Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра радиотехнических систем

В.Н. Левкович, А.С. Грицук, И.В. Коваленко

КОНСТРУИРОВАНИЕ ПРОГРАММ НА АССЕМБЛЕРЕ ДЛЯ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА PICMICRO

Учебное пособие
по курсу “Цифровые и микропроцессорные устройства»
для студентов специальностей
39 01 01 «Радиотехника» и 39 01 02 «Радиоэлектронные системы»
всех форм обучения

Минск 2004

УДК 004.31(075.8)
ББК 32.973 я 73
Л 37

Р е ц е н з е н т:
доцент кафедры сетей и устройств телекоммуникаций БГУИР,
канд. техн. наук. И.И. Астровский

Левкович В.Н.

Л 37 Конструирование программ на Ассемблере для микроконтроллеров семейства PICmicro: Учеб. пособие по курсу “Цифровые и микропроцессорные устройства» для студ. спец. 39 01 01 «Радиотехника» и 39 01 02 «Радиоэлектронные системы» всех форм обуч. /В.Н. Левкович, А.С. Грицук, И.В. Коваленко. –Мн.: БГУИР, 2004. - 80 с.: ил.
ISBN 985-444-647-6

В учебном пособии рассмотрены программные и аппаратные инструментальные средства, используемые в процессе конструирования программ для микроконтроллеров, дан краткий обзор и сравнительные возможности языков программирования, рассмотрена последовательность и порядок применения инструментальных средств на различных стадиях конструирования программы, приведено описание языка Ассемблер MPASM для однокристальных микроконтроллеров семейства PICmicro.

УДК 004.31(075.8)
ББК 32.973 я 73

ISBN 985-444-647-6

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящее пособие является очередным в серии пособий, выпускаемых преподавателями кафедры РТС для обеспечения учебного процесса по курсу «Вычислительные и микропроцессорные устройства».

Ранее изданы пособия по архитектуре и основам программирования базовой модели семейства PIC16 [1], а также по принципам работы в интегрированной среде разработки и отладки программ для микроконтроллеров семейства PICmicro MPLAB [2].

В настоящем пособии подробно изложен язык программирования Ассемблер MPASM.

В дальнейшем планируется выпуск пособия с изложением примеров программирования важнейших процедур для типовых применений микроконтроллеров, а также примеров ряда проектов, по уровню сложности соответствующих курсовым работам и проектам по названному курсу.

Завершить серию планируется изложением языка программирования С для PICmicro, а также примеров разработки проектов на этом языке.

Процесс создания программ для микроконтроллеров имеет много общего с написанием прикладных программ для персональных компьютеров (ПК). До появления исполняемого файла, пригодного для записи в память микроконтроллера, исходный текст программы проходит те же этапы компиляции, линковки и отладки, которые используются и при конструировании программ для ПК. Программирование микроконтроллеров также прошло большой путь развития от программирования в машинных кодах до применения современных интегрированных систем написания и отладки программ.

1 ОБЗОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

В настоящее время при разработке программного обеспечения (ПО) для микроконтроллеров широко используется персональный компьютер и целый ряд специальных прикладных программ, называемых инструментальными средствами. К ним относятся:

- редактор исходного кода;
- Ассемблер;
- компилятор;
- компоновщик (линкер);
- библиотекарь;
- отладчик (дебаггер);
- визуальный генератор исходного кода;
- интегрированная среда разработки (ИСР).

Рассмотрим назначение, основные свойства и наиболее известные реализации перечисленных инструментальных средств в применении к микроконтроллерам PIC.

1.1 Редактор исходного кода

Редактор исходного кода предназначен для создания и последующего редактирования исходных файлов программы. В качестве такого редактора можно использовать любой текстовый редактор, поддерживающий файлы с кодировкой ASCII. Однако существуют и специализированные редакторы исходного кода, предоставляющие ряд дополнительных сервисных услуг, таких, как выделение ключевых слов языка программирования, автоматическое форматирование исходного кода, быстрая и удобная навигация по исходному коду, отображение подсказок, генерация типовых программных конструкций и др.

Наиболее часто в качестве редактора исходного кода при написании программ для PIC используется встроенный редактор ИСР Microchip MPLAB.

1.2 Ассемблер

Основной задачей Ассемблера является преобразование файла исходного кода, написанного на языке Ассемблер, в файл объектного кода для последующей компоновки. При этом Ассемблер выполняет синтаксическую проверку текста программы и вычисление выражений. Кроме создания файла с объектным кодом Ассемблер генерирует различные файлы отчета, используемые для анализа результатов ассемблирования. Как правило, Ассемблер является программой командной строки, однако в некоторых реализациях существует возможность работать с Ассемблером и в диалоговом режиме.

Самым распространенным Ассемблером для PIC является универсальный макроАссемблер Microchip MPASM. Кроме MPASM существуют и Ассемблеры, разработанные другими фирмами, например Tech-Tools SPASM.

1.3 Компилятор

Основной задачей компилятора, так же как и Ассемблера, является преобразование файла исходного кода, написанного на соответствующем языке высокого уровня, в файл объектного кода для последующей компоновки. В дополнение к действиям, выполняемым Ассемблером, компилятор выполняет еще и оптимизацию исходного кода. Компилятор в отличие от Ассемблера может преобразовывать один оператор соответствующего языка в несколько команд микроконтроллера. Таким образом, компилятор является интеллектуальным генератором объектного кода, в то время как Ассемблер просто транслирует исходный код в объектный. Это является принципиальным

отличием компилятора от Ассемблера. Компилятор также генерирует различные файлы отчета, используемые для анализа результатов компиляции. Как правило, компилятор является программой командной строки.

Существуют компиляторы для различных языков программирования. Наибольшее распространение для PIC-контроллеров получил язык программирования C. Самые известные среди компиляторов C: Microchip MPLAB-C, CCS PCW, High-Tech PICC, B. Knudsen Data CC5X, FED C Compiler.

1.4 Компоновщик

Компоновщик выполняет объединение и преобразование (компоновку) файлов объектного кода в файл выполнимого кода для последующего программирования микроконтроллера. Как и Ассемблер, компоновщик создает файлы отчета и является программой командной строки.

Наиболее известным компоновщиком является Microchip MPLINK.

1.5 Библиотекарь

Библиотекарь служит для объединения файлов объектного кода в файл библиотеки для последующего использования в других программах, а также для редактирования существующих файлов библиотек. Библиотекарь является программой командной строки.

Распространенным библиотекарем является Microchip MPLIB.

1.6 Отладчик

Отладчик предназначен для проверки и отладки работы программы. При этом основными инструментами отладки являются возможности пошагового

выполнения программы, ее останова в заранее заданных точках, а также анализа состояния всех регистров и портов микроконтроллера.

Отладчики для микроконтроллеров подразделяются на две категории: эмуляторы и симуляторы.

Эмуляторы представляют собой программно-аппаратные средства, позволяющие отслеживать выполнение программы в реальной аппаратуре в режиме реального времени, а также в пошаговом режиме.

Симулятор представляет собой только программное средство, моделирующее процесс выполнения программы в микроконтроллере.

Достаточно часто отладчик является встроенным средством ИСР, что позволяет ему тесно взаимодействовать с редактором исходного кода. Однако существуют и отдельные программы-отладчики.

Среди симуляторов наибольшей популярностью пользуется встроенный в ИСР Microchip MPLAB симулятор Microchip MPLAB-SIM. Среди эмуляторов – Microchip MPLAB ICE, Microchip MPLAB ICD.

1.7 Визуальный генератор исходного кода

Визуальный генератор исходного кода позволяет путем установки параметров с помощью мыши в ряде диалоговых окон и использования механизма drag and drop легко получить исходный код для конфигурации периферийных модулей микроконтроллера, организации обработчиков прерываний и выполнения других стандартных действий. Визуальный генератор кода позволяет значительно сократить время написания исходного кода, но не обладает достаточной гибкостью и ограничен выполнением стандартных действий. При написании программы часто используется комбинированный подход, при котором сначала используется визуальный генератор, а затем редактируется полученный с его помощью код в редакторе. Визуальный генератор часто встраивается в ИСР.

Для PIC-контроллеров применяют визуальные генераторы исходного кода, например WIZ-C Visual Development.

1.8 Интегрированная среда разработки

Удобным средством при конструировании программ для микроконтроллеров является ИСР. Среда позволяет автоматизировать и упростить управление процессом создания программ. ИСР, как правило, объединяет в себе редактор исходного кода, Ассемблер и компилятор, компоновщик, симулятор и эмулятор, библиотекарь, визуальные средства генерации кода, программатор.

Самой популярной ИСР для PIC-контроллеров является Microchip MPLAB. Существуют и другие среды разработки, такие как IAR, WIZ-C.

Существует также альтернативный способ управления процессом конструирования программы – использование пакетных файлов (.bat). В этом случае создается пакетный файл, содержащий вызовы необходимых инструментальных средств и обработку результатов их вызова. Этот способ менее удобен, нежели конструирование программы с использованием ИСР, однако все же находит свое применение.

2 ОБЗОР ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

В настоящее время широкое распространение получили такие языки программирования, как Ассемблер, C, C++, Pascal, Object Pascal, Java, Basic и др. Все эти языки являются универсальными и предназначены для выполнения широкого круга задач.

2.1 Различия языков программирования

Различные языки программирования отличаются друг от друга областью применения, грамматикой, степенью абстрагирования от языка машинных кодов и аппаратных средств микропроцессора. Однако главным отличием языков программирования друг от друга является поддержка различных стилей программирования: процедурного программирования (использование подпрограмм, процедур и функций с передачей параметров и возвратом результата), модульного программирования (использование модулей), объектно-ориентированного программирования (использование иерархий полиморфных классов), обобщенного программирования (использование шаблонных функций и классов). Так, например, Ассемблер Microchip для микроконтроллеров PIC не поддерживает ни один из перечисленных стилей, C поддерживает только процедурный стиль программирования, Pascal и Object Pascal – все, кроме обобщенного, а C++ поддерживает все стили.

Примечание. Считается, что язык программирования поддерживает определенный стиль программирования, если предоставляет специальные программные конструкции для написания программ с использованием данного стиля. Однако даже если язык и не поддерживает конкретный стиль, то это не означает, что на нем невозможно программировать, придерживаясь данного стиля. Так, например, хотя Ассемблер иногда и не поддерживает процедурное программирование, а C – модульное, на Ассемблер возможна организация подпрограмм, а на C – модулей. В первом случае используется аппаратная поддержка процедурного программирования со стороны самого микропроцессора, а во втором – принцип раздельной компиляции.

2.2 Языки программирования для микроконтроллеров

В области программирования микроконтроллеров наибольшее распространение получили такие языки, как Ассемблер и С.

Ассемблер является языком низкого уровня, тесно связан с набором команд микроконтроллера и аппаратными средствами. Эти его особенности предоставляют программисту широкие возможности в управлении аппаратными средствами микроконтроллера и написании высокоэффективных с точки зрения времени выполнения и/или используемого объема памяти фрагментов исходного кода. С другой стороны, низкий уровень Ассемблер затрудняет разработку больших и сложных программ. Выход заключается в использовании языков более высокого уровня.

В 1983г. комитетом при Американском национальном институте стандартов (ANSI) был стандартизирован и рекомендован к применению в электронно-вычислительных системах язык программирования С. Важной особенностью языка С является то, что наряду с предоставлением более высокоуровневых программных конструкций язык сохраняет возможность работы с аппаратными средствами на низком уровне и создания высокоэффективного исходного кода.

При программировании микроконтроллеров иногда используются и другие языки, такие как Basic и Pascal, однако они не получили такого распространения, как С, в основном из-за слишком высокого предполагаемого уровня абстракции от аппаратных средств.

Остановимся более подробно на отличительных особенностях Ассемблер и С.

2.3 Особенности Ассемблер и С

Ассемблер относительно прост в освоении, тесно связан с архитектурой микроконтроллера, что делает его весьма полезным языком с методической точки зрения особенно на стадии первичного изучения микроконтроллеров. Кроме того, Ассемблер позволяет с точностью до команды контролировать генерируемый объектный код программы.

С обеспечивает поддержку различных типов данных (символьные, целочисленные, вещественные), символьных строк, массивов и указателей. Компилятор автоматически выполняет статическое и динамическое распределение памяти, а также статическую инициализацию переменных в памяти данных. Поддерживает большое число различных операций над переменными (арифметические, отношения, логические, присваивания, побитовые и др.), и предоставляет различные операторы (условия, циклов, передачи управления и др.). Язык полностью поддерживает стиль процедурного программирования, предоставляя операторы для определения и вызова функций, передачи и возврата из них параметров. *С* поддерживает типы, определяемые пользователем (структуры, объединения и перечисления), содержит мощный препроцессор и ряд стандартных библиотек для диагностики программ, работы с числами с плавающей точкой, математических вычислений, поддержки функций с переменным числом аргументов, ввода-вывода и др.

3 КОНСТРУИРОВАНИЕ ПРОГРАММЫ

Разработка непосредственно исходного кода программы является лишь составной частью более сложного процесса, называемого конструированием программы. Результатом конструирования программы является выполнимый файл, пригодный для программирования микроконтроллера, либо файл библиотеки, предназначенный для последующего использования в других программах и часто другими программистами.

На рисунке 1 приведена обобщенная схема процесса конструирования программы в общем случае.

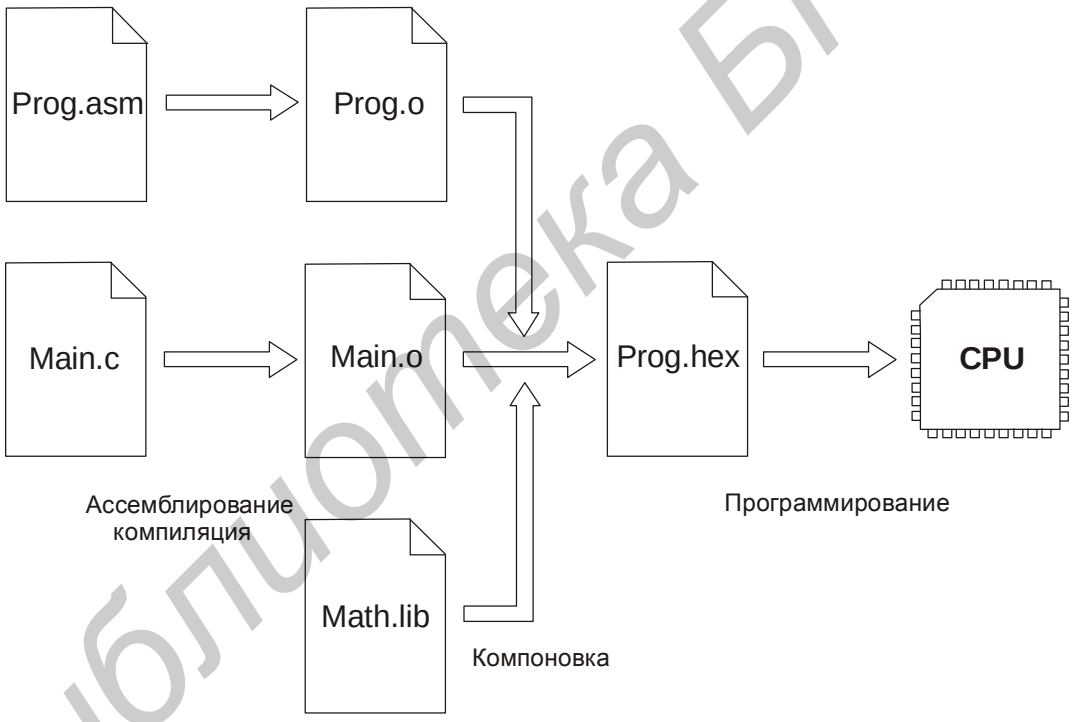


Рисунок 1- Обобщенная схема процесса конструирования программы

Процесс конструирования программы включает в себя три этапа:

- разработка исходного кода;
- ассемблирование и компиляция исходного кода;
- компоновка программы или библиотеки.

3.1 Разработка исходного кода

Этап разработки исходного кода меньше всего автоматизирован, и основную работу на нем выполняет сам программист. Исходными данными для программиста являются включаемые файлы для библиотек и файлов объектного кода (.inc), заголовочные файлы для библиотек и файлов объектного кода (.h), а также справочные файлы, содержащие информацию по использованию библиотек, предоставляемых, как правило, другими разработчиками. Включаемый файл содержит информацию о библиотеке или файле объектного кода, необходимую для работы Ассемблера. Этот файл включается программистом в файлы исходного кода программы посредством специальной директивы. Заголовочный файл содержит информацию о библиотеке или файле объектного кода, необходимую для работы компилятора С. Данный файл также включается программистом в файлы исходного кода при помощи специальной директивы. На этапе разработки исходного кода программист, использующий Ассемблер, создает файлы исходного кода программы на языке Ассемблер (.asm) и соответствующие им включаемые файлы (.inc). В случае использования С программист создает файлы исходного кода на С (.c) и соответствующие им заголовочные файлы (.h). В некоторых случаях исходный код программы состоит из файлов, написанных как на Ассемблер, так и на С. Кроме этого, в процессе разработки исходных кодов программист, как правило, создает файлы примечаний (.txt), предназначенные для последующего сопровождения программы. В редких случаях разработка программы ведется с “чистого листа”, без использования библиотек и соответствующих им включаемых и заголовочных файлов.

3.2 Ассемблирование и компиляция исходного кода

На этапе ассемблирования и компиляции основная работа выполняется Ассемблером и компилятором, которые осуществляют преобразование исходного кода программы в объектный код.

Исходными данными для Ассемблера являются файлы исходного кода на языке Ассемблер (.asm) и включаемые файлы (.inc), созданные программистом на предыдущем этапе. В случае успешного ассемблирования создаются файлы объектного кода программы (.o). Эти файлы являются промежуточными в процессе конструирования программы. Файл объектного кода не является выполнимым и не пригоден для программирования микроконтроллера. При этом он содержит машинно-зависимый код, а также дополнительную информацию для компоновщика. Файл объектного кода не содержит в явном виде информацию, доступную для анализа программистом. Кроме файлов объектного кода Ассемблер создает различные файлы отчета: файл ошибок (.err), файл листинга (.lst) и файл перекрестных ссылок (.xrf). Файл ошибок содержит ошибки, предупреждения и сообщения, сгенерированные во время ассемблирования. В случае наличия ошибок файл объектного кода не создается. Файл листинга содержит исходный код и объектный код программы, адреса размещения, символьную таблицу, карту распределения памяти программ, количество ошибок, предупреждений, сообщений и другую полезную информацию. Файл перекрестных ссылок содержит информацию по всем символическим именам, используемым в файле исходного кода.

Исходными данными для компилятора являются файлы исходного кода на языке С (.c) и заголовочные файлы (.h), созданные программистом на предыдущем этапе. В остальном процесс компиляции полностью аналогичен ассемблированию. Как правило, Ассемблер и компилятор способны обрабатывать только один файл исходного кода. Поэтому при наличии

нескольких файлов исходного кода необходимо каждый файл ассемблировать или компилировать отдельно.

Файлы объектного кода, сгенерированные различными компиляторами и Ассемблерами, имеют единый формат, что позволяет объединять их при компоновке. Эта особенность дает возможность при разработке программы использовать различные языки программирования. Кроме этого, большинство компиляторов С позволяет при необходимости включать в исходный код фрагменты кода, написанные на Ассемблере.

3.3 Компоновка программы или библиотеки

На этом этапе основную работу выполняют компоновщик или библиотекарь, которые преобразуют файлы объектного кода в выполнимый файл и файл библиотеки соответственно.

Исходными данными для компоновщика являются файлы объектного кода (.o), сгенерированные Ассемблером и компилятором на предыдущем этапе, а также файлы библиотек (.lib). В случае успешной компоновки генерируется выполнимый файл (.hex), пригодный для программирования микроконтроллера. Данный файл содержит выполнимый код программы в одном из так называемых hex-форматов (INHX8M, INHX8S, INHX32). Кроме выполнимого файла компоновщик создает различные файлы отчета: файл листинга (.lst), файл распределения памяти (.map), а также различные служебные файлы: файл для отладки (.cod) и COFF-файл (.out). Файл листинга содержит исходный код и объектный код всей программы, а также результат дизассемблирования. Файл распределения памяти содержит информацию о размещении секций программы, использовании памяти программы и информацию о распределении переменных и меток программы.

Библиотекарь используется для создания и модификации файлов библиотек (.lib). Файл библиотеки представляет собой коллекцию файлов

объектного кода, генерируемых Ассемблером и компилятором. Использование файлов библиотек позволяет избежать необходимости перечисления при компоновке большого количества файлов объектного кода. При компоновке файла библиотеки с другими файлами компоновщик включает в выполнимый файл не весь файл библиотеки, а только те его объектные модули, которые используются в программе. Таким образом, автоматически поддерживается минимальный размер выполнимого файла при модификации исходного кода программы.

Библиотекарь может выполнять несколько команд. Основной операцией является создание новой библиотеки. При этом исходными данными для библиотекаря является набор файлов объектного кода (рисунок 2). В случае успешного выполнения операции генерируется файл библиотеки. Кроме этого, возможно выполнение таких операций, как добавление файла объектного кода в библиотеку, извлечение файла объектного кода из библиотеки, удаление файла объектного кода из библиотеки и просмотр содержания файла библиотеки.

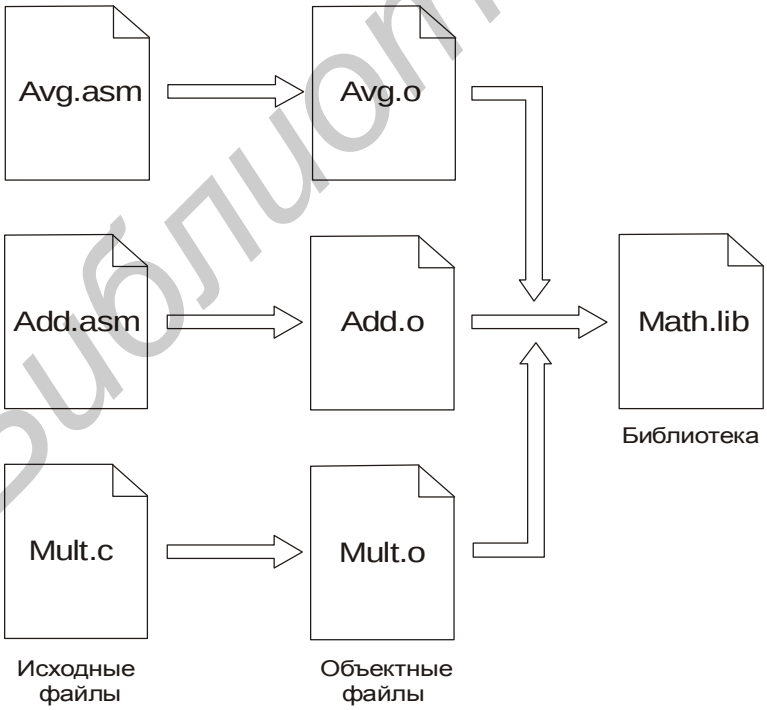


Рисунок 2 – Создание файла библиотеки

3.4 Пример

Проиллюстрируем процесс конструирования программы для микроконтроллера PIC16F873 на модельном примере.

Пусть в результате анализа поставленной задачи и проектирования структуры будущей программы было принято решение разработать программу program, состоящую из трех модулей: first, second и main. Для обеспечения высокой эффективности подпрограмм, содержащихся в модулях first и second, было принято решение разработать их на языке Ассемблер. Исходя из желания сократить время разработки, необходимости реализовать сложную функциональность программы и обеспечить возможность ее модификации в будущем, в качестве языка программирования для модуля main был выбран C. Кроме того, для еще большего сокращения времени разработки было принято решение распределить задачу между двумя программистами: первому поручили разработать модули first и second, а второму – разработать модуль main и скомпоновать программу.

Первый программист, разрабатывая модули first и second, использует библиотеку library фирмы firm. Для этого он включает в файлы исходного кода своих модулей first.asm и second.asm включаемый файл library.inc, поставляемый фирмой firm. При разработке модуля second программист решает использовать в нем некоторые подпрограммы из своего же модуля first. Для этого он создает включаемый файл first.inc и затем включает его в файл исходного кода second.asm. После завершения разработки модулей первый программист ассемблирует их и получает файлы объектного кода first.o и second.o:

```
mpasm.exe  /p16f873 /o+ first.asm
mpasm.exe  /p16f873 /o+ second.asm
```

Кроме этого, он создает два заголовочных файла first.h и second.h для того, чтобы второй программист имел возможность использовать модули first и

second при разработке своего модуля main. Для того чтобы предоставить второму программисту возможность работать одновременно с первым, последний начинает разработку своих модулей не с файлов исходного кода, а с заголовочных файлов и сразу передает их второму программисту. После этого оба программиста теоретически могут разрабатывать свои модули независимо друг от друга.

Второй программист, разрабатывая модуль main, также использует библиотеку library фирмы firm. Для этого он включает в файл исходного кода своего модуля main.c заголовочный файл library.h, который, как и включаемый файл library.inc, поставляется фирмой firm. Кроме этого, он использует модули first и second, разрабатываемые первым программистом. Для этого он включает в файл исходного кода main.c полученные от него заголовочные файлы first.h и second.h. Второй программист компилирует свой модуль и получает файл объектного кода main.o:

mcc.exe /p16f873 /o+ main.c

Так как модуль main не используется в других модулях программы, то второй программист не создает ни заголовочный файл main.h, ни включаемый файл main.inc.

Взаимодействие модулей программы prog показано на рисунке 3.

После того как все модули программы успешно сассемблированы и скомпилированы, а также файлы объектного кода first.o и second.o и файл библиотеки library.lib, поставляемый фирмой firm, переданы второму программисту, последний компонует программу и получает выполнимый файл program.hex:

mplink.exe 16f873.lkr main.o first.o second.o library.lib –o program.out

На практике процесс конструирования программы является многоитерационным и описанные выше действия полностью или частично выполняются не один раз. В этом случае удобно вызовы Ассемблера, компилятора и компоновщика с указанием параметров и имен файлов

оформить в виде пакетного файла (.bat) или воспользоваться интегрированной средой разработки, создав в ней соответствующий проект.

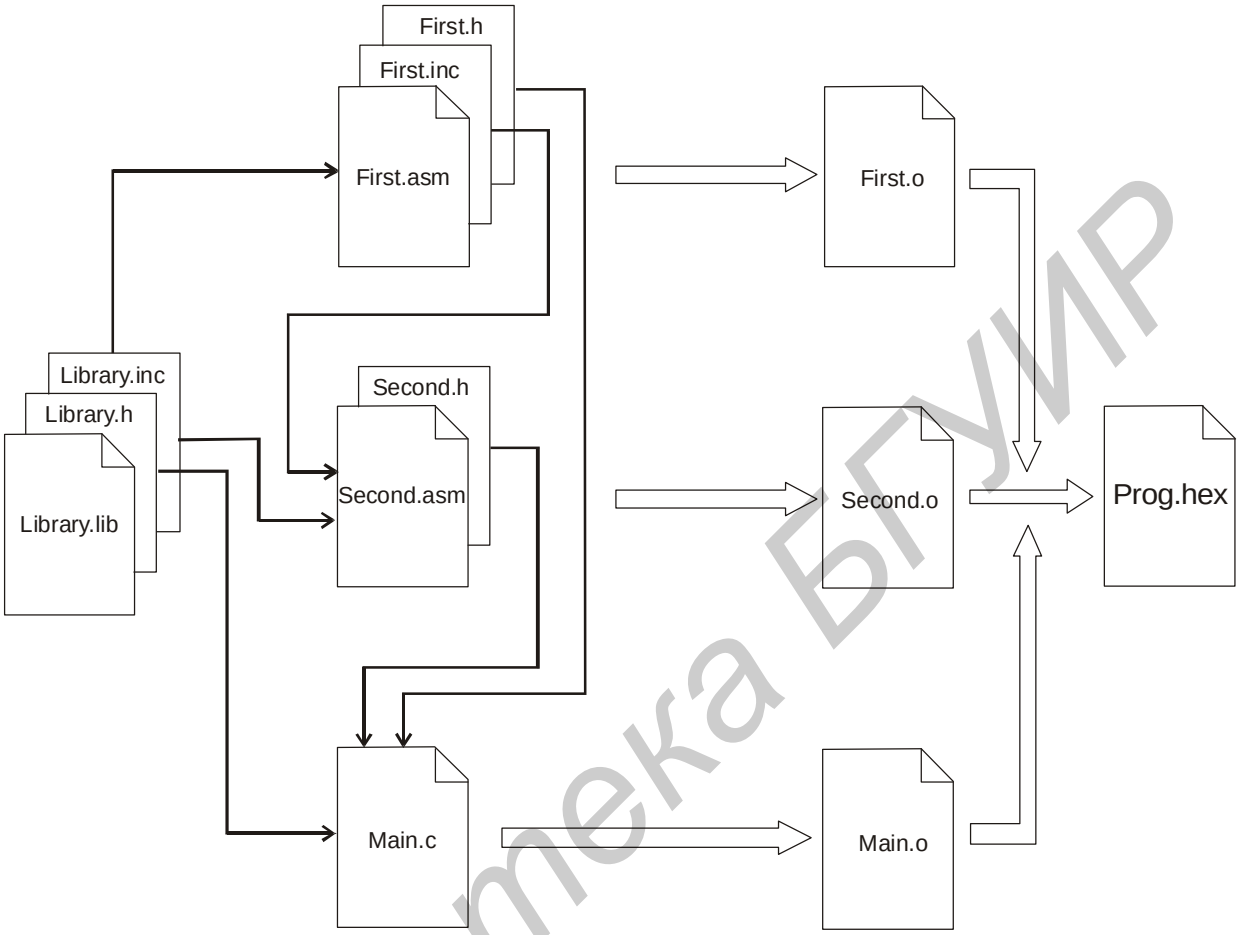


Рисунок 3 – Взаимодействие модулей программы prog

4 АССЕМБЛЕР MICROCHIP MPASM

Ассемблер Microchip MPASM предназначен для ассемблирования программ всех семейств микроконтроллеров PIC.

Ассемблер Microchip MPASM характеризуется такими особенностями, как:

- поддержка наборов команд всех семейств микроконтроллеров;
- интерфейс командной строки;
- широкий набор директив;
- гибкий язык макрокоманд;
- совместимость с интегрированной средой разработки Microchip MPLAB.

Ассемблер Microchip MPASM можно использовать в двух режимах:

- для генерации файла объектного кода, используемого впоследствии для компоновки с другими файлами объектного кода и библиотеками;
- для генерации файла выполнимого кода, пригодного для программирования микроконтроллера.

Последний режим не характерен для большинства Ассемблеров и не позволяет разрабатывать многомодульные программы с использованием различных языков программирования. Однако он оказывается удобным при построении небольших одномодульных программ, т.к. максимально упрощает процесс конструирования программы.

4.1 Файл исходного кода

Файл исходного кода должен иметь расширение .asm и может быть создан с использованием любого редактора, поддерживающего ASCII-файлы.

Каждая строка файла исходного кода является предложением языка Ассемблер, которое может включать в себя до четырех полей:

- поле меток;
- поле мнемоник;
- поле операндов;
- поле комментариев.

Наличие всех четырех полей не является обязательным. Важен порядок следования и расположение полей в предложении. Поля должны быть расположены в приведенной выше последовательности. Поле меток должно располагаться с первой позиции в предложении. Поле мнемоник должно располагаться, начиная со второй и далее позиции в предложении. Разделителями полей является один или более символов пробела или табуляции. Максимальная длина предложения не должна превышать 255 символов.

4.1.1 Поле меток

Имена меток должны начинаться с буквы или с символа подчеркивания (_) и могут содержать комбинации букв, цифр и знака вопроса. Длина метки не должна превышать 32 символа. Метка может заканчиваться двоеточием (:). При этом символ двоеточия не входит в имя метки.

4.1.2 Поле мнемоник

В этом поле могут использоваться мнемоники команд микроконтроллера (в том числе и псевдокоманды), имена директив Ассемблера и имена макрокоманд.

4.1.3 Поле операндов

Поле операндов содержит один или более операндов. В случае использования нескольких операндов они должны быть разделены запятыми (,).

4.1.4 Поле комментариев

Поле комментариев должно начинаться с символа точка с запятой (;) и продолжается до конца строки. Содержание поля комментариев Ассемблером игнорируется, поэтому оно может содержать любые символы.

4.1.5 Правила оформления

При написании исходного кода для повышения его читабельности желательно придерживаться определенных правил оформления. Эти правила не являются обязательными, однако часто используются многими программистами.

Файл исходного кода всегда снабжается некоторой шапкой, которая содержит такую информацию, как название и/или назначение модуля, имя автора, версия и дата, внесенные изменения и т.д. Имена мнемоник команд и псевдокоманд следует записывать полностью строчными буквами, а имена директив и операторов Ассемблера, а также макрокоманд – полностью прописными буквами, главным образом для того, чтобы легко отличать их от первых. В случае, если имя метки, переменной или другого идентификатора является составным (состоит из двух и более отдельных слов), следует использовать одну из двух распространенных схем записи составных имен. По первой схеме имя записывается полностью строчными буквами, а для разделения слов используется символ подчеркивания. Например: first_variable. По второй схеме символ подчеркивания не используется, также используются

строчные буквы, однако каждое слово в имени записывается с прописной буквы. Например: FirstVariable. Служебные имена следует начинать с одного или двух символов подчеркивания. После запятой в поле операндов принято оставлять один пробел. Одиночные пробелы также расставляют слева и справа от операторов при записи выражений. Для разделения полей в предложении следует использовать не пробелы, а один или два символа табуляции. Не следует комментировать каждую строку исходного кода. Всегда следует стараться выражать свои мысли не с помощью комментариев, а через соответствующие конструкции языка программирования. Полезно предложения Ассемблера, выполняющие какую-то одну операцию, группировать, используя пустые строки.

Кроме этого, при выборе имени переменных и констант рекомендуется использовать так называемую венгерскую запись (нотацию). Впервые она была применена сотрудником Microsoft - венгром по происхождению. Суть ее заключается в том, что в имя переменной или константы включается префикс, указывающий на ее тип. Это постоянно напоминает программисту о типе переменной, что особенно важно при программировании на Ассемблере, и исключает необходимость частого и утомительного поиска определения переменной. Префикс должен быть записан строчными буквами, представлять собой сокращение имени типа и быть значительно короче самого имени. Например:

```
bFirstVariable   RES  1     ; Переменная является байтом (byte – 8 бит).
wSecondVariable  RES  2     ; Переменная является словом (word - 2 байта).
aThirdVariable   RES  10    ; Переменная является массивом (array - 10 байт)
fFourthVariable  EQU  7     ; Переменная является флагом (flag - 1 бит)
```

Ниже приведен пример исходного кода небольшой программы для микроконтроллера PIC16F84 и Ассемблера Microchip MPASM.

```
; ***********************************************
; Пример программы для микроконтроллера PIC16F84
; ***********************************************
        LIST  P = 16F84, R = DEC
        #INCLUDE <p16f84.inc>
; ***** Вектора
        ORG  0x00                   ; Вектор сброса
        goto   Main                 ; Переход на начало

        ORG  0x04                   ; Вектор прерывания
        goto   Int                  ; Переход на обработчик

; ***** Переменные
Counter      EQU  0x0C              ; Счетчик

; ***** Обработчик прерываний
Int
        retfie                      ; Возврат из прерывания

; ***** Программа
Main
        clrf    Counter             ; Инициализация Counter

Loop
        movlw       99                  ; Сравнение с 99
        subwf       Counter, W

        btfsc       STATUS, Z           ; Равны ?
        goto        Label               ; Да.
```

```
        incf    Counter, F          ; Инкрементация Counter
        goto    Loop
Label   clrf    Counter             ; Инициализация Counter
        goto    Loop

        END
```

4.2 Числовые константы, операторы и выражения

Ассемблер Microchip MPASM предоставляет ряд операторов, которые могут использоваться в исходном коде для составления выражений. Выражения вычисляются Ассемблером во время ассемблирования. Поэтому в выражениях допускаются только те операнды, значения которых известны к моменту ассемблирования или могут быть вычислены при ассемблировании. Примером таких операндов являются числовые константы.

4.2.1 Числовые константы

Числовые константы могут быть записаны в нескольких системах счисления: шестнадцатеричной, десятичной, восьмеричной, двоичной и символьной (ASCII). По умолчанию действует шестнадцатеричная система счисления. Систему счисления по умолчанию можно изменить через специальные директивы (RADIX, LIST) или опции командной строки (r). Система счисления числовой константы может быть указана явно с использованием префиксов H, D, O, B, A.

Например:

H’4B’ – шестнадцатеричная

D’75’ – десятичная

O’113’ – восьмеричная

B’1001011’ – двоичная

A’K’ - символьная

Кроме того, можно использовать и три дополнительных префикса – 0x для шестнадцатеричной записи, . для десятичной и ‘’ для символьной.

Например:

0x4B – шестнадцатеричная

.75 – десятичная

‘K’ – символьная

Перед числовой константой могут стоять знаки + и -.

4.2.2 Операторы

Среди операторов Ассемблера Microchip MPASM можно выделить шесть основных групп:

- арифметические операторы,
- побитовые операторы,
- логические операторы,
- операторы отношения,
- операторы присваивания,
- другие операторы.

Каждый оператор характеризуется своим именем, выполняемой операцией, числом и типом операндов, приоритетом, ассоциативностью, коммутативностью и другими свойствами.

Операторы Ассемблера Microchip MPASM, сгруппированные по их приоритетам, приведены в таблице 1.

Таблица 1

Имя	Описание	Пример
1	2	3
$	Текущее значение программного счетчика	$ - 2
()	Скобки (изменение приоритета)	(1 + 2) * 3
!	Логическое отрицание	! (4 == 2)
-	Унарный минус (изменение знака)	- 1
+	Унарный плюс	+ 1
~	Побитовое отрицание (инверсия)	~ B’100010’
HIGH	Старший байт	HIGH(0x1234)
LOW	Младший байт	LOW(0x1234)
*	Умножение	2 * 4
/	Деление	9 / 2
%	Взятие по модулю	3 % 2
+	Сложение	2 + 4
-	Вычитание	2 – 4
<<	Сдвиг влево	B’00000001’ << 2
>>	Сдвиг вправо	B’00000100’ >> 2
>=	Больше или равно	4 >= 4
>	Больше	4 > 2
<=	Меньше или равно	4 <= 4
<	Меньше	4 < 8
==	Проверка на равенство	4 == 8
!=	Проверка на неравенство	4 != 8
&	Побитовое И	B’00110101’ & 0x01
|	Побитовое ИЛИ	B’00110100’ | 0x01
^	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ	B’00110101’ ^ 0x0F
&&	Логическое И	4 == 4 && 5 < 10
||	Логическое ИЛИ	4 == 4 || 5 > 10
=	Присваивание	I = 2
*=	Умножение с присваиванием	I *= 5
/=	Деление с присваиванием	I /= 5

Окончание таблицы 1

1	2	3
%=	Взятие по модулю с присваиванием	I %= 5
+=	Сложение с присваиванием	I += 5
-=	Вычитание с присваиванием	I -= 5
<<=	Сдвиг влево с присваиванием	I <<= 2
>>=	Сдвиг вправо с присваиванием	I >>= 2
&=	Побитовое И с присваиванием	I &= 0x0F
|=	Побитовое ИЛИ с присваиванием	I |= 0x0F
^=	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ с присваиванием	I ^= 0x0F
++	Инкремент на единицу	I ++
--	Декремент на единицу	I --

Большинство операторов Ассемблера Microchip MPASM заимствовано из языка С.

Оператор $ возвращает адрес первой следующей за ним команды микроконтроллера.

Разница между побитовыми операторами (~, &, |, ^) и логическими операторами (!, &&, ||) заключается в том, что побитовые операторы выполняют логическую операцию над каждой парой бит своих операндов, рассматривая их как массивы бит, а логические операторы выполняют логическую операцию, рассматривая свои операнды как логические: если операнд равен 0, то он представляет собой логическое значение «ложь» (false), если же он отличен от 0, то он представляет собой логическое значение «истина» (true). Результатом логических операторов, так же как и операторов отношения (==, !=, >, >=, <, <=), являются логические константы false или true.

Исходя из этого, следующие два выражения дают различные результаты:

0x0F & 0xF0 равно 0x00, 0x0F && 0xF0 равно 0x01.

Желательно в качестве операндов логических операторов использовать результаты либо операторов отношения, либо логических операторов.

Сложные операторы присваивания (*=, /= и др.) выполняют соответствующую операцию с левым и правым операндами и записывают

результат в левый операнд. С точки зрения программиста выражение I *= 5 полностью эквивалентно выражению I = I * 5, однако первое выражение выполняется Ассемблером быстрее, чем второе.

Одной из самых распространенных ошибок является использование оператора = при сравнении вместо оператора ==.

4.2.3 Выражения

При составлении выражений следует иметь в виду, что для их вычисления Ассемблер Microchip MPASM использует 32-битовую знаковую целочисленную арифметику. Так, например, оператор деления (/) выполняет целочисленное деление своих операндов.

Особое внимание следует уделять приоритетам входящих в него операторов или же использовать скобки для явного задания приоритетов.

4.3 Директивы

Директивы представляют собой команды Ассемблера и так же, как и команды микроконтроллера, включаются в файл исходного кода. Директивы имеют много общего с командами микроконтроллера, однако между ними существует и принципиальное различие: директивы выполняются Ассемблером во время ассемблирования, а не микроконтроллером во время выполнения программы. В основном они используются для управления процессом ассемблирования и не транслируются прямо, как команды микроконтроллера, в выполнимый или объектный код.

Ассемблер Microchip MPASM обладает широким набором различных директив. Часть из них может использоваться по желанию для повышения продуктивности разработки, гибкости и переносимости исходного кода, а часть является неизменным атрибутом любой, даже самой простой программы. Все

директивы Ассемблера Microchip MPASM могут быть разделены на 5 основных групп:

- директивы управления,
- директивы условного ассемблирования,
- директивы данных,
- директивы макрокоманд,
- директивы объектных файлов.

Краткое описание директив Ассемблера Microchip MPASM приведено в таблице 2.

Таблица 2

Имя	Описание	Синтаксис
1	2	3
Директивы управления		
CONSTANT	Определить константу	CONSTANT <label> [= <expr>,...,<label> [= <expr>]]
#DEFINE	Определить текстовую подстановку	#DEFINE <name> [[(<arg>,...,<arg>)]<value>]
END	Завершить исходный код	END
EQU	Определить константу	<label> EQU <expr>
ERROR	Создать сообщение об ошибке	ERROR "<text_string>"
ERRORLEVEL	Определить тип отображаемых сообщений	ERRORLEVEL 0|1|2|<+-><msg>
#INCLUDE	Включить дополнительный файл исходного кода	[#]INCLUDE <<include_file>> [#]INCLUDE "<include_file>"
LIST	Разрешить вывод в файл листинга. Определить параметры файла листинга	LIST [<option>[,...,<option>]]
MESSG	Создать сообщение	MESSG "<message_text>"
NOLIST	Запретить вывод в файл листинга	NOLIST
ORG	Определить начальный адрес	<label> ORG <expr>
PAGE	Начать новую страницу в файле листинга	PAGE
PROCESSOR	Определить модель микроконтроллера	PROCESSOR <processsor_type>

Продолжение таблицы 2

1	2	3
RADIX	Определить систему счисления по умолчанию	RADIX <default_radix>
SET	Определить переменную	<label> SET <expr>
SPACE	Вставить пустые строки в файл листинга	SPACE [<expr>]
SUBTITLE	Определить подзаголовок	SUBTITLE "<sub_text>"
TITLE	Определить заголовок	TITLE "<title_text>"
#UNDEFINE	Отменить текстовую подстановку	#UNDERFINE <label>
VARIABLE	Определить переменную	VARIABLE <label> [= <expr>,..., <label> [= <expr>]]
Директивы условного ассемблирования		
ELSE	Начать альтернативный блок	ELSE
ENDIF	Завершить условный блок	ENDIF
ENDW	Завершить условный цикл	ENDW
IF	Начать условный блок	IF <expr>
IFDEF	Проверить определение идентификатора	IFDEF <label>
IFNDEF	Проверить определение идентификатора	IFNDEF <label>
WHILE	Начать условный цикл	WHILE <expr>
Директивы данных		
__BADRAM	Определить недействительные ячейки памяти данных	__BADRAM <expr>
CBLOCK	Начать блок констант	CBLOCK [<expr>]
__CONFIG	Определить слово конфигурации	__CONFIG <expr> __CONFIG <addr>, <expr>
DA	Определить строку в памяти программ	[<label>] DA <expr> [, <expr2>, ...,<exprn>]
DATA	Определить числовые и текстовые данные в памяти программ	DATA <expr> [,<expr>,...,<expr>] DATA "<text_string>" [,"<text_string>",...]
DB	Определить байт в памяти программ	DB <expr>[,<expr>,...,<expr>]
DE	Определить данные в энергонезависимой памяти	DE <expr>[,<expr>,...,<expr>]
DT	Определить таблицу в памяти программ	DT <expr>[,<expr>,...,<expr>]

Продолжение таблицы 2

1	2	3
DW	Определить слово в памяти программ	DW <expr> [,<expr>,...,<expr>]
ENDC	Завершить блок констант	ENDC
FILL	Заполнить память программ	FILL <expr>, <count>
__IDLOCS	Определить ID ячейки	__IDLOCS <expr>
__MAXRAM	Определить максимально допустимый адрес в памяти программ	__MAXRAM <expr>
RES	Определить данные в памяти данных	RES <mem_units>
Директивы макрокоманд		
ENDM	Закончить определение макрокоманды	ENDM
EXITM	Выйти из макрокоманды	EXITM
EXPAND	Раскрывать макрокоманды в файле листинга	EXPAND
LOCAL	Объявить идентификатор локальным	LOCAL <label> [,<label>]
MACRO	Начать определение макрокоманды	<label> MACRO [<arg>,...,<arg>]
NOEXPAND	Не раскрывать макрокоманды в файле листинга	NOEXPAND
Директивы объектных файлов		
BANKISEL	Генерировать код установки банка для косвенной адресации	BANKISEL <label>
BANKSEL	Генерировать код установки банка	BANKSEL <label>
CODE	Начать секцию выполнимого кода	[<name>] code [<address>]
EXTERN	Объявить внешний идентификатор	EXTERN <label> [,<label>]
GLOBAL	Экспортировать идентификатор	GLOBAL <label> [.<label>]
IDATA	Начать секцию инициализированных данных	[<name>] IDATA [<address>]
PAGESEL	Генерировать код установки страницы	PAGESEL <label>

Окончание таблицы 2

1	2	3
UDATA	Начать секцию неинициализированных данных	[<name>] UDATA [<address>]
UDATA_ACS	Начать секцию доступа к неинициализированным данным	[<name>] UDATA_ACS [<address>]
UDATA_OVR	Начать секцию перекрывающихся неинициализированных данных	[<name>] UDATA_OVR [<address>]
UDATA_SHR	Начать секцию разделяемых неинициализированных данных	[<name>] UDATA_SHR [<address>]

4.3.1 Директивы управления

CONSTANT – Определить константу

Синтаксис: CONSTANT <label> = <expr> […, <label> = <expr>]

Описание: Присваивает идентификатору <label> значение выражения <expr>. Впоследствии изменить значение идентификатора невозможно.

Пример:

CONSTANT Const1 = 0x06, Const2 = 0x0A

CONSTANT Const3 = 0x10

Смотри также: EQU, VARIABLE, SET

#DEFINE – Определить текстовую подстановку

Синтаксис: #DEFINE <label> [<string>]

Описание: Определяет текстовую подстановку. При ассемблировании исходного кода все вхождения строки <label> будут заменены Ассемблером на строку <string>. Использование #DEFINE приводит к определению идентификатора <label>, которое может быть в дальнейшем проверено с помощью директив IFDEF и IFNDEF. Идентификаторы, созданные с помощью #DEFINE, недоступны для просмотра в Microchip MPLAB.

Директива #DEFINE заимствована из препроцессора языка C.

Пример:

```
#DEFINE     Flag
#DEFINE     Const           0x10
#DEFINE     Bit             STATUS, C
#DEFINE     Formula(x, y, z)    ((y) – (2 * ((z) + (x)))
```

Смотри также: #UNDEFINE, IFDEF, IFNDEF

END – Завершить исходный код

Синтаксис: END

Описание: Определяет конец исходного кода. Символы, следующие за директивой END, игнорируются Ассемблером.

EQU – Определить константу

Синтаксис: <label> EQU <expr>

Описание: Присваивает идентификатору <label> значение выражения <expr>. Впоследствии изменить значение идентификатора невозможно. Директива EQU полностью эквивалентна директиве CONSTANT за исключением того, что позволяет определить только одну константу.

Пример:

```
Const   EQU  0x0F
```

Смотри также: SET, CONSTANT, VARIABLE

ERROR – Создать сообщение об ошибке

Синтаксис: ERROR “<text_string>”

Описание: Директива ERROR позволяет создавать собственные сообщения об ошибках, идентичные сообщениям Microchip MPASM.

Текст <text_string> не должен превышать 80 символов.

Пример:

IF Const > .99

ERROR “Недопустимо большое значение Const !”

ENDIF

Смотри также: MESSG

***ERRORLEVEL** – Определить тип отображаемых сообщений*

Синтаксис: ERRORLEVEL {0|1|2|+<msgnum>|-<msgnum>} [, …]

Описание: Определяет тип сообщений, выводимых в файл ошибок и файл листинга (таблица 3). Аргументы 0, 1 имеют больший приоритет, чем аргументы -<msgnum> и +<msgnum>.

Таблица 3

Аргумент	Описание
0	Выводить сообщения, предупреждения и ошибки
1	Выводить предупреждения и ошибки
2	Выводить ошибки
-<msgnum>	Запретить вывод сообщения <msgnum>
+<msgnum>	Разрешить вывод сообщения <msgnum>

Пример:

ERRORLEVEL 1, -202

Смотри также: LIST

***#INCLUDE** – Включить дополнительный файл исходного кода*

Синтаксис: [#]INCLUDE <<include_file>>

[#]INCLUDE “<include_file>”

Описание: Включает дополнительный файл исходного кода <include_file> в текущий файл исходного кода. Использование директивы #INCLUDE эквивалентно вставке содержимого файла <include_file> начиная со строки, в которой расположена директива #INCLUDE. Допускается до 6 уровней вложенности директивы #INCLUDE. Имя включаемого файла может быть

окружено как двойными кавычками, так и угловыми скобками. <include_file> может включать полный путь доступа к включаемому файлу. В случае если путь не указан, Ассемблер Microchip MPASM осуществляет поиск включаемого файла в указанной последовательности в следующих каталогах:

- текущий каталог,
- каталог файлов исходного кода,
- каталог Ассемблера.

Директива #INCLUDE заимствована из препроцессора языка С.

Пример:

#INCLUDE <p16f84.inc>

#INCLUDE “c:\mpasm\p16f84.inc”

INCLUDE <library.inc>

***LIST** – Разрешить вывод в файл листинга. Определить параметры файла листинга*

Синтаксис: LIST [<list_option>, …, <list_option>]

Описание: В случае использования без аргументов директива разрешает вывод в файл листинга, если он был запрещен. При указании аргументов директива определяет некоторые параметры файла листинга и самого процесса ассемблирования. Список аргументов приведен в таблице 4.

Таблица 4

Аргумент	По умол-чанию	Описание
1	2	3
b = <number>	8	Установить длину символа табуляции, равную <number>
c = <number>	132	Установить длину строки, равную <number>
f = <format>	INHX8M	Установить формат выполнимого файла <format> Допустимые значения <format>: {INHX8M|INHX8S| INHX32}
free		Для совместимости с предыдущими версиями
fixed		Для совместимости с предыдущими версиями
mm = {on|off}	on	Создавать или нет карту распределения памяти
n = <number>	60	Установить число строк в странице

Окончание таблицы 4

1	2	3
p = <processor>		Установить модель микроконтроллера <processor>.
r = <radix>	hex	Установить систему счисления по умолчанию <radix>. Допустимые значения <radix>: {hex|dec|oct}
st = {on|off}	on	Создавать или нет символьную таблицу
t = {on|off}	off	Обрезать или переносить длинные строки
w = <level>	0	Установить тип выводимых сообщений <level>. Допустимые значения <level>: {0|1|2}. Смотри также ERRORLEVEL.
x = {on|off}	on	Раскрывать или нет макрокоманды

Пример:

```
LIST      p = 16F84, f = INHX8S, r = dec
LIST
```

Смотри также: ERRORLEVEL, EXPAND, NOEXPAND, NOLIST, PROCESSOR, RADIX

***MESSG** – Создать сообщение*

Синтаксис: MESSG “<text_string>”

Описание: Директива MESSG позволяет создавать собственные сообщения, идентичные сообщениям Microchip MPASM.

Текст <text_string> не должен превышать 80 символов.

Пример:

```
IF        Const > .99
MESSG   “Большое значение Const”
ENDIF
```

Смотри также: ERROR

***NOLIST** – Запретить вывод в файл листинга*

Синтаксис: NOLIST

Описание: Запрещает вывод в файл листинга, если он был разрешен.

Смотри также: LIST

ORG – Определить начальный адрес

Синтаксис: [<label>] ORG <expr>

Описание: Устанавливает начальный адрес в памяти программ для всех следующих команд микроконтроллера. Если используется идентификатор <label>, то ему будет присвоено значение <expr>.

Для микроконтроллеров семейства PIC18CXX допустимы только четные значения <expr>.

Недопустимо использовать директиву ORG при генерации файла объектного кода.

Пример:

```
        ORG  0x00
        goto  Main
IntVector ORG  0x04
        goto  Int
```

Смотри также: FILL, RES

PAGE – Начать новую страницу в файле листинга

Синтаксис: PAGE

Описание: Начинает новую страницу в файле листинга

Смотри также: LIST, SUBTITLE, TITLE

PROCESSOR – Определить модель микроконтроллера

Синтаксис: PROCESSOR <processor_type>

Описание: Устанавливает модель микроконтроллера <processor_type>

Пример:

```
PROCESSOR  16F84
```

Смотри также: LIST

RADIX – Определить систему счисления по умолчанию

Синтаксис: RADIX <default_radix>

Описание: Устанавливает систему счисления по умолчанию.

Допустимые значения <default_radix>: hex, dec, oct

Пример:

RADIX dec

Смотри также: LIST

SET – Определить переменную

Синтаксис: <label> SET <expr>

Описание: Присваивает идентификатору <label> значение выражения <expr>. Директива SET эквивалентна директиве EQU, за исключением того, что значение идентификатора <label> впоследствии может быть изменено.

Пример:

Variable SET 0x0A

Variable += 0x06

Смотри также: EQU, VARIABLE, CONSTANT

SPACE – Вставить пустые строки в файл листинга

Синтаксис: SPACE <expr>

Описание: Вставляет <expr> пустых строк в файл листинга.

Пример:

SPACE 3

Смотри также: LIST

SUBTITLE – Определить подзаголовок

Синтаксис: SUBTITLE “<sub_text>”

Описание: Устанавливает подзаголовок, используемый при создании файла листинга.

Текст <sub_text> не должен превышать 60 символов.

Пример:

SUBTITLE “A Subtitle”

Смотри также: TITLE

TITLE – Определить заголовок

Синтаксис: TITLE “<title_text>”

Описание: Устанавливает заголовок, используемый при создании файла листинга и отображаемый в первой строке каждой страницы.

Текст <title_text> не должен превышать 60 символов.

Пример:

TITLE “A Title”

Смотри также: SUBTITLE

#UNDEFINE – Отменить текстовую подстановку

Синтаксис: #UNDEFINE <label>

Описание: Отменяет текстовую подстановку <label>, определенную ранее с помощью директивы #DEFINE. Наличие текстовой подстановки может быть проверено с помощью директив IFDEF и IFNDEF.

Директива #UNDEFINE заимствована из препроцессора языка C.

Пример:

#UNDEFINE Flag

Смотри также: #DEFINE

VARIABLE – Определить переменную

Синтаксис: VARIABLE <label> = [<expr>] [, <label> = [<expr>] …]

Описание: Присваивает идентификатору <label> значение выражения <expr>. Директива VARIABLE эквивалентна директиве SET за исключением

того, что она позволяет определить сразу несколько переменных и не требует их обязательной инициализации.

Пример:

```
VARIABLE      Variable1 = 0x0A, Variable2 = 0x06
VARIABLE      Variable3
Variable3 = Variable1 + Variable2
```

Смотри также: CONSTANT, SET, EQU

4.3.2 Директивы условного ассемблирования

***ELSE** – Начало альтернативного блока программы условия IF*

Синтаксис: ELSE

Описание: Используется совместно с директивой IF для обеспечения альтернативного хода выполнения программы, соответствующего ложному выполнению условия. Директива ELSE может быть использована внутри регулярного блока программы или макроса.

Пример:

```
IF rate < 50
    DW slow
ELSE
    DW fast
ENDIF
```

Смотри также: ENDIF, IF

***ENDIF** – Окончание условного блока программы*

Синтаксис: ENDIF

Описание: Указывает окончание условного блока. Директива ENDIF может быть использована внутри регулярного блока программы или макроса.

Смотри также: ELSE, IF

ENDW – Завершает цикл While

Синтаксис: ENDW

Описание: Завершает цикл WHILE. Пока условие, указанное в директиве WHILE, остается истинным, программа будет выполняться между директивами WHILE и ENDW. Директива ENDW может быть использована внутри регулярного блока программы или макроса.

Пример:

Смотрите пример в описании директивы WHILE

Смотри также: WHILE

IF – Начало блока условия

Синтаксис: IF

Описание: Начало выполнения условного блока. Если выражение <expr> оценивается как истинное, то выполняется код программы после директивы IF. Иначе последующий текст программы игнорируется, пока не встретится директива ELSE или ENDIF.

Выражение, которое имеет значение нуль, рассматривается как логическая ЛОЖЬ. Выражение, имеющее любое другое значение, рассматривается как логическая ИСТИНА. Директивы IF и WHILE работают с логическим значением выражения. Логическая ИСТИНА гарантирует ненулевой результат выражения, а логическая ЛОЖЬ нулевой результат.

Пример:

```
IF version == 100
    movlw 0x0a
    movwf io_1
ELSE
    movlw 0x01a
    movwf io_2
```

ENDIF

Смотри также: ELSE, ENDIF

IFDEF – Выполнение, если определена символьная метка

Синтаксис: IFDEF

Описание: Если <label> была предварительно определена (#DEFINE), то выполняется текст программы, идущий непосредственно за директивой IFDEF. В противном случае последующий текст программы пропускается, пока не встретится директива ELSE или ENDIF.

Пример:

```
#DEFINE testing 1          ; установить testing “on”
:
:
IFDEF testing
<execute test code>        ; выполняемый код
ENDIF
```

Смотри также: #DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

IFNDEF – Выполнение, если символьная метка не определена

Синтаксис: IFNDEF

Описание: Если <label> не была предварительно определена или определение метки было отменено #UNDEFINE, то выполняется текст программы, идущий непосредственно за директивой IFNDEF. В противном случае последующий текст программы пропускается, пока не встретится директива ELSE или ENDIF.

Пример:

```
#DEFINE testing1           ; установить testing “on”
:
:
```

```
#UNDEFINE testing1          ; установить testing “off”
IFNDEF testing
:                           ; выполняемый код
:
ENDIF
END
```

Смотри также: #DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

WHILE – Цикл While

Синтаксис: WHILE

Описание: Выполняется программа между директивами WHILE и ENDW, пока значение <expr> истинно. Значение <expr>, равное нулю, рассматривается как ЛОЖЬ. Любое другое значение <expr> рассматривается как ИСТИНА. Логическая ИСТИНА гарантирует ненулевой результат выражения, а логическая ЛОЖЬ нулевой результат. Длина цикла не может быть более 100 строк программы. Максимальное число повторов программы внутри цикла 256.

Пример:

```
test_mac MACRO count
        VARIABLE i
        i = 0
        WHILE i < count
        movlw i
        i += 1
        ENDW
        ENDM
start
        test_mac 5
        END
```

Смотри также: ENDW, IF

4.3.3 Директивы данных

***__BADRAM** - Идентификация нереализованного ОЗУ*

Синтаксис: __BADRAM <expr>[-<expr>][, <expr>[-<expr>]]

Описание: Директивы __MAXRAM и __BADRAM определяют адреса нереализованных регистров ОЗУ. __BADRAM определяет индивидуальный адрес нереализованного регистра. Данная директива предназначена для использования совместно с директивой _MAXRAM. Каждое значение <expr> директивы __BADRAM должно быть меньше указанного в __MAXRAM. После директивы __MAXRAM в тексте программы директивами __BADRAM создается точная карта нереализованного ОЗУ.

Для указания диапазона адресов нереализованного ОЗУ используйте синтаксис <minloc> - <maxloc>.

Пример:

См. пример для __MAXRAM

Смотри также: __MAXRAM

***CBLOCK** – Определение блока констант*

Синтаксис: CBLOCK [<expr>]

<label>[:<increment>][,<label>[:<increment>]]

ENDC

Описание: Определить список именованных констант. Каждая именованная константа <label> имеет некоторое значение, описанное выше по тексту программы. Цель данной директивы состоит в том, чтобы указать адреса размещения нескольких констант. Список именованных констант заканчивается директивой ENDC.

<expr> - указывает стартовый адрес для первой константы. Если адрес не указан, то используется заключительное значение предыдущего CBLOCK. Если первый CBLOCK не имеет никакого значения <expr>, то размещение начинается с нулевого адреса.

<increment> - указывает приращения адреса для текущей именованной константы.

Именованные константы в одной строке разделяются запятыми.

Директива CBLOCK используется для размещения констант в памяти программ и памяти данных.

Пример:

```
    CBLOCK 0x20                 ; name_1 будет иметь адрес 20
        name_1, name_2    ; name_2 - 21
        name_3, name_4    ; name_4 - 23
    ENDC
    CBLOCK 0x30
        TwoByteVar: 0, TwoByteHigh, TwoByteLow
        Queue: QUEUE_SIZE
        QueueHead, QueueTail
        Double1:2, Double2:2
    ENDC
```

Смотри также: ENDC

__CONFIG – Установка битов конфигурации микроконтроллера

Синтаксис: __CONFIG <expr> OR __CONFIG <addr>, <expr>

Описание: Устанавливает биты конфигурации микроконтроллера в соответствии со значением <expr>. Для микроконтроллеров семейства PIC18CXXX дополнительно указывается адрес <addr> размещения конфигурационных бит. Подробное описание конфигурационных битов

смотрите в технической документации на соответствующий микроконтроллер.

Предварительно, перед директивой __CONFIG, надо указать тип микроконтроллера с помощью директивы LIST или PROCESSOR. Для микроконтроллеров семейства PIC17CXXX в директиве LIST необходимо указать выходной формат HEX файла INHX32.

Пример:

```
    LIST p=17c42, f = INHX32
__CONFIG H’FFFF’          ; по умолчанию
```

Смотри также: __IDLOCS, LIST, PROCESSOR

DA – Сохранение строки в памяти программ

Синтаксис: [<label>] DA <expr> [, <expr2>, ..., <exprn>]

Описание: Упаковывает в 14-битовый формат два 7-битовых символа ASCII. Используется для сохранения символьной строки в FLASH памяти программ микроконтроллера.

Пример:

DA "abcdef"

В памяти программ - 30E2 31E4 32E6 3380

DA "12345678" ,0

В памяти программ - 18B2 19B4 1AB6 0000

DA 0xFFFF

В памяти программ - 0x3FFF

DATA – Сохранение значений или текста в памяти программ

Синтаксис: [<label>] DATA <expr>,[,<expr>,...,<expr>]

[<label>] DATA "<text_string>"[,"<text_string>",...]

Описание: Инициализирует одно или более слов памяти программ. Данные могут быть в виде констант, внутренних меток или их выражений.

Данные также могут состоять из цепочки (одного) символов ASCII <text_string>. Один символ сохраняется в младшем байте памяти программ , в случае сохранения нескольких символов они упаковываются в слова по два знака. Если сохраняется нечетное число символов, то заключительный байт равен нулю. Во всех семействах микроконтроллеров, кроме PIC18CXXX, первый символ сохраняется в старшем байте слова. Для PIC 8CXXX первый символ сохраняется в младшем байте слова.

Эта директива может использоваться при генерации объектного файла. Дополнительную информацию смотрите в описании директивы IDATA.

Пример:

DATA reloc_label+10

DATA 1,2,ext_label

DATA "testing 1,2,3"

DATA ‘N’

DATA start_of_program

Смотри также: DB, DE, DT, DW, IDATA

***DB** – Побайтовое сохранение данных в памяти программ*

Синтаксис: [<label>] DB <expr>[,<expr>,...,<expr>]

Описание: Резервирует слово в памяти программ с сохранением 8-битового значения. Многозначные выражения последовательно заполняют слова памяти программ. В случае нечетного числа значений последний байт будет равен нулю.

Эта директива может использоваться при генерации объектного файла. Дополнительную информацию смотрите в описании директивы IDATA.

Пример:

DB ’t’, 0x0f, ’e’, 0x0f, ’s’, 0x0f, ’t’, ’\n’

Смотри также: DATA, DE, DT, DW, IDATA

DE – Резервирует 8-разрядное значение в EEPROM памяти

Синтаксис: [<label>] DE <expr>[,<expr>,...,<expr>]

Описание: Резервирует слово в EEPROM памяти для сохранения 8-битового значения <expr>. Старшие биты слова равны нулю. Каждое 8-разрядное значение сохраняется в отдельном слове.

Директива была разработана для PIC16F8X, но может быть использована и в других микроконтроллерах.

Пример:

```
ORG H’2100’                ; Инициализация EEPROM
DE "MY PROGRAM, V1.0", 0
```

Смотри также: DATA, DB, DT, DW

DT – Определяет таблицу данных

Синтаксис: [<label>] DT <expr>[,<expr>,...,<expr>]

[<label>] DT "<text_string>"[,"<text_string>",...]

Описание: Генерирует серию команд RETLW для 8-разрядных значений <expr>. Каждое значение <expr> сохраняется в отдельной команде RETLW.

Пример:

```
DT "A Message", 0
DT FirstValue, SecondValue, EndOfValues
```

Смотри также: DATA, DB, DE, DW

DW – Резервирует слова в памяти программ

Синтаксис: [<label>] DW <expr>[,<expr>,...,<expr>]

[<label>] DW "<text_string>"[,"<text_string>",...]

Описание: Резервирует слова в памяти программ для данных, заполняя пустые места определенными значениями. Для микроконтроллеров семейства PIC18CXXX директива DW работает подобно DB. Адрес последнего резервирования в памяти программ запоминается и увеличивается на единицу

при каждом сохранении значений. Выражения могут быть литеральными с сохранением в памяти программ аналогично директиве DATA.

Эта директива может использоваться при генерации объектного файла. Дополнительную информацию смотрите в описании директивы IDATA.

Пример:

```
DW 39, "diagnostic 39", (d_list*2+d_offset)
DW diagbase-1
```

Смотри также: DATA, DB, IDATA

***ENDC** – Окончание автоматического блока констант*

Синтаксис: ENDC

Описание: Используется совместно с директивой CBLOCK. Указывает окончание списка констант.

Смотри также: CBLOCK

***FILL** – Запись значения в память программ*

Синтаксис: [<label>] FILL <expr>, <count>

Описание: Записывает <count> слов программы (или байт для PIC18CXXX) <expr>. Инструкция Ассемблера может быть указана в круглых скобках.

Пример:

```
FILL 0x1009, 5
FILL (GOTO RESET_VECTOR), NEXT_BLOCK-$
```

Смотри также: DATA, DW, ORG

***__MAXRAM** – Определяет максимальный объем ОЗУ*

Синтаксис: __MAXRAM <expr>

Описание: Директивы __MAXRAM и __BADRAM определяют адреса нереализованных регистров ОЗУ. __MAXRAM определяет абсолютный максимальный адрес в ОЗУ и инициализирует карту доступного ОЗУ с

адресами меньше <expr>. Значение <expr> должно быть больше или равняться максимальному адресу банка 0 ОЗУ и меньше 1000h. Данная директива предназначена для использования совместно с директивой __BADRAM. После директивы __MAXRAM в тексте программы точная карта нереализованного ОЗУ создается директивами __BADRAM.

__MAXRAM может использоваться более одного раза в тексте программы, при этом повторно пересматривается максимальный объем ОЗУ и сбрасывается карта нереализованных регистров.

Пример:

```
LIST p=16c622
__MAXRAM H’0BF’
__BADRAM H’07’-H’09’, H’0D’-H’1E’
__BADRAM H’87’-H’89’, H’8D’, H’8F’-H’9E’
movwf H’07’          ; обращение к несуществующей ячейке ОЗУ
movwf H’87’          ; обращение к несуществующей ячейке ОЗУ
                     ; (вызовут сообщения об ошибках)
```

Смотри также: __BADRAM

***RES** – Резервирование памяти*

Синтаксис: [<label>] RES <mem_units>

Описание: Резервирует <mem_units> слов программы от текущего местоположения для хранения данных. В неперемещаемом коде программы <label> указывает адрес в памяти программ. В перемещаемом коде программы (при использовании MPLINK) директива RES может использоваться для резервирования памяти данных.

Для всех микроконтроллеров резервируется слово в памяти программ, кроме микроконтроллеров семейства PIC18, в которых резервируется байт памяти программ.

Пример:

```
Buffer  RES 64
```

Смотри также: FILL, ORG

4.3.4 Директивы макрокоманд

ENDM – Окончание макроса

Синтаксис: ENDM

Описание: Завершает макрос, открытый директивой MACRO.

Пример:

```
make_table MACRO arg1, arg2
        DW arg1, 0
        RES arg2
        ENDM
```

Смотри также: MACRO, EXITM

EXITM – Выход из макроса

Синтаксис: EXITM

Описание: Принудительный выход из макроса во время его выполнения. Эффект аналогичен выполнению директивы ENDM.

Пример:

```
test MACRO filereg
    IF filereg == 1
        EXITM
    ELSE
        ERROR "bad file assignment"
    ENDIF
ENDM
```

Смотри также: ENDM, MACRO

EXPAND – Включение текста макроса в файл листинга программы

Синтаксис: EXPAND

Описание: Разрешает включение в файл листинга программы полного текста макроса. Действие аналогично команде /m MPASM при его запуске из командной строки. Действует до директивы NOEXPAND.

Смотри также: MACRO, NOEXPAND

LOCAL – Объявить локальную переменную макроса

Синтаксис: LOCAL <label>[,<label>]

Описание: Объявляет, что указанные элементы данных должны рассматриваться только внутри макроса. <label> может иметь имя, идентичное другой метке, объявленной в основной программе, при этом ошибки не возникнет. Если макрокоманда вызывается рекурсивно, то при каждом обращении будет создаваться собственная локальная копия.

Пример:

```
<основная программа>
:
:
len  EQU 10            ; глобальная версия
size EQU 20            ; создание и изменение локальных переменных
                       ; на них не влияет
test MACRO size
     LOCAL len, label  ; локальные len и label
len SET size
label RES len
len SET len-20
ENDM                   ; конец
```

Смотри также: ENDM, MACRO

MACRO – Определить макрос

Синтаксис: <label> MACRO [<arg>,...,<arg>]

Описание: Макрос – последовательность инструкций, которые могут быть вставлены в код программы, используя единственный макрозапрос. Перед началом использования макроса его необходимо определить выше по тексту программы.

В теле макроса можно вызывать другой макрос или этот же макрос рекурсивно.

Пример:

```
Read    MACRO device, buffer, count
        movlw device
        movwf ram_20
        movlw buffer
        movwf ram_21
        movlw count
        call sys_21
        ENDM
```

Смотри также: ELSE, ENDIF, ENDM, EXITM, IF, LOCAL

NOEXPAND – Не разворачивать текст макроса

Синтаксис: NOEXPAND

Описание: Директива не разрешает разворачивать текст макроса в файле листинга программы при его вызове.

Смотри также: EXPAND

4.3.5 Директивы объектных файлов

BANKISEL – Выбор банка для косвенной адресации

Синтаксис: BANKISEL <label>

Описание: Используется при генерации объектного файла. Директива дает команду компилятору сгенерировать код настройки банка памяти данных для косвенного обращения к регистру <label>. Только одна метка может быть указана в директиве. Предварительно метка <label> должна быть объявлена и соответствовать назначению директивы.

Линковщик генерирует соответствующий код для выбора банка памяти. Для 14 – разрядных микроконтроллеров выполняется воздействие на бит IRP в регистре STATUS в соответствии с банком размещения регистра. Для 16-разрядных микроконтроллеров генерируется команда MOVLB или MOVLR. Если

пользователь сам выбирает рабочий банк памяти, то никаких дополнительных инструкций в код программы добавлено не будет.

Пример:

```
movlw       Var1
movwf       FSR
BANKISEL    Var1

movwf       INDF
```

Смотри также: BANKSEL, PAGESEL

BANKSEL – Выбор банка для прямой адресации

Синтаксис: BANKSEL <label>

Описание: Используется при генерации объектного файла. Директива дает команду компилятору сгенерировать код настройки банка памяти данных для прямого обращения к регистру <label>. Только одна метка может быть указана в директиве. Предварительно метка <label> должна быть объявлена и соответствовать назначению директивы.

Линковщик генерирует соответствующий код для выбора банка памяти. Для 12-разрядных микроконтроллеров устанавливает/сбрасывает бит в регистре FSR. Для 14-разрядных микроконтроллеров - изменяются биты в регистре STATUS. Для 16-разрядных микроконтроллеров генерируются команда MOVLB или MOVLR. Для усовершенствованных 16-разрядных микроконтроллеров будет сгенерирована команда MOVLB. Если в микроконтроллере только один банк памяти, никакой дополнительный код генерироваться не будет.

Пример:

```
BANKSEL     Var1
movwf       Var1
```

Смотри также: BANKISEL, PAGESEL

CODE – Начало кода объектного файла в памяти программ

Синтаксис: [<label>] CODE [<ROM address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции кода программы. Если <label> не указана, секции присваивается имя .code. Если не указан адрес секции, то ей будет присвоено текущее значение адреса в памяти программ.

Примечание. Не допускается использование двух одинаковых имен секций.

Пример:

```
Reset     CODE H'01FF'
          goto  Start
```

Смотри также: EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

***EXTERN** – Определение внешних меток*

Синтаксис: EXTERN <label>[,<label>]

Описание: Используется при генерации объектного файла. Объявляет имена меток, которые могут использоваться в текущем модуле, но определены как глобальные в других модулях. Директива EXTERN должна быть расположена раньше по тексту программы, чем использование <label>. При использовании директивы EXTERN должна быть указана хотя бы одна метка. Если метка определена в текущем модуле программы, то возникает двойная ошибка метки.

Пример:

```
EXTERN Function
...
call Function
```

Смотри также: GLOBAL, IDATA, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

GLOBAL – Внешняя метка

Синтаксис: GLOBAL <label>[,<label>]

Описание: Используется при генерации объектного файла. Объявляет имена меток, которые определены в текущем модуле программы и должны быть доступны другим модулям. Директива GLOBAL должна быть указана после описания соответствующей метки. По крайне мере одна метка должна быть описана в директиве GLOBAL.

Пример:

```
        UDATA
Var1 RES 1
Var2 RES 1
        GLOBAL Var1, Var2
        CODE
AddThree
        GLOBAL AddThree
        addlw 3
        return
```

Смотри также: EXTERN, IDATA, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

IDATA – Объявляет начало инициализации данных в объектном файле

Синтаксис: [<name>] IDATA [<address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции инициализации данных. Если <label> не определена, секция называется .idata. Если адрес инициализации не определен, то он будет назначен автоматически при связи объектных файлов. Никакой код не генерируется этой директивой. Линковщик формирует таблицу поиска для каждого байта, указанного в idata секции. Пользователь должен включать соответствующий код инициализации данных.

Данная директива не доступна для 12-разрядных микроконтроллеров.

Директивы RES, DB и DW могут использоваться для резервирования места под переменные. RES произведет установку нуля, DB будет

последовательно инициализировать байты ОЗУ. DW – последовательно, пословно инициализирует байты ОЗУ (младший байт /старший байт).

Пример:

```
        IDATA
LimitL  DW 0
LimitH  DW D'300'
Gain    DW D'5'
Flags   DB 0
String  DB 'Hi there!'
```

Смотри также: EXTERN, GLOBAL, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

PAGESEL – Произвести выбор страницы

Синтаксис: PAGESEL <label>

Описание: Используется при генерации объектного файла. Линковщик генерирует команды выбора страницы памяти программ в соответствии с указанной меткой <label>. Только одна метка может быть указана в директиве и она должна соответствовать назначению директивы. Необязательно предварительно объявлять метку.

Для микроконтроллеров с 12-разрядными командами будет изменено значение регистра STATUS. Для микроконтроллеров с 14/16-разрядными – будет изменено значение регистра PCLATH командами MOVLW и MOVWF. Если микроконтроллеры содержат только одну страницу памяти программ, то никакой дополнительный код не будет сгенерирован.

Для микроконтроллеров семейства PIC18CXXX директива не имеет смысла.

Пример:

```
PAGESEL GotoDest
goto GotoDest
....
PAGESEL CallDest
call CallDest
```

Смотри также: BANKISEL, BANKSEL

UDATA – Начало инициализации данных с обычным размещением в памяти (объектного файла)

Синтаксис: [<label>] UDATA [<RAM address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции данных с обычным размещением в памяти данных. Если секция не названа, ей присваивается имя .udata.

Если адрес не определен, то будет назначен текущий адрес инициализации.

Никакой код не генерируется в данной директиве. Директива RES должна использоваться для резервирования места под данные.

Примечание. В исходном файле две секции не могут иметь одно и то же имя.

Пример:

```
        UDATA
Var1    RES 1
Double  RES 2
```

Смотри также: EXTERN, GLOBAL, IDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

UDATA_ACS – Начало инициализации данных быстрого доступа (объектного файла)

Синтаксис: [<label >] UDATA_ACS [<RAM address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции данных быстрого доступа для микроконтроллеров семейства PIC18CXXX. Если секция не названа, ей присваивается имя .udata_acs.

Если адрес не определен, то будет назначен текущий адрес инициализации.

Никакой код не генерируется в данной директиве. Директива RES должна использоваться для резервирования места под данные.

Примечание. В исходном файле две секции не могут иметь одно и то же имя.

Пример:

```
          UDATA_ACS
Var1      RES 1
Double    RES 2
```

Смотри также: EXTERN, GLOBAL, IDATA UDATA, UDATA_OVR, UDATA_SHR

***UDATA_OVR** – Начало инициализации временных данных (объектного файла)*

Синтаксис: [<label >] UDATA_OVR [<RAM address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции временных данных. Если секция не названа, ей присваивается имя .udata_ovr. Место, зарезервированное в данной секции, доступно другим udata_ovr секциям с таким же именем. Это оптимальный метод резервирования памяти под временные переменные.

Если адрес не определен, то будет назначен текущий адрес инициализации.

Никакой код не генерируется в данной директиве. Директива RES должна использоваться для резервирования места под данные.

Примечание. Исключение к правилу, что две секции не могут иметь одно и то же имя в одном исходном файле.

Пример:

```
Temps     UDATA_OVR
Temp1     RES 1
Temp2     RES 1
Temp3     RES 1
Temps     UDATA_ovr
```

```
LongTemp1 RES 2      ; эта переменная имеет тот же адрес, что и Temp, Temp2
LongTemp2 RES 2      ; эта переменная имеет тот же адрес, что и Temp3
```

Смотри также: EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_SHR

UDATA_SHR – Начало инициализации разделяемых данных (объектного файла)

Синтаксис: [<label >] UDATA_SHR [<RAM address>]

Описание: Используется при генерации объектного файла. Объявляет начало секции разделяемых данных, доступных из всех банков памяти. Если секция не названа, ей присваивается имя .udata_shr.

Если адрес не определен, то будет назначен текущий адрес инициализации.

Никакой код не генерируется в данной директиве. Директива RES должна использоваться для резервирования места под данные.

Примечание. В исходном файле две секции не могут иметь одно и то же имя.

Пример:

```
Temps UDATA_SHR
Temp1 RES 1
Temp2 RES 1
Temp3 RES 1
```

Смотри также: EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_OVR

Некоторые директивы, имеющие различные названия и различный синтаксис, выполняют при этом одинаковые действия. Их наличие обусловлено необходимостью обеспечения совместимости с предыдущими версиями Microchip MPASM.

4.4 Команды и псевдокоманды

При описании команд и псевдокоманд микроконтроллеров PIC используются следующие условные обозначения, приведенные в таблице 5.

Таблица 5

Обозначение	Описание
b	Номер бита в пределах 8-битового регистра
d	Направление записи результата (W, F)
f	Адрес регистра
k	Константа или метка
W	Рабочий регистр

4.4.1 Команды

Команды микроконтроллеров семейства PIC16CXXX подразделяются на 4 основные группы:

- команды для работы с байтами,
- команды для работы с битами,
- команды для работы с константами,
- команды передачи управления.

Краткое описание команд микроконтроллеров семейства PIC16CXXX приведено в таблице 6.

Таблица 6

Мнемо-ника	Опе-ранды	Описание	Флаги
1	2	3	4
Команды для работы с байтами			
addwf	f, d	Сложить регистр и W	C, DC, Z
andwf	f, d	Выполнить логическое И с регистром и W	Z
clrf	f	Очистить регистр	Z
crlw		Очистить W	Z

Окончание таблицы 6

1	2	3	4
comf	f, d	Инвертировать регистр	Z
decf	f, d	Декрементировать регистр	Z
decfsz	f, d	Декрементировать регистр и пропустить следующую команду, если 0	
incf	f, d	Инкрементировать регистр	Z
incfsz	f, d	Инкрементировать регистр и пропустить следующую команду, если 0	
iorwf	f, d	Выполнить логическое ИЛИ с регистром и W	Z
movf	f, d	Переместить регистр в W	Z
movwf	f	Переместить W в регистр	
nop		Нет операции	
rlf	f, d	Сдвинуть регистр влево через флаг переноса	C
rrf	f, d	Сдвинуть регистр вправо через флаг переноса	C
subwf	f, d	Вычесть W из регистра	C, DC, Z
swapf	f, d	Поменять тетрады регистра местами	
xorwf	f, d	Выполнить логическое ИСКЛЮЧАЮЩЕЕ ИЛИ с регистром и W	Z
Команды для работы с битами			
bcf	f, b	Сбросить бит регистра	
bsf	f, b	Установить бит регистра	
btfsc	f, b	Пропустить, если сброшен бит в регистре	
btfss	f, b	Пропустить, если установлен бит в регистре	
Команды для работы с константами			
addlw	k	Сложить W и константу	C, DC, Z
andlw	k	Выполнить логическое И с константой и W	Z
iorlw	k	Выполнить логическое ИЛИ с константой и W	Z
movlw	k	Записать константу в W	
sublw	k	Вычесть константу из W	C, DC, Z
xorlw	k	Выполнить логическое ИСКЛЮЧАЮЩЕЕ ИЛИ с константой и W	Z
Команды передачи управления			
call	k	Перейти на подпрограмму	
goto	k	Перейти на метку	
retfie		Вернуться из прерывания	
return		Вернуться из подпрограммы	
retlw	k	Вернуться из подпрограммы и записать константу в W	
clrwdt		Сбросить сторожевой таймер	TO, PD
sleep		Войти в режим пониженного энергопотребления	TO, PD

4.4.2 Псевдокоманды

В дополнение к командам микроконтроллера Ассемблер Microchip MPASM предоставляет еще и ряд псевдокоманд, которые являются удобным сокращением наиболее часто используемых комбинаций нескольких команд микроконтроллера и их операндов. На этапе ассемблирования псевдокоманды заменяются Ассемблером эквивалентными командами микроконтроллера. Описание псевдокоманд для микроконтроллеров семейства PIC16CXXX и их эквивалентов приведено в таблице 7.

Таблица 7

Мнемо-ника	Опе-ранды	Описание	Флаги	Эквивалент
1	2	3	4	5
addcf	f, d	Добавить флаг переноса к регистру	Z	btfsc STATUS, C incf f, d
adddcf	f, d	Добавить флаг десятичного переноса к регистру	Z	btfsc STATUS, DC incf f, d
b	k	Перейти		goto k
bc	k	Перейти, если установлен флаг переноса		btfsc STATUS, C goto k
bdc	k	Перейти, если установлен флаг десятичного переноса		btfsc STATUS, DC goto k
bnc	k	Перейти, если сброшен флаг переноса		btfss STATUS, C goto k
bndc	k	Перейти, если сброшен флаг десятичного переноса		btfss STATUS, DC goto k
bz	k	Перейти, если установлен флаг нулевого результата		btfsc STATUS, Z goto k
bnz	k	Перейти, если сброшен флаг нулевого результата		btfss STATUS, Z goto k
clrc		Сбросить флаг переноса		bcf STATUS, C

Окончание таблицы 7

1	2	3	4	5
clrdc		Сбросить флаг десятичного переноса		bcf STATUS, DC
clrz		Сбросить флаг нулевого результата		bcf STATUS, Z
movfw	f	Записать содержимое регистра в рабочий регистр	Z	movf f, W
negf	f, d	Изменить знак регистра	Z	comf f, F incf f, d
setc		Установить флаг переноса		bsf STATUS, C
setdc		Установить флаг десятичного переноса		bsf STATUS, DC
setz		Установить флаг нулевого результата		bsf STATUS, Z
skpc		Пропустить, если установлен флаг переноса		btfss STATUS, C
skpdc		Пропустить, если установлен флаг десятичного переноса		btfss STATUS, DC
skpnc		Пропустить, если сброшен флаг переноса		btfsc STATUS, C
skpndc		Пропустить, если сброшен флаг десятичного переноса		btfsc STATUS, DC
skpz		Пропустить, если установлен флаг нулевого результата		btfss STATUS, Z
skpnz		Пропустить, если сброшен флаг нулевого результата		btfsc STATUS, Z
subcf	f, d	Вычесть флаг переноса из регистра	Z	btfsc STATUS, C decf f, d
subdcf	f, d	Вычесть флаг десятичного переноса из регистра	Z	btfsc STATUS, DC decf f, d
tstf	f	Сравнить регистр с нулем	Z	movf f, F

4.5 Стандартные включаемые файлы

Ассемблер Microchip MPASM распространяется вместе с набором стандартных включаемых файлов. Отдельный включаемый файл предоставляется для каждой модели микроконтроллера.

Стандартный включаемый файл имеет расширение .inc и содержит определения имен регистров специального назначения, имен битов регистров специального назначения, имен для указания регистра назначения, масок для формирования слова конфигурации и другую полезную информацию.

Стандартный заголовочный файл включается в исходный код программы при помощи директивы #INCLUDE. Однако перед его включением необходимо выбрать конкретную модель микроконтроллера одним из следующих трех способов:

- с помощью директивы PROCESSOR,
- с помощью директивы LIST,
- с помощью опции командной строки *p*.

Благодаря наличию директив LIST и NOLIST в файл листинга вместо содержимого всего стандартного заголовочного файла включается только строка комментария.

```
; P16F84.INC  Standard Header File, Version 2.00  Microchip Technology, Inc.
```

Имена, определяемые в стандартных заголовочных файлах, выбраны так, чтобы максимально соответствовать именам, используемым в техническом описании. Хотя использование стандартных включаемых файлов не является обязательным, Microchip настоятельно рекомендует включать именно их в файлы исходного кода.

К сожалению, кроме стандартных включаемых файлов Microchip не предлагает никакого завершенного пакета библиотек для разработки программ на языке Ассемблер.

4.6 Интерфейс командной строки

Ассемблер Microchip MPASM может быть вызван посредством командной строки с использованием следующего синтаксиса:

mpasm[.exe] [/<option>[+|-|<arg>] ...] <src_name>[.asm]
<option> - имя опции,
<arg> - аргумент опции,
<src_name> - имя файла исходного кода (без расширения).

За один вызов Ассемблер Microchip MPASM способен обработать только один файл исходного кода. Для ассемблирования нескольких файлов исходного кода необходимо вызвать Ассемблер соответствующее число раз.

4.6.1 Опции

Каждая опция командной строки имеет свой диапазон допустимых значений, а также значение по умолчанию.

Описание опций командной строки Ассемблера Microchip MPASM приведено в таблице 8.

Таблица 8

Имя	Аргументы	Описание	По умолч.
1	2	3	4
a	inhx8m inhx8s inhx32	Создавать выполнимый файл в формате IHX8M Создавать выполнимый файл в формате IHX8S Создавать выполнимый файл в формате IHX32	inhx8m

Продолжение таблицы 8

1	2	3	4
c	+ -	Различать строчные и прописные буквы Не различать строчные и прописные буквы	+
d	<variable [=value>]>	Определить переменную времени ассемблирования <variable> и инициализировать ее значением <value>	
e	+ - <err_name>	Создавать файл ошибок с именем <src_name>.err Не создавать файл ошибок Создавать файл ошибок с именем <err_name>	+
l	+ - <lst_name>	Создавать файл листинга с именем <src_name>.lst Не создавать файл листинга Создавать файл листинга с именем <lst_name>	+
m	+ -	Раскрывать макрокоманды в файле листинга Не раскрывать макрокоманды в файле листинга	+
o	+ - <obj_name>	Создавать файл объектного кода с именен <src_name>.o Не создавать файл объектного кода Создавать файл объектного кода с именем <obj_name>	
p	<processor>	Выбрать модель микроконтроллера <processor>. <processor> должен иметь формат вида 16F84.	
q	+ -	Отображать результаты ассемблирования на экране Не отображать результаты ассемблирования на экране	
r	hex dec oct	Установить шестнадцатеричную систему счисления по умолчанию Установить десятичную систему счисления по умолчанию Установить восьмеричную систему счисления по умолчанию	hex
t	<tab_size>	Установить длину символа табуляции в файле листинга, равную <tab_size> <tab_size> должен быть целым положительным числом	8
w	0 1 2	Выводить ошибки, предупреждения и сообщения Выводить только ошибки и предупреждения Выводить только ошибки	0

Окончание таблицы 8

1	2	3	4
x	+ - <xrf_name>	Создавать файл перекрестных ссылок с именем <src_name>.xrf Не создавать файл перекрестных ссылок Создавать файл перекрестных ссылок с именем <xrf_name>	
h		Отобразить справочную информацию	
?		Отобразить справочную информацию	

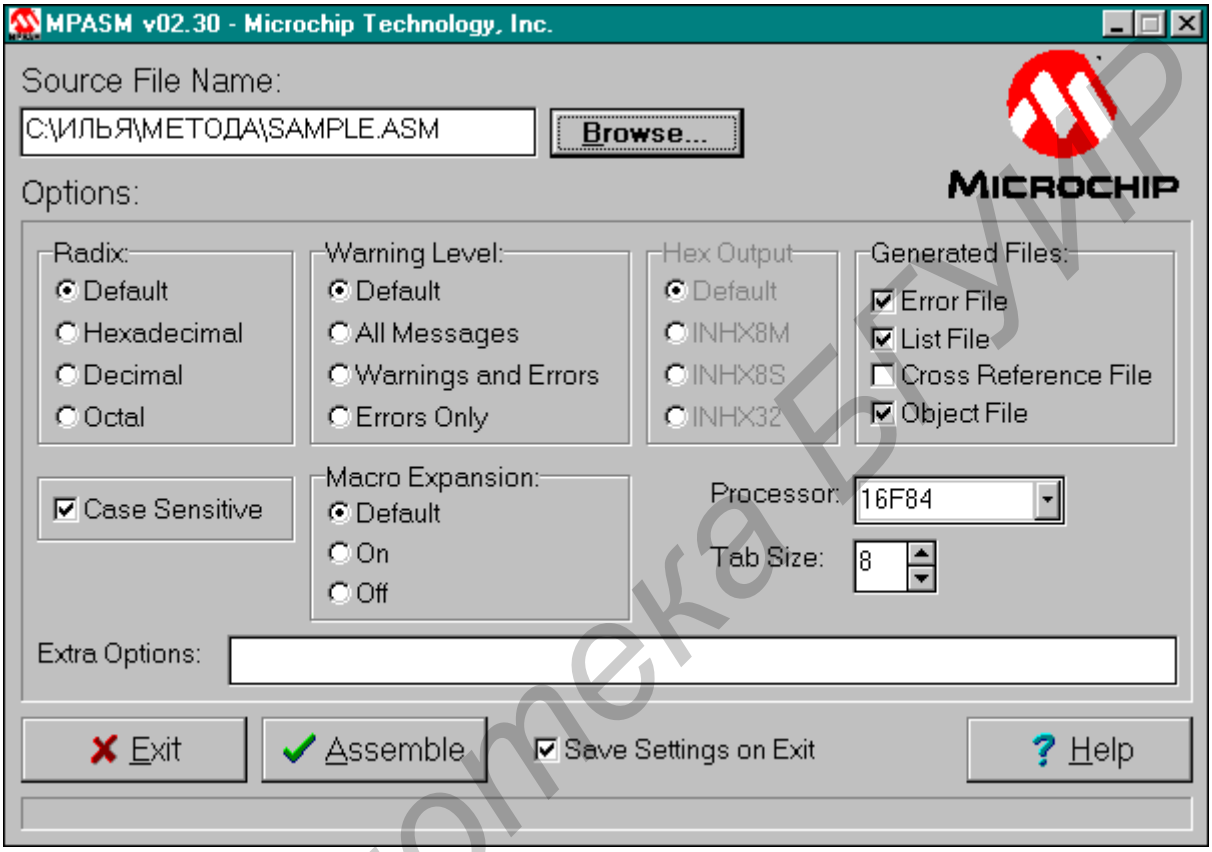
Опция p позволяет выбрать конкретную модель микроконтроллера. Преимуществом использования этой опции является отсутствие необходимости модифицировать файл исходного кода при его ассемблировании для различных моделей микроконтроллеров. Опция d предоставляет возможность определить и при желании инициализировать символическое имя. Использование этой опции эквивалентно использованию в первой строке файла исходного кода директивы #DEFINE. Опция d может использоваться в командной строке несколько раз для определения с возможной инициализацией сразу нескольких символических имен. Достоинством применения опции d является отсутствие необходимости модифицировать файл исходного кода.

Опция w позволяет фильтровать сообщения Ассемблера, выводимые на экран, в файл ошибок и файл листинга по типам.

При использовании опции *o* в командной строке опция *a* игнорируется. При использовании опций h и ? имя файла исходного кода, а также все остальные опции в командной строке игнорируются.

4.7 Диалоговый интерфейс

Ассемблер Microchip MPASM поддерживает также и диалоговый интерфейс с пользователем. Однако описание этого интерфейса не приводится,



т.к. он достаточно прост в освоении и традиционно не поддерживается большинством других Ассемблеров.

4.8 Файл листинга

Файл листинга имеет расширение .lst и содержит подробную информацию о проведенном процессе ассемблирования.

В первой строке каждой страницы файла листинга, генерируемого Ассемблером Microchip MPASM, содержится название и номер версии Ассемблера, имя файла исходного кода, дата и время ассемблирования, а также номер текущей страницы.

Поле LOC содержит относительные адреса, по которым будет расположен код. Поле VALUE содержит 32-битовые значения всех символических имен, определенных с помощью директив SET, EQU, VARIABLE, CONSTANT и CBLOCK. В поле OBJECT CODE отображаются коды машинных команд, которые будут выполняться микроконтроллером. Поле LINE содержит номера соответствующих строк файла исходного кода, а поле SOURCE TEXT – строки файла исходного кода.

Информация об ошибках, предупреждениях и сообщениях встраивается в файл листинга непосредственно перед строками, к которым она относится.

В символьной таблице SYMBOL TABLE перечислены все символические имена, используемые в файле исходного кода. Поле SYMBOL содержит символическое имя, а поле VALUE – его значение.

Карта использования памяти MEMORY USAGE MAP представляет использование памяти программ в графическом виде. Символом ‘X’ обозначаются использованные ячейки, а символом ‘-‘ – неиспользованные ячейки. В случае генерации файла объектного кода карта использования памяти не отображается.

В конце файла листинга приводится общее число отображенных и скрытых ошибок, предупреждений и сообщений.

Ниже приведен пример файла листинга, созданного Ассемблером Microchip MPASM.

```
MPASM 02.30 Released          SAMPLE.ASM   2-4-2001  20:46:40     PAGE  1
LOC    OBJECT CODE  LINE     SOURCE TEXT
  VALUE
                    00001 ; ******************************************
                    00002 ; Пример программы для микроконтроллера PIC16F84
                    00003 ; ******************************************
                    00004
                    00005         LIST    P = 16F84, R = DEC
                    00006         #INCLUDE <p16f84.inc>
```

```
                    00007
                    00008 ; ***** Вектора
                    00009
0000                00010         ORG     0x00        ; Вектор сброса
0000 2806           00011         goto    Main        ; Переход на начало
                    00012
0004                00013         ORG     0x04         ; Вектор прерывания
0004 2805           00014         goto    Int     ; Переход на обработчик
                    00015
                    00016 ; ***** Переменные
                    00017
  0000000C          00018 Counter EQU     0x0C        ; Счетчик
                    00019
                    00020 ; ***** Обработчик прерываний
                    00021
0005                00022 Int
0005 0009           00023         retfie      ; Возврат из прерывания
                    00024
                    00025 ; ***** Программа
                    00026
0006                00027 Main
0006 018C           00028         clrf    Counter   ; Инициализация Counter
                    00029
0007                00030 Loop
0007 3063           00031         movlw   99         ; Сравнение с 99
0008 020C           00032         subwf   Counter, W
                    00033
0009 1903           00034         btfsc   STATUS, Z              ; Равны ?
000A 280D           00035         goto    Label                  ; Да.
                    00036
000B 0A8C           00037         incf    Counter, F  ; Инкрементация Counter
000C 2807           00038         goto    Loop
                    00039
000D 018C           00040 Label   clrf    Counter    ; Инициализация Counter
000E 2807           00041         goto    Loop
                    00042
                    00043         END
```

4.9 Файл ошибок

Файл ошибок имеет расширение .err и содержит информацию об ошибках, предупреждениях и сообщениях сгенерированных Ассемблером на этапе ассемблирования.

Первое поле файла ошибок указывает на тип сообщения: ошибка (Error), предупреждение (Warning) или сообщение (Message). Второй столбец содержит имя и путь файла исходного кода. Третий столбец содержит номер строки файла исходного кода, к которой относится информация. Четвертый столбец содержит название ошибки, предупреждения или сообщения.

Ниже приведен пример файла ошибок, созданного Ассемблером Microchip MPASM.

```
Error[113]   SAMPLE.ASM 34 : Symbol not previously defined (STATU)
```

4.10 Выполнимый файл

Выполнимый файл имеет расширение .hex и содержит выполнимый код программы, пригодный для программирования микроконтроллера.

Ассемблер Microchip MPASM способен генерировать выполнимые файлы в трех различных шестнадцатеричных форматах Intel: INHX8M, INHX8S и INHX32.

Форматы называются шестнадцатеричными, т.к. содержат ASCII символы, которые представляют собой шестнадцатеричные цифры, отражающие реальные данные.

4.10.1 Шестнадцатеричный формат INHX8M

Формат INHX8M предназначен для записи массива 8-битовых чисел. Однако Ассемблер Microchip MPASM использует его для записи 12-, 14- и 16-битовых значений, в результате чего происходит удвоение реальных адресов.

Выполнимые файлы формата INHX8M используются большинством стандартных программаторов, таких, как PRO MATE 2, PICSTART, PICPro и др.

Каждая строка выполнимого файла начинается с символа двоеточия (:) и имеет следующий формат:

:BBAAAATTHHHH…HHHHCC

BB	поле из 2 цифр, содержащее число байт данных в строке
AAAA	поле из 4 цифр, содержащее адрес первого байта данных в строке
TT	поле из 2 цифр, содержащее тип данных в строке. Для последней строки поле TT содержит значение 01, а для всех остальных строк – значение 00
HHHH…HHH H	поле, содержащее данные. Данные расположены по принципу “младший байт по младшему адресу”
CC	поле из 2 цифр, содержащее контрольную сумму строки, которая вычисляется как дополнение до FF суммы всех предыдущих байт строки

Ниже приведен пример выполнимого файла в формате INHX8M, сгенерированного Ассемблером Microchip MPASM.

:020000000628D0
:08000800052809008C0163309A

:0E0010000C0203190D288C0A07288C01072802
:00000001FF

4.10.2 Шестнадцатеричный формат INHX8S

Формат INHX8S предназначен для записи массива 16-битовых чисел и подобен описанному выше формату INHX8M за исключение того, что младшие и старшие байты чисел хранятся в двух разных файлах, имеющих расширения .hxl и hxh, а адреса соответствуют реальным адресам.

Ниже приведен пример выполнимых файлов в формате INHX8S, сгенерированных Ассемблером Microchip MPASM.

:0100000006F9
:0B00040005098C630C030D8C078C07B2
:00000001FF

:0100000028D7
:0B0004002800013002192 80A280128FA
:00000001FF

4.10.3 Шестнадцатеричный формат INHX32

Формат INHX32 предназначен для записи массива 32-битовых чисел и подобен описанному выше формату INHX8M за исключением того, что в нем присутствуют два новых типа строк: строка сегментного адреса (TT = 02) и строка линейного адреса (TT = 04).

Ниже приведен пример выполнимого файла в формате INHX32, сгенерированного Ассемблером Microchip MPASM.

```
:020000040000FA
:020000000628D0
:08000800052809008C0163309A
:0E0010000C0203190D288C0A07288C01072802
:00000001FF
```

4.11 Файл объектного кода

Файл объектного кода имеет расширение .o и содержит машинно-зависимый код, а также дополнительную информацию, необходимую для работы компоновщика Microchip MPLINK.

4.12 Файл отладки

Файл отладки имеет расширения .cod и содержит информацию, необходимую для работы отладчика Microchip MPLAB-SIM.

ЛИТЕРАТУРА

1. Левкович В.Н. Архитектура и основы программирования однокристальных микроконтроллеров семейства PIC16F84: Метод. пособие к лаб. работам и курсовому проектированию по дисц. “Вычислительные и микропроцессорные устройства» для студентов спец. 39 01 02 «Радиоэлектронные системы» и 39 01 01 «Радиотехника». –Мн.: БГУИР, 2002. - 58 с.

2. Бурак А.И., Левкович В.Н. Интегрированная среда MPLab IDE разработки программ для микроконтроллеров PICmicro фирмы Microchip: Метод. пособие к лаб. работам по курсу «Цифровые и микропроцессорные устройства». – Мн.: БГУИР, 2003. – 31 с.

3. MPASM. Руководство пользователя /Пер. с англ. –М.: ООО «Микро-Чип», 2001. –62 с.

4. Яценков В.С. Микроконтроллеры Microchip. Практ. руководство – М.:Горячая линия - Телеком, 2002. –296 с.

Св. план 2004, поз.73

Учебное издание

Левкович Василий Николаевич,
Грицук Андрей Сергеевич,
Коваленко Илья Валерьевич

КОНСТРУИРОВАНИЕ ПРОГРАММ НА АССЕМБЛЕРЕ ДЛЯ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА PICMICRO

Учебное пособие
по курсу “Цифровые и микропроцессорные устройства»
для студентов специальностей
39 01 01 «Радиотехника» и 39 01 02 «Радиоэлектронные системы»
всех форм обучения

Редактор Т.Н. Крюкова
Корректор Е.Н. Батурчик

Подписано в печать 18.06.2004. Формат 60x84 1/16.
Бумага офсетная. Печать ризографическая. Усл. печ. л. 4,77.
Уч.-изд. л. 2,8. Тираж 150 экз. Заказ 83.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0133108 от 30.04.2004.
220013, Минск, П. Бровки, 6