Министерство образования и науки РФ
Федеральное государственное образовательное учреждение
высшего образования
«Владимирский Государственный Университет
имени А.Г. и Н.Г. Столетовых»

"Микропроцессорные устройства"

Методические указания к лабораторным работам по курсу
“Микропроцессорная техника в системах связи”,
для студентов дневного отделения
по направлению 11.03.02 «Инфокоммуникационные технологии и системы связи»

.

(электронный ресурс)

Владимир - 2018 г.

Составил – Давыдов Г.Д.

Методические указания к лабораторным работам по курсу “Микропроцессорная техника в системах связи” для студентов дневного отделения бакалавриата "Института инновационных технологий и радиоэлектроники" по направлению 11.03.02 «Инфокоммуникационные технологии и системы связи» федерального государственного образовательного учреждения высшего образования «Владимирский Государственный Университет имени А.Г. и Н.Г. Столетовых» (электронный ресурс) – Владимир: Изд-во ВлГУ,2018. - 37 с.

Так как данный курс читается в течении одного семестра и ему не прешествуют дисциплины по вычислительной технике и программированию, в качестве примера выбран микроконтроллер серии *PIC16F* , отличающийся относительной прозачностью структуры и простотой системы команд, но содержащий все элементы характерные для современной микропроцессорной техники. Методические указания содержат описания четырех лабораторных работ позволяющих ознакомиться с устройством и применением современных микроконтроллеров с *RISK* архитектурой.

В первой работе на примере готовой программы изучается практика трансяции программ с помощью среды проектирования программ *MPLab*. Во второй работе студенты практикуются в отладке простейшей программы на языке ассемблера. Третья работа посвящена моделированию функционирования микроконтроллера в составе электрической схемы генератора колебаний сложной формы. В последней работе цикла изучается устройство и программировани портов цифрового ввода/вывода.

Материалы могут быть полезны студентам электронных, компьютерных и информационных специальностей, а также аспирантам, преподавателям и всем желающим, которые занимаются изучением основ микропроцессорной техники.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные микроконтроллеры наряду с программируемыми интегральными микросхемами являются основными компонентами почти всех средств телекоммуникационной техники. На их основе специализированных цифровых устройств решаются такие задачи как построение: средств управления, синтезаторов частоты, цифровых фильтров, модуляторов и демодуляторов, реализующих современные сложные виды модуляции, и т.д.

На рынке представлен достаточно широкий спектр микроконтроллеров предназначенных для самых различных областей применения и назначения. К ним относятся: простейшие 8-ми разрядные микроконтроллеры низкой стоимости и с малым потреблением для целей ввода управляющих данных и индикации, 16-ти разрядные средней сложности и производительности, 32-х разрядные средней и высокой производительности, а так же многоразрядные процессоры для цифровой обработки сигналов. Эти микроконтроллеры значительно отличаются по сложности построения, но имеют много общего между собой в смысле идеологии построения, программирования и применения. Это обусловлено тем, что почти все современные микроконтроллеры построены по гарвардской RISK архитектуре с различными модификациями и расширениями. Дополнительным объединяющим фактором является широкое применение трансляторов и программных сред для автоматизации и управления процессом разработки программ на основе персональных компьютеров.

Методические указания к лабораторным работам по дисциплине "Микропроцессорная техника в системах связи" предназначены для студентов, обучающихся по специальности 11.03.02 "Инфокоммуникационные технологии и системы связи" во Владимирском Государственном Университете имени А.Г. и Н.Г. Столетовых. Указанная дисциплина согласно учебному плану читается в течение четвертого семестра. В связи с ограниченным объемом и отсутствиемна младших курсах дисциплин по прогаммированию и средствам вычислительной техники данный курс ориентирован на изучение студентами общих принципов построения микропроцессорных устройств, их программирования, отладки программ и применения в качестве встраиваемых элементов управления телекоммуникационными устройствами.

В соответствии с указанными ограничениями лабораторный цикл базируется на микроконтроллерах серии PIC16 классической RISK-архитектуры с моделированием функционирования микроконтроллера в среде проектирования *MPLab* и пакета *Multisim.*

Лабораторная работа № 1.

СРЕДА ПРОЕКТИРОВАНИЯ ПРОГРАМ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

ЦЕЛЬ РАБОТЫ

Изучение *RISK* архитектуры микропроцессоров на примере восьмиразрядных однокристальных микроконтроллеров фирмы *MICROCHIP* семейства ***PIC16F*** и системы проектирования программ микроконтроллерных устройств на основе среды ***MPLAB IDE***.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Архитектура микроконтроллеров *PIC16F87X*.

Микроконтроллеры ***PIC16F87X*** относятся к группе наиболее полных представителей семейства ***PIC16***.

Основные особенности архитектуры восьмиразрядных однокристальных микроконтроллеров ***PIC16F87X***:

- система прерываний (до 14 источников);
- 8-уровневый аппаратный стек;
- сторожевой таймер ***WDT*** с встроенным *RC*-генератором;
- режим энергосбережения;
- программный выбор параметров тактового генератора;

Арифметико-логическое устройство (АЛУ):

- высокоскоростная ***RISC*** архитектура арифметико-логического устройства;
- 35 инструкций;
- прямой, косвенный и относительный способы адресации;
- все команды выполняются за один командный цикл (4 такта) кроме инструкций переходов, выполняемых за два цикла;
- тактовая частота 0 – 20 МГц;

Память:

- 8192 14-разрядных слов памяти программ, выполненной по технологии ***FLASH*** (электрически перезаписываемое запоминающее устройство);
- 368 8-разрядных байт регистровой памяти (оперативное запоминающее устройство, ОЗУ, память данных или оперативная память);

- 256 8-разрядных байт флэш-памяти данных (электрически перезаписываемое запоминающее устройство).

Периферийные модули:

- 33 линии цифрового параллельного ввода-вывода (порты ***A*** – 6 разрядов; ***B***, ***C***, ***D*** – 8 разрядов; ***E*** – 3 разряда) с повышенной нагрузочной способностью (25 мА);

- таймер 0: 8-разрядный таймер/счетчик с 8-разрядным предделителем;

- таймер 1: 16-разрядный таймер/счетчик с возможностью подключения внешнего генератора;

- таймер 2: 8-разрядный таймер/счетчик с 8-разрядным предделителем и выходным делителем;

- два модуля сравнения/захват/ШИМ (***CCP***):

16-разрядный захват (максимальная разрешающая способность 12.5 нс);

16-разрядное сравнение (максимальная разрешающая способность 200 нс);

10-разрядный ШИМ;

- многоканальный 10-разрядный АЦП;

- последовательный синхронный порт ***MSSP***, работающий в режимах ведущий/ведомый ***SPI*** и ведущий/ведомый режим ***I2C***;

- последовательный синхронно-асинхронный приемопередатчик ***USART*** с поддержкой детектирования адреса;

- ведомый 8-разрядный параллельный порт (***PDP***) с поддержкой внешних сигналов ***–RD***, ***-WR***, ***-CS***.

Структурная схема микроконтроллера ***PIC16F788*** приведена на рис.1.

1.2. Среда проектирования программного обеспечения *MPLAB-IDE*

Интегрированная среда разработки и отладки ***MPLAB-IDE*** позволяет создавать и редактировать текст программ, преобразовывать текст программ в исполняемый код, а также осуществлять отладку перемещаемого (объектного) и абсолютного (исполняемого, т.е. двоичного) кода.

MPLAB-IDE включает в себя:

Менеджер проекта *MPLAB*, используемый для создания и работы с файлами, относящимися к проекту;

Редактор *MPLAB*, предназначенный для написания и редактирования исходного текста программы, шаблонов и файлов сценария линкера;

***MPLAB* ассемблер/компилятор/компоновщик/редактор библиотек** компилирует (преобразует в исполняемый код) исходный текст программы, связывая различные модули и библиотеки.

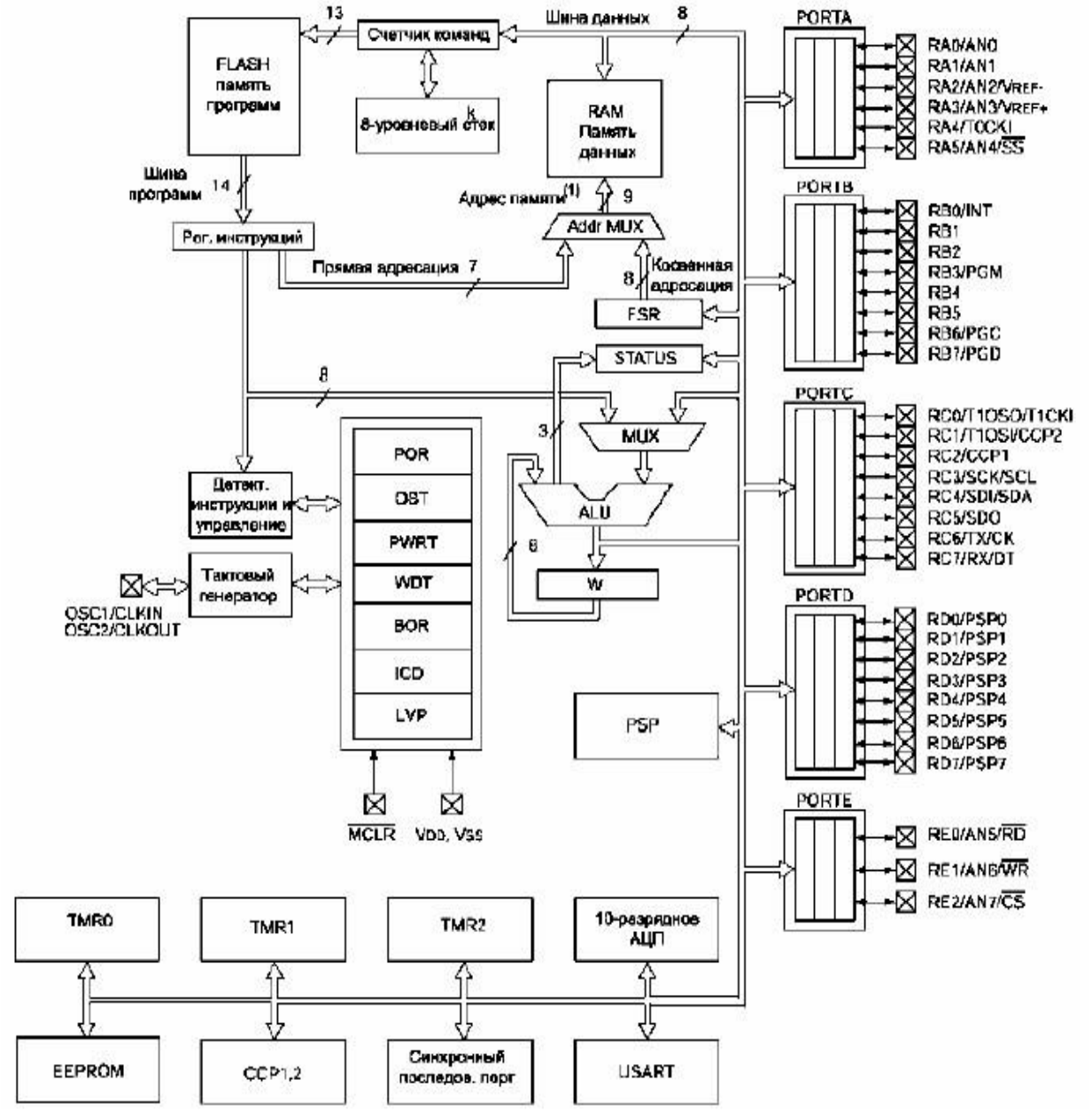

Рис. 1. Структурная схема микроконтроллера ***PIC16F788***

Редактор *MPLAB*, предназначенный для написания и редактирования исходного текста программы, шаблонов и файлов сценария линкера;

***MPLAB* ассемблер/компилятор/компоновщик/редактор библиотек** компилирует (преобразует в исполняемый код) исходный текст программы, связывая различные модули и библиотеки.

MPLAB-SIM - имитатор выполнения программы в микроконтроллере;

MPLAB ICD2 - программатор-отладчик, предназначенный для программирования встроенной памяти программ микроконтроллеров и обеспечивающий управляемую работу микроконтроллера в масштабе реального времени непосредственно в устройстве пользователя (внутрисхемная отладка и программирование).

1.3. Основы построения программ

Для того чтобы программа начала выполняться микропроцессором ее необходимо составить на одном из языков программирования, транслировать в исполняемый двоичный файл с расширением ***.hex*** и записать в программную память микропроцессора и включить питание.

Чтобы получить двоичный файл программы, требуется выполнить трансляцию с исходного языка в формате текстового редактора, например редактора Блокнот, в абсолютный двоичный код. В среде ***MPLAB-IDE*** это делается модулями компилятора *MPASM* (файл mpasmwin.exe) и компоновщика *MPLINK* (файл *mplink.exe*), запускаемыми директивами из меню среды.

Компилятор *MPASM* может использоваться в двух режимах:

для генерации абсолютного кода, который может быть загружен непосредственно в микроконтроллер;

для генерации объектных файлов, которые связываются с другими компилированными модулями.

По умолчанию *MPASM* генерирует абсолютный код (Рис.2). При компиляции исходного файла в этом режиме, все значения адресов должны быть явно указаны в исходном файле или во включаемых файлах. Если компиляция выполнена без ошибок, то будет создан двоичный код программы (*HEX* файл), который можно использовать для непосредственного программирования микроконтроллера.

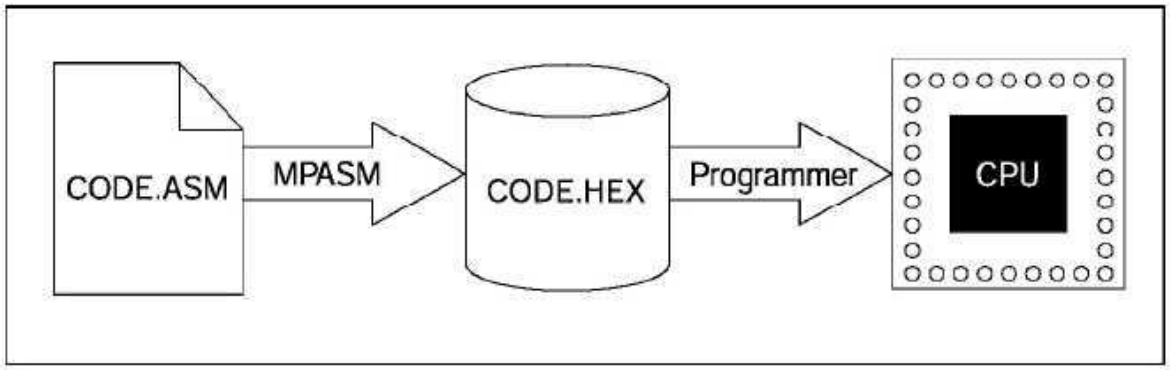

Рис. 2

Компилятор *MPASM* имеет возможность генерировать объектные модули в перемещаемом формате. Объектные модули являются промежуточными продуктами. Их применение позволяет разрабатывать программу по частям и использовать имеющиеся заготовки, модули от других программ и типовые библиотечные модули. Их начальный адрес и адреса переходов не определены и существуют в виде меток. С помощью программы компоновки *MPLINK* объектные модули объединяются в один модуль, связываются друг с другом и получают абсолютные адреса для окончательного формирования исполняемого (абсолютного) кода. Данный метод позволяет многократно использовать ранее подготовленные модули программы. Объектные файлы могут быть сгруппированы в библиотечные файлы с помощью программы *MPLIB*. Библиотеки могут указываться в качестве параметра во время компоновки. Компоновщик включает в исполняемый код только необходимые процедуры, а не все модули, содержащиеся в библиотеке.

Схема компиляции проекта из нескольких файлов приведена на рис.3.

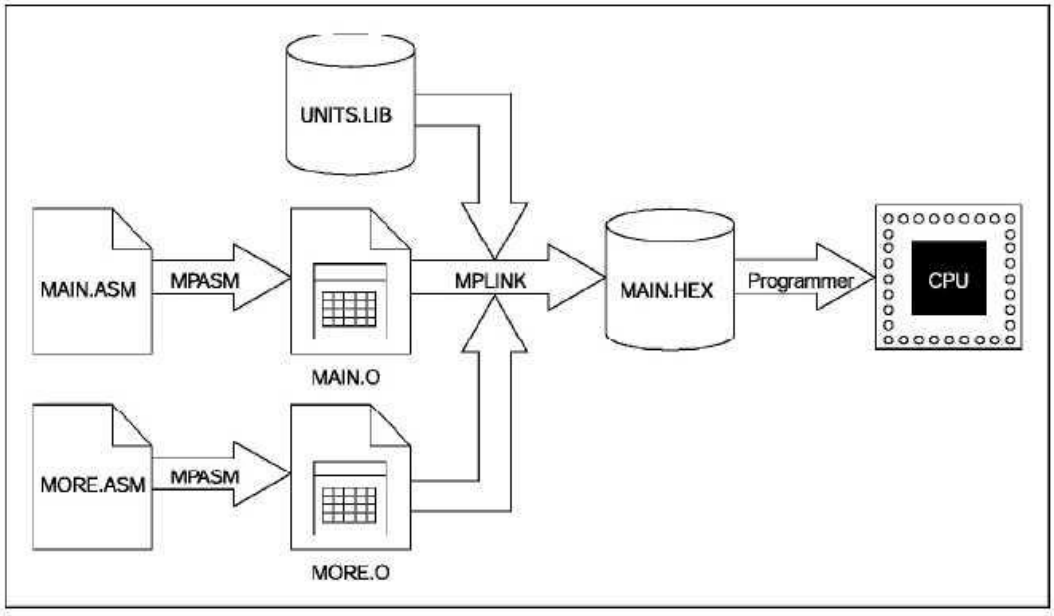

Рис 3

Запись программы в память производится специальным устройством - программатором. Процесс записи называют программированием микропроцессора или прошивкой. В настоящее время прошивка выполняется в автоматическом режиме из двоичного файла программы под управлением персонального компьютера (ПК). Программатор входом подключается к ПК, а выходом к микропроцессору. На ПК запускается ***MPLAB-IDE*** или специальная программа, управляющая процессом прошивки, которая запрашивает у пользователя путь к двоичному файлу программы.

1.4. Интерфейс *MPLAB-IDE*

На рабочем столе среды (рис. 4) находятся:

1. Главное текстовое меню.
2. Графическое меню.
3. Рабочая область.
4. Линейка состояния.

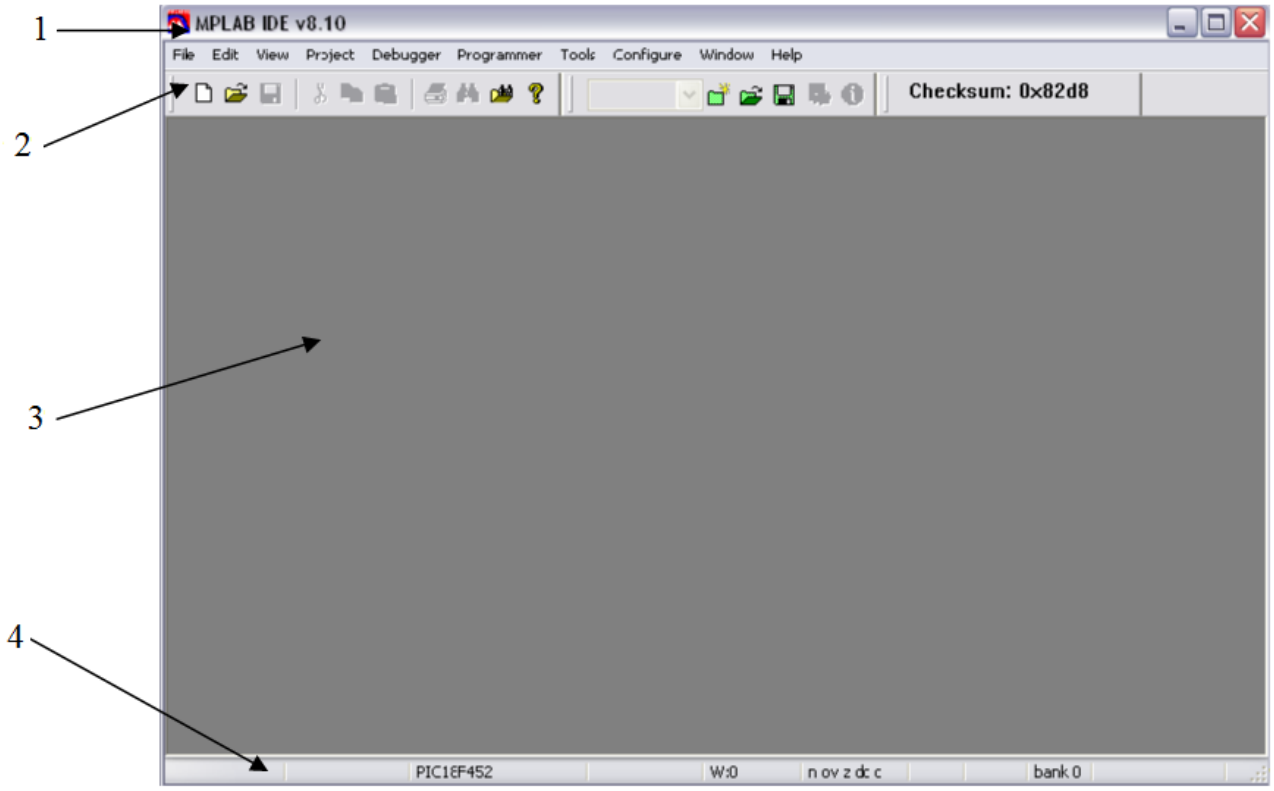

Рис.4. Рабочий стол среды ***MPLAB IDE***

В рабочей области *MPLAB IDE* можно одновременно открывать несколько рабочих окон подобных окнам Windows. К основным окнам можно отнести:

Workspace – описание проекта, содержащее состав проекта и назначение, входящих в него файлов.

Окно **редактор**а исходных текстов – с его помощью вводятся и редактируются тексты программных модулей.

Output –окно результатов работы с проектом.

Каждое окно открывается автоматически по мере необходимости. Принудительно окно может быть открыто с помощью пункта *View* главного меню. Закрытие окна не влияет на данные, содержащиеся в нем.

1.5. Подготовка к проектированию

В процессе проектировании программного обеспечения среда ***MPLAB IDE*** ведет автоматическое сопровождение проекта. Для этого все файлы, необходимые для построения программы, помещаются в отдельный каталог проекта. Исходные файлы, библиотеки и другие, ранее подготовленные файлы, необходимые для построения программы, копируются в каталог проекта программистом. Все вновь создаваемые средой файлы автоматически помещаются в этот каталог. При создании проекта средой набор файлов, предоставленный пользователем, дополняется двумя служебными файлами с расширениями ***.mcp*** и ***.mcv***, в которых система хранит данные о составе и связях исходных и созданных при построении файлов проекта. Сюда же среда помещает промежуточные, вспомогательные и выходные файлы, включая файл абсолютного модуля программы, который и является основным итогом работы.

Свой каталог проекта следует создать до начала работы в среде. Особенностью данной среды проектирования является ограниченная обработка данных в альтернативных шрифтах. Альтернативные шрифты (кириллица) могут использоваться только в поле комментарий текста программы и в обрабатываемых проектируемой программой данных. В остальных случаях должен использоваться латинский шрифт. По этой причине имя каталога проекта и путь к нему могут содержать **только** символы **латинского** алфавита. Практически следует создать свой каталог проекта в специально выделенном рабочем каталоге ***C:Work***, куда средствами *ОС Windows* надо скопировать все исходные файлы. В качестве имени каталога лучше использовать свою фамилию латинскими буквами с номером группы.

В данной лабораторной работе используется простейший режим построения программы (трансляции): из исходного текста сразу в абсолютный двоичный формат. Поэтому проект перед построением может содержать только один файл - файл исходного текста программы в формате ASCI. Такой режим работы транслятора считается демонстрационным. В лабораторной работе в качестве примера исходного текста программы используем стандартный файл шаблона программы

C:\Program Files\Microchip\MPASM Suite\Template\Object\PIC16F877tmpo.asm.

В шаблонах среды **MPLAB IDE** прописаны типовые установки микроконтроллера пригодные во многих случаях. В эту заготовку можно в дальнейшем вписать свою программу. Шаблон рассчитан на создание программы в перемещаемом формате. С целью упрощения лабораторная работа выполняется в абсолютном формате, поэтому при трансляции появятся сообщения об ошибках связанных с наличием директив определения

данных перемещаемого формата. Строки с ошибками можно просто удалить или закомментировать.

1.6. Настройка *MPLAB-IDE*

Настройка ***MPLAB IDE*** состоит в задании типа микроконтроллера, слова конфигурации и средств отладки.

Задание типа микроконтроллера производится в п. ***Configure>Select Device…*** главного меню (Рис.3).

На панели этого меню виде светящихся индикаторных светодиодов показаны средства проектирования и отладки программы, которые могут быть применены с данным типом микроконтроллера: программаторы, языки программирования и отладчики. Зеленый цвет индикатора означает наличие и доступность средства, красный - средство отсутствует в данной комплектации оборудования или не может быть применено. В данной лабораторной работе используется ассемблер и моделирующая программа *MPLAB SIM*, называемая симулятором. Такие средства как внутрисхемный аппаратный программатор-отладчик *MPLAB ICD* и аппаратный эмулятор *MPLAB ICE*, в этой лабораторной работе не используются и к компьютеру не подключены.

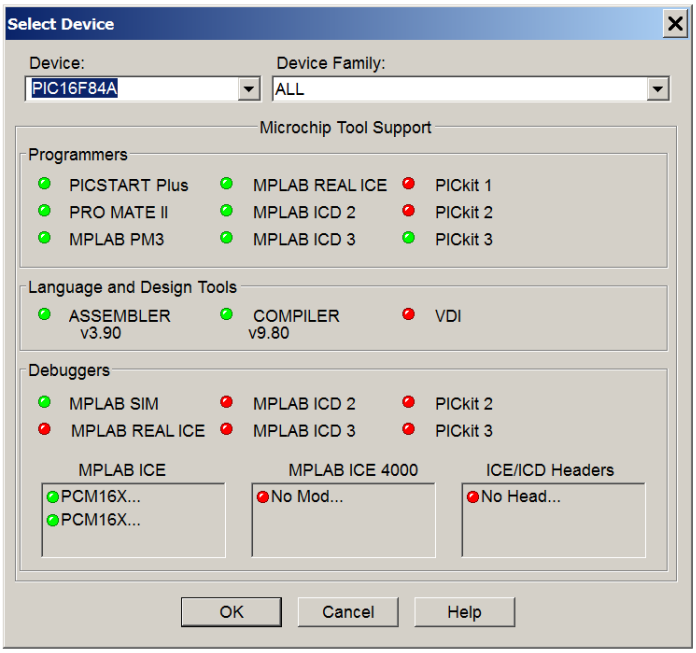

Рис.3. Окно выбора типа микроконтроллера

Следующим этапом настройки среды *MPLAB* является задание конфигурации процессора. Под этим понимается изменение внутренней схемы и режимов работы узлов контроллера, которые допускают многовариантное использование. Режимы работы этих узлов задаются аппаратно содержимым ячейки конфигурации с адресом индивидуальным

для каждого процессора (для *PIC16F84A* – 2007). От отдельных битов её включаются или выключаются различные устройства микроконтроллера. По умолчанию средой *MPLAB IDE* в ней задаётся код 3FFF. Для конкретных типов микроконтроллеров семейства код конфигурации может иметь разный смысл, так как состав устройств в процессоре зависит от типа, и они отличаются друг от друга.

Конфигурация может задаваться в исходном тексте головной управляющей программы или в среде *MPLAB IDE* с помощью меню ***Configure>Configuration Bits…*** (Рис.4). При задании конфигурации в среде MPLAB IDE метка в строке ***Configuracion Bits set in code*** убирается.

Самым значимым узлом является тактовый генератор (OSC - Oscillator). По умолчанию обычно устанавливается RC –генератор. Самый медленный и простой. Часто это встроенный в микроконтроллер генератор, не требующий внешних соединений, но обычно требуется присоединить к контактам микросхемы процессора резистор с конденсатором. Наиболее часто применяют кварцевый или пьезокерамический резонатор.

В задании предусматривается выключение сторожевого таймера (*WDT*), чтобы упростить учебные программы, и запрещение записи во внешнюю память программ (*WRT_ENABLE*), так как она считается отсутствующей.

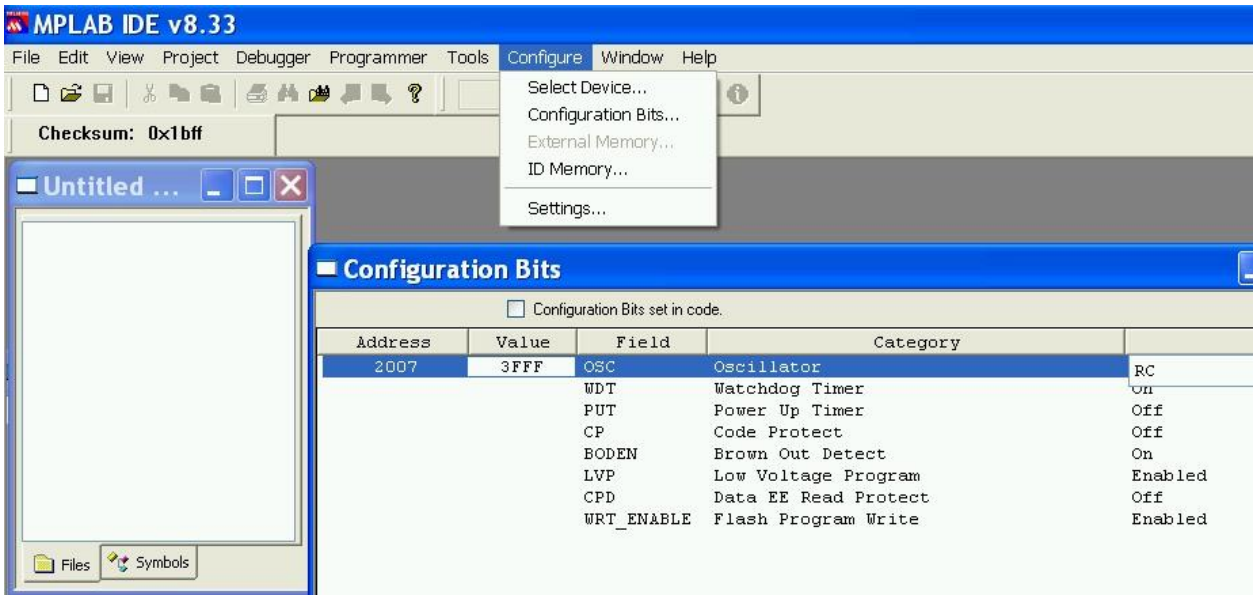

Рис. 4. Конфигурация микроконтроллера

Для выбора инструментального средства отладки, используемого в проекте используйте пункт *Debugger>Select Tool*(Рис.5) в главном текстовом меню.

На первых этапах используют симулятор ***MPLAB SIM.*** С ним можно выполнить предварительную отладку программы путем моделирования работы программы, т.е. в

виртуальном режиме без использования аппаратных средств. Это удобно, так как на первом этапе отладки можно обойтись без изготовления макета.

После того как MPLAB настроен, в линейке состояния включатся индикаторы “***MPLAB SIM***” и “***PIC16F877***”.

1.7. Проектирование в среде *MPLAB IDE*

1.7.1. Создание проекта в *MPLAB*

Создание проекта является первым действием в процессе проектирования в среде. Собственно создание проекта производится с помощью пункта ***Project>Project Wizard*** в главном меню MPLAB.

На первом шаге (Рис.5) выбираем тип микроконтроллера.

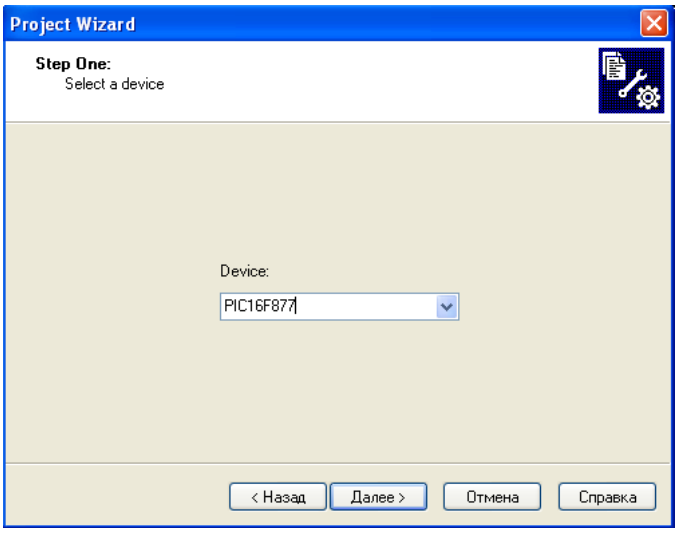

Рис.5.

На втором шаге определяем используемые средства трансляции программы с помощью окна показанного на рис. 7. Для программирования на языке ассемблера меню ***Active Toolsuite*** надо выбрать ***Microchip MPASM Toolsuite,*** далее в меню ***Toolsuite Contents*** выбрать ***MPASM Assembler*** **(*mpasmwin.exe*).** В меню ***Location*** указанный путь должен совпадать с фактическим местонахождением файла mpasmwin.exe. Если имеется расхождение, отыскать файл компилятора с помощью клавиши ***Browse…*** и указать его правильный путь.

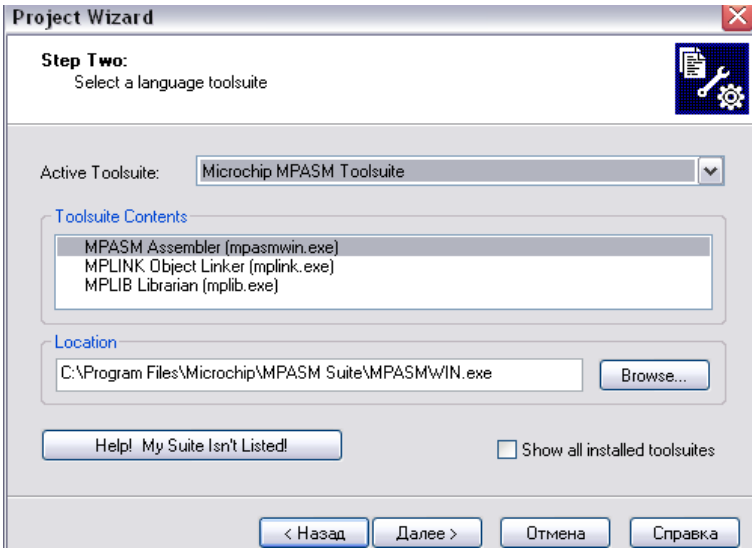

Рис.6.

На третьем шаге (Рис.7) с помощью клавиши ***Browse…***. задается путь сохранения результатов и имя проекта в каталоге проекта, созданном при подготовке

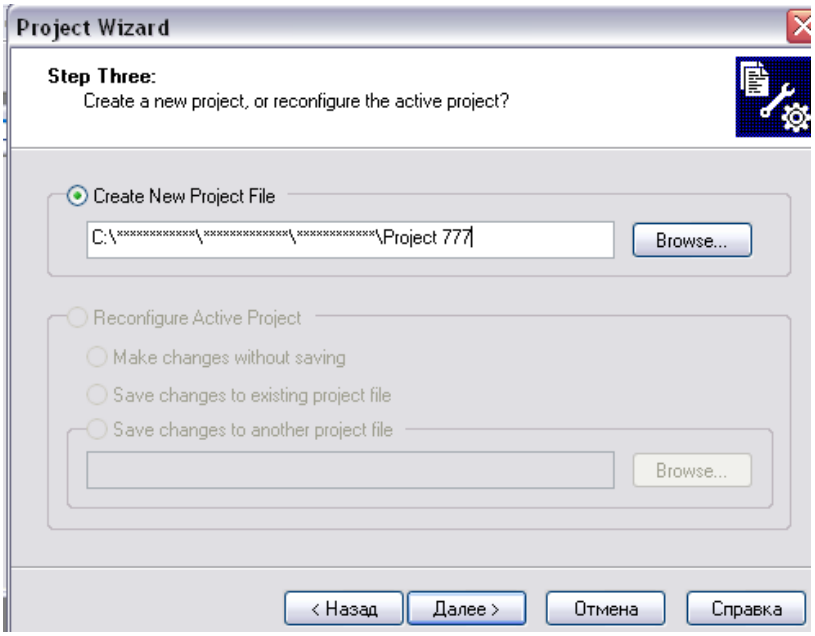

Рис.7.

После третьего шага заготовка проекта фактически создана. Если создается пустой проект, т.е. не содержащий файлов, следующий шаг надо пропустить, нажав клавишу **Далее >**.

Если же имеются ранее заготовленные файлы, которые требуется включить в проект, это можно сделать на четвертом шаге с помощью панели ***Step Four.*** Для присоединения

достаточно найти нужный файл на диске и выбрать *Add* (Рис.8 и 9). При этом присоединять файлы можно только из каталогов пользователя, в которых пользователю доступны не только чтение, но и перезапись файлов. Файлы из системных каталогов присоединять не следует: они по факту не присоединятся (сработает защита операционной системы) или могут искажаться с нарушением работы среды программирования.

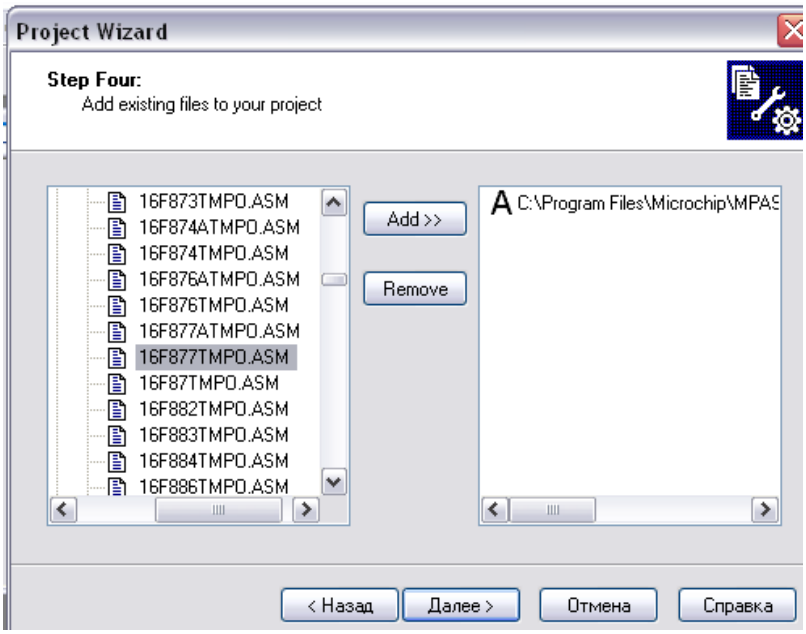

Рис.8.

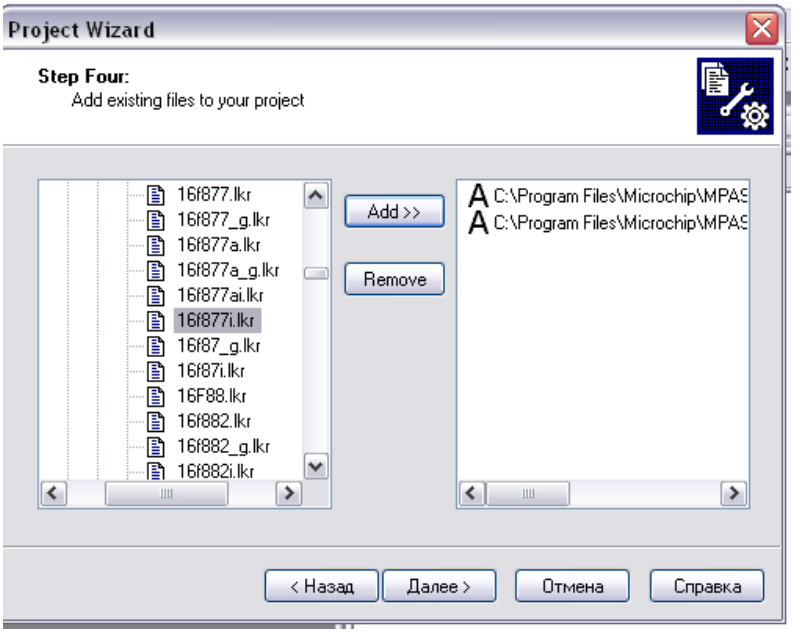

Рис.9.

После завершения четвертого шага клавишей **Далее >**, в итоговом меню проверяем правильность имен и подтверждаем это клавишей **Готово**. При этом проект сохраняется в каталоге проекта, в котором записываются служебные файлы с расширениями ***.mcp***, ***.mcw*** и ***.mcs***, хранящие данные о состоянии проекта.

1.7.2. Присоединение рабочих файлов к проекту

Присоединить файлы к созданному проекту можно с помощью оператора *Proect>Add Files to proect* **…** . Так как работа выполняется студентами в непривилегированном режиме, присоединять файлы можно только из своего каталога, созданного при подготовке (См. п.2.2.3). Присоединяемые файлы должны иметь расширения имен соответствующие их роли в системе программирования среды ***MPLAB IDE***. В случае программирования на языке ассемблера возможны следующие расширения: ***.asm*** - исходный текст головной программы и модулей на языке ассемблера в коде *ASCII*, ***.h*** - стандартные заголовки среды в коде *ASCII*, ***.inc*** - стандартные подключаемые файлы среды в коде *ASCII*, ***.lkr*** - файлы управления компоновщиком, ***.o*** объектные модули пользователя, ***.lib*** -библиотечные файлы.

1.7.3. Построение проекта

После создания проекта, в рабочей области должно быть окно ***Workspace*** с названием вашего проекта и деревом включенных в него файлов. Если этого окна нет, то надо в главном текстовом меню выбрать пункт *View>Project* для появления окна (Рис.10).

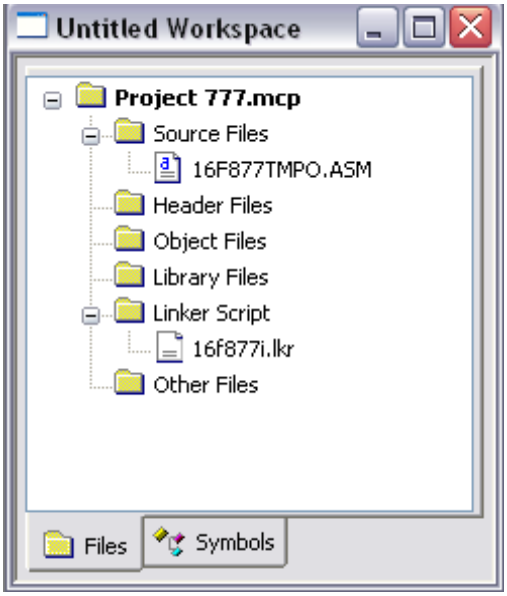

Рис.10.

Трансляция программ *PIC*-контроллеров производится компилятором *MPASM*, который текстовый файл программы на языке ассемблера преобразует в двоичный код программы перемещаемого формата (объектный код в виде файла с расширением ***.o***), и компоновщиком *MPLINK*, который объединяет все объектные файлы проекта в один файл в двоичном абсолютном коде с расширением имени файла ***.hex***. Все создаваемые транслятором файлы помещаются в каталог проекта.

Запуск процесса трансляции производится с помощью п. *Project>Build All*. Этим запуститься процесс автоматической компиляции и компоновки. По умолчанию трансляция выполняется в абсолютном формате, но некоторые виды ошибок ставят это под сомнение и среда выводит запрос пользователю с требованием указать формат результата. В лабораторной работе в этом случае следует выбрать абсолютный формат.

Если в процессе трансляции не выявляются ошибки в окне ***OUTPUT*** появится сообщение об успешной трансляции ***BUILD SUCCEEDED*** (Рис. 11).

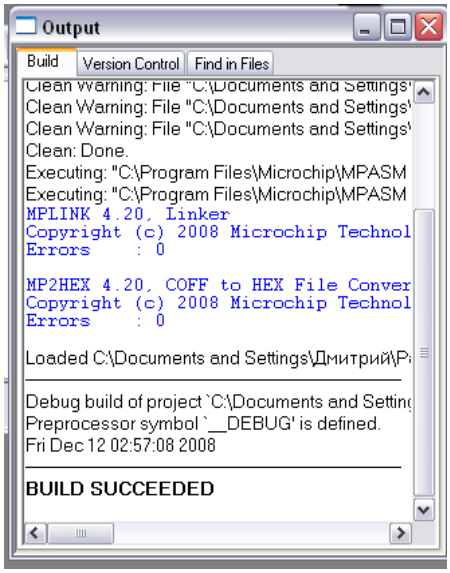

Рис.11.

При наличии ошибок выводится сообщение *BUILD FAILED*, в котором приводится список ошибок с их кодами, номерами строк исходного файла, в которых предполагаются ошибки (Рис.12).

Сообщение представляет протокол трансляции, в котором строки, начинающиеся со слова ***Error***, содержат перечень ошибочных строк исходного текста. После имени файла, к которому относится формальная ошибка, указывается номер строки с ошибкой и краткое описание ее. По номеру строки в тексте файла находят эту строку, определяют состав ошибки и исправляют ее.

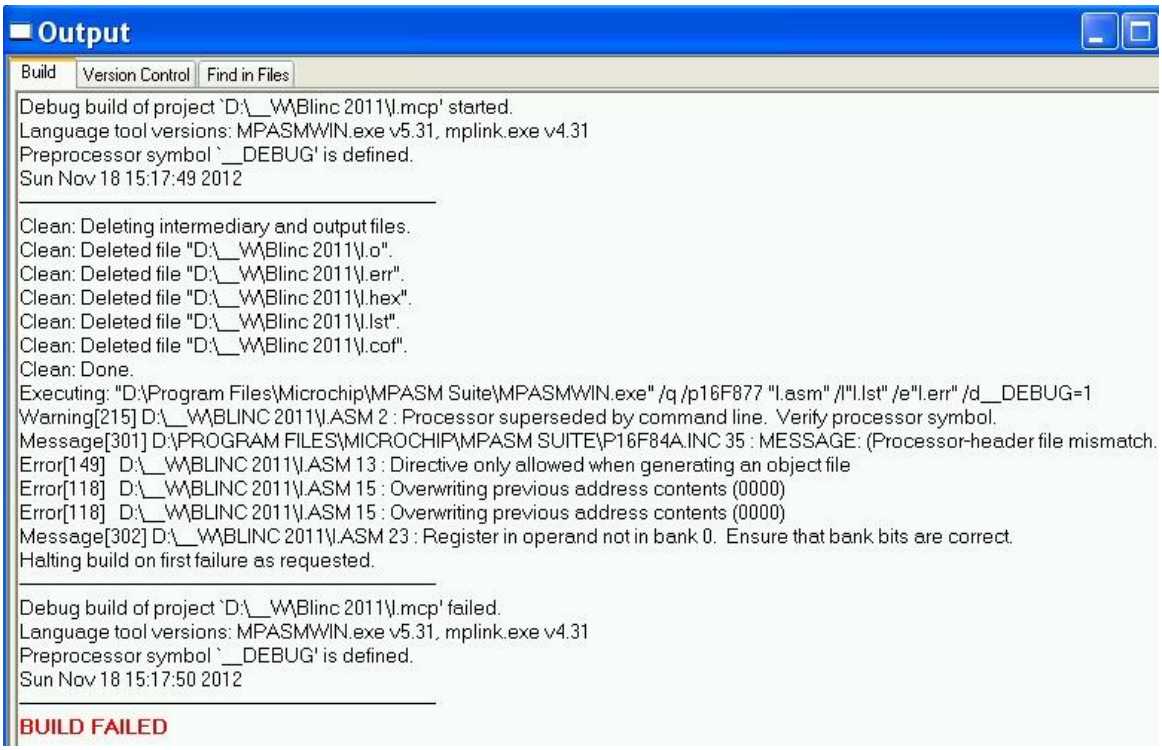

Рис.12

Подсчет номера строки по тексту программы может быть весьма затруднительным. Это можно произвести с помощью файла листинга программы, который создается транслятором. Листинг имеет расширение ***.lst***. Этот файл для каждой строки текста программы содержит текст строки, ее номер, двоичный код, полученный в итоге трансляции, и адрес памяти, куда помещается код. Второй способ поиска строки основан на использовании свойств встроенного редактора исходного текста. Они устанавливаются с помощью контекстного меню окна редактора, который автоматически нумерует все строки текста. Можно так же найти нужную строку в тексте программ двойным щелчком ЛКМ по строке с ошибкой в окне ***Output***.

В данной лабораторной работе в качестве исходного текста программы используется абсолютно правильный текст шаблона программы, не содержащий ошибок. Однако он настроен на создание объектных файлов в перемещаемом формате и содержит несколько директив недопустимых при построении в абсолютном формате непосредственно из исходных текстов. В лабораторной работе применяется именно такой режим трансляции, в котором указанные директивы не требуются и недопустимы. Следовательно, их надо отыскать и удалить. Практически не следует пытаться за один проход удалить все неверные директивы, так как часть из ошибок могут быть вторичными и исчезнут сами, когда предыдущий текст будет правильным. Поэтому предпочтительно ошибки исправлять небольшими частями и только очевидные. Иначе неправильные исправления породят новые ошибки, которые придется отыскивать и восстанавливать.

1.7.4. Тестирование кода программы с помощью MPLAB SIM в пошаговом режиме и автоматическом с остановами.

Выбираем в главном текстовом меню п. *Debugger>Select Tool>MPLAB SIM*, включив тем самым MPLAB симулятор.

Если программа запущена ранее ее необходимо остановить с помощью п. *Debugger>Halt* и установить счетчик команд в исходное состояние с помощью п. *Debugger>Reset* после чего программа анализируется симулятором, и отмечается начало программы (Рис.13).

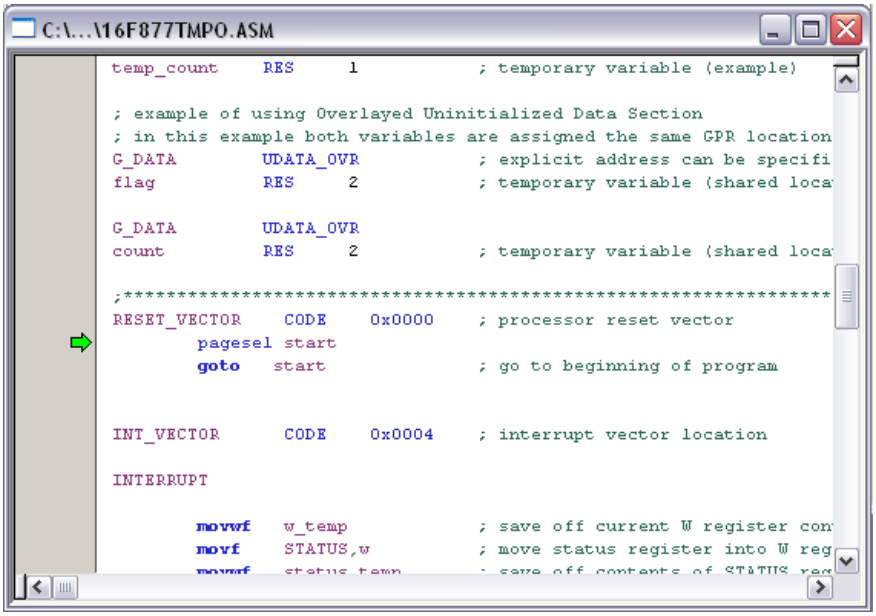

Рис.13.

Отлаживать программу можно в пошаговом режиме и в автоматическом. В пошаговом режиме имитируется выполнение процессором одной команды в ответ на пуск программы с помощью одного из пунктов меню *Debager*: *Step Into, Step Over* или *Step Out*. Выполнив одну команду, MPLAB SIM останавливает процесс и можно проанализировать результаты выполнения команды.

Три пошаговых режима отличаются только способом обработки подпрограмм (процедур). Так как лабораторной работе нет подпрограмм или процедур, все три режима внешне ведут себя одинаково. При наличии подпрограмм следует выбирать один из режимов в зависимости от степени отлаженности подпрограмм и головной программы. В первом случае выполняется вход в подпрограмму в пошаговом режиме аналогично всем другим командам. Во втором – симулятор при достижении команд подпрограммы переходит в

автоматический режим, а после выхода из подпрограммы снова работает в пошаговом режиме. Третий пошаговый режим работает почти как второй, но переходит в пошаговый режим перед командой выхода из подпрограммы.

В автоматическом режиме отладку производят, устанавливая отладочные остановы в желаемых местах. На текст программы они не влияют. Установить останов перед выбранной командой можно директивой *Set Breakpoint* контекстного меню, предварительно выбрав нужную команду щелчком правой кнопки мыши. После пуска программа выполняется до первого останова. После анализа содержимого регистров можно продолжить выполнение программы до следующего останова. Удалить отладочные установы можно также с помощью контекстного меню п. *Remove Breakpoint* или п. *Breakpoints*.

1.7.5. Запуск программы

После выполнения трансляции директивой *Project>Build All* автоматически запускается выбранный отладчик. В данной работе это MPLAB SIM. Слева от первой исполняемой строки программы появляется зеленая стрелка. Если она не появилась, значит, отключился отладчик и его надо снова активизировать п. *Debugger>Select Tool>MPLAB SIM*.

Для пуска в пошаговом режиме выберите меню *Debuger>Step Into*. После анализа результатов выполнения текущей команды для выполнения следующего шага пуск повторяют. Основные пункты управления пуском программы продублированы горячими клавишами на клавиатуре и кнопками на инструментальной панели (Рис.14).

Рис.14

Повторный запуск программы начинают с меню *Debuger>Reset*. Счетчик команд автоматически принимает значение 0x00.

Команда *Debuger>Run* выполняет запуск программы в автоматическом режиме, при котором после выполнения очередной команды сразу начинается выполнение следующей. Так как выполнение команд происходит очень быстро, не удается проанализировать результаты выполнения команд и программных блоков. По этой причине автоматический режим в отладке используется только для оценки общих результатов работы программы, которая уже отлажена по модулям.

Для наблюдения процесса передачи управления в автоматическом режиме в замедленном темпе применяется режим анимации, запускаемый командой *Debuger>Animate*. В режиме

анимации наблюдатель успевает отследить перемещение зеленого маркера и зафиксировать факт выполнения каждой команды.

2. ЛАБОРАТОРНОЕ ЗАДАНИЕ

2.1. Изучить по данному пособию архитектуру PIC-микроконтроллеров.

2.2. Открыть MPLAB и изучить рабочие поля, меню и инструментальную панель, используя пункт меню **Help.**

2.3. Создать папку для рабочего места проекта, имя и путь которой содержат только символы латиницы. Скопировать в него имеющиеся файлы. В данной работе это только файл программы в виде исходного текста на языке ассемблера. так как лабораторная работа выполняется в абсолютном формате (Демонстрационный режим) и файлы компоновки или файлы модулей здесь не требуются.

В качестве примера исходного текста используйте универсальный шаблон-заготовку из состава среды **MPLAB IDE**

C:\Program Files\Microchip\MPASM Suite\Template\Object\PIC16F877tmpo.asm.

2.4. Настроить MPLAB на проектирование микропроцессорного изделия с процессором **PIC 16F877** и виртуальной отладкой с помощью отладчика **MPLAB SIM**.

2.5. Задать отладочную конфигурацию основных устройств процессора. Для этого снять пометку в окне Configuration Bits set in code, выключить сторожевой таймер и возможность записи во внешнюю программную память (поля WDT и WRT_ENABLE).

2.6. Создать пустой проект.

2.7. Присоединить к проекту исходный текст программы.

2.8. Выполнить компиляцию и компоновку программы.

2.9. Ознакомиться с файлами, созданными в процессе трансляции. Изучить их состав, назначение и форматы.

2.10. Пользуясь файлом с расширением **.lst**, найти и исправить все ошибки, добившись успешной трансляции.

2.11. Запустить выполнение программы в пошаговом режиме в виртуальном отладчике MPLAB SIM. Перед выполнением программы ознакомиться с двоичным кодом программы в памяти программ и содержимым регистров в оперативной памяти. В процессе пошагового выполнения программы наблюдать изменение содержимого регистров оперативной памяти.

2.12. Запустить программу в режиме анимации в виртуальном отладчике и наблюдать процесс ее выполнения.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Из каких программ состоит транслятор?
2. В каком формате компилятор выводит результаты своей работы?
3. Зачем нужны модули программ и где они совместно с библиотечными модулями собираются в абсолютную программу, загружаемую в программную память микроконтроллера?
4. Какой шаг создания проекта следует пропустить при создании пустого проекта?
5. Как запустить отладчик в шаговом режиме?

Лабораторная работа № 2

СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ *PIC16F87X*

ЦЕЛЬ РАБОТЫ

Изучение системы команд ассемблера восьмиразрядных однокристальных микроконтроллеров фирмы ***MICROCHIP*** семейства ***PIC16***. Освоение основ разработки и отладки программ для встраиваемых микроконтроллеров.

1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Язык ассемблера для *PIC16*

В этом языке каждая строка исходного текста программы считается одной ассемблерной командой или оператором. Все команды макроассемблера являются либо машинными **командами** (**инструкциями** в переводах документации), либо **директивами** макроассемблера. Инструкции транслируются в двоичные коды машинных команд, заносятся в программную память микроконтроллера и исполняются им в процессе функционирования. Директивы служат для управления самим компилятором макроассемблера и в исполняемые коды не транслируются.

Инструкции описываются в документации на микроконтроллер. Краткое описание инструкций приведено ниже. Там же приведены мнемонические (символьные) обозначения инструкций на языке ассемблера *PIC*-микроконтроллеров. Директивы рассмотрены в руководстве на макроассемблер *MPASM* (файл *mpasm.pdf* или *mpasm.doc*).

1.1.1. Правила подготовки исходного файла

Управляющая программа микроконтроллера составляется на языке макроассемблера MPASM и часто называется исходным файлом. Подготовленный на языке ассемблера исходный текст программы должен быть транслирован, т.е. переведен в двоичный код. Чтобы трансляция прошла до конца, текст программы должен удовлетворять следующим формальным требованиям:

1. Каждая строка программы рассматривается как новая операция. Максимальная ширина строки 255 символов.

2. Каждая строка исходного файла может содержать до четырех информационных полей:

метка – это мнемоническая ссылка на адрес команды, константы или переменной. Метки должны начинаться с **первой позиции строки**. Метки от мнемоник должны отделяться двоеточием, пробелами или символами табуляции;

команда – мнемоническое обозначение кода операции. Мнемоники команд должны начинаться не ближе второй позиции строки. Если метка перед командой отсутствует, свободные позиции в начале строки заполняются символами пробелов или табуляции;

операнды – следуют за мнемоникой команды и отделяются от команды одним или несколькими пробелами. Между собой они должны разделяться запятыми;

комментарии – могут следовать за операндами, мнемониками и метками, если они есть. Комментариям должен предшествовать символ точки с запятой. Любой текст после точки с запятой до конца строки воспринимается транслятором как комментарии.

3. Допускаются строки, не содержащие команд. Такие строки могут оставаться пустыми или содержать метки и/или комментарии.

Структура исходного текста программы в простейшем случае видна из приведенного ниже примера.

Пример записи исходного файла MPASM:

```
        list  p=16c54
Dest    equ   H'0B'
        org   H'01FF'
        goto  Start

Start   movlw   H'A'
        movwf   Dest
        bcf     Dest,3
        goto Start
        end
```

В общем случае программу надо начинать с названия программы и объяснения основных ее функций. Это делается с помощью комментарий.

1.1.2. Средства ввода исходного текста

Исходный файл программы может быть создан в любом текстовом редакторе *ASCII* (например, в редакторе Блокнот). В этом случае присвоенное ему редактором расширение надо вручную изменить на ***.asm***. После этого файл исходного текста должен быть присоединен к проекту.

В среде *MPLAB* имеется специализированный текстовый редактор, который выделяет поля команды разным цветом, что существенно уменьшает число формальных ошибок при подготовке исходного текста программы.

Запуск текстового редактора *MPLAB* можно произвести директивами ***File/New*** или ***File/Add New File to Project…*** . При использовании первой директивы имя файла задается после ввода текста при закрытии файла. Так как назначение файла в процессе ввода текста неизвестно, цветовая разметка полей отсутствует. Вторая директива более удобна для ввода исходного текста, так как она запрашивает имя файла при его создании. Если задать имя с расширением **.asm**, текст будет вводится с цветовой разметкой полей и при сохранении будет автоматически присоединен к проекту.

1.1.3. Инструкции семейства *PIC16*, используемые в работе

Основные особенности системы команд восьмиразрядных однокристальных микроконтроллеров *PIC16F*:

- ***RISC*** архитектура арифметико-логического устройства;
- 35 инструкций;
- непосредственный, прямой и косвенный способы адресации;
- все команды выполняются за один командный цикл (4 такта) кроме инструкций переходов, выполняемых за два цикла;
- тактовая частота 0 – 20 МГц.

Каждая инструкция транслируется в машинный (двоичный) код длиной 14 бит (машинное слово), разделенный на несколько полей: поле кода команды (***OPCODE***- код операции), указывающий на тип исполняемой команды, и одно или несколько полей **операндов**. Для исполнения программа из символьного формата исходного файла преобразуется в двоичный код. В виде двоичных слов она заносится в программную память микропроцессора. Процесс занесения программы в память, как и процесс разработки программы, называется программированием.

Инструкции подразделяются на байт-ориентированные, бит-ориентированные, символьные и инструкции управления.

Байт-ориентированные инструкции

Байт-ориентированные инструкции выполняют действия над восьмиразрядными числами из памяти данных. Регистры памяти данных часто называют файловыми регистрами, потому что они представляют организованную группу данных, объединенных сквозной адресацией, т.е. - файл. Поэтому в мнемонике *MPASM* используется символ ***f*** для обозначения адреса регистра в памяти данных.

Почти все байтовые команды имеют одинаковый синтаксис:

[*label*] КОП f,d

Первое поле *label* – поле метки изображено в прямоугольных скобках. Это означает, что оно не является обязательным.

Поле КОП – обязательное поле, обозначающее мнемонический код выполняемой операции.

Несмотря на использование одноадресной системы команд, большинство операций *PIC* микроконтроллеров имеют два операнда. Для байтовых команд это:

регистр памяти данных – источник данных (при описании команд обозначается символом **f**;

указатель приемника результатов (обозначается символом d):

d=0 – результат помещается в аккумулятор (регистр W),

d=1 – результат помещается в исходный регистр, по умолчанию принимается d=1.).

При выполнении команд действия выполняются над машинными словами длиной в восемь разрядов в формате с фиксированной запятой. Так как реальные числа могут быть значительно большей разрядности, действия над ними приходится выполнять по частям с учетом переносов. Для учета полных результатов операций по частям в процессоре используется специальный регистр состояния STATUS, в котором хранятся признаки результатов предыдущих операций, называемые флагами. Каждый флаг занимает один разряд регистра STATUS и может принимать единичное (Да) или нулевое (Нет) значение. Значения разрядов устанавливаются аппаратно после выполнения каждой операции. В процессорах PIC16 имеются три разряда результатов:

C (Carry) – перенос из старшего или младшего разряда сумматора;

DC – перенос между третьим и четвертым разрядами сумматора для ускорения операций десятичной арифметики;

Z – нулевой результат операции.

Набор байт-ориентированных инструкций можно разделить на несколько групп по выполняемым операциям:

- арифметические операции;
- команды пересылки;
- операции преобразования содержимого регистров.

Арифметические операции

ADDWF - Сложение содержимого регистров W и f. Если d=0 - то результат сохраняется в регистре W, если d=1, то в регистре f.

Синтаксис: [*label*] **ADDWF** f,d

Операнды: 0<=f<=127 d E [0,1]

Изменяет флаги: C, DC, Z

Код в памяти программ: 00 0111 dfff ffff

Пример #1: **ADDWF** FSR, 0

Перед выполнением: W = 0x17, FSR = 0xC2

После выполнения: W = 0xD9 FSR = 0xC2

SUBWF - Вычитание содержимого регистра W из f. Выполняется аналогично операции сложения.

Команды пересылки

MOVF - Пересылка содержимого регистра f в регистр назначения. Если d=0, регистром назначения будет W. Если d=1, регистром назначения будет f. Установкой d=1 удобно проверять файловый регистр f, так как инструкция изменяет состояние бита Z.

MOVWF - Пересылка содержимого регистра W в регистр f.

Синтаксис: [*label*] **MOVWF** f – операнд d в этой команде отсутствует.

Операции преобразования содержимого регистров

NOP - Пустая команда

По этой команде процессор выполняет стандартный машинный цикл длительностью четыре такта, но никаких действий не производит. Используется для создания программной временной задержки или замены лишних команд, введенных ранее.

CLRF, CLRW - Операции **очистки** регистра и аккумулятора соответственно. Оператор d в этих команда лишний и в формате команды отсутствует.

INCF, DECF - Увеличение (**инкремент**) и уменьшение (**декремент**) значения регистра f на 1. Эти команды удобны для создания всевозможных счетчиков.

INCFSZ, DECFSZ - **инкремент** и **декремент** регистра f **с проверкой** результата на равенство нулю. Если итог операции равен нулю, следующая команда пропускается. С помощью этих команд удобно программировать циклы с фиксированным числом повторений. Эти команды можно было бы отнести не только к группе байтовых команд, но и к группе команд передачи управления. Они, как и команды передачи управления, могут менять порядок выполнения команд. Выполняются они также за два машинных цикла.

Бит-ориентированные инструкции

Бит-ориентированные инструкции выполняют установку или проверку состояния одного бита в слове памяти данных с адресом **f**. Номер бита в мнемонике задается символом **b.** Все битовые команды работают одинаково. Сначала считывается весь регистр, далее изменяется заданный бит и результат записывается обратно в регистр (чтение-изменение-запись).

BCF f,b - очистка (запись нуля) бита с номером **b** в регистре **f**.

BSF f,b - установка в единицу бита в регистре **f**.

BTFSC f,b - проверка бита на равенство бита нулю. Если значение бита равно нулю, пропускается одна команда. Иначе выполняется очередная команда.

BTFSS f,b - проверка бита на равенство бита единице. Если значение бита равно единице, пропускается одна команда. Иначе выполняется очередная команда.

Символьные инструкции

Символьные инструкции выполняют действия с константами (литерами), записываемыми непосредственно в коде команды и в дальнейшем хранящимися в памяти программ в одной ячейке с кодом инструкции. Константа в мнемонике ассемблера обозначается символом **k**. Для символьных команд и команд управления k обозначает 8-ми или 9-ти битную константу.

ADDLW - Сложение литеры и W. Содержимое регистра-аккумулятора W складывается с 8-ми битной литерой k и результат помещается в регистр W.

Синтаксис: [*label*] ADDLW k

Изменяет флаги: C, DC, Z

MOVLW - Пересылка литеры в регистр W. Значение поля k команды длиной 8 разрядов из программной памяти пересылается в регистр W.

Синтаксис: [*label*] MOVLW k

Команды передачи управления

В лабораторной работе используется только одна команда передачи управления – безусловный переход. Кроме нее эта группа команд языка MPASM включает еще несколько команд, влияющих на порядок передачи управления: обращение к процедуре, возврат из процедуры, очистка сторожевого таймера, переход в режим сна и т.д. Все они перечислены в сводной таблице инструкций.

GOTO - Безусловный переход. Данная инструкция совершает безусловный переход, при этом 11 бит непосредственного адреса загружаются в PC<10:0>. Остальная старшая часть адреса, содержащая номер страницы, загружается из PCLATH<4:3>. Команда выполняется за 2 командных цикла.

Синтаксис: [*label*] GOTO k

1.1.4 Сводные таблицы инструкций PIC16

Описания всех инструкций в краткой форме приведены в таблицах 1,2 и 3. Таблицы взяты с сайта http://www.microchip.ru/?mid=38&tab=18.

В табл.1 сведены все обозначения, используемые в описаниях команд. В табл.2 приведены три формата, которые может принимать команда. Во всех примерах для обозначения шестнадцатеричного числа используется формат: 0xhh, где h - шестнадцатеричная цифра. В табл. 3 приведено краткое описание всех машинных команд (инструкций) микроконтроллеров серии PIC16.

Табл.1

Поле	Описание
f	Адрес файлового регистра (от 0x00 до 0x7F)
W	Рабочий регистр (аккумулятор)
b	номер бита внутри 8-ми битного регистра
k	Символьное поле, константа или метка
x	Любое значение разряда (=0 или 1) Компилятор сгенерирует код с x=0 -это нужно для совместимости со всеми программными продуктами Microchip.
d	У места, где сохранять результат: d=0 (сохранять в W), d=1 (сохранять в f). По умолчанию d=1
label	Имя метки
TOS	Вершина стека
PC	Счетчик команд (программный счетчик)
PCLATH	Записываемый буфер для старших 5-ти бит PC (выбор страницы памяти программ)
GIE	Флаг разрешения глобальных прерываний
WDT	Сторожевой таймер
TO	бит Таймаута (Time-out bit)
PD	бит понижения питания (Power-Down bit)
dest	Назначение, либо регистр W либо другой регистр указанный в

описании команды

[ ]	Опционально, т.е. необязательное использование записи, которая заключена в квадратные скобки
( )	Содержимое
->	Занести в
< >	Битовое поле в регистре
E	принадлежит
italic	Поле заполненное курсивом определяется пользователем

Табл. 2

Байт-ориентированные операции с файловым регистром

13	8	7	6	0
OPCODE		d	f(FILE #)	

d = 0 - запись результата в W
d = 1 - запись результата в f
f = 7-ми битный адрес файлового регистра

Бит-ориентированные операции с файловым регистром

13	10	9	7	6	0
OPCODE		b (BIT #)		f(FILE #)	

b = 3-х битный адрес
f = 7-ми битный адрес файлового регистра

Символьные команды и команды ветвления (общий вид)

13	8	7	0
OPCODE		k (символ)	

k = 8-ми битное непосредственное значение

Команды ветвления - GOTO и CALL

13	11	10	0
OPCODE		k (символ)	

k = 11-ти битное непосредственное значение

Табл.3

Мнемоника в ассемблере	Краткое описание	Циклов	14-ти битный код	Изменяет флаги	Прим.
БАЙТ-ОРИЕНТИРОВАННЫЕ КОМАНДЫ					
ADDWF f,d	Сложение W и f	1	00 0111 DFFF FFFF	C, DC, Z	1, 2
ANDWF f,d	Поразрядная операция "И" с W и f	1	00 0101 DFFF FFFF	Z	1, 2
CLRF f	Очистка регистра f	1	00 0001 1FFF FFFF	Z	2
CLRW	Очистка регистра W	1	00 0001 0000 0011	Z	
COMF f,d	Инвертирование битов регистра f	1	00 1001 DFFF FFFF	Z	1, 2
DECF f,d	Уменьшение значения регистра f	1	00 0011 DFFF FFFF	Z	1, 2
DECFSZ f,d	Уменьшение значения регистра f, пропуск следующей инструкции если результат равен нулю.	1(2)	00 1011 DFFF FFFF	нет	1, 2, 3
INCF f,d	Увеличение значения регистра f на 1	1	00 1010 DFFF FFFF	Z	1, 2
INCFSZ f,d	Увеличение значения регистра f, пропуск следующей инструкции если результат равен нулю.	1(2)	00 1111 DFFF FFFF	нет	1, 2, 3
IORWF f,d	Логическая операция включающего ИЛИ W и f	1	00 0100 DFFF FFFF	Z	1, 2
MOVF f,d	Пересылка содержимого регистра f	1	00 1000 DFFF FFFF	Z	1, 2
MOVWF f	Пересылка содержимого регистра W в регистр f	1	00 0000 1FFF FFFF	нет	
NOP	Пустая команда	1	00 0000 0XX0 0000	нет	
RLF f,d	Циклический сдвиг влево через флаг переноса	1	00 1101 DFFF FFFF	C	1, 2
RRF f,d	Циклический сдвига вправо через флаг переноса	1	00 1100 DFFF FFFF	C	1, 2
SUBWF f,d	Вычитание W из f	1	00 0010 DFFF FFFF	C, DC, Z	
SWAPF f,d	Обмен местами полу-байт регистра f	1	00 1110 DFFF FFFF	нет	
TRIS f	Загрузка регистра TRIS	1	00 0000 0110 0FFF	нет	
XORWF f,d	Логическая операция исключающего ИЛИ с W и f	1	00 0110 DFFF FFFF	Z	
БИТ-ОРИЕНТИРОВАННЫЕ КОМАНДЫ					
BCF f,d	Очистка бита в f	1	01 00BB BFFF FFFF	нет	1, 2
BSF f,b	Установка бита в f	1	01 01BB BFFF FFFF	нет	1, 2
BTFSC f,b	Проверка на равенство бита нулю, пропускаем след.	1(2)	01 10BB BFFF FFFF	нет	3

	команду если да.				
BTFSS f,b	Проверка на равенство бита 1, пропускаем след. команду если да.	1(2)	01 11BB BFFF FFFF	нет	3
СИМВОЛЬНЫЕ КОМАНДЫ И КОМАНДЫ ВЕТВЛЕНИЯ					
ADDLW k	Сложение литеры и W	1	11 111X KKKK KKKK	C, DC, Z	
ANDLW k	Логическая операция "И" с символом и W	1	11 1001 KKKK KKKK	Z	
CALL k	Вызов процедуры	2	10 0KKK KKKK KKKK	нет	
CLRWDT	Сброс Сторожевого Таймера (WDT)	1	00 0000 0110 0100	TO, PD	
GOTO k	Безусловный переход	2	10 1KKK KKKK KKKK	нет	
IORLW k	Логическая операция включающего ИЛИ с символом и W	1	11 1000 KKKK KKKK	Z	
MOVLW k	Пересылка литеры в регистр W	1	11 00XX KKKK KKKK	нет	
OPTION	Загрузка данных в регистр OPTION	1	00 0000 0110 0010	нет	
RETFIE	Возврат управления после прерывания	2	00 0000 0000 1001	нет	
RETLW k	Возврат с литерой в W	2	11 01XX KKKK KKKK	нет	
RETURN	Возврат из процедуры	2	00 0000 0000 1000	нет	
SLEEP	Переход в режим "сна"	1	00 0000 0110 0011	TO, PD	
SUBLW k	Вычитание W из литеры	1	11 110X KKKK KKKK	C, DC, Z	
XORLW k	Логическая операция исключающего ИЛИ с символом и W	1	11 1010 KKKK KKKK	Z	

2.1.5. Директивы MPASM

Директивы управляют процессом трансляции и не переводятся в исполняемые микропроцессором инструкции.

Существует пять основных типов директив:

директивы контроля - управляют созданием разделов условно компилированного кода:

директивы данных - управляют распределением памяти и назначением символических имен переменным и константам;

директивы листинга - определяют формат и состав файла листинга (файл с расширением .LST в каталоге, где построен проект). Эти директивы позволяют: указывать заголовки, нумеровать страницы и настраивать другие параметры;

макро директивы - управляют работой макросов и распределением данных в теле макроса:

директивы объектного файла - используются только при создании объектного файла.

В полном объеме директивы описаны в руководстве к макроассемблеру MPASM. Здесь приводятся описания только директив, используемых в лабораторной работе и некоторых основных директив.

INCLUDE - Подключение дополнительного исходного файла

Синтаксис: include <<include_file>>

или include “<include_file>”

Указанный файл читается как исходный текст программы. Эффект аналогичен копирования полного текста программы указанного файла в место расположения директивы. После окончания компиляции подключенного модуля, компиляция продолжается в исходной программе. Допускается до шести уровней вложения файлов.

<include_file> может быть указан в кавычках или угловых скобках. Если указан полный путь к файлу, то поиск файла будет происходить только в указанной директории. Если путь к подключаемому файлу не указан, то поиск файла будет выполняться в текущей рабочей директории, директории исходного файла и директории выполняемого MPASM.

Пример:

```
include   "c:\sys\sysdefs.inc"    ; системные параметры
include   <regs.h>                ; список регистров
```

LIST - Список параметров трансляции

Синтаксис: **list** [<list option>, …,<list option>]

Директива LIST должна быть размещена на отдельной строке. Она изменяет параметры компиляции исходного файла и генерации файла листинга. Один или несколько следующих параметров может быть указан в директиве LIST:

Параметр	Значение по умолчанию	Описание
b=nnn	8	Число пробелов при табуляции
C=nnn	132	Число символов в строке
f=<format>	INHX8M	Формат выходного HEX файла <format>: |NHX32, INHX8M, |NHX8S
free	FIXED	Использование свободного формата печати
fixed	FIXED	Использование фиксированного формата печати
mm={ON|OFF}	On	Включать карту памяти в файл листинга программы
n=nnn	60	Число строк на одной странице
p=<type>	HET	Тип микроконтроллера. Например. PIC16C54
r=<radix>	hex	Система счисления по умолчанию: hex, dec, oct
st={ON|OFF}	On	Включать таблицу символов в файл листинга программы
t={ON|OFF}	Off	Усечение длинных строк
w={0|1|2}	0	Установка уровня сообщений. См. ERRORLEVEL
x={ON|OFF}	On	Включать полный текст макроса

Примечание: Все параметры директивы LIST указываются в десятичных числах.

Пример

```
list   p=17c42,   f=INHX32,   r=DEC
```

__*CONFIG* - Установка битов конфигурации микроконтроллера

Синтаксис: _ _config <значение> или _ _config <адрес>,<значение>

или

_ _config <значение текст>& … &<значение текст>

Устанавливает биты конфигурации микроконтроллера в соответствии с числовым полем <значение> или текстовыми полями обозначений параметров. Набор

конфигурационных битов индивидуален для каждого типа и подробно описан в технической документации на соответствующий микроконтроллер.

Предварительно, перед директивой *CONFIG*, надо указать тип микроконтроллера с помощью директивы *LIST* или *PROCESSOR*.

Пример

```
                list   p=16F84A
_               _ _config H'FFFF'       ; Конфигурация по умолчанию
```

или

```
_               _ _config_XT_OSC&_PWRT_ON&_WDT_OFF
```

EQU - Определение константы ассемблера

Синтаксис

<label> equ <expr>

Описание

Присваивает значение <expr> метке <label>.

Пример

```
four equ 4      : присваивает значение 4 константе с именем four
```

ORG – Установить абсолютный адрес начала модуля программы в памяти программ

Синтаксис: <label> org <expr>

Установить адрес начала части программы равным числовому значению <expr>. Если указана метка <label> то она будет иметь адрес равный <expr>. Если в тексте граммы не встречается директива ORG, то считается, что программа начинается с нулевого адреса. Данная директива требует создания модуля в абсолютном формате и не может быть использована при генерации объектного файла. В случае трансляции в объектном формате используется подобная директива ***Code***.

Пример

```
int_l   org   0x2 0             ; Вектор с адресом 20
int_2   org   int_l+0xl0   ; Вектор с адресом 30
```

END – признак окончания текста программы. Не выполняет функцию останова микроконтроллера.

В табл.4 представлен более полный список директив поддерживаемых MPASM.

Табл.4

Директива	Описание	Синтаксис
_ _BADRAM	Идентификация нереализованного ОЗУ	_ _badram <expr>[-<expr>][, <expr>[-<expr>]]
BANKISEL	Выбор банка для косвенной адресации	bankisel <label>
BANKSEL	Выбор банка для прямой адресации	banksel <label>
CBLOCK	Определение блока констант	cblock [<expr>]
CODE	Начало кода объектного файла в памяти программ	[<name>] code [<address>]
_ _CONFIG	Установка битов конфигурации микроконтроллера	_ _config <expr> OR _ _config <addr>, <expr>
CONSTANT	Определить символьную константу	constant <label>[=<expr>,...,<label>[=<expr>]]
DA	Сохранение строки в памяти программ	[<label>] da <expr> [, <expr2>, ..., <exprn>]
DATA	Сохранение значений или текста в памяти программ	[<label>] data <expr>,[,<expr>,...,<expr>] [<label>] data "<text_string>"[,"<text_string>",...]
DB	Побайтное сохранение данных в памяти программ	[<label>] db <expr>[,<expr>,...,<expr>] [<label>] db "<text_string>"[,"<text_string>",...]
DE	Резервирует 8-разрядное значение в EEPROM памяти	[<label>] de <expr>[,<expr>,...,<expr>] [<label>] de "<text_string>"[,"<text_string>",...]
#DEFINE	Определяет замену текста	#define <name> [<string>] #define <name> [<arg>,...,<arg>] <string>
DT	Определяет таблицу данных	[<label>] dt <expr>[,<expr>,...,<expr>] [<label>] dt "<text_string>"[,"<text_string>",...]
DW	Резервирует слова памяти программ	[<label>] dw <expr>[,<expr>,...,<expr>] [<label>] dw "<text_string>"[,"<text_string>",...]
ELSE	Начало альтернативного блока программы условия IF	else
END	Окончание программы	end
ENDC	Окончание автоматического блока констант	endc
ENDIF	Окончание условного блока программы	endif
ENDM	Окончание макроса	endm
ENDW	Завершает цикл While	endw

EQU	Определение константы ассемблера	<label> equ <expr>
ERROR	Формирует сообщение об ошибке	error "<text_string>"
ERRORLEVEL	Настройка параметров вывода сообщений об ошибках	errorlevel 0|1|2| <+|-><message number>
EXITM	Выход из макроса	exitm
EXPAND	Включение текста макроса в файл листинга программы	expand
EXTERN	Определение внешних меток	extern <label>[,<label>]
FILL	Запись значения в память программ	[<label>] fill <expr>, <count>
GLOBAL	Внешняя метка	global <label>[,<label>]
IDATA	Объявляет начало инициализации данных в объектном файле	[<name>] idata [<address>]
_ _IDLOCS	Установка значения ID	_ _idlocs <expr>
IF	Начало блока условия	if <expr>
IFDEF	Выполнение, если определена символьная метка	ifdef <label>
IFNDEF	Выполнение, если символьная метка не определена	ifndef <label>
INCLUDE	Подключение дополнительного исходного файла	include <<include_file>> |"<include_file>"
LIST	Список параметров	list [<list_option>,...,<list_option>]
LOCAL	Объявить локальную переменную макроса	local <label>[,<label>]
MACRO	Определить макрос	<label> macro [<arg>,...,<arg>]
_ _MAXRAM	Определяет максимальный объем ОЗУ	_ _maxram <expr>
MESSG	Сформировать сообщение	messg "<message_text>"
NOEXPAND	Не разворачивать текст макроса	noexpand
NOLIST	Выключить вывод в файл листинга	nolist
ORG	Установить адрес программы	<label> org <expr>
PAGE	Вставить страницу в файл листинга программы	page
PAGESEL	Произвести выбор страницы	pagesel <label>
PROCESSOR	Выбор типа микроконтроллера	processor <processsor_type>

1.2. Организация памяти программ

Семейство **PIC16F7X** имеет 13-разрядный счетчик команд, с помощью которого адресуется пространство 8К×14 памяти программ. После начальной установки (сброса микроконтроллера) программный счетчик устанавливается на 0000h, а любое прерывание вызывает переход на адрес 0004h.

Организацию памяти программ поясняет рис.1.

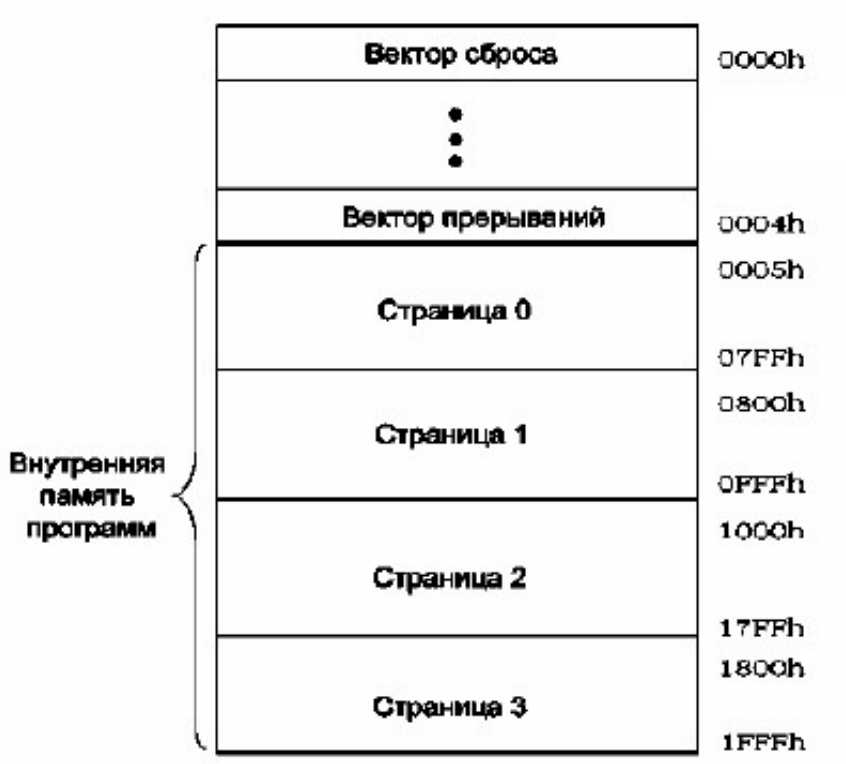

Рис. 1. Организация памяти программ

После включения питания процессора вырабатывается импульс сброса, который приводит все устройства микроконтроллера в исходное состояние. По окончании импульса сброса микроконтроллер запускается с команды, расположенной по адресу 0000h. Эта ячейка называется вектором сброса. В ней должна быть помещена команда перехода на начало программы, которая может начинаться не ранее адреса 0005h.

В последних ячейках памяти располагаются калибровочные константы и другие служебные данные. В специальных ячейках за пределами поля адресации записывается слово конфигурации.

1.3. Организация памяти данных

Память данных состоит из восьмиразрядных регистров общего и специального (SFR) назначения, объединенных общим полем адресации. **Регистры специального назначения** выполняют функцию управления аппаратными средствами и влияют на свойства микроконтроллера. Каждый из них может иметь специфичную индивидуальную структуру, определяемую функциями регистра. Часть разрядов в них может отсутствовать или взаимодействовать друг с другом. Поэтому применение специальных регистров должно проводиться строго по назначению. Каждый регистр специального назначения имеет индивидуальное мнемоническое имя, зависящее от его назначения.

Регистры общего назначения имен не имеют и занимают все оставшиеся адреса имеющейся памяти данных. Все они одинаковы, взаимозаменяемы и являются просто оперативными ячейками памяти размером в один байт. Любая команда программы выполняется с ними корректно.

Память данных разделена на четыре банка. Биты RP1 и RP0 регистра STATUS (соответственно биты 6 и 5) предназначены для управления банками данных. В таблице показано состояние управляющих битов при обращении к банкам памяти данных.

RP1:RP0	Банк
00	0
01	1
10	2
11	3

Объем одного банка памяти составляет 128 байт (7Fh). В начале каждого из банков размещены регистры специального назначения. Регистры общего назначения текущего банка используются пользователем как статическое ОЗУ данных. Некоторые, наиболее часто используемые регистры специальных функций, отображаются во всех четырех банках памяти данных. Карта памяти данных применительно к ***PIC16F788*** приведена на Рис. 2.

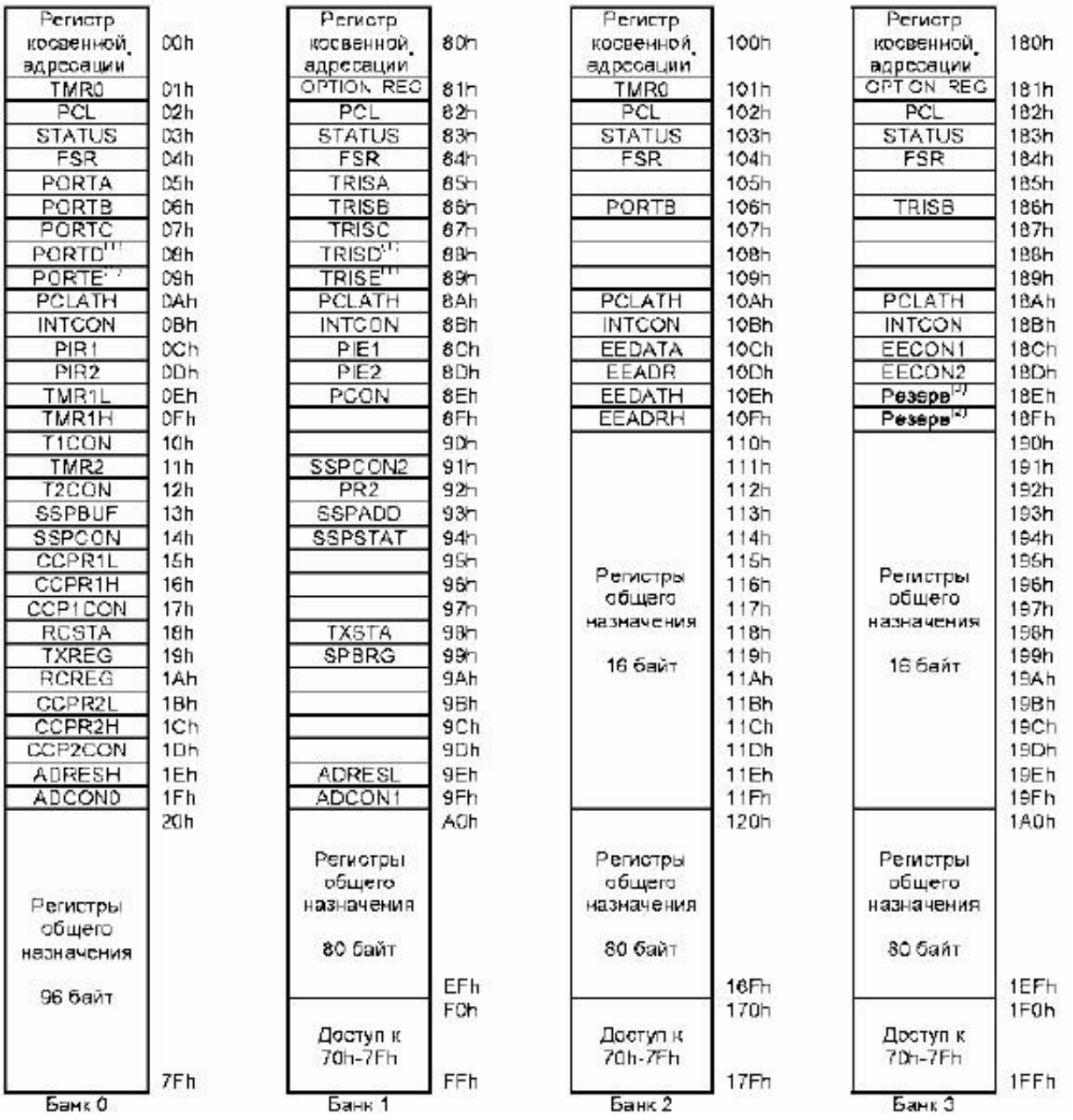

Рис. 2. Карта памяти данных

1.4. Стек:

Микроконтроллеры семейства *PIC16F87X* имеют 8-уровневый 13-разрядный аппаратный стек. Стек не имеет отображения на память программ и память данных. Значение счетчика команд заносится в вершину стека при выполнении инструкции перехода или обработки прерывания. Обратное чтение из стека в счетчик команд происходит при возвращении из подпрограммы или процедуры обработки прерываний. Стек работает как циклический буфер. После 8 записей в стек, девятая запись заменяет первую.

1.5. Отладка программы

В среде *MPLAB* отладку программы можно выполнять несколькими способами:

- в моделирующей программе *MPLAB SIM*, которая имитирует работу выбранного микропроцессора (виртуально). Этот режим используется на первых этапах отладки управляющей программы;

- с помощью аппаратных эмуляторов, построенных на высокопроизводительных микропроцессорах, имитирующих работу целевого процессора и вырабатывающих на специальных разъемах соответствующие электрические сигналы, к которым для отладки можно подключать электрические цепи проектируемого устройства;

- на макетной или реальной плате микропроцессорного устройства с помощью специальной головки внутрисхемной отладки. Этот режим используется на последних стадиях отладки управляющей программы или всего изделия в целом. Внутрисхемная отладка требует наличия в микропроцессоре специальных встроенных устройств и специального разъема на печатной плате. Не все микропроцессоры поддерживают этот режим.

В лабораторной работе выполняется только отладка в *MPLAB SIM*. Процесс отладки программ в остальных режимах подобен отладке в *MPLAB SIM* и состоит, прежде всего, в проверке правильности выполнения каждой команды программы в пошаговом режиме, режиме анимации и в автоматическом режиме. Автоматический режим эффективен только после завершения отладки в первых двух и выполняет в основном контрольные функции.

Отладка в пошаговом режиме состоит в последовательном выполнении команд с контролем состояния микропроцессора после выполнения каждой команды. После пуска с помощью п. *Debuger>Step Into, MPLAB SIM* выполняет очередную команду и приостанавливает процесс до следующего пуска. Это дает возможность проверить состояние микропроцессора перед выполнением следующей команды и правильность выполненной команды, сравнив фактические результаты с ожидаемыми.

Среда *MPLAB IDE* содержит несколько специальных средств контроля состояния памяти данных, программной памяти и специальных регистров. Они собраны в п. *View* главного меню:

View>Spetial Function Registers – выводит на экран содержимое всех специальных регистров, включая регистр-аккумулятор (**WREG**), счетчик команд (**PCL**, **PCLATH**) и регистр состояния (**STATUS**). В этом и других меню регистры, состояние которых изменилось во время выполнения команды, выделяются красным цветом;

View>Program Memory – выводит в шестнадцатеричном коде содержимое всех ячеек памяти программ и дизассемблированный (восстановленный по двоичному коду) текст программы на языке ассемблера;

View>File Registers - выводит в шестнадцатеричном коде содержимое всех ячеек памяти данных, включая специальные регистры;

View>Watch – позволяет создавать закладки и отслеживать содержимое выбранных программистом регистров специального и общего назначения по их символьным именам.

2. ЛАБОРАТОРНОЕ ЗАДАНИЕ

2.1. Изучить правила подготовки текстов программ по описанию ассемблера MPASM или по данным методическим указаниям.

2.2. Создать новый пустой проект без файла исходного текста программы.

2.3.Создать файл исходного текста программы по приведенному ниже примеру с ошибками и присоединить его к проекту, воспользовавшись встроенным редактором MPLAB с помощью меню File/Add New File to Project…

Текс примера с ошибками:

```
;
;    ПРОГРАММА ДВОИЧНОГО СЧЕТЧИКА ДО 10
;

listp=16f877
include<p16f877.inc>
c1 equ 0x0c          ;   определение переменной c1 по адресу – 0x0c

org 0x00             ; Определить пусковой адрес программы
                     ; в векторе сброса 0x00
goto start           ;переход на начало программы

org 0x04             ;начало размещения программы в памяти
start
movlw 0xF6           ;установить начальное значение счетчика
movwf c1             ;записать его в регистр счетчика
loop
incfsz c1, F         ;инкремент счетчика и пропуск следующей команды,
                     ;  если результат равен нулю
goto loop            ; продолжить
goto bug             ; повторить счет сначала ( с метки start)
end
```

2.4. Попытаться построить проект компиляцией и компоновкой. Ознакомиться с файлами, созданными компилятором в вашем рабочем каталоге.

2.5. Исправить формальные ошибки исходного текста, пользуясь номерами строк с ошибками, найденными компилятором, файлом листинга (расширение **.lst**) из каталога проекта и правилами подготовки исходного файла.

2.6. Разобрать все команды программы и составить алгоритм программы – примера, пользуясь комментариями и описаниями команд.

2.7. Построить проект компиляцией и компоновкой, исправив обнаруженные ошибки.

2.8. Ознакомиться с файлами, созданными компилятором в вашем рабочем каталоге после успешной трансляции

2.9. В пошаговом режиме отладчика проверить правильность выполнения команд, контролируя порядок выполнения команд и результат с помощью пункта View пакета MPLAB. Исправить вновь обнаруженные ошибки.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Опишите порядок выполнения операций по разработке и применению программного обеспечения для микроконтроллера.
2. Какие операции выполняют текстовый редактор, компилятор и редактор связей?
3. Какие форматы файлов создаются средой *MPLab* в процессе создания проекта и программы?
4. Чем отличается объектный файл от абсолютного?
5. Зачем нужен файл с расширением *.lst*?
6. Почему надо удалять команды перемещаемого формата при трансляции в абсолютном формате?
7. Для чего нужен пошаговый режим работы отладчика?
8. Как выполнять отладку в пошаговом режиме?

Лабораторная работа № 3.

ОТЛАДКА МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ В СРЕДЕ СХЕМОТЕХНИЧЕСКОГО МОДЕЛИРОВАНИЯ MULTISIM

ЦЕЛЬ РАБОТЫ

Изучение технологии и средств отладки микроконтроллерных систем путем комплексного моделирования программной и аппаратной части микропроцессора совместно с внешней электрической схемой системы, включая аналоговые и цифровые элементы.

1. КРАТКИЕ СВЕДЕНИЯ ПО МОДЕЛИРОВАНИЮ

Моделирование PIC-микроконтроллеров совместно с окружающими элементами электрической принципиальной схемы в среде NI Multisim 10 имеет следующие особенности:

- работа тактового генератора как элемента электрической схемы не моделируется в полном объеме, а имитируется наличие тактовых импульсов от условного заведомо работающего генератора, рабочую частоту, которого можно менять. В результате схему тактового генератора можно даже не изображать на рабочем поле;

- управляющая программа вводится на языках СИ или Ассемблере в виде исходного текста с помощью встроенного редактора или текстового файла с соответствующим расширением. В двоичном коде, т.е. в виде HEX-файла абсолютного формата могут вводиться только данные для внешней флэш-памяти. В данной работе такие данные отсутствуют, поэтому вводить в двоичном формате ничего не надо.

1.1. Создание проекта микропроцессорного устройства

Построение модели с микропроцессорным устройством в среде Multisim начинается с создания проекта, без которого моделирование микропроцессорного устройства (МПУ) не может быть выполнено.

Создание проекта начинается с создания папки для проекта. Средствами операционной системы необходимо создать свою папку, путь к которой должен содержать только символы латиницы. Эту папку проекта можно создать заранее или, в крайнем случае, непосредственно перед заданием пути, которое легче выполнить с помощью клавиши **Browse…** в . В лабораторной работе для рабочих пространств выделен каталог **C:WORK**, в котором и следует работать. Свой каталог, в том числе и при работающем MPLAB, создается через

клавишу **Пуск** операционной системы Windows. Имя рабочего пространства также должно состоять только из символов латиницы и не допускается кириллица. В эту папку надо скопировать файл исходного текста программы Blinc.asm

Непосредственно создание проекта начинается с помещения на рабочее поле Multisim уловного графического изображения микропроцессора с помощью п. **Place>Component** главного меню Multisim на всплывающей панели, показанной на рис.1.

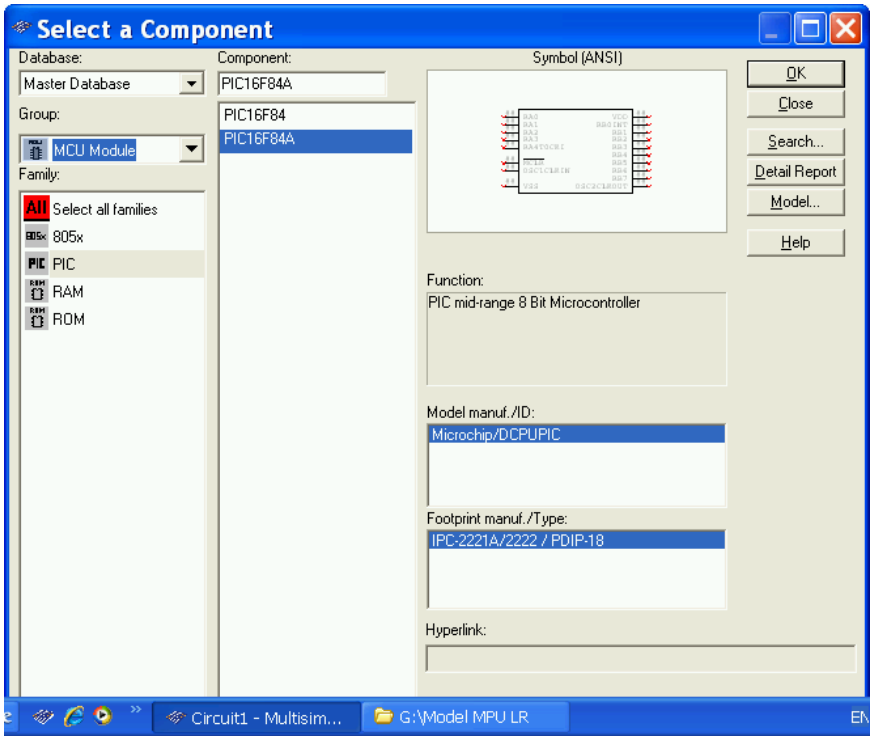

Рис. 1

После помещения обозначения на рабочее поле автоматически запускается помощник по созданию проекта и появляется панель, показанная на рис.2, в которой надо ввести полный путь к созданной ранее своей папке для рабочих мест проектов и имя рабочего места для текущего проекта.

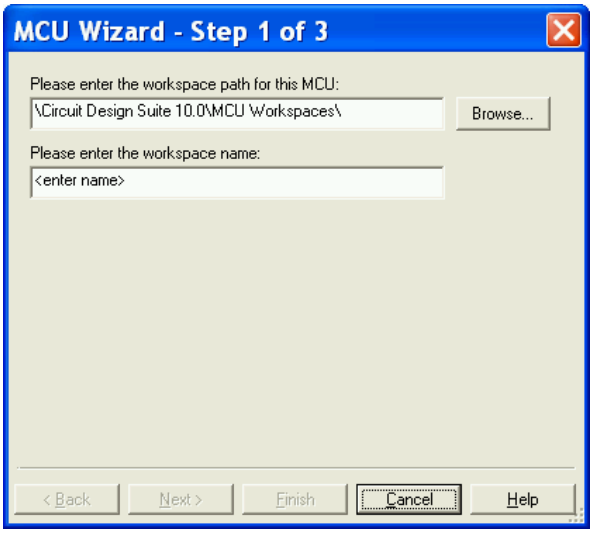

Рис. 2

Меню второго шага, показанное на рис.3, позволяет выбрать тип проекта, язык программирования, транслятор и имя проекта.

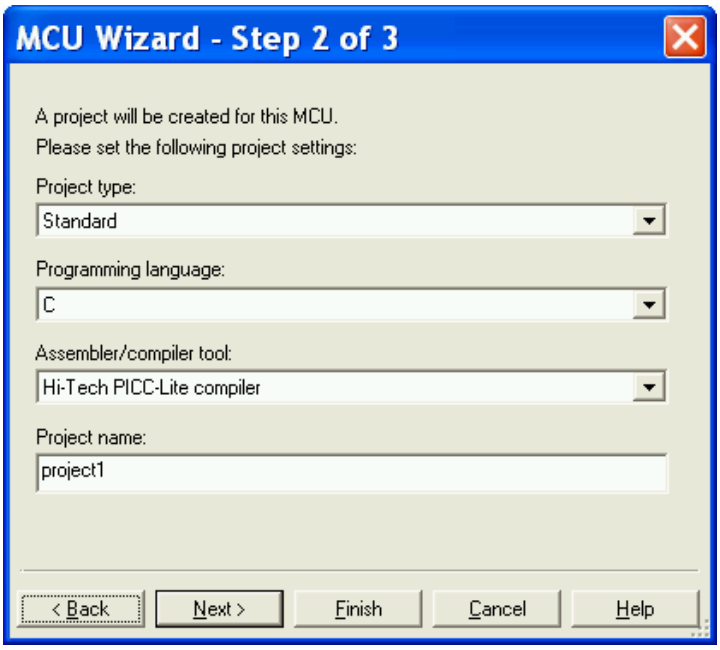

Рис. 3

Типов проекта в данной версии пакета Multisim может быть только два: стандартный с занесением программы в память программ для последующего исполнения процессором и загрузка данных в энергонезависимую флэш-память процессора. В лабораторной работе используется стандартный проект.

Файл используемой готовой программы записан на языке ассемблера. Этот язык и надо установить.

Помощник по заданному языку попытается самостоятельно найти программы компилятора и компоновщика. В данной версии MPLAB файлы компилятора (**MPASMWIN.exe**), компоновщика (**mplink.exe**) и библиотек (**mplib.exe**) находятся в папке **C:\Program Files\Microchip\MPASM Suite**. В процессе создания проекта необходимо проверить правильно ли выбраны эти файлы и есть ли они на месте. Если они отсутствуют или искажены, необходимо их восстановить.

Последним задается имя проекта. Оно должно содержать только символы латинского алфавита. Под этим именем среда создает папку с файлами проекта.

После ввода всех данных, как обычно нажимается клавиша **Next >** и происходит переход к третьему шагу (Рис.4).

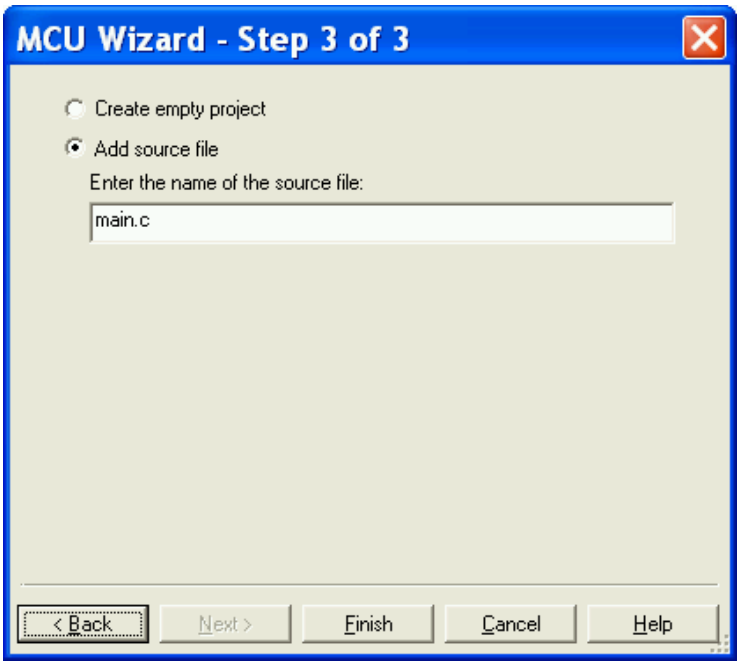

Рис.4

На третьем шаге создается пустой проект или присоединяется файл исходного текста, если он есть. В первом случае надо установить метку в кружке - **Create empty project**. Во втором случае надо установить метку в кружке – **Add source file** и ввести полный путь к нужному файлу исходного текста. По умолчанию помощник присоединяет в качестве исходного текста файл–шаблон с именем **main**. Этот файл – заготовка, в которую можно ввести текст своей программы с помощью встроенного редактора. Текст исходного файла после присоединения появляется автоматически в рабочем поле в окне программы. Вызвать его на экран можно щелчком ЛКМ по вкладке с именем программы внизу рабочего поля.

При отсутствии исходного файла можно присоединить шаблон исходного файла и отредактировать его с помощью встроенного редактора.

Завершается создание проекта клавишей **Finish**. После этого проект сохраняется в заданном рабочем пространстве.

1.2. Присоединение файла к проекту

Присоединение файла к проекту производится с помощью п. **MCU>MCU PIC16F84A U1>MCU Code Manager…** главного меню пакета. Окно менеджера кодов показано на рис.5.

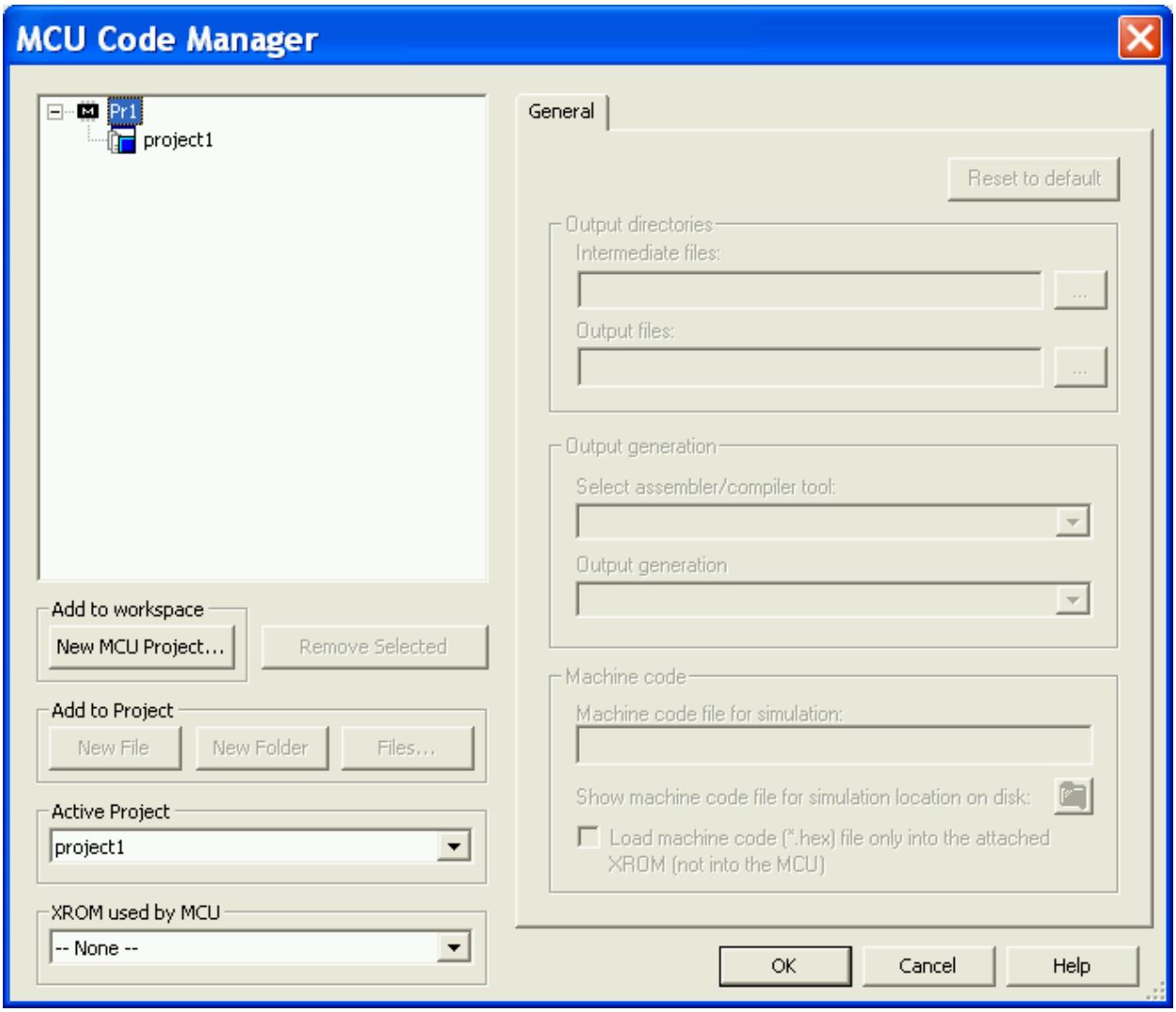

Рис. 5

Для присоединения файла исходного текста к пустому проекту надо в окне менеджера кодов левой кнопкой мыши (ЛКМ) выделить активный проект и нажать ЛКМ на клавишу **Files…**. После этого с помощью открывающегося нового меню найти присоединяемый файл и выделив его открыть. В результате выбранный файл копируется в каталог проекта и присоединяется к проекту. На рабочем поле появляется дополнительная закладка встроенного редактора с текстом программы, если присоединенный файл является файлом исходного текста. Эта закладка позволяет редактировать текст программы.

Программа Blinc, используемая в лабораторной работе, приведена ниже.

```
#include <P16F84A.INC>
list p=16f84a
```

```
Reg     equ   0x20

    __CONFIG _WDT_OFF & _XT_OSC;Конфигурация процессора

            ORG 0x00
            goto Start          ;  Секция
            ORG 0x05            ;        запуска


Start       CLRW
            MOVWF Reg
Blinc       COMF Reg, F

Vyvod       bsf STATUS,RP0          ; Установка
            clrf TRISB          ;        всех разрядов
            bcf  STATUS,RP0         ;                порта B в режим вывода
            movf Reg,W          ; Вывод во все разряды порта B
            movwf PORTB             ;        нового кода

            goto Blinc
            end
```

В процессе работы эта программа периодически выводит единицы и нули на выходные контакты устройства цифрового ввода-вывода PortB.

1.3. Трансляция программы

Трансляция программы производится с помощью п. **MCU>MCU PIC16F84A U1>Build** главного меню. Она может выполняться до окончательного ввода схемы модели или после. При этом в нижнем окне **Spreadsheet** на закладке программы появляется сообщение о результатах трансляции. Если количество ошибок трансляции равно нулю, трансляция считается успешной и программу можно запускать, но до пуска программы необходимо достроить электрическую схему устройства. Если окно **Spreadsheet** после построения

программы не появилось, его следует вызвать, установив пометку напротив п. **View>Spreadsheet View** главного меню.

1.4. Ввод схемы

Схема лабораторной модели представлена на рис. 6.

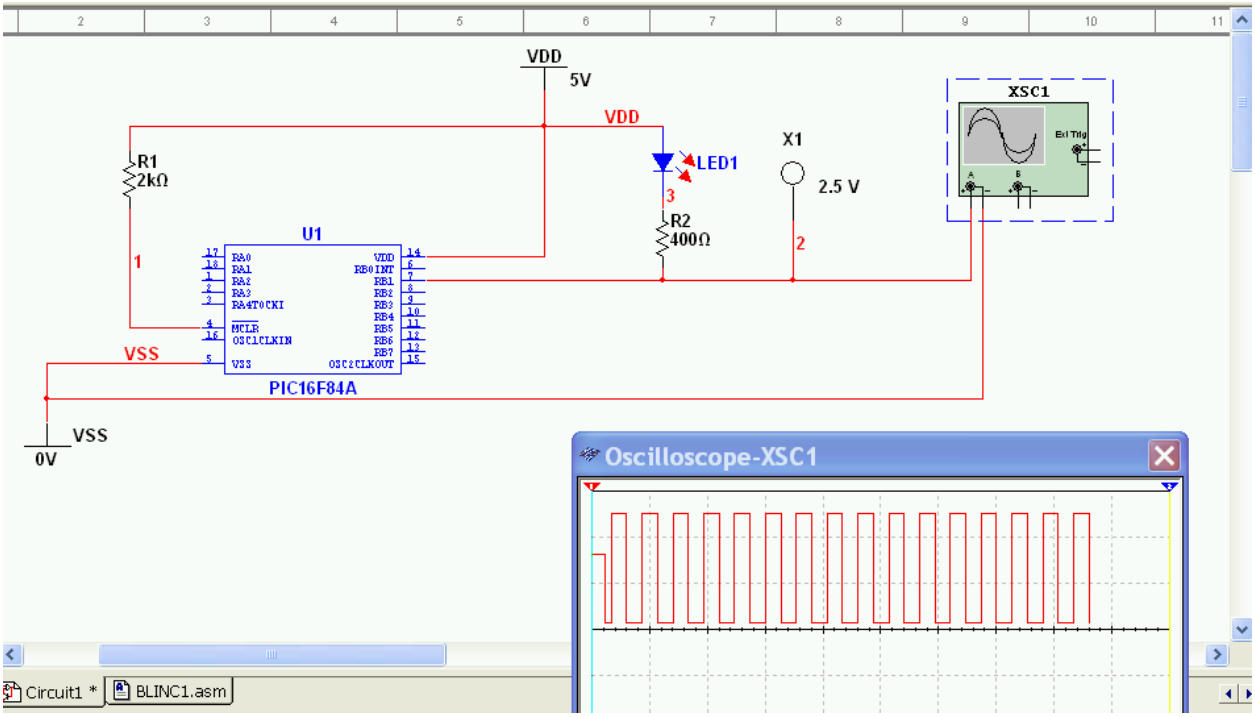

Рис. 6

Ввод электрической схемы микропроцессорного устройства удобно начинать с обязательной части, которая включает элементы необходимые для правильного функционирования самого микропроцессора. К таким элементам реальной схемы относятся:

- элементы тактового генератора;
- элементы питания;
- схема сброса микропроцессора.

Наиболее просто в данном пакете моделирования решена проблема с элементами обеспечения работы тактового генератора. В PIC-микроконтроллерах может использоваться внешний тактовый генератор, внутренний усилитель генератора с внешней резонансной цепью или встроенный *RC*-генератор. В любом случае этот генератор является аналоговым устройством и при настройке микропроцессорного устройства, в конечном итоге контролируется внешними измерительными приборами. По этой причине никакое

моделирование не может правильно отразить работу тактового генератора. С целью разрешения этой ситуации работа тактового генератора в среде *Multisim* моделируется упрощено, а именно тактовый генератор считается всегда исправным и работающим на частоте, которую можно установить на панели свойств микропроцессора (Рис.7). Вызвать эту панель можно через п. **Propertiies** контекстного меню микропроцессора, открывающееся после щелчка ПКМ по обозначению микропроцессора.

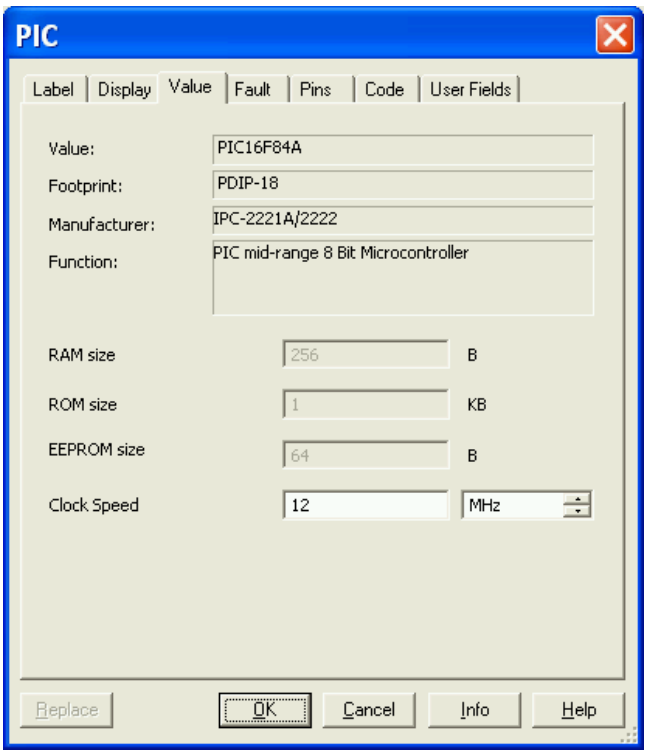

Рис. 7

Микроконтроллеры серии PIC16 способны работать при напряжении питания в диапазоне соответствующем интегральным микросхемам TTL- и CMOS-технологий. Обычно это положительное напряжение 4 – 6 В. В лабораторной работе следует взять 5В. Микроконтроллеры серии PIC16 выполнены по технологии CMOS. Полюса источника питания для CMOS микросхем имеют устоявшиеся обозначения: VDD – для полюса питания (В данном случае +5В) и VSS – для потенциала нулевого провода (В данном случае 0В). В общем случае следует присоединять модели тех источников, которые обозначены на фирменном изображении микросхемы.

Последняя группа элементов относится к схеме сброса микроконтроллера. Согласно обозначению микроконтроллеров серии PIC16 они имеют инверсный вход сброса (черта над обозначением входа сброса MCLR). Поэтому если подать нулевою потенциал на этот

контакт, процессор будет удерживаться в состоянии останова и работать не будет. Для начала работы необходимо подать на вход сброса положительный потенциал близкий к напряжению питания. Это можно сделать с помощью специальной схемы управления сбросом, если требуется управление процессором по этому входу. В данном случае такого управления не требуется и положительный потенциал можно подать через резистор 1 – 5 кОм. Достаточно присоединить этот резистор одним выводом к напряжению питания, а другим – к входу MCLR.

Для визуальной индикации выводимых данных можно использовать светодиоды. Так как все разряды порта меняются одинаково, здесь включен только один светодиод LED1. Чтобы ограничить величину тока через него, последовательно с ним надо включать резистор R2. При нулевом напряжении на выходе светодиод светится, а при единице – гаснет.

Таким образом, в процессе работы микропроцессора по программе светодиод должен моргать. Однако, если тактовая частота будет высокой, эти моргания заметить будет трудно. Для фиксации быстрых процессов дополнительно подключен осциллограф. Для удобства можно также подключить виртуальный индикатор X1, который светится при напряжении на контакте больше порогового значения.

2. ЛАБОРАТОРНОЕ ЗАДАНИЕ

2.1. Создать пустой проект микропроцессорного устройства в среде Multisim на микроконтроллере PIC16F84A для отладки программы на языке ассемблера.

2.2. Присоединить к проекту готовый файл программы blinc.asm, которая обеспечивает периодическое включение и выключение светодиодов присоединенных к контактам цифрового порта ввода/вывода **PORTB**.

2.3. Составить схему тестового устройства в среде Multisim.

2.4. Транслировать программу. Пользуясь файлом blinc.lst, созданной встроенным транслятором Multisim, определить наличие ошибок в исходном тексте и исправить их.

2.5. Включить отладочный режим и проверить работу тестовой программы по шагам.

2.6. Запустить программу в автоматическом режиме. Снять осциллограммы напряжений на выходах **PORTB**.

2.7. Изменить частоту тактового генератора так чтобы изменение состояния выходов происходило каждую миллисекунду условного времени модели.

2.8. Составить подпрограмму временной задержки и встроить ее в программу **blinc.asm**, подобрав параметры задержки так, чтобы изменение состояния выходов происходило каждые две миллисекунды условного времени модели при тактовой частоте микропроцессора 1МГц.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какими возможностями преимуществами обладает *Multisim* для отладки управляющих программ микроконтроллеров?

2. В каком формате задается программа микроконтроллера для отладки в *Multisim* ?

3. Какие режимы отладки реализованы в *Multisim* ?

4. Как моделируется тактовый генератор?

5. Как изменить частоту тактового генератора в модели?

6. Как воспользоваться пошаговым режимом среды *Multisim* ?

Лабораторная работа № 4.

ПОРТЫ ЦИФРОВОГО ВВОДА/ВЫВОДА МИКРОКОНТРОЛЛЕРОВ PIC16F84A

ЦЕЛЬ РАБОТЫ

Изучение устройства и программирования работы портов цифрового ввода/вывода (ЦВВ) восьмиразрядных однокристальных микро-контроллеров семейства *PIC16*.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Описание портов ввода/вывода

Универсальные порты ЦВВ позволяют PIC-микроконтроллерам контролировать работу и управлять другими устройствами передавая им или принимая от них данные в параллельном коде, т.е. в виде напряжений одновременно на нескольких контактах порта. Порты ЦВВ как и все периферийные устройства соединяются с ядром микропроцессора через шину данных (рис1).

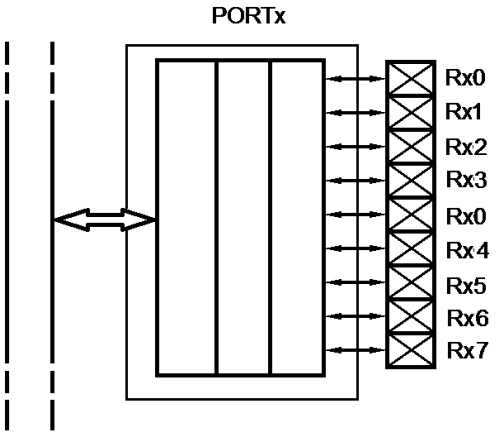

Рис. 1

В микроконтроллерах *PIC16* порты ЦВВ имеют разрядность до одного байта, т.е. до восьми разрядов. Порты несколько различаются по свойствам и разрядности. В некоторых портах отдельные разряды могут иметь индивидуальные особенности или отсутствовать, однако типовой порт считается восьмиразрядным. Все разряды портов построены на основании типовой структуры одного разряда, приведенной на рис. 2.

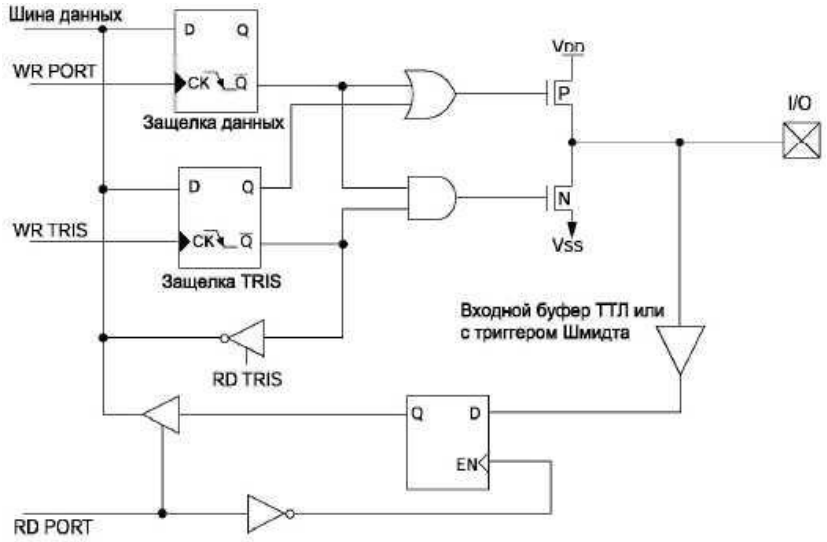

Рис. 2. Типовая структурная схема одного разряда порта ввода/вывода

Для управляющей программы каждый порт ЦВВ представлен двумя специальными регистрами памяти данных:

Регистр ***PORTx***, в который выводимый байт записывается через шину данных по сигналу управления шиной WR PORT, который вырабатывается автоматически ядром микропроцессора при выполнении команды записи в регистр *PORTx*. На рис.2 из технического описания один разряд порта изображен в виде синхронного D-триггера и называется “ Защелка данных”. Данные из регистра PORTx через логическую схему состоящую из логических элементов И и ИЛИ поступают в выходной усилитель на комплементарных полевых транзисторах, соединенных стоками между собой и с соответствующим выводом I/O на корпусе микропроцессора.

Регистр ***TRISx*** служит для управления направлением передачи данных, т.е. он устанавливает режим или вывода данных из микропроцессора, или ввода. На рис. 2 отдельный разряд этого регистра называется “Защелка TRIS”. Запись восьмиразрядного кода в регистр *TRISx* определяет направление передачи. Каждый разряд может устанавливаться в '0' или '1' независимо от других разрядов. Это позволяет индивидуально устанавливать направление ввода/вывода для каждого разряда, выбранного порта. Если бит *TRIS* установлен в '1' то соответствующий разряд порта ввода/вывода работает как вход, а если бит *TRIS* сброшен в '0', то - как выход.

1.2. Семи-сегментный индикатор

Светодиодный семисегментный индикатор представляет собой группу светодиодов расположенных в определенном порядке и объединенных конструктивно. Зажигая одновременно несколько светодиодов можно формировать на индикаторе символы цифр. Индикаторы различаются по типу соединения светодиодов – общий анод, общий катод, по количеству отображаемых разрядов – однораразрядные, двух разрядные и т.д. и по цвету – красные, зеленые, желтые и т.д.

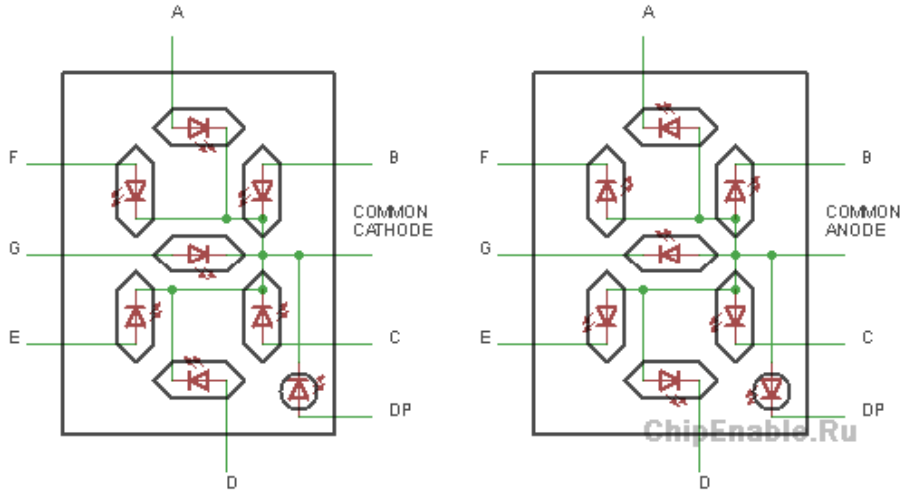

Рис. 3

Семисегментным индикатором можно управлять статически или динамически. При статическом управлении разряды индикатора подключены к микроконтроллеру независимо друг от друга и информация на них выводится постоянно.

Чтобы зажечь на индикаторе какую-то цифру нужно настроить порты, к которым подключен индикатор, в режим выхода, “открыть” транзистор (в данном случае подать на базу “единицу”) и установить в порту микроконтроллера её код. В зависимости от того, куда подключены сегменты индикатора – коды могут быть разные. Для нашего случая коды цифр будут выглядеть так.

Рис.4

Такие индикаторы при различных комбинациях светящихся элементов высвечивают цифры от 0 до 9. В данном случае предпочтительно использовать индикаторы с общим анодом. Для цифры 0 необходимо погасить сегмент *g*, а остальные должны светится. Для цифры 1 — светятся сегменты *b и c;* сегменты *a, d, e, /, g* погашены и т. д. Сегмент будет гореть, если на него будет подано напряжение логического нуля. Сегмент будет погашен, если на него будет подано напряжение логической единицы. Это соответствует подключению общего для всех сегментов электрода к напряжению питания. Двоично-десятичный код, который необходимо вывести следует в программе предварительно преобразовать в код семисегментного индикатора. Запишем таблицу истинности для данного преобразования кодов.

Таблица преобразования десятичного кода в семисегментный

Десятичная цифра	Код 8421				Состояние сегментов						
	X4	X3	X2	X1	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

Это преобразование программными средствами можно выполнить с помощью специальной процедуры реализующей таблицу.

1.3. Виртуальный цифроаналоговый преобразователь (ЦАП).

Простейший восьмиразрядный виртуальный ЦАП находится в группе **Mixed>ADC-DAC** под именем **VDAC**. Для подключения его к микропроцессору необходимо соединить входы ЦАП с выходами выбранного порта микропроцессора и подать опорное напряжение на входы **Vref+** и **Vref-**. Аналоговое напряжение пропорциональное входному коду снимается с вывода **Output**. Возможный пример подключения ЦАП к модели микропроцессора показан на рис. 5.

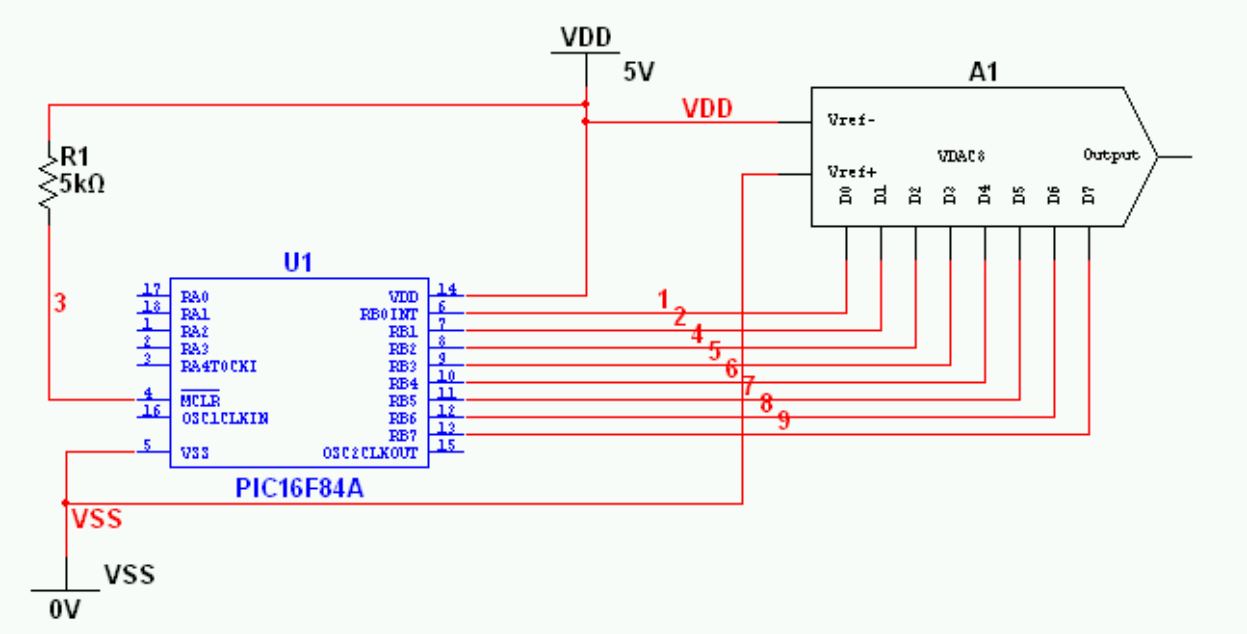

Рис. 5

1.3. Примеры программной реализации некоторых функций

1.3.1. Организация таблиц

В PIC-микроконтроллерах создание таблиц констант удобно выполнять с помощью команд вызова процедуры и возврата из процедуры. Таблица оформляется в виде подпрограммы, а именно процедуры в тексте которой и заносятся константы в виде команд возврата с операндами – константами.

CALL - Вызов процедуры. Вызов подпрограммы. Во-первых, адрес следующей после CALL команды (PC+1) помещается на вершину стека. Одинадцатибитный непосредсдственный адрес загружается в счетчик комманд PC биты <10:0>. Верхние биты PC <10:9> загружаются из PCLATH, бит PC<8> обнуляется. CALL выполняется за два командных цикла.

RETLW Возврат с литерой в W. В регистр W загружается значение 8-биной литеры 'k'. Счетчик программы загружается с вершины стека(адрес возврата). Команда выполняется за два командных цикла.

Синтаксис: [*label*] RETLW k

Ниже приведен обобщенный пример реализации таблицы. Сама таблица оформлена в виде процедуры с именем TABL. Перед обращением к таблице в регистр аккумулятор любым способом заносится номер элемента, вызываемого из таблицы. В примере это выполнено чтением из ячейки оперативной памяти с именем INDEX, где он должен быть подготовлен заранее.

```
TABLE    ADDWF  PCL  ;W=смещение
         RETLW  k1;Начало таблицы TABLE
         RETLW  k2
         *****  **
         *****  **
         *****  **
         RETLW    kn; конец таблицы TABLE
```

```
 MOVF INDEX,0
CALL  TABLE ; Обращение к таблице, аккумулятор W
                    ; содержит номер константы таблицы
         *  ;W теперь содержит
         *  ;значение из таблицы.
           ;


         end
```

Первой командой **ADDWF PCL** в процедуре **TABLE** выполняется увеличение содержимого счетчика команд **PCL** на величину номера, выбираемого элемента из таблицы,

занесенного перед обращением в аккумулятор **W**. Номер элемента задается считая с нуля. Таким образом, происходит безусловный переход к команде возврата из процедуры **RETLW**, содержащей в качестве операнда с непосредственной адресацией, выбранную константу. При исполнении этой команды в **W** заносится значение константы и происходит возврат к вызывающей программе, которая и использует константу из аккумулятора.

1.3.2. Организация ввода и вывода с помощью портов ЦВВ

Для реализации процесса ввода/вывода через порты ЦВВ необходимо выполнить три программных процедуры:

- начальный сброс портов;
- инициализация портов - задание направления ввода или вывода;
- ввод или вывод данных.

Начальный сброс портов ЦВВ требуется для обеспечения их защиты от аварийных режимов в момент включения питания, если контакты портов подключены к внешним источникам напряжения. В этом случае необходимо перевести порты в режим ввода. Для этого достаточно записать единицы во все разряды регистров TRISx каждого порта. Пример начального сброса PORTA приведен ниже.

Пример: Начальный сброс PORTA

```
BSF     STATUS, RP0     ; Выбрать банк 1
MOVLW 0xFF              ; Запись единиц во все разряды
                        ; направления каналов PORTA
MOVWF   TRISA           ; Настроить каналы как входы,
BCF     STATUS, RP0     ; вернуться в банк 0
```

Здесь считается, что программа работает с банком 0 оперативной памяти, так как при включении питания этот банк устанавливается автоматически импульсами сброса при пуске процессора. Регистр TRISA расположен в первом банке оперативной памяти, поэтому перед записью в него необходимо перейти к работе с банком 1. После сброса порта снова надо вернуться в банк 0.

Для инициализации необходимо определить начальное значение на выходах порта путем записи в регистр PORTx соответствующего кода (Обычно 0 во всех разрядах) и направление

передачи для каждого канала (разряда) путем записи в регистр TRISx кода, в котором разряд устанавливаемые на вывод равны 0, а на ввод – 1. Пример инициализации приведен ниже.

Пример: Инициализация PORTA

```
BCF         STATUS, RP0   ; Выбрать банк 0
MOVLW 0xFF            ; Занесение начальных состояний в
MOVWF    PORTA              ; защелку PORTA

BSF         STATUS, RP0       ; Выбрать банк 1
MOVLW 0xCF            ; Значение для инициализации
                                ; направления каналов PORTA
MOVWF     TRISA               ; Настроить RA<3:0> как входы,
                                ; настроить RA<5:4> как
                                ;  выходы.
```

Теперь для вывода данных достаточно записывать их в регистр PORTx (В примере PORTA) и результаты будут появляться на выходах. На разряды настроенные для ввода данные в PORTx не влияют.

Для ввода данных по соответствующим входам достаточно считывать их из регистра опять же PORTx. При этом в разрядах, настроенных на ввод будем получать значения соответствующие внешнему напряжению на входах, а в разрядах настроенных вывод получим нули.

2. ЛАБОРАТОРНОЕ ЗАДАНИЕ

2.1. Изучить устройство портов ЦВВ микроконтроллера PIC16F84A по справочнику, семи-сегментного светодиодного индикатора по техническому описанию и виртуального аналого-цифрового преобразователя среды *Multisim*.

2.2. Составить принципиальную схему устройства вывода данных через порты цифрового ввода/вывода микроконтроллера согласно индивидуальному заданию.

2.3. Составить общий и подробные алгоритмы программы управления индикаторными устройствами PIC16F84A согласно индивидуальному заданию.

2.4. По подробному алгоритму составить программу для микроконтроллера PIC16F84A.

2.5. Отладить написанную программу в среде MPLAB.

2.6. Промоделировать работу схемы устройства в среде *Multisim*.

3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Каждый вариант программы должен предусматривать возможность приостановки процесса вывода результатов, управляемый выключателем по одному из входов ЦВВ.

1. Счетчик импульсов с одноразрядным десятичным индикатором.

2. Двоичный счетчик со светодиодным двоичным восьмибитным индикатором.

3. Программа иллюстрации двоичного циклического сдвига байта влево и вправо со виртуальным индикатором.

4. Бегущий огонь на линейке светодиодов.

5. Генератор пилообразного напряжения с виртуальным ЦАП.

6. Генератор ступенчатого напряжения с уровнями 0, 0,3 и 1 от максимального напряжения выводимого виртуальным ЦАП.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему выходной каскад разряда порта должен обеспечивать три состояния?
2. Зачем нужны регистры *TRIS* и *PORT*?
3. Как установить единичное значение на нескольких разрядах порта, если остальные разряды используются в режиме входа?
4. Почему считываются все восемь разрядов порта, если требуется значение только одного разряда, работающего в режиме ввода? Как выделить этот разряд?
5. Как преобразовать двоично-десятичный код в код семисегментного индикатора?
6. Как перевести разряд порта в режим аналогового ввода?

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Цифровая схемотехника : учеб. пособие для вузов / Е.П. Угрюмов. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2010. — 809 с.: ил. - ISBN 978-5-9775-0162-0.

2. Цифровые устройства и микропроцессоры: учеб. пособие / А. В. Микушин, А. М. Сажнев, В. И. Сединин. — СПб.: БХВ-Петербург, 2010. — 832 с.: ил. — (Учебная литература для вузов). - ISBN 978-5-9775-0417-1.

3. Цифровые устройства и микропроцессоры /Д.А.Безуглов, И.В.Калиенко. – Ростов н/Д.: Феникс, 2006. – 480с.;

4. Брей Б. Применение микроконтроллеров PIC18: Пер. с англ. – СПб.: «МК-пресс»,2008. – 576. с.

5. Multisim. Руководство пользователя. National Instruments Corporation. 2007. - 491 с.

6. Шестеркин А.Н. Система моделирования и исследования радиоэлектронных устройств Multisim 10. – М.: ДМК Пресс,2012 – 360 С. ISBN 978-5-94074-756-7.

7. MPASM. Руководство пользователя. – М. :ООО "Микро-Чип", 2001 – 61 с.