд представляется числом или символическим именем.

Используемые в качестве операндов символические имена и метки должны быть определены, а числа представлены с указанием системы счисления.

Директивы Ассемблера. Ассемблирующая программа транслирует исходную программу в объектные коды. Хотя транслирующая программа берет на себя многое из рутинных задач программиста, такие как присвоение действительных адресов, преобразование чисел, присвоение действительных значений символьным переменным и т.п., программист все же должен указать ей некоторые параметры: начальный адрес рабочей программы, конец ассемблируемой программы, форматы данных и т.п. Всю эту информацию программист вставляет в исходный текст своей прикладной программы в виде директив Ассемблера, которые только управляют процессом трансляции и не преобразуются в коды объектной программы.

Директива ORG задает программе-ассемблеру адрес ячейки памяти, в которой должна быть расположена следующая за ней команда прикладной программы.

Директивой EQU можно любому символическому имени, используемому в программе, поставить в соответствие определенный операнд.

Например, запись

```
TMR0    equ   0x01
```

приводит к тому, что в процессе ассемблирования всюду, где встретится символическое имя TMR0, оно будет заменено числом 01h.

Директивой END программист дает программе-ассемблеру указание об окончании трансляции.

В результате трансляции должна быть получена карта памяти программ, где каждой ячейке памяти поставлен в соответствие хранящийся в ней код.

Ниже приведен фрагмент рабочей программы, содержащий операнды с различным способом задания и директивы Ассемблера:

```
Init1     equ   b'11111111'   ; переменной с именем Init1 присвоить
                               ; значение 11111111 в двоичной системе
          ……..
          movlw  Init1         ; загрузить константу Init1 в регистр W
          ……..
          movlw  .25           ; загрузить число 25 (десятичное)
          ……..
          movlw  d'55'         ; загрузить число 55 (десятичное)
          ……..
          movlw  0x0A     ; загрузить число 0Ah (шестнадцатеричное)
          ……..
          movlw   h'CF'    ; загрузить число CFh (шестнадцатеричное)
          ……..
          end              ; конец текта программы
```

2.2. Структура рабочей программы

Обычно программа для МК PIC16 состоит из трех основных секций:

1. Секция заголовка.
2. Секция описания.
3. Рабочая секция.

В секции заголовка обычно помещаются директивы Ассемблера, определяющие тип микропроцессора, имена подключаемых стандартных файлов и т.д.

В секции описания определяются логические имена для всех используемых в программе ресурсов: битовых и байтовых переменных, регистров пользователя.

Рабочая секция программы начинается с выражения org 0x000,

которое является указателем для Ассемблера о том, что код следующий за этим выражением начинается с нулевого адреса памяти программ. Рабочая секция обязательно должна заканчиваться директивой end, означающей, что за ней отсутствуют исполняемые команды.

3. Порядок выполнения работы

При выполнении лабораторной работы № 2 мы будем использовать папку с именем Lab2, которую необходимо предварительно создать.

С этой целью откройте вашу рабочую папку (с именем, соответствующем вашей фамилии) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab2.

В дальнейшем Вы будете записывать и хранить в этой папке все файлы при выполнении лабораторной работы № 2. Полный путь к этой папке будет такой:

e:\Users\MPT\PM-31\Ivanov\Lab2

3.1. Запуск MPLAB IDE

Запуск MPLAB IDE выполняется с помощью ярлычка с надписью MPLAB IDE на рабочем столе компьютера.

После запуска на экране появляется рабочий стол среды разработки.

Первое, что необходимо сделать для работы в MPLAB IDE, это выбрать тип микроконтроллера, который будет использоваться в разработке. С этой целью выполните команду меню **Configure > Select Device.** В диалоговом окне Select Device выберите микроконтроллер PIC16F84A из списка. Затем щелкните по кнопке [OK] для подтверждения выбора и закрытия окна.

3.2. Создание исходного файла базовой программы

Используя пункт меню **File > New** создайте исходный файл и с помощью команды **File > Save As…** сохраните его под именем base.asm в папке e:\...\Lab2.

Примечание. Рекомендуем присваивать имя предполагаемого файла, даже если его еще нет в окне редактирования. Дело в том, что

в этом случае редактор MPLAB IDE использует «цветное кодирование». Это означает, что команды Ассемблера отображаются полужирным шрифтом синего цвета, комментарии – светло-зеленым, а директивы – светло-синим цветом. Это позволяет контролировать правильность набора текста программы.

Текст базовой программы с именем base, которая будет использоваться в данной лабораторной работе, имеет следующий вид.

```
;**********************************************************
; base – базовая программа для исследования команд МК
;**********************************************************
        ; секция заголовка
      list p=16f84a          ; директива, определяющая тип процессора
                             ; и систему счисления hex по умолчанию
     #include <p16f84a.inc>  ; директива включения файла описания
                             ; регистров микроконтроллера PIC16F84A
     __CONFIG   _WDT_OFF  &  _HS_OSC
        ; секция описания
Init1     equ  b'11111111'      ; константа для настройки порта A
Init2     equ  b'00000000'      ; константа для настройки порта B
Temp1     equ  0x10         ; регистр для временного хранения данных
Temp2     equ   0x11         ; регистр для временного хранения данных
        ; рабочая секция
          org    0x000             ;  начальный адрес после сброса МК
reset:    goto   main              ; переход на начало программы
          org    0x010        ; адрес размещения основной программы
main:
          clrf    PORTA            ; очистка регистров-защелок
          clrf    PORTB            ; портов A и B
          bsf    STATUS, RP0     ; выбор банка 1
          movlw  Init1             ; настроить все линии
          movwf  TRISA            ; порта A  на ввод
          movlw  Init2            ; настроить все линии
          movwf  TRISB            ; порта B  на вывод
          bcf   STATUS, RP0       ; выбор банка 0
;**********************************************************
; здесь будут находиться  команды примеров
;
;**********************************************************
          goto   $                  ; зацикливание программы
```

```
end                         ; конец базовой программы
```

Рассмотрим работу этой программы. Вначале она указывает тип используемого микроконтроллера и систему счисления по умолчанию hex (шестнадцатеричную). Идущие далее ассемблерные директивы equ определяют константы Init1 и Init2, используемые для настройки портов МК. Символические имена Temp1 и Temp2 назначают адреса ячеек памяти данных (регистров общего назначения) для хранения промежуточных данных.

В программе строка "__CONFIG ...." является директивой Ассемблера, под руководством которой в выходной hex-файл заносится информация о битах конфигурации микроконтроллера. Она означает:

_WDT_OFF – сторожевой таймер отключен;

_HS_OSC – высокочастотный кварцевый резонатор.

Директива org 0x000 устанавливает стартовый адрес программного кода равным 0, т.е. соответствующим начальному состоянию счетчика команд МК после сброса. Команда goto main вместе с ассемблерной директивой org 0x010 и меткой main обеспечивают переход на адрес памяти программ 0x010, начиная с которого и размещается основная часть программы. Это необходимо для того, чтобы обойти адрес 0x004, используемый в качестве вектора прерываний, и тем самым зарезервировать его для возможных будущих применений. Для подпрограммы обработки прерывания можно будет использовать ячейки памяти программ с адреса 0x004 по 0x00F.

С метки main начинается программа инициализации портов МК. Вначале программа очищает (сбрасывает в нуль) триггеры-защелки данных портов A и B. Эта операция рекомендуется фирмой Microchip для того, чтобы исключить неопределенность в состояниях регистров портов после включения электропитания. Затем командой bsf STATUS, RP0 устанавливается (записывается 1) бит RP0 в регистре STATUS и производится переключение на 1-й банк памяти данных, где расположены регистры управления направлением передачи информации портов TRISA и TRISB.

Команда movlw Init1 загружает в рабочий регистр W значение, присвоенное константе Init1. Это значение равно b'11111111'. Символ b'…' означает, что данные заданы в двоичном формате. Двоичное представление удобно использовать в тех случаях, когда пред-

полагается операция с битами в регистре.

Следующая команда movwf TRISA пересылает содержимое рабочего регистра W в регистр управления направлением передачи TRISA порта A. Задание 1 в разряде регистра TRISA определяет, что соответствующий разряд порта A настраивается на ввод. В нашем случае все разряды порта A устанавливаются на ввод, т.е. будут являться входами МК. Обратите внимание, что порт A имеет только 5 разрядов, и старшие 3 бита числа, записываемого в регистр TRISA, также имеющего 5 разрядов, не используются.

Следующая команда movlw Init2 загружает в рабочий регистр W значение, присвоенное имени Init2 и равное b'00000000'. Команда movwf TRISB пересылает содержимое рабочего регистра W в регистр управления направления передачи TRISB порта B. Задание 0 в разряде регистра TRISB определяет, что соответствующий разряд порта B устанавливается на вывод. В нашем случае все разряды порта B будут настроены на вывод, т.е. являются выходами МК.

И, наконец, с помощью команды bcf STATUS, RP0 сбрасывается бит RP0 в регистре STATUS и происходит возврат в 0-й банк памяти данных, где располагаются необходимые для работы программы регистры и порты.

Коды примеров программ, которые будут использоваться в лабораторной работе, следует вставлять на место закомментированной строки «; Здесь будут находиться команды примеров». Вы будете заменять эту строку на реальные команды, ассемблировать получившуюся программу, исследовать ее работу с помощью симулятора MPLAB SIM и смотреть, что получилось.

В конце базовой программы стоит команда goto $. Оператор Ассемблера "$" означает текущее значение программного счетчика PC. Поэтому команда goto $ означает переход туда, где мы в данный момент находимся. Такой цикл бесконечен, поскольку не существует способа (кроме сброса микроконтроллера) выхода из него. Команда goto $ часто применяется для остановки выполнения программы при отладке.

Директива end указывает ассемблеру на конец текста программы.

Задание. Наберите текст базовой программы в окне открытого файла base.asm.

Примечание. При наборе текста программы необходимо при-

держиваться определенных правил. Все метки и символические имена (в данном примере Init1, Init2, Temp1, Temp2, reset, main) должны начинаться с начала строки (колонка номер 1). Мнемоники команд микропроцессора и директивы Ассемблера (в данном примере list, #include, org, end) должны размещаться со 2-й и далее колонок строки. В последней строке текста программы обязательно должна быть директива end. Хотя в языке Ассемблера символы, написанные заглавными и строчными буквами, транслируются одинаковыми кодами, современный стиль программирования рекомендует мнемокоды команд, директивы и метки записывать строчными буквами, а имена регистров специальных функций микроконтроллера и их отдельных битов – заглавными буквами. Имена регистров пользователя (в данном примере Temp1, Temp2) и имена констант (Init1, Init2) будем записывать строчными буквами, причем первая буква имени обязательно заглавная.

Обратите внимание, что перед директивой __CONFIG надо обязательно ввести два символа нижнего тире.

Используя пункт меню **Project >Quickbuild base.asm**, выполните ассемблирование исходного текста программы base.asm. Ассемблирование считается успешным при появлении в окне результатов сообщения “BUILD SUCCEEDED”.

3.4. Изучение и исследование команд пересылки данных

Микроконтроллеры семейства PIC16 имеют следующие команды пересылки данных:

```
movlw  k   ; пересылка константы  k  (8-ми разрядное значение)
           ; в рабочий регистр  W
 movwf   f  ; пересылка содержимого рабочего регистра W в регистр f
```

Команду movlw k часто называют командой загрузки рабочего регистра W.

Общим свойством этих двух команд является то, что они не изменяют флаги C, DC, Z (биты признаков) в регистре состояния STATUS.

Команда

```
movf  f, d      ; выполняет пересылку содержимого регистра  f,
                ; а символ  d определяет, куда будет помещен результат.
```

При d = 0 результат помещается в рабочий регистр W, а если d =

1, то результат записывается в использованный в команде регистр f. Эта команда изменяет бит признака нулевого результата (флаг Z).

Данная команда, в основном, используется для пересылки содержимого какого-либо регистра в рабочий регистр W (при d = 0). Если же установить d = 1, то эта команда загрузит регистр сам в себя, но при этом бит признака нуля Z установится в соответствии с содержимым этого регистра. Поэтому эта команда может использоваться для проверки содержимого регистра на нуль.

При написании подобных команд удобнее вместо цифр 0 или 1 подставлять символы W или F, которые обозначают регистр результата. Эти символы определены в файле p16f84a.inc, который был включен в базовую программу. Таким образом в дальнейшем будем писать команду movf f, W вместо movf f, 0 и команду movf f, F вместо movf f, 1.

Рассмотрим пример применения команд пересылки данных:

```
movlw   b'01010101' ; загрузить в рабочий регистр число в двоичном
                      ; формате
movwf   PORTB       ; переслать содержимое регистра W в порт B
movlw   0x0F        ; загрузить в рабочий регистр W число 0Fh
movwf   Temp1         ; переслать содержимое регистра W
                      ; в регистр Temp1
movlw   0x00          ; загрузить в регистр W число 0
movf    Temp1, W      ; переслать содержимое регистра Temp1
                      ; в  регистр W
movlw  .10          ; загрузить в регистр W десятичное число 10
movwf  Temp2          ; переслать содержимое регистра W
                      ; в регистр Temp2
```

Задание. Вставьте команды примера в базовую программу. С помощью пункта меню **Project >Quickbuild base.asm** выполните ее ассемблирование. Выберите симулятор MPLAB SIM в качестве инструмента для отладки программы с помощью пункта меню **Debug > Tools** .

Затем создайте окно для наблюдения переменных, используя пункт меню **View > Watch .** Последовательно введите в окно наблюдения имена регистров специальных функций WREG (рабочий регистр W), PORTB (порт B), STATUS (регистр состояния), а затем имена регистров пользователя Temp1 и Temp2 (регистров общего на-

значения).

Примечание. По умолчанию значения регистров отображаются в окне наблюдения в hex-кодах. Для отслеживания состояний регистров часто более удобен двоичный формат, показывающий отдельные разряды. Можно изменить формат отображения в окне наблюдения. С этой целью после ввода имени переменной, например, PORTB нужно поместить курсор на колонку с надписью Value и щелкнуть правой кнопкой мыши. В раскрывшемся списке надо выбрать строку Binary.

Проверку работы команд примеров удобно выполнять в пошаговом режиме. С этой целью выполните сброс микроконтроллера с помощью клавиши F6 или щелчком мыши по иконке с надписью Reset графического меню отладки. Зеленая стрелка появится в окне файла base.asm на строке reset goto main с адресом 0x000. Для пошагового выполнения программы используйте клавишу F7 или щелкайте по иконке с надписью Step Into графического меню.

Внимательно наблюдайте за изменением содержания регистров в окне Watch при выполнении команд примера. Сделайте выводы о влиянии команд пересылки на флаги микроконтроллера.

3.5. Изучение и исследование команд сброса (очистки)

В микроконтроллерах семейства PIC16 имеется две команды сброса (очистки):

```
clrw            ; сброс (очистка) содержимого рабочего регистра W
clrf   f        ; сброс (очистка) любого регистра  f.
```

Общим свойством команд сброса является то, что они устанавливают бит признака нуля (флаг) Z = 1 в регистре STATUS.

Например, с использованием команд сброса можно вывести нули на выводы порта B следующими способами:

```
  clrw                ; очистить рабочий регистр W
  movwf   PORTB       ; переслать содержимое  W  в порт B
или
  clrf   PORTB        ; очистить порт B
```

Задание 1 для самостоятельной работы. Напишите программу, которая выполняет следующие действия:

1) переслать десятичное число 25 в регистр Temp1;

2) вывести нули в порт B;
3) очистить рабочий регистр W;
4) переслать содержимое регистра Temp1 в порт B;
5) очистить регистр Temp1.

Запишите текст полученной программы в базовую.

Замечание. Так как в отчете по данной лабораторной работе будут требоваться тексты программ к заданиям для самостоятельной работы, то можно сохранить новый текст базовой программы в вашей папке под именем, например, base1.asm с помощью команды меню **File > Save as…**

Выполните ассемблирование разработанной программы и проверьте ее работу в пошаговом режиме. По изменению содержимого регистров в окне наблюдения сделайте выводы о соответствии выполнения программы заданному алгоритму.

3.6. Исследование команд сложения

Микроконтроллеры семейства PIC16 имеют две команды сложения:

```
addlw  k      ; сложение содержимого рабочего регистра W и
      ;  константы  k, заданной  в команде, результат – в регистре W
addwf   f, d   ; сложение содержимого рабочего регистра W с
               ; содержимым любого регистра  f.
```

Символ d определяет регистр, куда будет помещен результат выполнения команды. Если d = 0, то результат помещается в рабочий регистр W, а если d = 1, то результат записывается в использованный в команде регистр f. При написании команд удобнее вместо цифр 0 или 1 подставлять символы W или F, которые обозначают регистр результата.

Эти команды модифицируют биты признаков C, DC и Z в регистре STATUS. После выполнения команды сложения можно проверить эти биты и определить, является ли результат нулевым, положительным или отрицательным.

Следующий пример демонстрирует работу команд сложения:

```
clrw                    ; очистить рабочий регистр W
```

```
addlw   0x02      ; сложить содержимое регистра W с константой  02h
movwf   Temp1       ; переслать содержимое регистра  W
                    ; в регистр Temp1
movlw  0x34         ; загрузить в рабочий регистр число 34h
movwf   Temp2       ; переслать содержимое регистра W
                    ;  в регистр  Temp2
addwf   Temp2, W   ; сложить содержимое регистра W с содержимым
                    ; регистра Temp2, результат – в регистре W
movlw  0x34         ; загрузить в рабочий регистр число 34h
movwf   Temp2       ; переслать содержимое регистра W
                    ; в регистр  Temp2
addwf   Temp2, F    ; сложить содержимое регистра  W с содержимым
                    ; регистра Temp2, результат – в регистре Temp2
```

Задание. Запишите пример в базовую программу. Выполните ее ассемблирование. Исследуйте работу программы в пошаговом режиме, наблюдая за изменением содержимого регистров.

3.7. Команды инкремента и декремента

В микроконтроллерах семейства PIC16 имеются команды инкремента регистра incf f, d и декремента регистра decf f, d. Команда incf увеличивает содержимое заданного регистра на 1, а decf – уменьшает на 1. Результат может быть помещен обратно в заданный регистр (при d=1), либо в рабочий регистр W (при d=0). Если результат операции будет нулевой, то установится бит признака Z=1.

Следующий пример демонстрирует работу команд инкремента и декремента:

```
clrf   Temp1       ; очистить регистр Temp1
incf   Temp1, F    ; инкремент регистра, результат – в регистре Temp1
incf   Temp1, F
movlw  0xFF        ; загрузить в регистр W число  FFh
movwf  Temp2   ; переслать содержимое регистра W в регистрTemp2
decf   Temp2, F    ; декремент регистра, результат – в Temp2
decf   Temp2, F
```

Задание. Запишите пример в базовую программу и выполните ее ассемблирование. Проверьте работу программы в пошаговом режиме, наблюдая за изменением содержимого регистров Temp1, Temp2 и W.

Сделайте выводы о свойствах команд инкремента и декремента.

3.8. Исследование логических команд

Микроконтроллеры семейства PIC16 имеют большой набор команд, реализующих основные логические операции: И, ИЛИ, Исключающее ИЛИ, НЕ.

Есть три логические команды операций с константами (8-разрядными данными):

```
andlw  k   ; логическое  И константы  k  и рабочего регистра W,
                ; результат – в регистре W
iorlw  k   ; логическое  ИЛИ  константы  k  и рабочего регистра W,
                ; результат – в регистре W
xorlw  k   ; логическое  Исключающее ИЛИ константы  k  и  рабочего
                ; регистра W, результат – в регистре W
```

Эти команды очень часто используются для изменения значений разрядов (битов) числа, находящегося в рабочем регистре W. В этом случае константу k называют маской.

Команда andlw k сбрасывает в нуль (очищает) разряд числа в рабочем регистре W, если в соответствующем разряде маски будет записан 0. И не изменяет его, если в разряде маски записана 1.

Команда iorlw k устанавливает в единицу разряд числа в рабочем регистре W, если в соответствующем разряде маски будет записана 1. И не изменяет его, если в этом разряде записан 0.

Команда xorlw k инвертирует разряд числа в рабочем регистре W, если в соответствующем разряде маски записана 1. И не изменяет его, если в этом разряде записан 0.

Задание. Запишите в базовую программу команды примера:

```
movlw   b'11000011'  ; записать в регистр W значение  b'11000011'
andlw   b'01111110'  ; сбросить в нуль 7-й и 0-й  разряды регистра W
iorlw   b'00101000'  ; установить единицы в 5-м и 3-м разрядах
                     ; регистра W
xorlw   b'10101000'  ; инвертировать 7-й, 5-й, 3-й разряды регистра W
```

Занесите команды программы в базовую, выполните ее ассемблирование и проверьте работу в пошаговом режиме, наблюдая за состоянием разрядов регистра W.

В микроконтроллерах PIC16 имеется три логические команды операций с регистрами:

```
andwf   f, d      ; логическое  И рабочего регистра W и регистра  f
iorwf   f, d      ; логическое  ИЛИ рабочего регистра W и регистра  f
xorwf   f, d      ; логическое  Исключающее ИЛИ рабочего
                    ; регистра W и  регистра  f
```

Результат операции помещается либо в рабочий регистр W (при d = 0), либо обратно в заданный регистр f (при d = 1).

Эти команды часто применяются для изменения отдельных разрядов числа, находящегося в произвольном регистре f. При этом маска должна находиться в рабочем регистре W.

Задание 2 для самостоятельной работы. Напишите программу, которая выполняет следующий алгоритм:

1) вывести нули в порт B;
2) установить единицы в 7-м, 3-м, 1-м, 0-м разрядах порта B;
3) сбросить в нуль 3-й и 1-й разряды порта B;
4) инвертировать 5-й, 3-й и 0-й разряды порта B

Вставьте команды разработанной программы в базовую, выполните ее ассемблирование и проверьте работу в пошаговом режиме, наблюдая за изменением содержимого регистра W и порта B.

В микроконтроллерах PIC16 имеется команда, выполняющая логическую операцию НЕ (инвертирование):

```
comf   f, d   ; инвертирование регистра f
```

Эта команда поразрядно инвертирует содержимое любого заданного регистра. При d = 0 результат заносится в рабочий регистр W, а при d = 1 результат заносится в регистр f. При получении нулевого результата устанавливается флаг Z = 1.

Задание 3 для самостоятельной работы. Разработайте программу, которая записывает в регистр Temp1 число b’01010101’, а затем преобразует его в дополнительный код. Результат надо сохранить в регистре Temp1.

Примечание. Дополнительный код беззнакового числа – это его

обратный код плюс единица. Обратный код числа – это поразрядная инверсия всех его разрядов.

Вставьте полученную программу в базовую и выполните ее ассемблирование. Проверьте работу программы в пошаговом режиме.

3.9. Команды сброса и установки битов

Команда очистки (сброса) бита bcf f, b и команда установки бита bsf f, b используются для работы с отдельными битами в регистрах. Параметр b означает номер бита, с которым производится операция, и может принимать значения от 0 до 7.

Вот пример программы, которая устанавливает в единицу, а затем сбрасывает в нуль 7-й разряд (линию) порта B:

```
clrf  PORTB        ; вывести нули в порт B
bsf   PORTB, 7     ; вывести 1 в 7-й разряд  порта  B
bcf   PORTB, 7     ; вывести 0 в 7-й разряд порта  B
```

Задание. Занесите пример в базовую программу, выполните ее ассемблирование и проверьте работу с помощью симулятора в пошаговом режиме.

3.10. Исследование команд условных переходов

В микроконтроллерах семейства PIC16 отсутствуют команды условных переходов. Вместо них применяются такие, которые позволяют пропустить выполнение следующей команды программы. Имеется два варианта подобных команд.

Первый вариант – это команды инкремента incfsz f,d и декремента decfsz f,d с пропуском следующей команды при нулевом результате. С точки зрения обработки данных они работают аналогично командам incf f,d и decf f,d. Основное отличие от этих команд заключается в том, что при нулевом результате выполнения команды incfsz f,d или decfsz f,d пропускается следующая за ней команда. Это означает, что команды incfsz f,d и decfsz f,d могут использоваться для организации программных циклов. Другая особенность этих команд состоит в том, что они не влияют на флаги регистра STATUS.

Вот типичный пример использования цикла:

```
        movlw  .10          ; загрузить число 10 в рабочий регистр W
        movwf  Temp1        ; переслать содержимое W регистр Temp1
loop:
        decfsz  Temp1, F    ; декремент регистра Temp1
                            ; и проверка его на 0
        goto   loop         ; повторение цикла 10 раз, если содержимое
                            ; регистра Temp1 не равно 0
        movf  Temp1, W      ; переслать содержимое Temp1 в регистр  W
        movwf   PORTB       ; вывод содержимого W в порт B
```

Задание. Занесите пример в базовую программу, выполните ее ассемблирование и проверьте работу с помощью симулятора в пошаговом режиме. Наблюдайте за изменением содержимого регистра Temp1 при каждом проходе цикла. Какая информация будет на выводах порта B ?

Второй вариант команд перехода – это команды btfsc f, b и btfss f, b.

Эти команды проверяют состояние заданного бита в любом регистре, и в зависимости от результата пропускают или нет следующую команду. Параметр b означает номер бита, который проверяется, и может принимать значения от 0 до 7.

Команда btfsc f, b проверяет заданный бит регистра на нуль и если это выполняется, то следующая за ней команда программы пропускается.

Команда btfss f, b проверяет, равен ли заданный бит единице, и если это так, то следующая за ней команда программы пропускается.

Вот простой пример для PIC16F84A, у которого порт A настроен на ввод, а порт B на вывод:

```
        clrf  PORTB         ; вывести нули в порт B
loop:
        btfsc   PORTA, 2    ; Проверить бит с номером 2 на
                            ; входе порта A
        goto  loop          ; цикл ожидания появления низкого уровня
                            ; (логического 0) на входе порта A
        bsf  PORTB,7        ; вывести 1 в 7-й разряд порта B,
                            ; если  RA2 = 0
```

В этом примере проверяется 2-й разряд порта А (условно обозначается в программах как RA2) и, если на этом выводе будет низкий уровень (RA2 = 0), то выводится 1 в 7-й разряд (на линию 7) порта В.

3.11. Исследование команд вызова и возврата из подпрограммы

В микроконтроллерах семейства PIC16 имеется несколько команд для работы с подпрограммами. Сначала рассмотрим команды call k и return.

Команда call используется для перехода на подпрограмму по адресу, задаваемому в команде, а команда return – для возврата из подпрограммы. Адрес команды, следующей за командой call, запоминается в стеке. В PIC16 стек используется только для вызова подпрограмм и возврата. Глубина стека, т.е. число ячеек в нем, равна 8. Поэтому из основной программы можно сделать не более 8 вложенных вызовов подпрограмм. После возврата из подпрограммы выполнение основной программы продолжается со следующей за call командой. Регистры W и STATUS при вызове подпрограммы не сохраняются, поэтому, если необходимо, их можно сохранить в отдельных ячейках памяти (регистрах общего назначения). Вот простой пример использования подпрограммы:

```
start:
        clrf  PORTB          ; очистить порт B
        bsf  PORTB, 7        ; вывести 1 на 7-ю линию порта B
        call  turnoff        ; вызов подпрограммы с именем turnoff
        goto  start          ; идти на начало

turnoff:                     ; подпрограмма сброса в 0 линии порта B
        bcf  PORTB, 7        ; вывести 0 на 7-ю линию порта B
        return               ; возврат из подпрограммы
```

Задание. Занесите пример в базовую программу, выполните ее ассемблирование и проверьте работу с помощью симулятора в пошаговом режиме. Какая информация будет на выводах порта B?

Команды retlw k и retfie

Эти команды предназначены для возврата из подпрограмм. Команда retlw возвращает в рабочий регистр W константу, заданную в этой команде, а команда retfie разрешает прерывания. Команда retlw часто используется для чтения таблиц, находящихся в памяти программ.

Пусть, например, в рабочем регистре W содержится смещение от начала таблицы. Тогда получить нужный элемент можно следующей процедурой:

```
movlw   0x02        ; задать смещение
call  showsym       ; вызов подпрограммы
movwf   PORTB       ; вывести элемент таблицы в порт B
goto   $            ; зацикливание программы

showsym:            ; подпрограмма чтения элементов таблицы
        addwf  PCL, F     ; сложение  регистра W  с  содержимым
                          ; регистра PCL и пересылка результата в PCL
        retlw   0xAA      ; 0-й элемент таблицы
        retlw   0xBB      ; 1-й элемент
        retlw   0xCC      ; 2-й элемент
        retlw   0xDD      ; 3-й элемент
        retlw   0xEE      ; 4-й элемент
```

Программа начинает свою работу с пересылки константы 02h в рабочий регистр W. Затем производится вызов подпрограммы showsym. Ее работа начинается с добавления содержимого регистра W к текущему состоянию младшего байта счетчика команд PCL, и результат помещается в PCL. Таким образом, производится дополнительное смещение счетчика команд на величину, которая была передана в рабочем регистре. Например, если было W=0, то счетчик команд не изменится, и будет выполнена следующая команда retlw 0xAA, которая вызовет возврат из подпрограммы с записью числа AAh в регистр W. Если, как было в приведенной программе, W=02h, то к содержимому PCL будет добавлено число 02h, и произойдет дополнительное смещение на 2 шага. В результате будет выполнена команда retlw 0xCC, которая вызовет возврат из подпрограммы с записью CCh в регистр W.

После возврата из подпрограммы производится пересылка содержимого W в порт B, т.е. вывод в порт. В результате на выводах

порта должно быть 11001100.

Задание. Занесите пример в базовую программу, выполните ее ассемблирование и проверьте работу с помощью симулятора в пошаговом режиме. Исследуйте ее работу при различных значениях смещения, например, 0, 1, 3.

4. Содержание отчета

Наименование и цель работы. Текст базовой программы. Тексты программ к заданиям 1, 2, 3 для самостоятельной работы. Комментарии во всех программах обязательны!

Контрольные вопросы

1. Как производится настройка портов МК PIC16F84A на ввод?
2. Как производится настройка портов МК PIC16F84A на вывод?
3. Какие команды сложения имеются в МК семейства PIC16?
4. Какими командами можно изменять значения отдельных битов регистров?
5. Как можно проверить значение флага переноса C в регистре STATUS?
6. Как можно проверить на нуль сигнал на линии 3 порта A, настроенного на ввод?
7. Как можно проинвертировать 4-й разряд порта B, настроенного на вывод?

Лабораторная работа № 3

Изучение и исследование среды разработки электронных устройств PROTEUS

1. Цель работы

Изучить основы работы со средой проектирования электронных устройств Proteus. Изучить и исследовать методику разработки микроконтроллерных устройств с помощью Proteus.

2. Основные теоретические сведения

Proteus (произносится Протеус) – это пакет программ, объединяющий в себе две основные программы: ISIS – средство разработки и отладки в режиме реального времени электронных схем и ARES – средство разработки печатных плат. Разработчиком пакета Proteus является фирма Labcenter Electronics (Великобритания). Сайт разработчика: http://www.labcenter.co.uk.

Отличие Proteus от аналогичных по назначению пакетов программ, например, Electronics Workbench Multisim, MicroCap заключается в развитой системе симуляции (интерактивной отладки в режимах реального времени и пошаговом) для различных семейств микроконтроллеров: MCS-51 (Intel), PIC (Microchip), AVR (Atmel) и др. Proteus имеет обширные библиотеки компонентов, в том числе периферийных устройств (светодиодные и ЖК-индикаторы, температурные датчики, часы реального времени), интерактивных элементов ввода-вывода (кнопки, переключатели, виртуальные порты) и виртуальных измерительных приборов, которые не всегда присутствуют в других подобных программах.

В данной лабораторной работе будет рассматриваться только программа ISIS.

3. Порядок выполнения работы

3.1. Запуск программы

Внимание! Перед запуском программы создайте на диске E в своей рабочей папке (с именем, соответствующим вашей фамилии) новую папку с именем Lab3, в которую вы будете помещать исследуемые программы. Полный путь к этой папке может быть, например, таким:

E:\Users\MPT\PC-31\Ivanov\Lab3

Запуск программы ISIS.exe выполняется с помощью ярлычка с надписью ISIS на рабочем столе компьютера. После запуска программы на экране появится основное окно.

Самое большое пространство отведено под окно редактирования Edit Window. Именно в нем происходят все основные процессы создания, редактирования и отладки схемы устройства.

Слева вверху расположено маленькое окно предварительного просмотра Overview Window. С его помощью можно перемещаться по окну редактирования.

Под окном предварительного просмотра находится Object Selector – список выбранных в данный момент компонентов, символов и других элементов. Выделенный в списке объект отображается в окне предварительного просмотра.

Все возможные функции и инструменты Proteus доступны через меню, расположенное в самом верху основного окна, через пиктограммы, находящиеся под меню и с левой стороны основного окна, а также через «горячие клавиши».

В самом низу основного окна расположены слева направо:

- кнопки вращения и разворота объекта вокруг своей оси;
- панель управления интерактивной симуляцией (выглядит как магнитофонная и функции такие же: Play – ПУСК, Step – ПОШАГОВЫЙ РЕЖИМ, Pause - ПАУЗА, Stop – СТОП);
- строка состояния (в ней отображаются ошибки, подсказки, текущее состояние процесса симуляции и т.д.;
- координаты курсора, отображаемые в дюймах.

3.2. Изучение основных функций Proteus

3.2.1. Для освоения основных функций программы откройте один из уже имеющихся демонстрационных проектов. С этой целью выполните команду меню **File > Open Design.** Загрузите файл AC01.dsn из папки SAMPLES. Полный путь к этому файлу: SAMPLES\Interactive Simulation\Animated Circuits\AC01.dsn.

В основном окне появится схема простейшей электрической цепи, состоящей из генератора переменного тока и лампочки.

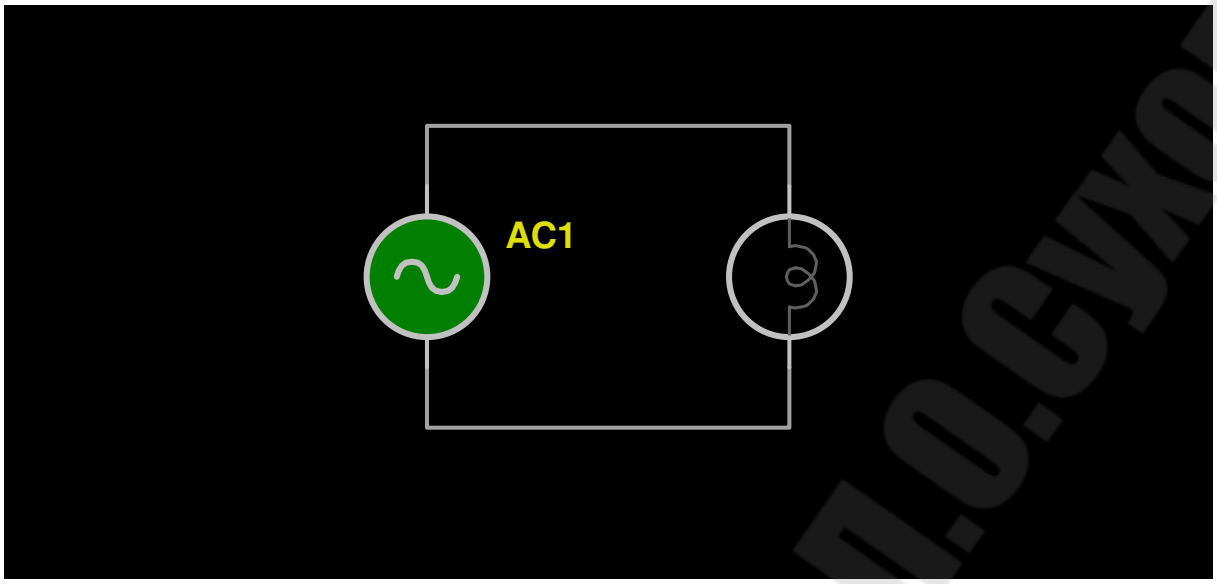

Рисунок 1. Схема электрической цепи переменного тока

Запустите проект, нажав с помощью мыши на панели управления кнопку [Play] – ПУСК.

На схеме появились цветные стрелки, меняющие направления с определенной частотой. Эта схема демонстрирует действие переменного тока в электрической цепи. Частота генератора снижена до 0,5 Гц для наглядности. Цвет и яркость проводов определяют полярность и уровень напряжения, стрелки – направление тока.

Рассмотрим методы навигации и манипулирования объектами в ISIS. Это можно сделать только на остановленном проекте. С этой целью остановите выполнение программы, нажав кнопку [Stop] – СТОП.

3.2.2. Изменение вида, отображенного в окне редактирования, производится двумя операциями: масштабированием и позиционированием.

Масштабирование – это настройка масштаба рисунка. Есть несколько способов увеличить или уменьшить участок схемы:

- наведите курсор мыши на место, которое хотите увеличить или уменьшить, и покрутите ролик мыши вперед, чтобы увеличить масштаб, или назад, чтобы уменьшить;
- наведите курсор мыши на место, которое хотите увеличить или уменьшить, и нажмите одну из клавиш F6 или F7 соответственно;
- используйте иконки Zoom In (увеличить) или Zoom Out (уменьшить) на панели инструментов.

Клавиша F8 может использоваться в любое время, чтобы отобразить рисунок целиком в окне редактирования.

Позиционирование – это настройка участка для изображения. Существует несколько способов позиционирования. Самый быстрый – использование окна предварительного просмотра. Для этого нужно навести курсор мыши в центр нового участка в окне предварительного просмотра и щелкнуть левой кнопкой мыши. В окне появится изображение фигурного креста, означающего, что включен режим слежения. При перемещении мыши по окну предварительного просмотра рисунок двигается по окну редактирования. Можно одновременно с перемещением рисунка изменять его масштаб, прокручивая ролик мыши. Для выхода из режима позиционирования со слежением следует вновь щелкнуть левой кнопкой мыши. Курсор мыши примет обычный вид стрелки.

Задание 1. Исследуйте все рассмотренные методы навигации в ISIS на схеме открытого проекта. Если изображение схемы исчезнет с экрана, то нажатием клавиши F8 его можно опять вернуть в центр окна редактирования.

3.2.3. Для визуальной помощи в окне редактирования может быть отображена сетка точек, которая помогает в выравнивании элементов и проводников. Включение и отключение сетки выполняется из меню **View** (Вид) командой **Grid** (Сетка). Можно также управлять отображением сетки с помощью иконки с надписью Toggle Grid на панели инструментов.

Задание 2. Выключите, а затем включите отображение сетки различными способами.

3.2.4. Теперь рассмотрим методы манипулирования объектами, то есть перемещения и изменения их параметров.

Для того, чтобы манипулировать объектами, их нужно сначала выделить, а это можно сделать только на остановленном проекте. Есть несколько способов выделения объектов в ISIS.

Первый способ. Для выделения одного объекта надо поместить курсор на изображение объекта, при этом курсор примет вид ладони, а вокруг объекта появится пунктирный прямоугольник. Затем надо щелкнуть правой кнопкой мыши. Изображение объекта изменит свой цвет на оранжевый, а на экране появится контекстное меню, со-

держащее доступные действия для этого объекта. Основные пункты меню:

Drag Object - переместить объект
Edit Component – редактирование свойств
Delete Object - удалить объект
Rotate Clockwise - повернуть по часовой стрелке на 90º
Rotate Anti-Clockwise – повернуть против часовой стрелки на 90º.

Для снятия выделения надо щелкнуть правой кнопкой мыши на пустом месте схемы и выбрать пункт Clear Selection (Очистить выделение) в появившемся контекстном меню.

Можно также для снятия выделения просто щелкнуть левой кнопкой мыши на пустом месте схемы. После снятия выделения цвет объекта изменится на первоначальный.

Задание 3. Выделите на схеме лампочку и последовательно выполните все перечисленные пункты меню следующими способами:

- после выбора пункта Drag Object на схеме появится контур лампочки зеленого цвета. Переместите мышь на новое место (контур лампочки будет следовать за мышью), а затем щелкните левой кнопкой, чтобы разместить объект;
- после выбора пункта Edit Properties на экране появится окно редактирования свойств выбранного компонента, в данном случае лампочки. Будут указаны ее напряжение питания и внутреннее сопротивление. Изменять ничего не надо, а просто закрыть окно редактирования;
- после выбора пункта Delete Object изображение лампочки исчезнет со схемы. Отменить указанное действие можно с помощью иконки отмены с надписью Undo Changes на панели инструментов.

Второй способ выделения. Рассмотрим его на примере генератора переменного напряжения. Подведите курсор мыши к изображению генератора и щелкните левой кнопкой. Изображение станет оранжевого цвета и курсор превратится в крест. Это указывает, что объект выделен и включен режим слежения. Затем нажмите левую кнопку и, не отпуская ее, перемещайте мышь по окну. Появится контур изображения генератора зеленого цвета. Поместите мышь на новое место и отпустите кнопку. Схема примет новый вид. Щелкните последовательно клавишами ‘+’ и ‘-‘ цифровой клавиатуры, чтобы повернуть изображение генератора на 90º. Щелкните левой кнопкой мыши на пустом месте схемы, чтобы снять выделение с генератора.

Третий способ выделения удобен для группы объектов. С этой целью поместите курсор мыши в левый верхний угол окна редактирования и нажмите левую кнопку. Затем, не отпуская кнопку мыши, растяните появившийся прямоугольник к правой нижней части окна. Отпустите кнопку мыши. На экране появится прямоугольник, ограниченный штриховыми линиями, а все, что находится внутри него, окажется выделенным. Используйте метки установки размеров, чтобы прямоугольник выделения охватывал только объекты схемы. Теперь поместите курсор мыши внутрь прямоугольника. Курсор превратится в изображение креста, что указывает на включение режима слежения. Нажмите левую кнопку и, не отпуская ее, передвиньте мышь. При этом прямоугольник выделения и все объекты внутри него также будут передвигаться. Поместите объекты на новое место и отпустите кнопку мыши. Затем щелкните левой кнопкой на пустом участке схемы, чтобы убрать выделение.

Имеется возможность изменить параметры любого компонента схемы. Рассмотрим это на примере генератора AC1. Поместите курсор на изображение генератора и дважды щелкните левой кнопкой. На экране появится диалоговое окно Edit Component - редактирование свойств компонента. В строке Component Reference стоит AC1 – наименование на схеме. В строке Amplitude – величина амплитуды напряжения 12 V, а в строке Frequency – частота 0.5 Hz. Обратите внимание, что дробная часть отделяется от целой точкой, как это принято в английской нотации. Не делайте никаких изменений, просто закройте окно редактирования.

3.2.5. Теперь научимся применять органы управления схем, которые называются в Proteus активаторами. С этой целью откройте следующий демонстрационный проект, выполнив команду меню **File > Open Design.**

На экране появится диалоговое окно с сообщением:

Save changes to current design? (Сохранить изменения в проекте?) с возможностью выбора: Yes, No, Cancel. Нажмите на кнопку [No], что изменений делать не надо.

Откройте файл Basic07.dsn. Полный путь к нему: SAMPLES\Interactive Simulation\Animated Circuits\Basic07.dsn.

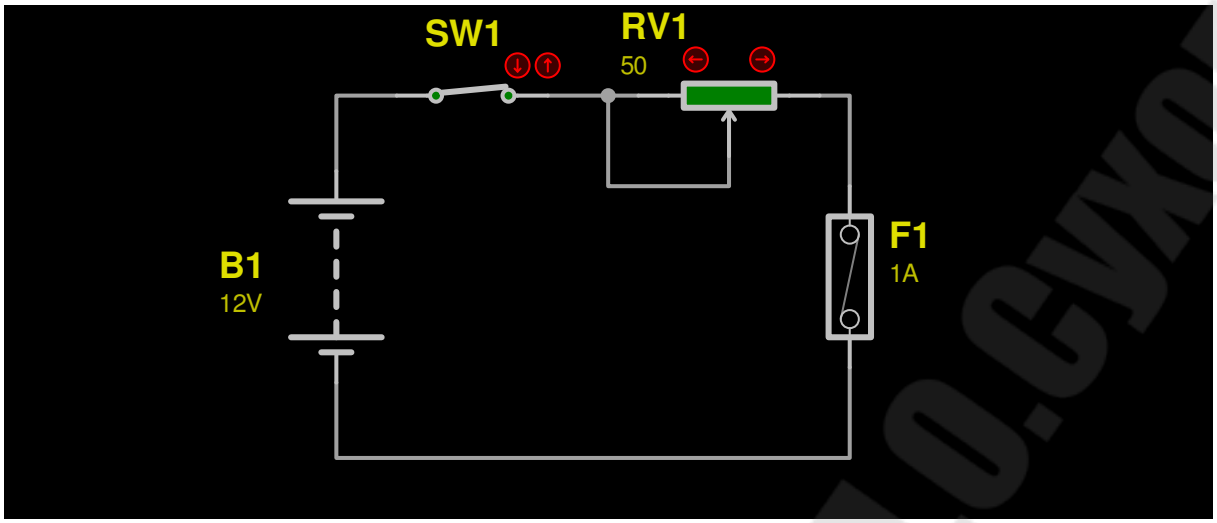

Рисунок 2. Схема цепи постоянного тока

В проекте изображена простейшая схема цепи постоянного тока, состоящая из батареи, переключателя (тумблера), реостата и предохранителя. Запустите проект на выполнение. У тумблера и реостата имеются красные стрелки в кружках. Это и есть активаторы. Щелкая по ним левой кнопкой мыши, можно переключать тумблер или перемещать движок реостата, изменяя таким образом его сопротивление. Для удобства наблюдения увеличьте изображение схемы. Попробуйте переключать тумблер, щелкая по его активаторам. Затем, щелкая по активаторам реостата, изменяйте положение его движка. В заключение выключите тумблер и поставьте движок реостата в среднее положение.

Запустите проект на выполнение. Затем включите тумблер. Цветные стрелки показывают направление тока в цепи. Теперь перемещайте движок реостата вправо. При определенном положении движка сгорает предохранитель, и ток в цепи прекращается. Остановите выполнение проекта и поставьте движок реостата в среднее положение. При повторном запуске схема вновь заработает.

3.3. Разработка проекта управления светодиодом от микроконтроллера

Вы овладели необходимым минимумом работы в Proteus, настало время создания своих проектов. Первым таким проектом будет разработка простейшего микроконтроллерного устройства (МКУ), в

котором МК управляет светодиодом, то есть включает или выключает его по определенной программе.

3.3.1. Создайте новый проект, используя команду меню **File > New Design**. На экране появится диалоговое окно Create New Design, предлагающее выбрать шаблон для создания нового проекта. Щелкните мышью по варианту DEFAULT (проект с параметрами по умолчанию), а затем по кнопке [OK]. На экране появится основное окно Proteus с синим прямоугольником в окне редактирования, в котором вы будете создавать схему МКУ.

Принципиальная схема МКУ будет иметь вид, приведенный на рисунке 3.

В качестве микроконтроллера выбран PIC16F84A. На схеме не изображены цепи питания и сброса МК, а также задания частоты тактового генератора. Все это эмулируется программно, и нет необходимости добавлять их в схему. Однако, если вы будете разрабатывать проект до этапа изготовления печатной платы, то эти цепи и элементы придется добавить.

Итак, для построения схемы МКУ нам нужен микроконтроллер PIC16F84A, светодиод красного свечения и резистор с сопротивлением 300 Ом. Получить все эти элементы можно из библиотек компонентов, которые имеются в Proteus.

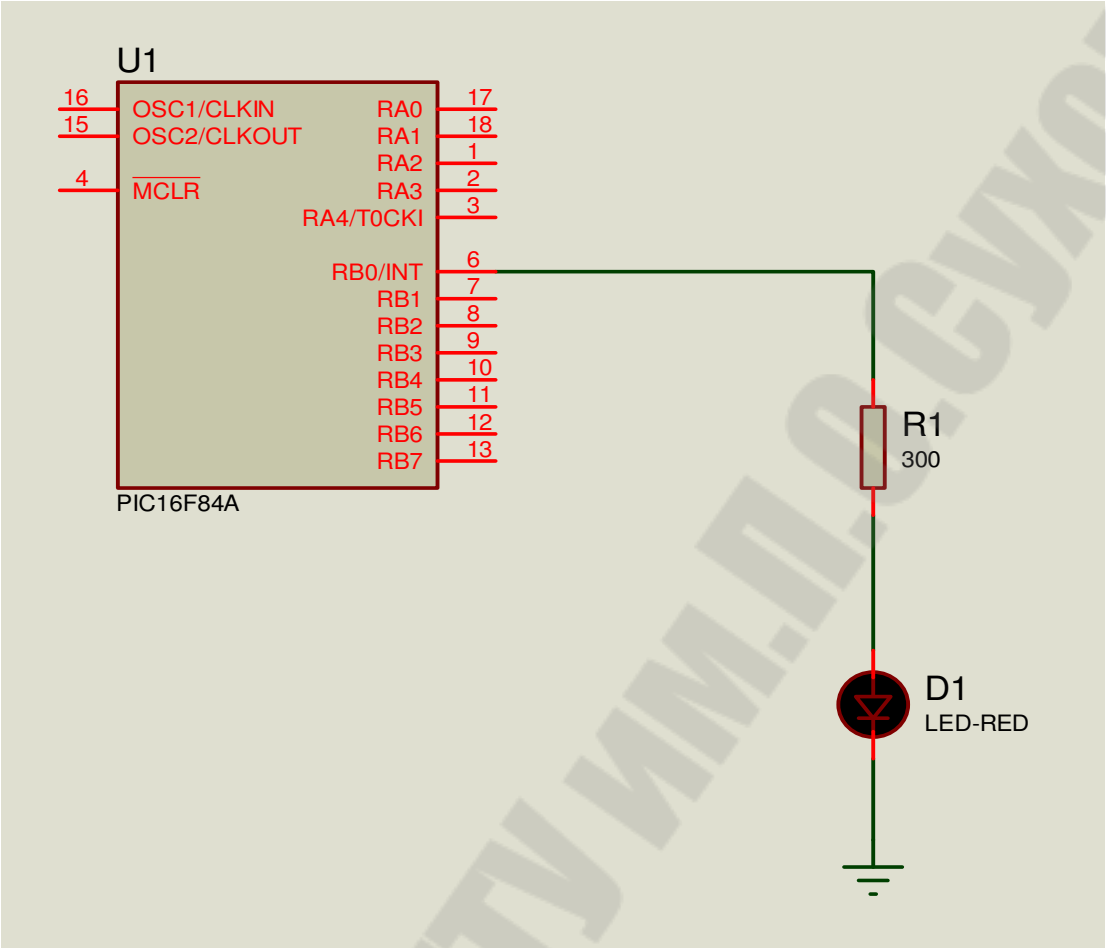

Рисунок 3. Принципиальная схема МКУ

3.3.2. Выбор элементов из библиотек производится следующим образом. Необходимо перейти в режим компонентов Component Mode, щелкнув мышью по соответствующей иконке на панели инструментов. Затем надо щелкнуть мышью по кнопке P в верхнем левом углу переключателя объектов Object Selector, либо дважды щелкнуть мышью в поле Object Selector. На экране появится окно Pick Devices библиотеки компонентов.

Компоненты (элементы схем) можно выбирать по категориям Category, подкатегориям Subcategory, по производителю Manufacturer или же искать по ключевым словам Keywords. В случае, если вы знаете название элемента, то лучший способ – искать по нему. Итак, найдем микроконтроллер. Наберите в окне Keywords слово pic16f84a. В окне результата Results появится строка с описанием выбранного компонента. Подведите курсор мыши к этой строке и щелкните левой кнопкой. Строка поменяет цвет на синий, что говорит о выделении данного компонента. Для перемещения выбранного компонента, в

данном случае PIC16F84A, в список Object Selector нужно дважды щелкнуть по выделенной строке.

Для выбора следующего элемента схемы – светодиода очистите строку Keywords и введите слово LED-RED (красный светодиод). В окне результата появится строка с описанием светодиода LED-RED из библиотеки ACTIVE. Щелкните мышью по этой строке для выделения, а затем еще дважды для перемещения выбранного элемента в список Object Selector.

Для выбора следующего элемента схемы – резистора очистите строку Keywords и введите слово RES (резистор). В окне результата появится список резисторов из библиотеки. Выберите строку RES DEVICE. Этот вид модели резистора используется для большинства цифровых схем. Для перемещения выбранного резистора в список Object Selector щелкните два раза по этой строке левой кнопкой мыши.

Итак, все элементы схемы выбраны и появились в списке Object Selector. Теперь можно закрыть библиотеку. Для этого нужно закрыть окно библиотеки или, что проще, нажать клавишу [Enter].

3.3.3. Теперь, когда элементы выбраны, необходимо перейти к следующему этапу разработки: их размещению в окне редактировании. Начнем с микроконтроллера. Щелкните левой кнопкой мыши по строке с текстом PIC16F84A. В окне предварительного просмотра появится изображение микроконтроллера. Если расположение выводов, то есть ориентация корпуса, соответствует исходной схеме, приведенной на рисунке 3, то переместите указатель мыши в середину окна редактирования и щелкните левой кнопкой. Контур микроконтроллера появится под указателем мыши и будет следовать за ним, когда мышь перемещается по окну редактирования. Поместите контур в верхнюю часть окна и щелкните еще раз левой кнопкой мыши. Изображение микроконтроллера после этого прорисуется полностью и будет помещено на схему.

Теперь щелкните мышью по строке RES. Изображение резистора появится в окне предварительного просмотра. Если его ориентация не соответствует рисунку 3, то можно развернуть его нажатием на иконки вращения панели инструментов. Затем надо поместить курсор мыши в окно редактирования и щелкнуть левой кнопкой. Перемещая мышь, надо разместить контур резистора в нужном месте схемы и еще раз щелкнуть левой кнопкой.

Далее щелкните мышью по строке LED-RED. Если ориентация светодиода будет соответствовать рисунку 3, то поместите его на схему в окне редактирования.

Нам не хватает еще одного важного элемента – «общего провода» или «корпуса». Элементы такого типа выбираются в режиме Terminals Mode, для чего требуется щелкнуть по иконке с соответствующей надписью на панели инструментов. После этого в окне Object Selector появится список доступных элементов. Точный русскоязычный эквивалент этих терминов в настоящее время не определился. Будем называть эти элементы клеммами. Выберите из списка элемент Ground (земля, общий провод) и поместите его под светодиодом. В итоге должно получиться расположение элементов, приведенное на рисунке 4.

3.3.4. Теперь необходимо соединить элементы между собой согласно принципиальной схеме, приведенной на рисунке 3. Рассмотрим методику соединения или по-другому «разводки» на примере соединения вывода 6 (RB0/INT) микроконтроллера с выводом резистора R1.

Курсор мыши, при расположении его в произвольном участке схемы, имеет вид бесцветного карандаша. Подведите курсор к выводу 6 (RB0/INT) микроконтроллера. Цвет курсора изменится на зеленый, а на выводе появится красный квадратик, показывающий, что соединение возможно. Щелкните левой кнопкой мыши. Курсор мыши изменится на бесцветный карандаш, чтобы показать, что происходит разводка, и предполагаемый путь будет следовать за движением мыши к концу проводника. При соприкосновении курсора с выводом резистора R1 цвет курсора вновь изменится на зеленый, и появится красный квадратик. Это говорит о том, что соединение возможно. Щелкните левой кнопкой мыши. На схеме появится проводник, соединяющий вывод 6 микроконтроллера с верхним выводом резистора R1.

По аналогии выполните соединения нижнего вывода резистора R1 с анодом светодиода, а затем катода светодиода с клеммой общего провода («землей»).

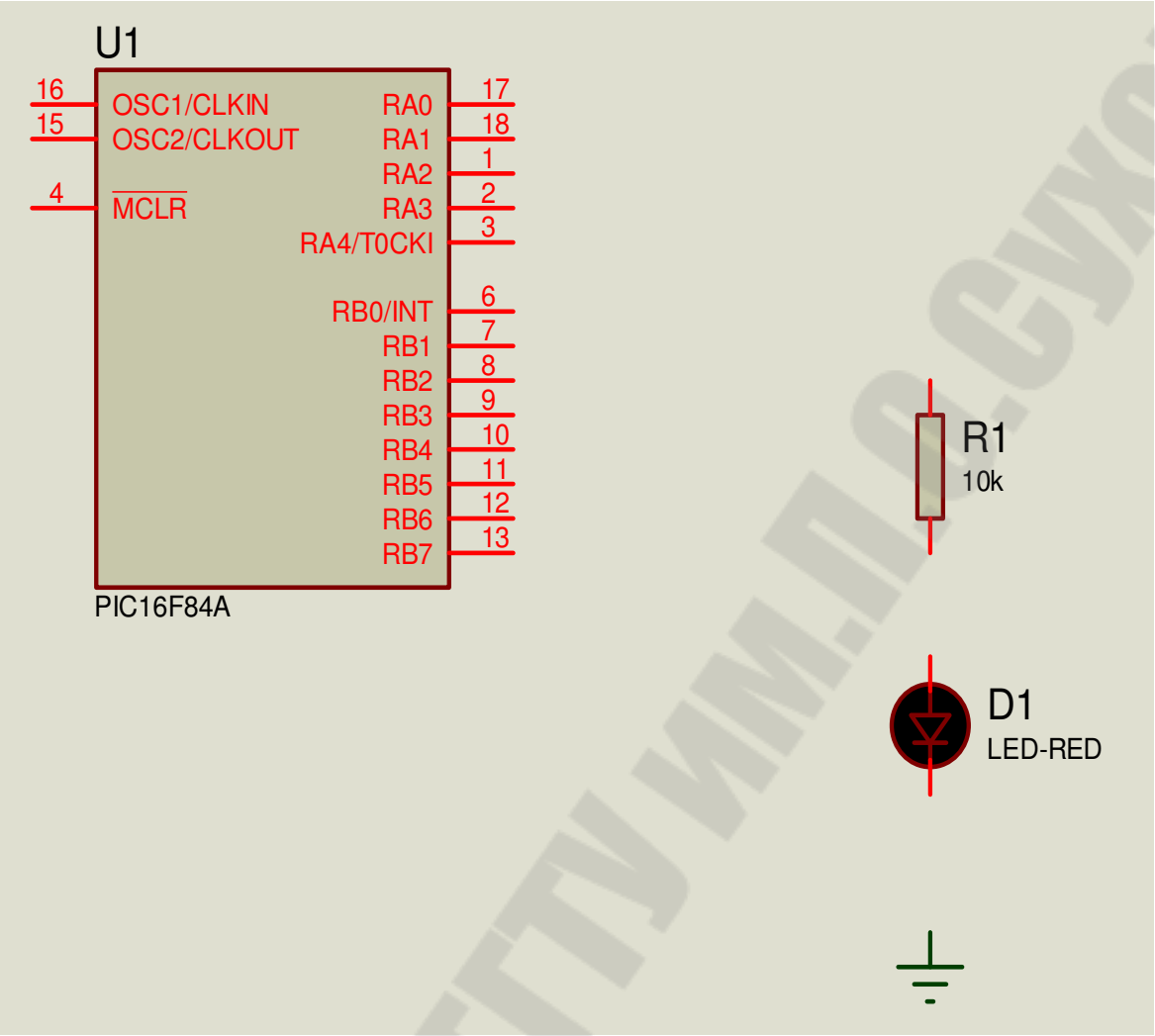

Рисунок 4. Расположение элементов на схеме МКУ

3.3.5. Теперь необходимо изменить значение сопротивления резистора на 300 Ом и изменить тип модели резистора R1 на цифровой (digital). С этой целью подведите курсор к изображению резистора R1 и дважды щелкните левой кнопкой мыши. Откроется окно редактирования свойств компонента. Введите в поле Resistance число 300 (сопротивление), а затем нажмите кнопку в поле Mode Type и выберите пункт DIGITAL. Нажмите кнопку [OK] для подтверждения выбора параметров. Выбор типа модели резистора digital (цифровой) нужен для того, чтобы симулятор не тратил время на обсчет аналоговых свойств резистора. Это позволяет разгрузить процессор компьютера при моделировании цифровых схем, где элементы работают только при двух уровнях напряжений – высоком и низком. Итак, схема МКУ готова.

Теперь нужно сохранить проект в вашей папке. Выберите пункт меню **File > Save Design As…** Раскройте папку e:\...\Lab3 и сохраните проект под именем led.dsn.

3.3.6. Разработка программы работы МКУ. Предположим, что после включения питания микроконтроллера (его сброса) светодиод будет мигать с частотой 1 Гц. Текст программы на языке Ассемблера может иметь следующий вид.

```
;*****************************************************
; led.asm – программа управления светодиодом
;*****************************************************
      list p=16f84a
      #include<p16f84a.inc>
       __CONFIG  _WDT_OFF  &  _HS_OSC
Count1  equ  0x0C          ; регистры хранения переменных
Count2  equ  0x0D
Count3  equ  0x0E          ; для подпрограммы временной задержки
        org  0x000         ; начальный адрес размещения программы
        clrf  PORTB            ; очистить регистр-защелку порта B
        bsf   STATUS, RP0  ; выбрать банк 1
        clrf  TRISB            ; настроить все линии порта B на вывод
        bcf   STATUS, RP0  ; выбрать банк 0
cycle:
        bsf   PORTB, 0     ; включить светодиод
        call   del500ms    ; задержка на 500 мс
        bcf   PORTB, 0     ; выключить светодиод
        call   del500ms    ; задержка на 500 мс
        goto cycle         ; зацикливание программы

   #include ”del500ms.asm”  ; подключить файл  del500ms.asm
                           ; подпрограммы  временной задержки
        end
```

В этой программе строка __CONFIG _CP_OFF ….. является директивой Ассемблера, под руководством которой в выходной hex-файл заносится информация о битах конфигурации микроконтроллера. Она означает:

_WDT_OFF – сторожевой таймер отключен;

_HS_OSC – высокочастотный кварцевый резонатор.

Задание 1. Запустите текстовый редактор (блокнот Windows) с помощью комбинации клавиш Shift + F4. В открывшемся диалоговом окне введите имя файла led.asm. Затем наберите текст программы и сохраните его в папке e:\....\Lab3, используя команду меню **Файл > Сохранить как …** .

Для работы программы необходима подпрограмма временной задержки del500ms на 500мс. Эта подпрограмма размещается в файле del500ms.asm, который должен находиться также в папке e:\...\Lab3. Текст подпрограммы задержки при частоте резонатора 4 МГц следующий.

```
;*****************************************************
; del500ms.asm – подпрограмма задержки на 500 мс
; при тактовой частоте 4 МГц
;*****************************************************
del500ms:
            clrf  Count1
            clrf  Count2
            movlw  .3
            movwf  Count3
loop:
            decfsz  Count1, F
            goto  loop
            decfsz  Count2, F
            goto  loop
            decfsz  Count3, F
            goto  loop
            return
```

Задание 2. Запустите текстовый редактор (блокнот Windows) с помощью комбинации клавиш Shift + F4. В открывшемся диалоговом окне введите имя файла del500ms.asm. Затем наберите текст программы и сохраните его в папке e:\....\Lab3, используя команду меню **Файл > Сохранить как …** .

3.3.7. Теперь необходимо получить выходной hex-файл для загрузки его в память МК. С этой целью откройте меню **Source** и выберите пункт **Add/Remove Source File** (добавить/удалить исходный файл). В появившемся диалоговом окне нажмите кнопку New (новый). Откроется окно New Source File, в котором нужно выбрать исходный файл. Найдите файл led.asm из вашей папки e:\...\Lab3 и на-

жмите кнопку [Открыть]. Имя файла led.asm должно появиться в строке Source Code Filename. Затем нажмите на кнопку выбора в окне Code Generation Tool (средство для генерации кода) и выберите ассемблер MPASMWIN. Подтвердите свой выбор, нажав кнопку [OK].

Теперь необходимо выполнить компиляцию и получить выходной hex-файл. Для этого выполните команду меню **Source > Build All**. Начнется компиляция программы, а затем откроется окно BUILD LOG, в котором будут сообщения о результатах построения проекта. Если ошибок нет, то появятся кружки (семафоры) зеленого цвета и сообщение об успешной компиляции Source code build completed OK. Если же вы допустили ошибки, то семафоры будут красного цвета и появятся строки с сообщениями об ошибках.

Закройте окно сообщений компилятора.

3.3.8. И, наконец, нужно записать полученный hex-код программы в память МК, или, как часто говорят, запрограммировать МК. С этой целью наведите курсор мыши на изображение МК и дважды щелкните левой кнопкой мыши. Откроется окно редактирования свойств компонента Edit Component. Щелкните по кнопке в строке Program File. Откроется окно Select File Name с hex-файлами из папки с вашими проектами e:\...\Lab3. Выберите файл с именем led.hex и нажмите кнопку [Открыть].

В строке Processor Clock Frequency (тактовая частота процессора) выставьте 4 МГц. В строке Program Configuration Word (слово конфигурации) изменять ничего не надо, так как соответствующие данные есть в исходном файле. Остальные установки также изменять пока нет необходимости.

Нажмите кнопку [OK] для подтверждения выбора параметров. Щелкните по пустому месту схемы для снятия выделения с МК.

3.3.9. Теперь можно проверить работу МКУ. С этой целью с помощью кнопки [Play] – ПУСК запустите проект на выполнение. Если замигал светодиод, то все в порядке.

Обратите внимание на маленькие квадратики около выводов МК, резистора и светодиода. Цвет этих квадратиков говорит о логических уровнях на выводах элементов в данный момент времени. Синий цвет – низкий, красный – высокий уровень.

3.4. Разработка проекта управления светодиодом с помощью кнопки

3.4.1. Теперь разработаем МКУ, в котором управление светодиодом будет выполняться с помощью кнопки. Принципиальная схема такого МКУ может иметь вид, приведенный на рисунке 5.

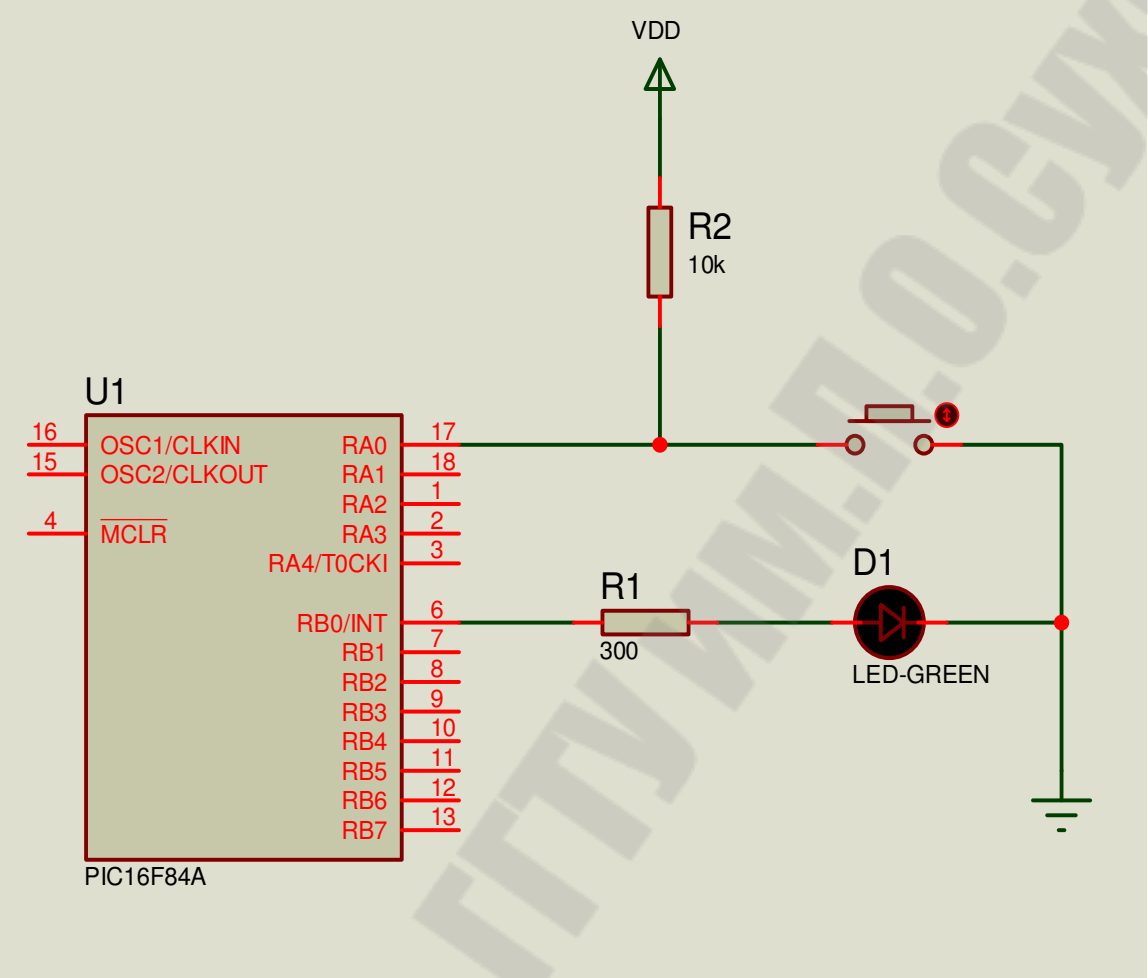

Рисунок 5. Принципиальная схема МКУ с кнопкой

Создайте новый проект, используя команду меню **File > New Design.** На экране появится диалоговое окно, в котором спрашивается, нужно ли сохранять изменения в закрываемом проекте. Нажмите на кнопку [OK]. После этого появится диалоговое окно Create New Design, предлагающее выбрать шаблон для создания нового проекта. Щелкните мышью по варианту DEFAULT (проект с параметрами по умолчанию), а затем по кнопке [OK]. На экране появится основное окно Proteus с синим прямоугольником в окне редактирования, в котором вы будете создавать схему микроконтроллерного устройства.

Согласно схеме на рисунке 5 для построения МКУ нам нужен микроконтроллер PIC16F84A, светодиод зеленого свечения, кнопка и два резистора.

Выбор элементов схемы произведите по методике, изложенной в п. 3.3.2.

Для выбора зеленого светодиода введите в строку Keywords слово LED-GREEN (зеленый светодиод). Для выбора кнопки введите в строку Keywords слово BUTTON (кнопка). В окне Results выберите строку BUTTON ACTIVE.

Остальные элементы выбираются аналогично п. 3.3.2. Когда все необходимые элементы будут выбраны, нужно закрыть библиотеку.

3.4.2. Далее необходимо разместить элементы в окне редактирования согласно схеме, приведенной на рисунке 5. Начните с микроконтроллера. Затем последовательно разместите резистор R1 и светодиод, потом – резистор R2 и кнопку.

Для выбора клемм общего провода и питания V_{DD} щелкните по иконке с надписью Terminals Mode на панели инструментов. Выберите из списка в окне Object Selector клемму Ground (земля, общий провод) и поместите ее в нужное место схемы. Затем выберите клемму Power (питание) и поместите ее над резистором R2 согласно рисунку 5.

Далее необходимо соединить элементы между собой согласно принципиальной схеме по методике, изложенной в п. 3.3.4.

3.4.3. Теперь нужно скорректировать параметры некоторых элементов схемы. С этой целью выделите резистор R1, перейдите в режим редактирования свойств и измените величину его сопротивления на 300 Ом (слово Ом вводить не надо), а также тип модели на DIGITAL. Далее выделите резистор R2 и измените тип его модели также на DIGITAL.

И, наконец, выделите клемму питания и перейдите в режим редактирования. В раскрывшемся окне на вкладке Label (метка) щелкните по кнопке в строке String. Из раскрывшегося списка выберите пункт VDD. Щелкните по кнопке подтверждения выбора [OK]. На схеме над изображением клеммы питания появится надпись V_{DD}. Это означает, что на клемму подано положительное напряжение V_{DD} (по умолчанию +5 В), которое используется для питания микроконтроллера.

Щелкните левой кнопкой по пустому месту схемы, чтобы убрать возможные выделения элементов.

Если полученная схема в окне редактирования Proteus соответствует приведенной на рисунке 5, то разработку схемы МКУ можно считать законченной.

Теперь нужно сохранить проект в вашей папке. Выберите пункт меню **File > Save Design As…** Раскройте вашу папку e:\...\Lab3 и сохраните проект под именем led_but.dsn.

3.4.4. Следующим этапом проектирования МКУ является разработка программы работы микроконтроллера. Предположим, что после включения питания МК (его сброса) светодиод не горит. При нажатии на кнопку светодиод начинает мигать с частотой 1 Гц (как в предыдущей программе). При отжатии кнопки (размыкании ее контакта) светодиод гаснет. Для упрощения программы будем предполагать, что даже при кратковременном замыкании контакта кнопки светодиод будет гореть в течение 0,5 с. Текст программы работы МКУ может иметь следующий вид.

```
;*****************************************************
; led_but.asm – программа управления светодиодом от кнопки
;*****************************************************
        list p=16f84a
        #include<p16f84a.inc>
        __CONFIG  _WDT_OFF  &  _HS_OSC
Count1  equ  0x0C           ; регистры хранения переменных
Count2  equ  0x0D           ; для подпрограммы
Count3  equ  0x0E           ; временной задержки
        org 0x000
        clrf  PORTA         ; очистить регистр-защелку порта A
        clrf  PORTB         ; очистить регистр-защелку порта B
        bsf  STATUS, RP0    ; выбрать банк 1
        movlw 0xFF          ; настроить все линии
        movwf  TRISA        ; порта A на ввод
        clrf  TRISB         ; настроить все линии порта B на вывод
        bcf  STATUS, RP0    ; выбрать банк 0
 wait:
        btfsc  PORTA, 0     ; пропустить команду, если RA0 = 0
        goto  wait          ; идти на метку, если  RA0 = 1
        bsf  PORTB, 0       ; включить светодиод
        call del500ms       ; задержка на 0,5 с
        bcf  PORTB, 0       ; выключить светодиод
        call del500ms       ; задержка на 0,5 с
        goto  wait          ; зацикливание программы

        #include ”del500ms.asm”   ; подключить файл  del500ms.asm
```

; подпрограммы задержки на 500 мс

end

Задание. Наберите текст программы управления светодиодом и сохраните его в папке e:\...\Lab3 под именем led_but.asm.

3.4.5. Далее нужно получить выходной hex-файл программы и загрузить его в память МК, то есть запрограммировать МК. Это можно сделать по методике, изложенной в пп. 3.3.7, 3.3.8.

3.4.6. После записи hex-файла программы в МК надо проверить работу МКУ. С этой целью с помощью кнопки [Play] – ПУСК запустите проект на выполнение. Согласно алгоритму программы после запуска МКУ светодиод должен быть погашен. Нажмите кнопку на схеме МКУ, щелкнув по кружку активатора около кнопки. Контакт кнопки должен замкнуться, а светодиод начать мигать с частотой 1 Гц. После этого вновь щелкните мышью по кружку активатора. Контакт кнопки должен разомкнуться, а светодиод погаснуть. Если все эти действия МКУ выполняет правильно, то можно сделать вывод, что программа работает согласно заданному алгоритму.

В заключение проверьте работу МКУ при кратковременном нажатии на кнопку. Это будет соответствовать кнопке без фиксации замкнутого положения контактов. С этой целью щелкните мышью по середине кнопки. Контакт кнопки должен кратковременно замкнуться, а затем разомкнуться. При этом светодиод должен загореться на время 0,5 с.

Задание 1 для самостоятельной работы. Используя схему МКУ, приведенную на рисунке 5, разработайте программу, выполняющую следующий алгоритм. После включения питания микроконтроллера светодиод горит. При нажатии на кнопку (даже кратковременного) диод гаснет на 5 секунд, а затем вновь загорается. Текст программы сохраните в файле с именем led_but_2.asm в папке e:\...\Lab3. Проверьте правильность работы программы на открытом проекте led_but.dsn.

Указание. Для получения временной задержки на 5 секунд можно 10 раз в цикле вызывать подпрограмму задержки del500ms. Для организации цикла (счета повторений) нужно объявить дополнительный регистр, например, с именем Count4 по адресу 0x0F.

3.5. Разработка проекта управления от микроконтроллера

семисегментным светодиодным индикатором

3.5.1. Теперь разработаем МКУ, в котором микроконтроллер выводит данные на семисегментный светодиодный индикатор. Принципиальная схема такого МКУ может иметь вид, приведенный на рисунке 6. В этой схеме используется индикатор с общими катодами. Сегменты индикатора подключены к выводам порта B, а общий вывод индикатора (катоды светодиодов) – к общему проводу схемы. Сегмент индикатора будет светиться при высоком уровне на соответствующем выводе порта B. На изображении индикатора выводы сегментов располагаются с левой стороны сверху вниз по порядку: a, b, c, d, e, f, g.

3.5.2. Создайте новый проект, используя команду меню **File > New Design.**

3.5.3. Откройте библиотеку компонентов и выберите из нее микроконтроллер PIC16F84A. Затем выберите в окне Category пункт Optoelectronics (оптоэлектронные приборы) и выделите в окне Results строку 7SEG-COM-CATODE. Это 7-ми сегментный индикатор красного цвета свечения с общими катодами светодиодов. Щелкните два раза по строке, а затем закройте библиотеку.

3.5.4. Теперь разместите элементы МКУ в окне редактирования. Вначале поместите микроконтроллер. Постарайтесь, чтобы вывод RB0/INT оказался на линии точек сетки. Это в дальнейшем упростит рисование соединений в схеме. Затем поместите в окно редактирования индикатор. Постарайтесь, чтобы верхний вывод индикатора (сегмент a) оказался на одной горизонтальной линии с выводом RB0/INT микроконтроллера. Для удобства размещения увеличьте масштаб отображения элементов. Затем перейдите в режим Terminals Mode и выберите из списка клемму Ground. Поместите ее ниже общего вывода индикатора.

3.5.5. Выполните соединение элементов между собой согласно принципиальной схеме, приведенной на рисунке 6.

После завершения разводки необходимо сохранить проект. Для этого выполните команду меню **File > Save Design As…** Раскройте папку e:\...\Lab3 и сохраните в ней проект под именем ind.dsn.

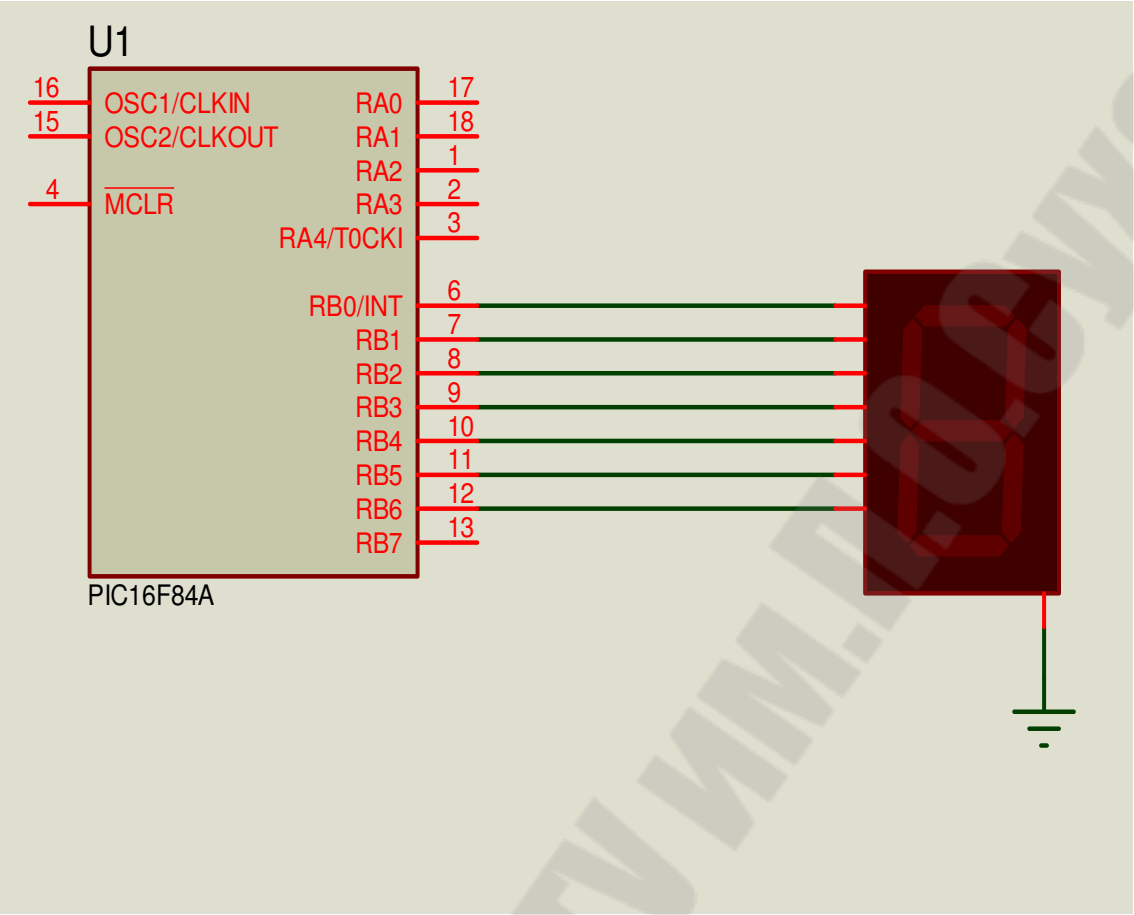

Рисунок 6. Схема МКУ для управления индикатором

3.5.7. Следующим этапом проектирования МКУ является разработка программы для МК. Пусть после включения питания на индикатор постоянно выводится определенный код символа.

Текст программы, реализующий заданный алгоритм, может иметь следующий вид.

```
;*****************************************************
; ind.asm – программа вывода на индикатор цифры 6
;*****************************************************
        list p=16f84a
        #include<p16f84a.inc>
        __CONFIG   _WDT_OFF   &   _HS_OSC
Number   equ  0x06              ; символ для вывода – цифра 6
         org  0x000
         clrf  PORTB            ; очистить триггеры-защелки порта B
         bsf  STATUS, RP0       ; выбрать банк 1
         clrf  TRISB            ; настроить все линии порта B на вывод
```

```
        bcf  STATUS, RP0   ; выбрать банк 0
again:
        movlw  Number      ; переслать в регистр W символ
        call  crosscode    ; вызов подпрограммы перекодировки
                           ; в семисегментный код,
                           ; в регистре  W – семисегментный код
        movwf  PORTB       ; вывод на индикатор
        goto  again        ;  зацикливание программы

        #include ”cross.asm”    ; подключение файла cross.asm
                                 ;  подпрограммы перекодировки
        end
```

Задание 1. Наберите текст программы управления индикатором и сохраните в папке e:\...\Lab3 под именем ind.asm.

Для работы программы необходима подпрограмма перекодировки из позиционного двоичного кода в семисегментный. Текст подпрограммы имеет следующий вид.

```
;*************************************************
; cross.asm – подпрограмма перекодировки (получения
;              семисегментного кода)
; входной параметр:  регистр W – число в двоичном коде;
; выходной параметр:  регистр W – семисегментный код числа
;*************************************************
crosscode:
        andlw  0x0F     ; обнулить старшую тетраду регистра W
        addwf  PCL,F    ; сложить рег. W с PCL и переслать
                        ; результат в PCL
        retlw  0x3F  ; возврат из подпрограммы с кодом символа ‘0’
        retlw  0x06  ; возврат из подпрограммы с кодом символа ‘1’
        retlw  0x5B  ; возврат из подпрограммы с кодом символа ‘2’
        retlw  0x4F   ;  возврат из подпрограммы с кодом символа ‘3’
        retlw  0x66    ; возврат из подпрограммы с кодом символа ‘4’
        retlw  0x6D   ; возврат из подпрограммы с кодом символа ‘5’
        retlw  0x7D   ; возврат из подпрограммы с кодом символа ‘6’
        retlw  0x07    ; возврат из подпрограммы с кодом символа ‘7’
        retlw  0x7F   ; возврат из подпрограммы с кодом  символа ‘8’
        retlw  0x6F   ; возврат из подпрограммы с кодом символа ‘9’
        retlw  0x77   ; возврат из подпрограммы с кодом символа ‘A’
```

```
retlw  0x7C   ; возврат из подпрограммы с кодом символа ‘B’
retlw  0x39   ; возврат из подпрограммы с кодом символа ‘C’
retlw  0x5E   ; возврат из подпрограммы с кодом символа ‘D’
retlw  0x79   ; возврат из подпрограммы с кодом символа ‘E’
retlw  0x71   ;  возврат из подпрограммы с кодом символа ‘F’
```

Задание 2. Наберите текст подпрограммы перекодировки и сохраните его в файле с именем cross.asm в папке e:\...\Lab3.

3.5.8. Получите выходной hex-файл по методике, изложенной в п. 3.3.7, и загрузите его в память МК (тактовая частота процессора должна быть 4 МГц).

3.5.9. Проверьте работу МКУ, запустив проект на выполнение. Если на индикаторе появилась цифра 6, то все в порядке.

Задание 2 для самостоятельной работы. Используя схему МКУ, приведенную на рисунке 6, разработайте программу, выполняющую следующий алгоритм. После включения питания микроконтроллера на индикатор выводится символ F, а затем он начинает мигать с частотой 1 Гц (подобно светодиоду в схеме на рисунке 1). Текст программы сохраните в файле с именем ind_2.asm в папке e:\...\Lab3. Проверьте правильность работы программы на открытом проекте ind.dsn.

Задание 3 для самостоятельной работы. Используя схему МКУ, приведенную на рисунке 6, разработайте программу, выполняющую следующий алгоритм. После включения питания микроконтроллера на индикатор через каждую секунду выводятся последовательно символы от 0 до F, то есть все 16 значений из подпрограммы crosscode. Цикл вывода бесконечен. Текст программы сохраните в файле с именем ind_3.asm в папке e:\...\Lab3. Проверьте правильность работы программы на открытом проекте ind.dsn.

4. Содержание отчета

Наименование и цель работы. Схемы МКУ и тексты программ всех заданий для самостоятельной работы (комментарии в программах обязательны!).

Контрольные вопросы

1. Какие функции выполняет программный пакет Proteus?

2. Из каких частей состоит пакет Proteus?

3. Как выбираются компоненты принципиальной схемы в Proteus?

4. Каким образом можно перемещать схему по экрану дисплея?

5. Как можно удалить из схемы какой-либо компонент?

6. Каким образом можно изменить величину сопротивления резистора на схеме?

7. Как можно загрузить программу в микроконтроллер в среде Proteus?

Литература

1. Новиков, Ю.В. Основы микропроцессорной техники: курс лекций для вузов / Ю.В. Новиков, П.К. Скоробогатов. – М.: ИНТУИТ РУ, 2009.

2. Микропроцессорные системы: учебное пособие для вузов / Е.К. Александров и др. Под общ. ред. Д.В. Пузанкова. – СПб.: Политехника, 2002.

Содержание

Виноградов Эдуард Михайлович

ПРОГРАММИРОВАНИЕ PIC-МИКРОКОНТРОЛЛЕРОВ НА ЯЗЫКЕ АССЕМБЛЕР

Практикум
по дисциплине «Микропроцессорная техника»
для студентов специальности 1-36 04 02
«Промышленная электроника»
дневной и заочной форм обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 20.10.16.
Рег. № 11Е.
http://www.gstu.by