MPASM™

Руководство пользователя

Перевод основывается на технической документации DS33014G
компании Microchip Technology Incorporated, USA.

MPASM USER'S GUIDE

Содержание

1. Предварительная информация о MPASM

1.1 Введение

В этой главе будут рассмотрены основные характеристики MPASM.

1.2 Основные части раздела

Что такое MPASM
Назначение MPASM
Совместимость кода программы
Совместимость с инструментальными средствами

1.3 Что такое MPASM

MPASM – бесплатная, универсальная программа компиляции исходного текста программы на языке ассемблер для микроконтроллеров PICmicro компании Microchip Technology Incorporated.

Ассемблер MPASM работает под управлением операционных систем MS-DOS V5.0 и Microsoft Windows 95/98/NT на PC совместимых компьютерах.

1.4 Назначение MPASM

MPASM обеспечивает универсальный инструмент разработки программ для 12/14/16-разрядных микроконтроллеров PICmicro.

Основные достоинства ассемблера MPASM:

- поддержка всех инструкций микроконтроллеров PICmicro;
- интерфейс командной строки;
- оконный интерфейс;
- система директив;
- поддержка макросов;
- совместимость с MPLAB IDE.

1.5 Совместимость кода программы

Поскольку MPASM является универсальным решением для всех типов микроконтроллеров PICmicro то, например программа, написанная для PIC16C54, может быть легко перенесена на микроконтроллер PIC16C71. При переносе программы следует изменить инструкции, которые связаны с аппаратными особенностями микроконтроллеров, а остальная часть директив и макрокоманд останется без изменений.

1.6 Совместимость с инструментальными средствами

MPASM совместим со всеми инструментами Microchip включая: MPLAB SIM, MPLAB ICE, PRO MATE, PICSTART Plus.

MPASM гарантирует совместимость синтаксиса текста программы с последующими версиями.

2. Установка и начало работы с MPASM

2.1 Введение

В этой главе будет рассмотрена установка MPASM на ваш компьютер с кратким обзором ассемблера.

2.2 Основные части раздела

Установка MPASM
Краткий обзор ассемблера
Входные и выходные файлы MPASM

2.3 Установка MPASM

Существуют три версии MPASM:
- MPASM.EXE для MS DOS V 5.0 и выше;
- MPASM_DP.EXE расширенная DOS версия;
- MPASMWIN.EXE для Microsoft Windows 3.x/95/98/NT (рекомендуется).

MPASM.EXE – имеет интерфейс командной строки, может использоваться в DOS или в окнах DOS под управлением операционной системы Microsoft Windows 3.x/95/98/NT. Допускается использование совместно с MPLAB, рекомендуется MPASMWIN.EXE.

MPASM_DP.EXE – имеет оконный интерфейс, может использоваться в DOS или в окнах DOS под управлением операционной системы Microsoft Windows 3.x/95/98/NT. Допускается использование совместно с MPLAB, рекомендуется MPASMWIN.EXE.

MPASMWIN.EXE – версия MPASM для операционной системы Microsoft Windows 3.x/95/98/NT имеет оконный интерфейс. Допускается использование совместно с MPLAB или автономно.

Если Вы собираетесь использовать MPASM совместно с MPLAB IDE, то необходимо установить MPASM, включив его в список установочных компонентов при инсталляции MPLAB IDE.

Если предполагается использовать MPASM отдельно от MPLAB IDE, то создайте дополнительную директорию на диске для размещения файлов MPASM, и разархивируйте в нее установочный файл, взятый с CD Microchip или WEB узла Microchip (www.microchip.com или www.microchip.ru).

2.4 Краткий обзор ассемблера

MPASM может использоваться в двух случаях:
- для генерации абсолютного кода, который может быть загружен непосредственно в микроконтроллер;
- для генерации объектных файлов, которые связываются с другими компилированными модулями.

Абсолютный код – режим работы программы MPASM по умолчанию.

При компиляции исходного файла в этом режиме, все значения должны быть явно указаны в исходном файле или во включаемых файлах. Если компиляция выполнена без ошибок, то будет создан HEX файл кода программы, который можно использовать для непосредственного программирования микроконтроллера.

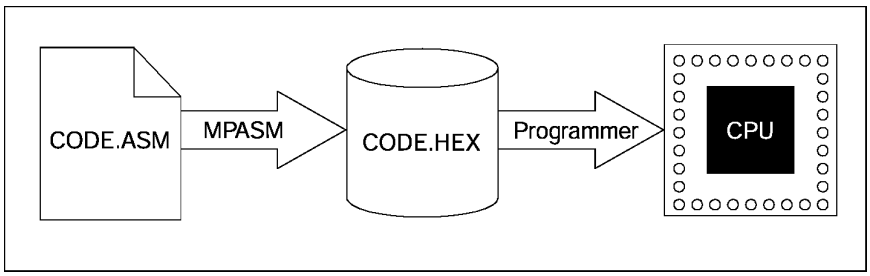

Рис 2.1

MPASM так же имеет возможность генерировать объектные модули, которые могут быть связаны друг с другом с использованием линкера MPLINK для окончательного формирования исполняемого (абсолютного) кода. Данный метод позволяет многократно использовать отлаженные модули программы. Объектные файлы могут быть сгруппированы в библиотечные файлы с помощью программы MPLIB. Библиотеки могут указываться в качестве параметра во время линковки и, таким образом, в исполняемый код будет включены только необходимые процедуры.

Компиляция проекта из нескольких файлов.

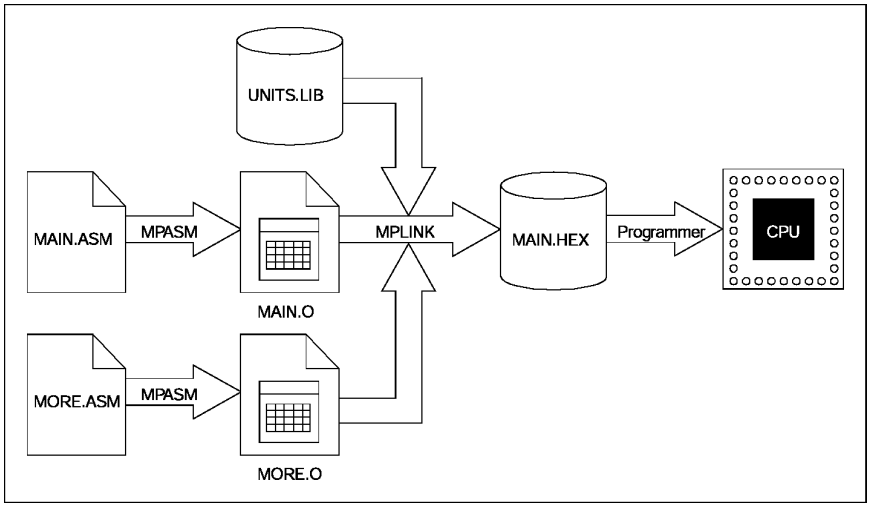

Рис 2.2

Группировка объектных файлов в библиотеку.

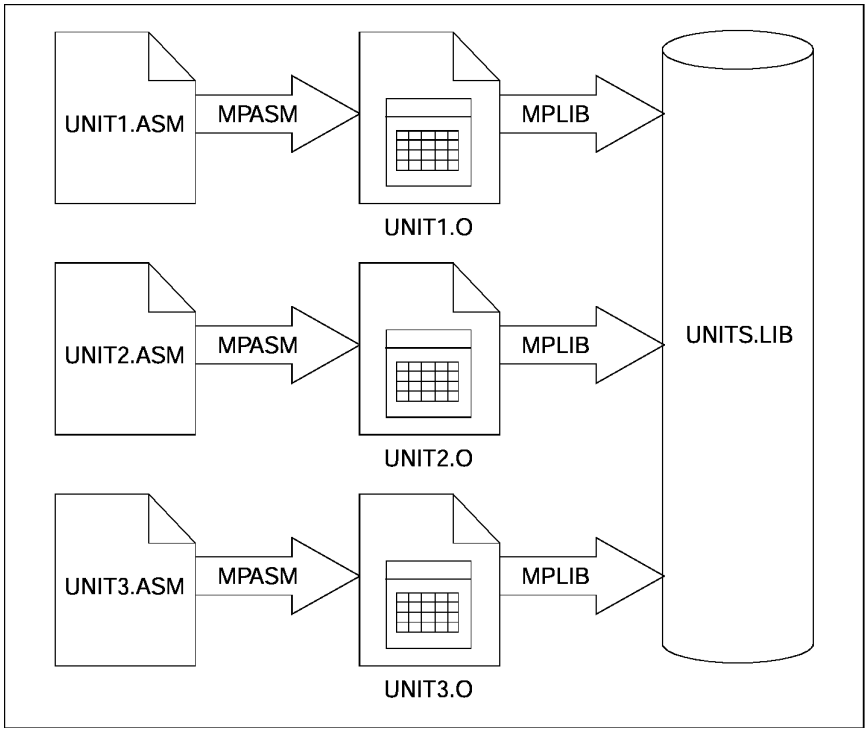

Рис 2.3

2.5 Входные и выходные файлы MPASM

Типы файлов, связанные с ассемблером MPASM.

Тип файла	Описание
.ASM	Исходный файл MPASM, <Source_name> .ASM
.LST	Файл листинга программы, <Source_name> .LST
.ERR	Список ошибок, возникших при компиляции, <Source_name> .ERR
.HEX	Файл кода программы, <Source_name> .HEX
.HXL/ .HXH	Файлы кода программы, отдельно младшие и старшие байты кода, <Source_name> .HXL < Source_name > .HXH
.COD	Файл для отладчика, <Source_name> .COD
.O	Объектный файл программы, <Source_name> .O

2.5.1 Исходный файл (.ASM)

Исходный файл программы может быть создан в любом текстовом редакторе ASCII. Текст программы должен удовлетворять следующим требованиям:

Каждая строка исходного файла может содержать до четырех информационных полей:

- метка;
- мнемоника команды;
- операнды команды;
- комментарии.

Необходимо соблюдать порядок расположения информационных полей в строке. Метки должны начинаться с первой колонки. Мнемоники команд должны начинаться со второй (и далее) колонки. Операнды следуют за мнемоникой команды. Комментарии могут следовать за операндами, мнемониками и метками и могут начинаться в любой колонке. Максимальная ширина колонки 255 символов. Метки от мнемоник должны отделяться двоеточием, пробелами или символами табуляции, операнды должны разделяться запятыми.

Например:

```
        ;
        ; Пример исходного файла MPASM.
        ;
              list   p=16c54
        Dest  equ    H'0B'

              org    H'01FF'
              goto   Start

              org    H'0000'

        Start movlw  H'0A'
              movwf  Dest
              bcf    Dest, 3
              goto   Start

              end
```

2.5.1.1 Метки

Метка должна начинаться в колонке 1. За ней может следовать двоеточие (:), пробелы, символы табуляции или конец строки.

Метка должна начинаться с символа латинского алфавита или символа подчеркивания (_) и может состоять из алфавитно-цифровых символов латинского алфавита, символа подчеркивания (_) или знака вопроса (?).

Максимальная длина метки 32 символа.

По умолчанию метки чувствительны к регистру символов, этот параметр может быть изменен в командной строке при запуске MPASM. Если в имени метки используется двоеточие, то отделенная часть трактуется как оператор, а не как часть имени метки.

2.5.1.2 Мнемоники

Мнемоника инструкций микроконтроллера, директивы ассемблера и макрокоманды должны начинаться во второй (и далее) колонке. Если в той же строке имеется метка, то она должна быть отделена двоеточием или одним (и более) символом пробела (табуляции).

2.5.1.3 Операнды

Операнды должны быть отделены от мнемоники одним (или более) символом пробела (табуляции). Многократные операнды разделяются запятыми.

2.5.1.4 Комментарии

Любой текст после (;) трактуется как комментарий и все символы до конца строки игнорируются. Допускаются строковые константы содержащие (;), как комментарий они не воспринимаются.

2.5.2 Формат файла листинга (.LST)

Пример:

```
MPASM 01.99.21 Intermediate        MANUAL.ASM          5-30-1997 15:31:05 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT

VALUE
                00001 ;
                00002 ; Пример исходного файла MPASM.
                00003 ;
                00004         list p=16c54
0000000B        00005 Dest    equ    H'0B'
                00006
01FF            00007         org    H'01FF'
01FF    0A00    00008         goto   Start
                00009
0000            00010         org    H'0000'
                00011
0000    0C0A    00012 Start   movlw  H'0A'
0001    002B    00013         movwf  Dest
0002    0A00    00014         goto   Start
                00015
                00016         end


MPASM 01.99.21 Intermediate        MANUAL.ASM          5-30-1997 15:31:05 PAGE 2

SYMBOL TABLE
  LABEL                VALUE

Dest                   0000000B
Start                  00000000
__16C54                00000001

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : XXX------------ ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used: 4
Program Memory Words Free: 508

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 0 reported, 0 suppressed
```

Формат файла листинга, генерируемого MPASM, следующий:

Имя файла и версия, дата и время компиляции, номер страницы выводятся в начале каждой страницы.

Первая колонка цифр указывает базовый адрес кода в памяти. Вторая колонка показывает 32-разрядное значение всех символьных переменных созданных директивами SET, EQU, VARIABLE, CONSTANT или CBLOCK. Третья колонка предназначена для машинного кода, выполняемого микроконтроллером. Четвертая колонка содержит номер строки соответствующего исходного файла

Остаток строки зарезервирован для исходного текста, который породил машинный код.

Ошибки, предупреждения и сообщения вставляются между строк исходного кода и относятся к следующей по тексту строке исходного кода.

Таблица символов (SYMBOL TABLE) показывает все символьные переменные, определенные в программе.

Карта использования памяти (MEMORY USAGE MAP) дает представление об использовании памяти в графическом виде. Символ “X” показывает использованный участок, а “-” отмечает участок памяти не используемый данным объектом. При генерации объектного файла карта памяти не выводится.

2.5.3 Формат файла списка ошибок, возникших при компиляции, (.ERR)

По умолчанию MPASM формирует файл списка ошибок при выполнении компиляции исходных файлов. Этот файл может быть полезен при отладке программы. MPLAB автоматически открывает этот файл в случае возникновения ошибок.

Структура файла ошибок:
<тип> [<номер>]<файл><строка><описание>

Например:
```
Error[113] C:\PROG.ASM 7 : Symbol not previously defined (start)
```

Список возможных ошибок смотрите в приложении B.

2.5.4 Форматы файлов кода программы (.HEX, .HXL, .HXH)

MPASM способен создавать файлы кода различных форматов. Подробное описание форматов смотрите в приложении A.

2.5.5 Файл для отладчика (.COD)

При компиляции исходного текста программы формируется дополнительный файл, используемый для отладки программы средствами MPLAB IDE.

2.5.6 Объектный файл программы (.O)

Объектные файлы являются перемещаемым кодом, который генерируется из файлов с исходным текстом.

3. DOS версия MPASM

3.1 Введение

Эта глава посвящена описанию версий MPASM для ОС DOS (MPASM.EXE и MPASM_DP.EXE). DOS версия (MPASM.EXE) выполняется из командной строки DOS или DOS окна в Windows. Расширенная DOS версия (MPASM_DP.EXE) работает точно также, и может использоваться, когда для DOS версии не хватает памяти.

3.2 Основные части раздела

Интерфейс командной строки
Оконный интерфейс

3.3 Интерфейс командной строки

MPASM может быть запущен из командной строки:

MPASM [/ <параметр> [/ <параметр> …]] [<имя файла>]
или
MPASM_DP [/ <параметр> [/ <параметр> …]] [<имя файла>]

Где:

/ <параметр> - один из параметров компиляции
/ <имя файла> - имя исходного файла

Например:

Если исходный файл test.asm находится в текущей директории, то команда компиляции может быть следующая:
MPASM /e /l test

Параметры настройки, принятые по умолчанию, могут быть изменены следующим образом:

/ <параметр>	Включает параметр
/ <параметр> +	Включает параметр
/ <параметр> -	Выключает параметр
/ <параметр> <имя файла>	Включает параметр и перенаправляет вывод в указанный файл.

Если имя исходного файла не указано, то MPASM запускается в режиме оконного интерфейса.

Параметры командной строки компилятора:

Параметр	По умолчанию	Описание
?	-	Вызов краткой помощи
a	INHX8M	Устанавливает формат HEX файла: /a<hex-формат> Где <hex-формат> может принимать одно из значений INHX8M INHX8S INHX32
c	Включено	Включение/выключение чувствительности к регистру символов
d	-	Определять символы: /dDebug /dMax=5 /dString="abc"
e	Включено	Включение/выключение/установка директории для файла списка ошибок / e включено / e + включено / e – выключено / e <директория> имя файла - включено, установлена директория и имя файла
h	-	Вызов краткой помощи
l	Включено	Включение/выключение/установка директории для файла листинга программы / l включено / l + включено / l – выключено / l <директория> имя файла - включено, установлена директория и имя файла
m	Включено	Включение/выключение полного теста макроса в файле листинга программы
o	Выключено	Включение/выключение/установка директории для объектного файла / o включено / o + включено / o – выключено / o <директория> имя файла - включено, установлена директория и имя файла
p	-	Указание типа микроконтроллера / p <тип микроконтроллера> где <тип микроконтроллера> - наименование типа микроконтроллера PICmicro, например PIC16C54
q	Выключено	Включение/выключение режима подавления вывода на экран.
r	HEX	Устанавливает система счисления по умолчанию / r <формат> Где < формат> может принимать одно из значений HEX – шестнадцатеричная DEC – десятичная OCT – восьмеричная
t	8	Длина символа табуляции / t <длина>
w	0	Устанавливает уровень детализации вывода сообщений на экран / w <уровень> Где < уровень> может принимать одно из значений 0 – все сообщения 1 – сообщения об ошибках и предупреждения 2 – только сообщения об ошибках
x	Выключено	Включение/выключение/установка директории для файла кросс ссылок / x включено / x + включено / x – выключено / x <директория> имя файла -включено, установлена директория и имя файла

3.4 Оконный интерфейс

Оконный интерфейс MPASM для операционной системы DOS V 5.0 и выше работает в текстовом режиме монитора. Пример рабочего окна показан на рисунке. Здесь Вы можете указать имя исходного файла и другие параметры компиляции.

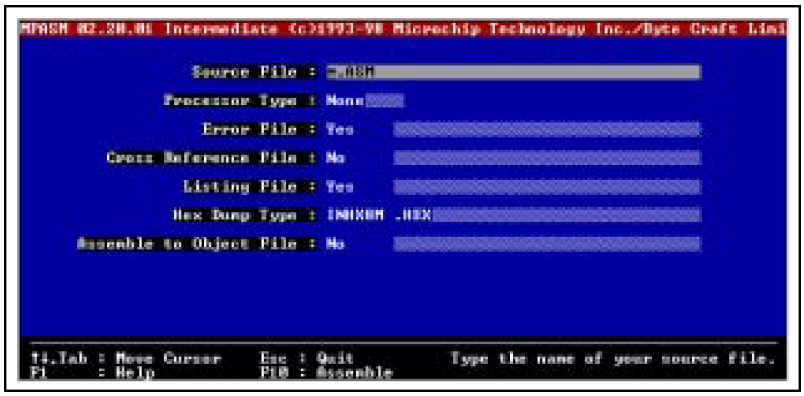

Рис 3.1

3.4.1 Source File

Укажите имя исходного файла. Имя файла может содержать путь DOS и символы расширения. Если Вы используете символы расширения (* или ?), Вам будет показан список всех файлов удовлетворяющих маске, из которых вы можете выбрать необходимый файл. Для автоматического ввода в строку “*.ASM” нажмите кнопку <TAB>.

3.4.2 Processor Type

Если Вы не указали тип микроконтроллера в исходном файле, укажите его в этом поле. С помощью стрелок на клавиатуре перейдите в поле выбора типа микроконтроллера и нажатием клавиши <RET> выберете нужный тип.

3.4.3 Error File

Файл ошибок компиляции (<sourcename>.ERR) создается по умолчанию. С помощью стрелок на клавиатуре перейдите в поле указания имени файла. Клавишей <RET> выберете создавать (YES) или нет (NO) файл. Нажмите на клавишу <TAB> для перевода курсора в затененную область и ввода нового имени файла. В имени файла ошибок не допускаются групповые символы.

3.4.4 Cross Reference File

По умолчанию файл перекрестных ссылок <sourcename>.XRF не создается. С помощью стрелок на клавиатуре перейдите в поле указания имени файла. Клавишей <RET> выберете создавать (YES) или нет (NO) файл. Нажмите на клавишу <TAB> для перевода курсора в затененную область и ввода нового имени файла. В имени файла перекрестных ссылок не допускаются групповые символы.

3.4.5 Listing File

Файл листинга программы (<sourcename>.LST) создается по умолчанию. С помощью стрелок на клавиатуре перейдите в поле указания имени файла. Клавишей <RET> выберете создавать (YES) или нет (NO) файл. Нажмите на клавишу <TAB> для перевода курсора в затененную область и ввода нового имени файла. В имени файла листинга программы не допускаются групповые символы.

3.4.6 HEX Dump Type

Укажите тип и имя .HEX файла. С помощью стрелок на клавиатуре перейдите в поле указания имени файла. Клавишей <RET> выберете тип HEX файла. Нажмите на клавишу <TAB> для перевода курсора в затененную область и ввода нового имени файла.

3.4.7 Assemble to Object File

В этом пункте можно разрешить генерацию перемещаемого объектного кода, который может использоваться линкером для создания шестнадцатеричного файла. Имя объектного файла может быть изменено, так же как и имя файла ошибок.

4. Windows версия MPASM

4.1 Введение

В главе рассмотрены вопросы работы версией MPASM для Windows (MPASMWIN.EXE) как отдельного приложения, так и в составе MPLAB IDE.

4.2 Основные части раздела

Оконный интерфейс
Работа с MPASM в интегрированной среде MPLAB IDE
Настройка MPLAB IDE для работы с MPASM
Компиляция исходного текста программы
Возможные ошибки

4.3 Оконный интерфейс

MPASM для операционной системы Windows 3.x/95/98/NT поддерживает графический интерфейс настройки параметров компиляции исходного файла программы.

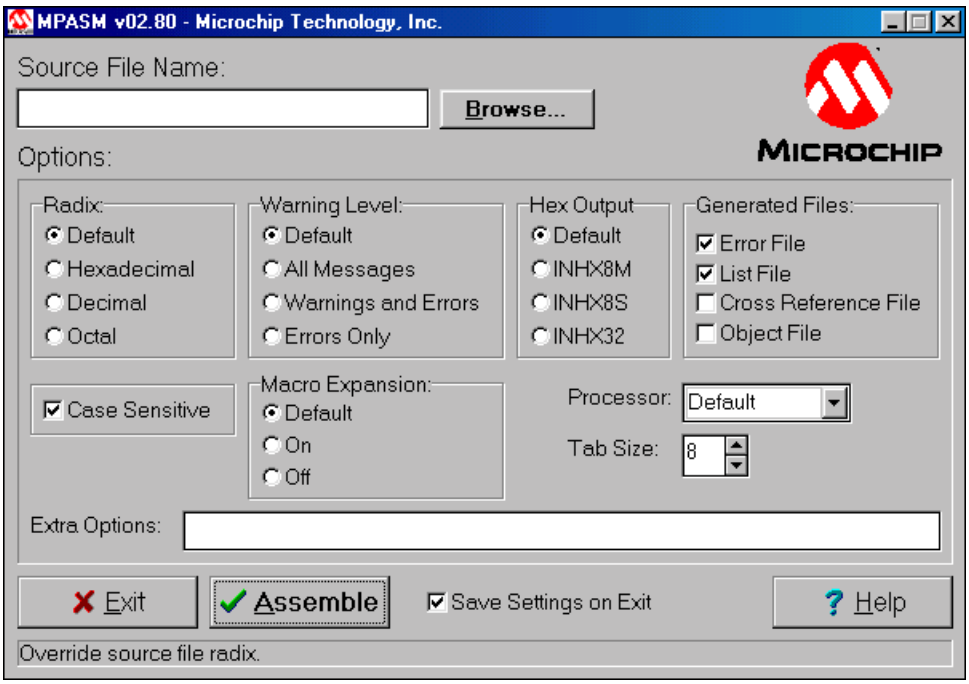

Рис. 4.1

Примечание. При запуске MPASM из интегрированной среды разработки MPLAB IDE, интерфейс настройки параметров не доступен. Для правильной настройки параметров компиляции обратитесь к соответствующему разделу документации MPLAB User's Guide.

Нажмите кнопку *Browse* для выбора файла исходного текста программы. Укажите необходимые параметры и нажмите кнопку *Assemble* для начала компиляции

Radix	- система счисления по умолчанию;
Warning Level	- уровень вывода сообщений;
Hex Output	- формат HEX файла;
Generated Files	- создаваемые файлы;
Case Sensitivity	- включение/выключение чувствительности к регистру символов;
Macro Expansion	- разворачивать макрос в файле листинга программы;
Processor	- тип микроконтроллера;
Tab Size	- длина табуляции;
Extra Options	- использовать настройки, указанные в командной строке (см. раздел 3.3);
Save Settings on Exit	- сохранить настройки MPASM в файле MPLAB.INI. Эти настройки будут использоваться при последующих запусках MPASMWIN.

4.4 Работа с MPASM в интегрированной среде MPLAB IDE

MPLAB проект состоит из узлов, показанных на рисунке 4-2, которые представляют файлы используемые в проекте:

Целевой узел – заключительный вывод
- Шестнадцатеричный файл кода

Узлы проекта – компоненты
- Исходные файлы проекта

В этой главе рассмотрены аспекты взаимодействия MPLAB и MPASM. Дополнительную информацию о создании проектов в среде MPLAB смотрите в технической документации MPLAB User's Guide (DS51025).

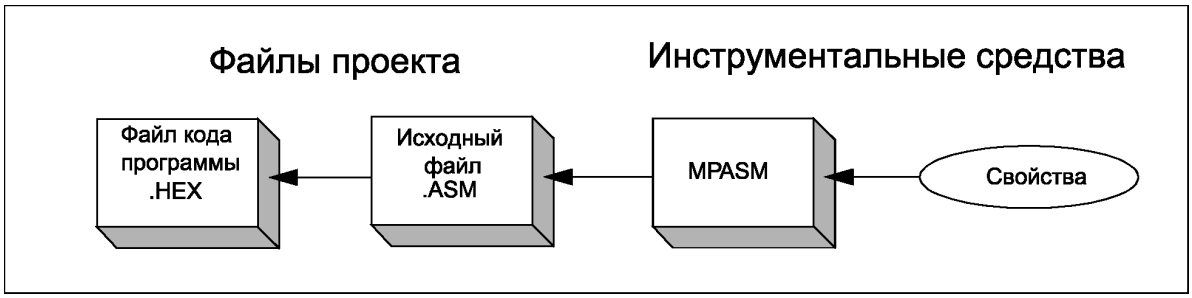

Рис 4.2

Создание проекта необходимо для того, что бы объединить работу компилятора и линкера в генерации кода программы (.HEX) из исходных файлов. На диаграмме показано соотношение заключительного файла кода .HEX и исходного фала .ASM.

4.5 Настройка MPLAB IDE для работы с MPASM

Необходимо выполнить следующие шаги для работы с MPASM в среде MPLAB:
1. После создания проекта (*Project>New Project*), откроется диалоговое окно настройки проекта. Выберите HEX файл (например, tutor84.hex) в списке файлов проекта *Project Files* и нажмите кнопку *Node Properties* для настройки параметров компиляции.

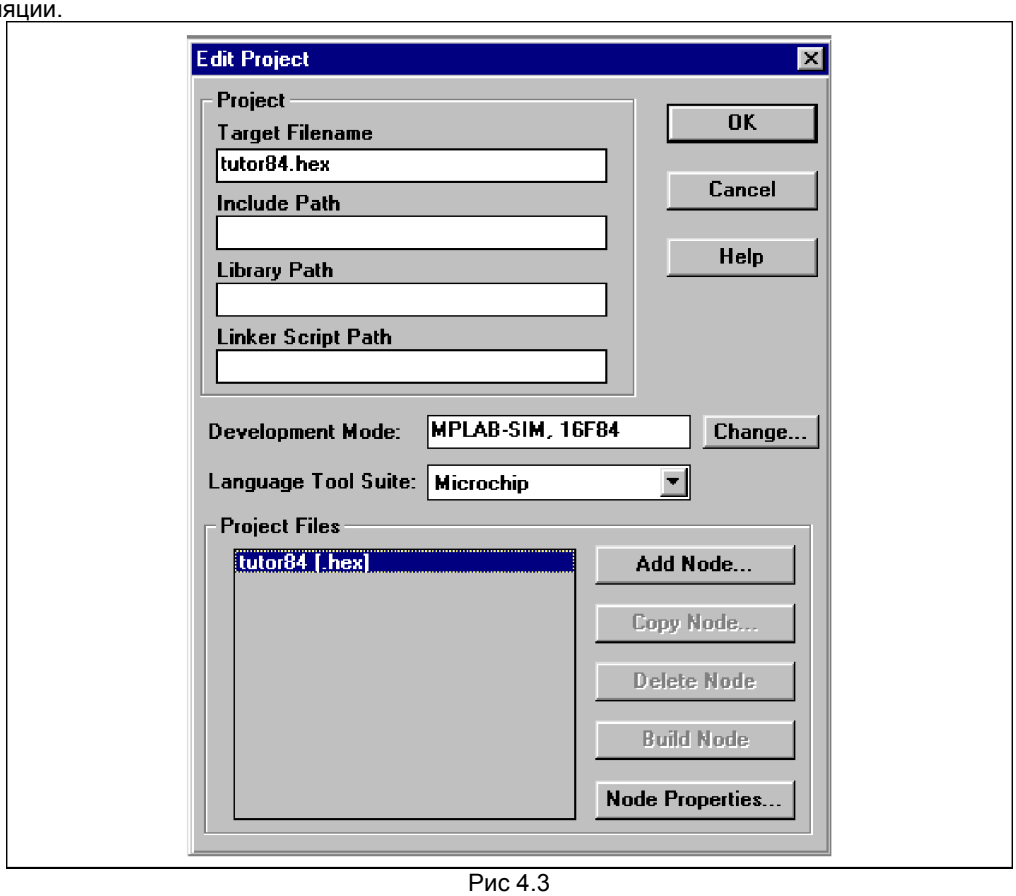

Рис 4.3

2. В диалоговом окне настройки параметров укажите MPASM как средство компиляции, а так же если необходимо, настойте другие параметры компиляции.

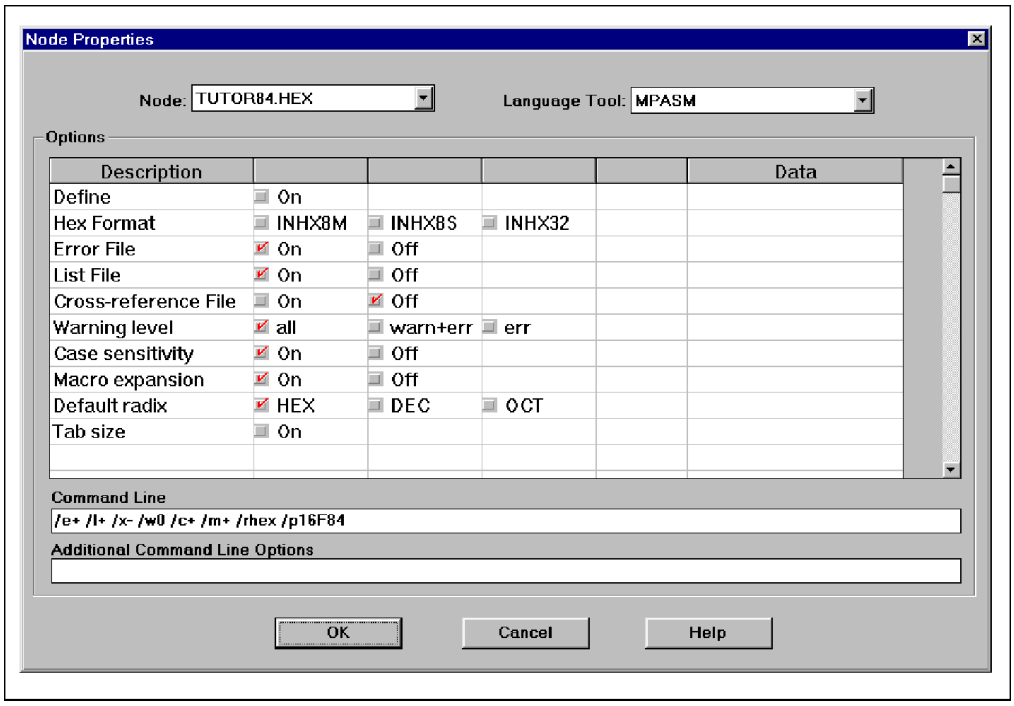

Рис 4.4

3. Присоедините исходный файл к проекту (нажмите кнопку *Add Node* в диалоговом окне настройки проекта).

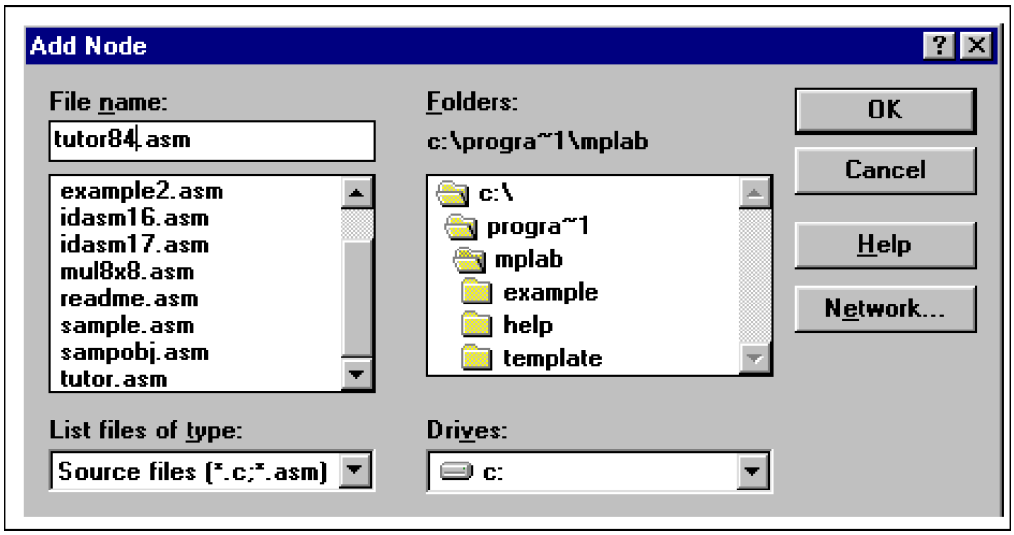

Рис 4.5

4. Нажмите OK для подтверждения настроек.

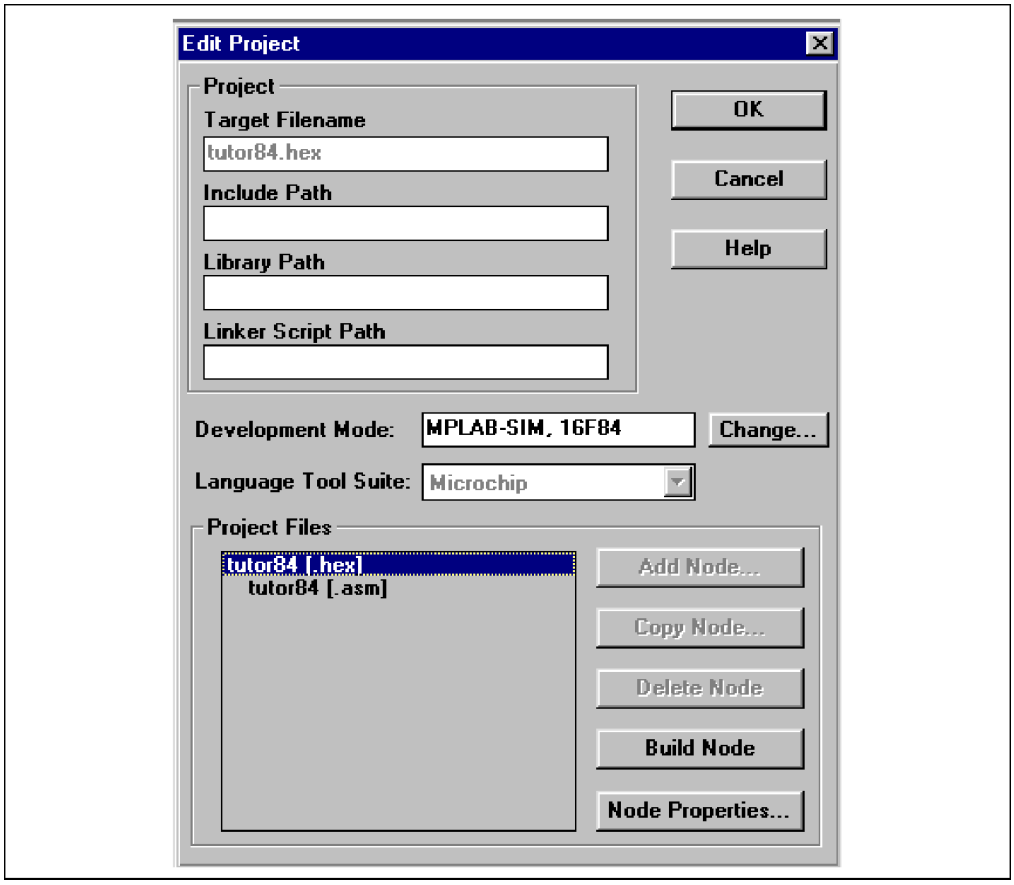

Рис 4.6

4.6 Компиляция исходного текста программы

Запуск компиляции проекта выполняется в MPLAB IDE командой *Project>Make Project*. HEX файлы автоматически загружаются в память программ.

Вместе с файлом кода создаются дополнительные файлы. Подробную информацию о дополнительных файлах генерируемых при компиляции исходного файла смотрите в разделе 2.5.

4.7 Возможные ошибки

В случае возникновения ошибок в работе MPASM при компиляции проекта рекомендуется проверить следующие параметры:

Выберите пункт меню *Project>Install Language Tool...* и проверьте правильность указания пути к файлу MPASMWIN.EXE и установку флага *Windowed*. Как альтернатива, может быть указан файл MPASM.EXE, но в этом случае должен быть установлен флаг *Command-Line*.

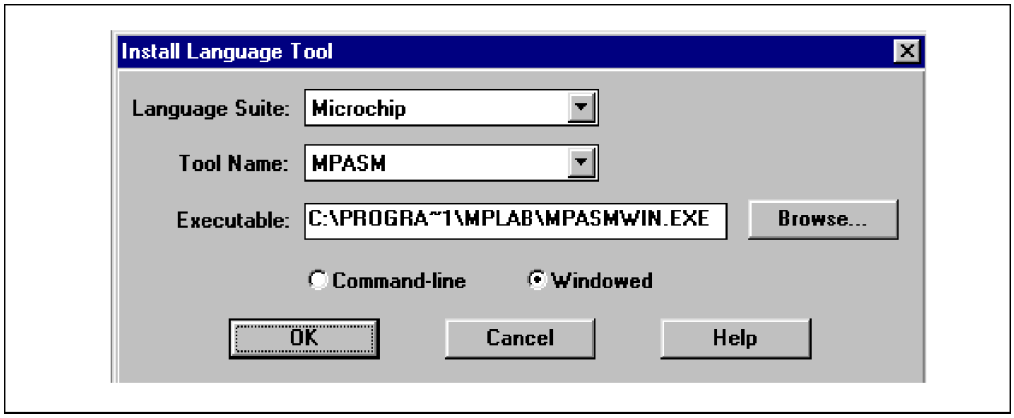

Рис 4.7

Если Вы используете MPASM.EXE DOS версии и получаете предупреждение о нехватки памяти в компьютере, проверьте параметры работы DOS программы в среде Windows. Используя программу «Проводник», включаемую в операционную систему Windows, нажмите правой кнопкой «мыши» на файле MPASM.EXE и выберите пункт «Свойства».

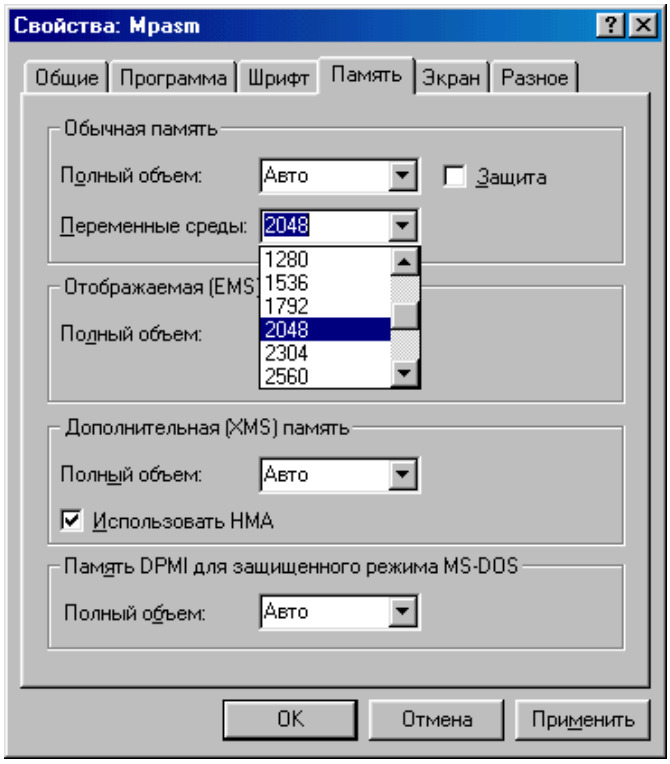

Рис 4.8

Выберите параметр «Переменные среды» (Initial Environment) равный 2048. Если одновременно используется большое количество приложений, параметры которых указаны в файле Autoexec.bat, необходимо выбрать большее значение параметра «Переменные среды».

5. Директивы MPASM

5.1 Введение

В этой главе будут подробно описаны все директивы MPASM.

Директивы MPASM – команды, которые входят в состав исходного текста программы, но непосредственно не включаются в выходной код. Они используются для управления MPASM, параметрами ввода-вывода и распределением данных.

Многие директивы ассемблера имеют дополнительные имена и форматы. Они предназначены для обеспечения совместимости с более ранними версиями ассемблера и индивидуальными методами программирования. Для получения минимального объема кода программы рекомендуется использовать директивы MPASM, описанные в данном документе.

5.2 Типы директив MPASM

Существует пять основных типов директив:

- директивы контроля – управляют созданием разделов условно компилированного кода;
- директивы данных – управляют распределением памяти и назначением символических имен переменным и константам;
- директивы листинга – определяют формат и состав файла листинга. Эти директивы позволяют: указывать заголовки, нумеровать страницы и настраивать другие параметры;
- макро директивы – управляют работой макросов и распределением данных в теле макроса;
- директивы объектного файла – используются только при создании объектного файла.

5.3 Список директив MPASM

В таблице представлен список директив поддерживаемых MPASM. В остальной части этой главы будет подробно описана каждая директива. В описании всех директив будут содержаться пункты:

- синтаксис;
- описание;
- пример.

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

EQU	Определение константы ассемблера	<label> equ <expr>
ERROR	Формирует сообщение об ошибке	error "<text_string>"
ERRORLEVEL	Настройка параметров вывода сообщений об ошибках	errorlevel 0|1|2| <+|-><message number>
EXITM	Выход из макроса	exitm
EXPAND	Включение текста макроса в файл листинга программы	expand
EXTERN	Определение внешних меток	extern <label>[,<label>]
FILL	Запись значения в память программ	[<label>] fill <expr>, <count>
GLOBAL	Внешняя метка	global <label>[,<label>]
IDATA	Объявляет начало инициализации данных в объектном файле	[<name>] idata [<address>]
_ _IDLOCS	Установка значения ID	_ _idlocs <expr>
IF	Начало блока условия	if <expr>
IFDEF	Выполнение, если определена символьная метка	ifdef <label>
IFNDEF	Выполнение, если символьная метка не определена	ifndef <label>
INCLUDE	Подключение дополнительного исходного файла	include <<include_file>> |"<include_file>"
LIST	Список параметров	list [<list_option>,...,<list_option>]
LOCAL	Объявить локальную переменную макроса	local <label>[,<label>]
MACRO	Определить макрос	<label> macro [<arg>,...,<arg>]
_ _MAXRAM	Определяет максимальный объем ОЗУ	_ _maxram <expr>
MESSG	Сформировать сообщение	messg "<message_text>"
NOEXPAND	Не разворачивать текст макроса	noexpand
NOLIST	Выключить вывод в файл листинга	nolist
ORG	Установить адрес программы	<label> org <expr>
PAGE	Вставить страницу в файл листинга программы	page
PAGESEL	Произвести выбор страницы	pagesel <label>
PROCESSOR	Выбор типа микроконтроллера	processor <processsor_type>
RADIX	Система счисления по умолчанию	radix <default_radix>
RES	Резервирование памяти	[<label>] res <mem_units>
SET	Определение константы	<label> set <expr>
SPACE	Вставить пустые строки	space <expr>
SUBTITLE	Определение подзаголовка программы	subtitle "<sub_text>"
TITLE	Определение заголовка программы	title "<title_text>"
UDATA	Начало инициализации данных с обычным размещением в памяти (для объектного файла)	[<label>] udata [<RAM address>]
UDATA_ACS	Начало инициализации данных быстрого доступа (для объектного файла)	[<label >] udata_acs [<RAM address>]
UDATA_OVR	Начало инициализации временных данных (для объектного файла)	[<label >] udata_ovr [<RAM address>]
UDATA_SHR	Начало инициализации разделяемых данных (для объектного файла)	[<label >] udata_shr [<RAM address>]
#UNDEFINE	Отменить замену текста	#undefine <label>
VARIABLE	Определение символьной переменной	variable <label>[=<expr>,...,<label>[=<expr>]]
WHILE	Цикл While	while <expr>

5.4 _ _BADRAM - Идентификация нереализованного ОЗУ

5.4.1 Синтаксис

```
_ _badram <expr>[-<expr>][, <expr>[-<expr>]]
```

5.4.2 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, точная карта нереализованного ОЗУ создается директивами _ _BADRAM.

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

5.4.3 Пример

См. пример для _ _MAXRAM

5.4.4 См. также

_ _MAXRAM

5.5 BANKISEL - Выбор банка для косвенной адресации

5.5.1 Синтаксис

```
bankisel <label>
```

5.5.2 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.

Линкер 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 генерируются команда MOVLB или MOVLR. Если пользователь сам выбирает рабочий банк памяти, то никаких дополнительных инструкций в код программы добавлено не будет.

Дополнительную информацию смотрите в главе 6.

5.5.3 Пример

```
        movlw       Var1
        movwf       FSR
        bankisel    Var1

        movwf       INDF
```

5.5.4 См. также

BANKSEL, PAGESEL

5.6 BANKSEL - Выбор банка для прямой адресации

5.6.1 Синтаксис

```
banksel <label>
```

5.6.2 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.

Линкер 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 сгенерированна команда MOVLB. Если в микроконтроллере только один банк памяти никакой дополнительный код генерироваться не будет.

Дополнительную информацию смотрите в главе 6.

5.6.3 Пример

```
        banksel     Var1
        movwf       Var1
```

5.6.4 См. также

BANKISEL, PAGESEL

5.7 CBLOCK - Определение блока констант

5.7.1 Синтаксис

```
cblock [<expr>]
        <label>[:<increment>][,<label>[:<increment>]]
endc
```

5.7.2 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.

5.7.3 Пример

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

5.7.4 См. также

ENDC

5.8 CODE - Начало кода объектного файла в памяти программ

5.8.1 Синтаксис

```
[<label>] code [<ROM address>]
```

5.8.2 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.

Дополнительную информацию смотрите в главе 6.

5.8.3 Пример

```
        RESET         code H'01FF'
                      goto START
```

5.8.4 См. также

EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

5.9 _ _CONFIG - Установка битов конфигурации микроконтроллера

5.9.1 Синтаксис

```
_ _config <expr> OR _ _config <addr>, <expr>
```

5.9.2 Описание

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

5.9.3 Пример

```
        list p=17c42,f=INHX32
        _ _config H'FFFF' ;Конфигурация по умолчанию
```

5.9.4 См. также

_ _IDLOCS, LIST, PROCESSOR

5.10 CONSTANT - Определить символьную константу

5.10.1 Синтаксис

```
constant <label>[=<expr>,...,<label>[=<expr>] ]
```

5.10.2 Описание

Создает символьную константу для использования в выражениях MPASM. Существующая константа не может быть определена повторно, а выражения используемые при определении константы должны быть полностью разрешимы. Это основное отличие между константами определенными директивой CONSTANT и VARIABLE, SET. Иначе, константы и переменные могут поочередно использоваться в выражениях.

5.10.3 Пример

```
        variable RecLength=64

        constant BufLength=512
        .
        .
        .
        .
        constant MaxMem=RecLength+BufLength
```

5.10.4 См. также

VARIABLE, SET

5.11 DA - Сохранение строки в памяти программ

5.11.1 Синтаксис

```
[<label>] da <expr> [, <expr2>, ..., <exprn>]
```

5.11.2 Описание

Упаковывает в 14-битный формат два 7-битных символа ASCII. Используется для сохранения символьной строки в FLASH памяти программ микроконтроллера.

5.11.3 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
```

5.12 DATA - Сохранение значений или текста в памяти программ

5.12.1 Синтаксис

```
[<label>] data <expr>,[,<expr>,...,<expr>]
[<label>] data "<text_string>"[,"<text_string>",...]
```

5.12.2 Описание

Инициализирует одно или более слов памяти программ. Данные могут быть в виде констант, внутренних/внешних 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 PIC18CXXX первый символ сохраняется в младшем байте слова.

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

5.12.3 Пример

```
data reloc_label+10
data 1,2,ext_label
data "testing 1,2,3"
data 'N'
data start_of_program
```

5.12.4 См. также

DB, DE, DT, DW, IDATA

5.13 DB - Побайтное сохранение данных в памяти программ

5.13.1 Синтаксис

```
[<label>] db <expr>[,<expr>,...,<expr>]
```

5.13.2 Описание

Резервирует слово в памяти программ с сохранением 8-битного 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.

5.13.3 Пример

```
db 't', 0x0f, 'e', 0x0f, 's', 0x0f, 't', '\n'
```

5.13.4 См. также

DATA, DE, DT, DW, IDATA

5.14 DE - Резервирует 8-разрядное значение в EEPROM памяти

5.14.1 Синтаксис

```
[<label>] de <expr>[,<expr>,...,<expr>]
```

5.14.2 Описание

Резервирует слово в EEPROM памяти для сохранения 8-битное 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.

5.14.3 Пример

```
org H'2100'                 ; Инициализация EEPROM
de "My Program, v1.0", 0
```

5.14.4 См. также

DATA, DB, DT, DW

5.15 #DEFINE - Определить замену текста

5.15.1 Синтаксис

```
#define <name> [<string>]
#define <name> [<arg>,...,<arg>] <string>
```

5.15.2 Описание

Данная директива определяет правила замены текста. В тексте программы строка <name> будет заменена последовательностью символов <string>. В случае использования директивы без указания параметра <string> последовательность <name> отмечается MPASM для последующей проверки IFDEF.

Эта директива подражает директиве #define стандарта ANSI 'C'. Символы, определенные данным методом не доступны для просмотра в среде MPLAB IDE.

5.15.3 Пример

```
     #define         length 20
     #define         control 0x19,7
     #define         position(X,Y,Z) (Y-(2 * Z +X))
     :
     :
     test_label      dw position(1, length, 512)
                     bsf    control      ; установка бита 7 в регистре 0x19
```

5.15.4 См. также

#UNDEFINE, IFDEF, IFNDEF

5.16 DT - Определяет таблицу данных

5.16.1 Синтаксис

```
[<label>] dt <expr>[,<expr>,...,<expr>]
[<label>] dt "<text_string>"[,"<text_string>",...]
```

5.16.2 Описание

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

5.16.3 Пример

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

5.16.4 См. также

DATA, DB, DE, DW

5.17 DW - Резервирует слова памяти программ

5.17.1 Синтаксис

```
[<label>] dw <expr>[,<expr>,...,<expr>]
[<label>] dw "<text_string>"[,"<text_string>",...]
```

5.17.2 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.

5.17.3 Пример

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

5.17.4 См. также

DATA, DB, IDATA

5.18 ELSE - Начало альтернативного блока программы условия IF

5.18.1 Синтаксис

```
else
```

5.18.2 Описание

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

5.18.3 Пример

```
speed macro rate
   if rate < 50
          dw slow
   else
          dw fast
endif
endm
```

5.18.4 См. также

ENDIF, IF

5.19 END - Окончание программы

5.19.1 Синтаксис

```
end
```

5.19.2 Описание

Указывает окончание текста программы.

5.19.3 Пример

```
list p=17c42
:          ; текст программы
:
end        ; конец всех команд
```

5.20 ENDC - Окончание автоматического блока констант

5.20.1 Синтаксис

```
endc
```

5.20.2 Описание

Используется совместно с директивой CBLOCK. Указывает окончание списка констант.

5.20.3 См. также

CBLOCK

5.21 ENDIF - Окончание условного блока программы

5.21.1 Синтаксис

```
endif
```

5.21.2 Описание

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

5.21.3 См. также

ELSE, IF

5.22 ENDM - Окончание макроса

5.22.1 Синтаксис
```
endm
```

5.22.2 Описание
Завершает макрос, открытый директивой MACRO.

5.22.3 Пример
```
make_table macro arg1, arg2
          dw arg1, 0
          res arg2
          endm
```

5.22.4 См. также
MACRO, EXITM

5.23 ENDW - Завершает цикл While

5.23.1 Синтаксис
```
endw
```

5.23.2 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.

5.23.3 Пример
Смотрите пример в описании директивы WHILE

5.23.4 См. также
WHILE

5.24 EQU - Определение константы ассемблера

5.24.1 Синтаксис
```
<label> equ <expr>
```

5.24.2 Описание
Присваивает значение <expr> константе <label>.

5.24.3 Пример
```
four equ 4       ; присваивает значение 4 константе four
```

5.24.4 См. также
SET

5.25 ERROR - Формирует сообщение об ошибке

5.25.1 Синтаксис
```
error "<text_string>"
```

5.25.2 Описание
Сообщение <text_string> (длиной от 1 до 80 символов) будет напечатано в списках ошибок MPASM.

5.25.3 Пример
```
error_checking macro arg1
   if arg1 >= 55
          error "error_checking-01 arg out of range"
   endif
endm
```

5.25.4 См. также
MESSG

5.26 ERRORLEVEL - Настройка параметров вывода сообщений об ошибках

5.26.1 Синтаксис

```
errorlevel {0|1|2|+<msgnum>|-<msgnum>} [, ...]
```

5.26.2 Описание

Указание типов сообщений, которые будут включены в файл списка ошибок.

Параметр	Эффект
0	Вывод сообщений, предупреждений и ошибок
1	Вывод предупреждений и ошибок
2	Вывод только ошибок
+<msgnum>	Разрешить вывод сообщение с кодом <msgnum>
-<msgnum>	Запретить вывод сообщение с кодом <msgnum>

Полный список ошибок смотрите в приложении B. Сообщения об ошибках не могут быть запрещены. Для уровней 0, 1 и 2 может быть запрещен/разрешен вывод каждого сообщения в отдельности.

5.26.3 Пример

```
errorlevel 1, -202
```

5.26.4 См. также

LIST

5.27 EXITM - Выход из макроса

5.27.1 Синтаксис

```
exitm
```

5.27.2 Описание

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

5.27.3 Пример

```
test macro filereg
   if filereg == 1
          exitm
   else
          error "bad file assignment"
   endif
endm
```

5.27.4 См. также

ENDM, MACRO

5.28 EXPAND - Включение текста макроса в файл листинга программы

5.28.1 Синтаксис

```
expand
```

5.28.2 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.

5.28.3 См. также

MACRO, NOEXPAND

5.29 EXTERN - Определение внешних меток

5.29.1 Синтаксис

```
extern <label>[ ,<label>]
```

5.29.2 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.

Дополнительную информацию смотрите в главе 6.

5.29.3 Пример

```
     extern    Function
     ...
     call      Function
```

5.29.4 См. также

GLOBAL, IDATA, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

5.30 FILL - Запись значения в память программ

5.30.1 Синтаксис

```
[<label>] fill <expr>, <count>
```

5.30.2 Описание

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

5.30.3 Пример

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

5.30.4 См. также

DATA, DW, ORG

5.31 GLOBAL - Внешняя метка

5.31.1 Синтаксис

```
global <label>[ ,<label>]
```

5.31.2 Описание

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

Дополнительную информацию смотрите в главе 6.

5.31.3 Пример

```
               udata
     Var1      res    1
     Var2      res    1
               global Var1, Var2
               code
     AddThree
               global AddThree
               addlw  3
               return
```

5.31.4 См. также

EXTERN, IDATA, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

5.32 IDATA - Объявляет начало инициализации данных в объектном файле

5.32.1 Синтаксис

```
[<name>] idata [<address>]
```

5.32.2 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. Линкер 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).

Дополнительную информацию смотрите в главе 6.

5.32.3 Пример

```
            idata
     LimitL   dw 0
     LimitH   dw D'300'
     Gain     dw D'5'
     Flags    db 0
     String   db 'Hi there!'
```

5.32.4 См. также

EXTERN, GLOBAL, TEXT, UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

5.33 _ _IDLOCS - Установка значения ID

5.33.1 Синтаксис

```
_ _idlocs <expr>
```

5.33.2 Описание

Для микроконтроллеров PIC12CXXX, PIC14000 и PIC16XXX указывается 4 шестнадцатиразрядных полубайта ID. Для микроконтроллеров семейства PIC18CXXX указывается два шестнадцатиразрядных значения ID <expr1> и <expr2>. Данная директива не имеет силу для микроконтроллеров PIC17CXXX.

Например, если значение <expr> равно 1AF, то первое значение – нуль (в младшем адресе); второе – один; третье – десять; четвертое - пятнадцать. Прежде чем использовать эту директиву необходимо указать тип микроконтроллера директивой LIST или PROCESSOR.

5.33.3 Пример

```
     _ _idlocs H'1234'
```

5.33.4 См. также

LIST, PROCESSOR, _ _CONFIG

5.34 IF - Начало блока условия

5.34.1 Синтаксис

```
if <expr>
```

5.34.2 Описание

Начало выполнения условного блока. Если выражение <expr> оценивается истинным, 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 не нулевой результат выражения, а логическая ЛОЖЬ нулевой результат.

5.34.3 Пример

```
     if version == 100
        movlw 0x0a
        movwf io_1
     else
        movlw 0x01a
        movwf io_2
     endif
```

5.34.4 См. также

ELSE, ENDIF

5.35 IFDEF – Выполнение, если определена символьная метка

5.35.1 Синтаксис

```
ifdef <label>
```

5.35.2 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.

5.35.3 Пример

```
      #define testing 1                    ; установить testing “on”
      :
      :
      ifdef     testing
                <execute test code>        ; выполняемый код
      endif
```

5.35.4 См. также

#DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

5.36 IFNDEF - Выполнение, если символьная метка не определена

5.36.1 Синтаксис

```
ifndef <label>
```

5.36.2 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.

5.36.3 Пример

```
      #define   testing1                   ; установить testing “on”
      :
      :
      #undefine testing1                   ; установить testing “off”
                ifndef testing
                :                          ; выполняемый код
                :
                endif

                end                        ; конец условия
```

5.36.4 См. также

#DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

5.37 INCLUDE - Подключение дополнительного исходного файла

5.37.1 Синтаксис

```
include <<include_file>>
include "<include_file>"
```

5.37.2 Описание

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

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

5.37.3 Пример

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

5.38 LIST - Список параметров

5.38.1 Синтаксис

```
list [<list_option>,...,<list_option>]
```

5.38.2 Описание

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

Параметр	Значение по умолчанию	Описание
b=nnn	8	Число пробелов при табуляции
c=nnn	132	Число символов в строке
f=<format>	INHX8M	Формат выходного HEX файла <format>: INHX32, INHX8M, INHX8S
free	FIXED	Использование свободного формата печати
fixed	FIXED	Использование фиксированного формата печати
mm={ON|OFF}	On	Включать карту памяти в файл листинга программы
n=nnn	60	Число строк на одной страницы
p=<type>	НЕТ	Тип микроконтроллера. Например, PIC16C54
r=<radix>	hex	Система счисления по умолчанию: hex, dec, oct
st={ON|OFF}	On	Включать таблицу символов в файл листинга программы
t={ON|OFF}	Off	Усечение длинных строк
w={0|1|2}	0	Установка уровня сообщений. См. ERRORLEVEL
x={ON|OFF}	On	Включать полный текст макроса

Примечание. Все параметры директивы LIST указываются в десятичных числах.

5.38.3 Пример

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

5.38.4 См. также

ERRORLEVEL, EXPAND, NOEXPAND, NOLIST, PROCESSOR, RADIX

5.39 LOCAL - Объявить локальную переменную макроса

5.39.1 Синтаксис

```
local <label>[,<label>]
```

5.39.2 Описание

Объявляет, что указанные элементы данных должны рассматриваться только внутри макроса. <label> может иметь имя идентичное другой метки объявленной в основной программе, при этом ошибки не возникнет.

Если макрокоманда вызывается рекурсивно, то при каждом обращении будет создаватся собственная локальная копия.

5.39.3 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 макроса
```

5.39.4 См. также

ENDM, MACRO

5.40 MACRO - Определить макрос

5.40.1 Синтаксис

```
<label> macro [<arg>,...,<arg>]
```

5.40.2 Описание

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

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

Дополнительную информацию смотрите в главе 7.

5.40.3 Пример

```
     Read      macro  device, buffer, count
               movlw  device
               movwf  ram_20
               movlw  buffer
               movwf  ram_21
               movlw  count
               call   sys_21
               endm
```

5.40.4 См. также

ELSE, ENDIF, ENDM, EXITM, IF, LOCAL

5.41 _ _MAXRAM - Определяет максимальный объем ОЗУ

5.41.1 Синтаксис

```
_ _maxram <expr>
```

5.41.2 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.

5.41.3 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
                      ; вызовет сообщение об ошибке
```

5.41.4 См. также

_ _BADRAM

5.42 MESSG - Сформировать сообщение

5.42.1 Синтаксис

```
messg "<message_text>"
```

5.42.2 Описание

Включает информационное сообщение (не более 80 символов) в файл листинга программы. Директива не формирует коды ошибок.

5.42.3 Пример

```
     mssg_macro macro
               messg "mssg_macro-001 invoked without argument"
     endm
```

5.42.4 См. также

ERROR

5.43 NOEXPAND - Не разворачивать текст макроса

5.43.1 Синтаксис
```
noexpand
```

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

5.43.3 См. также
EXPAND

5.44 NOLIST - Выключить вывод в файл листинга

5.44.1 Синтаксис
```
nolist
```

5.44.2 Описание
Выключить вывод в файл листинга.

5.44.3 См. также
LIST

5.45 ORG - Установить адрес программы

5.45.1 Синтаксис
```
<label> org <expr>
```

5.45.2 Описание
Установить <expr> адрес программы. Если указана метка <label> то она будет иметь адрес <expr>. Если в тексте программы не встречается директива ORG, то считается, что программа начинается с нулевого адреса.
Для микроконтроллеров PIC18CXXX допускается только значение <expr>.
Данная директива не может быть использована при генерации объектного файла.

5.45.3 Пример
```
int_1 org 0x20
   ; Вектор с адресом 20
int_2 org int_1+0x10
   ; Вектор с адресом 30
```

5.45.4 См. также
FILL, RES

5.46 PAGE - Вставить страницу в файл листинга программы

5.46.1 Синтаксис
```
page
```

5.46.2 Описание
Вставить страницу в файл листинга программы.

5.46.3 См. также
LIST, SUBTITLE, TITLE

5.47 PAGESEL - Произвести выбор страницы

5.47.1 Синтаксис

```
pagesel <label>
```

5.47.2 Описание

Используется при генерации объектного файла. Линкер 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. Не обязательно 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.

Дополнительную информацию смотрите в главе 6.

5.47.3 Пример

```
pagesel GotoDest
goto GotoDest
....
pagesel CallDest
call CallDest
```

5.47.4 См. также

BANKISEL, BANKSEL

5.48 PROCESSOR - Выбор типа микроконтроллера

5.48.1 Синтаксис

```
processor <processsor_type>
```

5.48.2 Описание

Устанавливает тип микроконтроллера <processsor_type>.

5.48.3 Пример

```
processor 16C54
```

5.48.4 См. также

LIST

5.49 RADIX - Система счисления по умолчанию

5.49.1 Синтаксис

```
radix <default_radix>
```

5.49.2 Описание

Указывает системы счисления по умолчанию: hex, dec, oct. Первоначально установлена шестнадцатеричная система счисления (hex).

5.49.3 Пример

```
radix dec
```

5.49.4 См. также

LIST

5.50 RES - Резервирование памяти

5.50.1 Синтаксис
```
[<label>] res <mem_units>
```

5.50.2 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.

5.50.3 Пример
```
      buffer res 64
```

5.50.4 См. также
FILL, ORG

5.51 SET - Определение константы

5.51.1 Синтаксис
```
<label> set <expr>
```

5.51.2 Описание
Присваивает <label> значение <expr> для использования в выражениях MPASM. Действует аналогично директиве EQU за исключением того, что значение <label> может быть впоследствии переопределено директивой SET.

5.51.3 Пример
```
      area      set 0
      width     set 0x12
      length    set 0x14
      area      set length * width
      length    set length + 1
```

5.51.4 См. также
EQU, VARIABLE

5.52 SPACE - Вставить пустые строки

5.52.1 Синтаксис
```
space <expr>
```

5.52.2 Описание
Вставляет в файл листинга программы <expr> пустых строк.

5.52.3 Пример
```
      space 3
```

5.52.4 См. также
LIST

5.53 SUBTITLE - Определение подзаголовка программы

5.53.1 Синтаксис
```
subtitle "<sub_text>"
```

5.53.2 Описание
<sub_text> последовательность ASCII символов (не более 60) вставляемая во вторую строку колонтитула страниц файла листинга программ.

5.53.3 Пример
```
      subtitle "diagnostic section"
```

5.53.4 См. также
TITLE

5.54 TITLE - Определение заголовка программы

5.54.1 Синтаксис

```
title "<title_text>"
```

5.54.2 Описание

<title_text> последовательность ASCII символов (не более 60) вставляемая в верхнюю строку колонтитула страниц файла листинга программ.

5.54.3 Пример

```
     title "<title_text>"
```

5.54.4 См. также

LIST, SUBTITLE

5.55 UDATA - Начало инициализации данных с обычным размещением в памяти (для объектного файла)

5.55.1 Синтаксис

```
[<label>] udata [<RAM address>]
```

5.55.2 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.

Ни какой 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 тоже имя.

Дополнительную информацию смотрите в главе 6.

5.55.3 Пример

```
             udata
     Var1    res 1
     Double  res 2
```

5.55.4 См. также

EXTERN, GLOBAL, IDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

5.56 UDATA_ACS - Начало инициализации данных быстрого доступа (для объектного файла)

5.56.1 Синтаксис

```
[<label >] udata_acs [<RAM address>]
```

5.56.2 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.

Ни какой 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 тоже имя.

Дополнительную информацию смотрите в главе 6.

5.56.3 Пример

```
             udata_acs
     Var1    res 1
     Double  res 2
```

5.56.4 См. также

EXTERN, GLOBAL, IDATA UDATA, UDATA_OVR, UDATA_SHR

5.57 UDATA_OVR - Начало инициализации временных данных (для объектного файла)

5.57.1 Синтаксис

```
[<label >] udata_ovr [<RAM address>]
```

5.57.2 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 другими udata_ovr секциями с таким же именем. Это оптимальный метод резервирования памяти под временные переменные.

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

Ни какой 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 тоже имя в одном исходном файле.

Дополнительную информацию смотрите в главе 6.

5.57.3 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 Temp1, Temp2
LongTemp2 res 2         ; эта переменная имеет тот же адрес что и Temp3
```

5.57.4 См. также

EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_SHR

5.58 UDATA_SHR - Начало инициализации разделяемых данных (для объектного файла)

5.58.1 Синтаксис

```
[<label >] udata_shr [<RAM address>]
```

5.58.2 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.

Ни какой 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 тоже имя.

Дополнительную информацию смотрите в главе 6.

5.58.3 Пример

```
Temps udata_shr
Temp1 res 1
Temp2 res 1
Temp3 res 1
```

5.58.4 См. также

EXTERN, GLOBAL, IDATA, UDATA, UDATA_ACS, UDATA_OVR

5.59 #UNDEFINE - Отменить замену текста

5.59.1 Синтаксис

```
#undefine <label>
```

5.59.2 Описание

Отменяет определенную директивой #DEFINE замену текста. Указанная в директивы метка должна быть предварительно определена в MPASM. После выполнения директивы метка удаляется из таблицы символов.

5.59.3 Пример

```
#define length 20
:
:
#undefine length
```

5.59.4 См. также

#DEFINE, IFDEF, INCLUDE, IFNDEF

5.60 VARIABLE - Определение символьной переменной

5.60.1 Синтаксис

```
variable <label>[=<expr>,...,<label>[=<expr>] ]
```

5.60.2 Описание

Директивой объявляются переменные, используемые в выражениях MPASM. Переменные и константы могут использоваться в выражениях попеременно.

Директива VARIABLE определяет переменную, которая является функциональным эквивалентом созданной директивой SET. Отличием является то, что в директиве VARIABLE не требуется указывать конкретное значение при объявлении.

Примечание. Значение переменных не может быть обновлено в пределах операнда. Изменение переменных должно быть выполнено на отдельных строках программы.

5.60.3 Пример

Пример смотрите в описании директивы CONSTANT

5.60.4 См. также

CONSTANT, SET

5.61 WHILE - Цикл While

5.61.1 Синтаксис

```
while <expr>
:
:
endw
```

5.61.2 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 не нулевой результат выражения, а логическая ЛОЖЬ нулевой результат. Длина цикла не может быть более 100 строк программы. Максимальное число повторов программы внутри цикла 256.

5.61.3 Пример

```
test_mac  macro count
          variable i
i = 0
          while i < count
          movlw i
i += 1
          endw
          endm
start
          test_mac 5
          end
```

5.61.4 См. также

ENDW, IF

6. Использование MPASM для создания перемещаемых объектов

6.1 Введение

Начиная с версии MPASM v2.00 и MPLINK v1.00, пользователи имеют возможность выполнять связь объектных модулей для генерации HEX кода программы. Написание исходного текста программы, который будет компилирован в объектный файл, несколько отличается от создания программы с непосредственной компиляцией в HEX файл. Подпрограммы, разработанные для компиляции непосредственно в HEX файл, потребуют незначительных изменений для получения корректного перемещаемого объектного модуля.

6.2 Основные части раздела

Файлы сценария
Память программ
Операнды инструкций
Распределение ОЗУ
Биты конфигурации и ID
Обращение к меткам других модулей
Работа с банками ОЗУ
Недопустимые директивы
Формирование объектного файла
Пример программы

6.3 Файлы сценария

Microchip разработал стандартные файлы сценария (например p17c756.inc), которые должны использоваться при получении объектного кода. Данные файлы определяют параметры и набор регистров специального назначения конкретного микроконтроллера.

6.4 Память программ

Тексту программы должна предшествовать директива CODE, определяющая секцию перемещаемого кода.

6.4.1 Абсолютный код

```
Start   CLRW
        OPTION
        :
```

6.4.2 Перемещаемый код

```
        CODE
Start   CLRW
        OPTION
        :
```

Если более чем одна секция CODE определена в исходном файле, то каждая секция должна иметь свое уникальное имя. Если имя секции не определено, то данной секции присваивается имя .code.

Каждая секция памяти программ должна быть смежна в пределах исходного текста. Секции нельзя разбивать на части. Физический адрес кода в памяти программ может быть установлен, указав необязательный параметр директивы CODE <ROM address>.

Указание физического адреса может быть необходимо в следующих случаях:

- определение вектора прерывания;
- для расположения кода программы в пределах страницы памяти программ.

6.4.3 Пример перемещаемого кода

```
Reset   CODE H'0lFF'
        GOTO Start

Main    CODE
        CLRW
        OPTION
```

6.5 Операнды инструкций

Существуют некоторые ограничения по использованию операндов в инструкциях. Операнды в инструкциях должны иметь следующую форму:

```
[HIGH|LOW|UPPER] (<relocatable symbol> + <constant offset>)
```

Где:
`<relocatable symbol>` - любая метка, которая определяет адрес в памяти программ или данных;
`<constant offset>` - выражение, которое разрешено и имеющее значение в пределах от –32768 до +32767.

Значение `<relocatable symbol>` или `<constant offset>` может быть пропущено.

Операнды формы:
`<relocatable symbol> - <relocatable symbol>`
Значение будет уменьшаться до некоторой постоянной, если оба символа определены в том же самом коде или секции данных.

Для обозначения изменяемых битов в выражении используются дополнительные указатели:
LOW – изменяются биты 0-7;
HIGH – изменяются биты 8-15;
UPPER – изменяются биты 16-21.

6.6 Распределение ОЗУ

Распределение ОЗУ должно быть выполнено в секции данных. Существует пять типов секций данных:

UDATA – Неинициализированные данные. Это наиболее общий тип размещения данных. Ячейки, зарезервированные в этой секции, не инициализируются, а обращение к данным производится за счет меток или косвенно.

UDATA_ACS – Неинициализированные данные доступа. Эта секция данных используется для переменных, которые будут помещены в память быстрого доступа микроконтроллеров семейства PIC18CXXX. Для обращения к памяти быстрого доступа используются специальные команды.

UDATA_OVR – Неинициализированные временные данные. Используется для переменных, которые могут иметь одинаковый адрес в этом же или других связываемых объектных модулях. Типичное использование данной секции – временные переменные.

UDATA_SHR – Неинициализированные разделяемые данные. Данные этой секции будут размещены в ячейках ОЗУ, которые доступны из всех банков памяти данных.

IDATA – Инициализированные данные. Линкер сформирует таблицу поиска, которая позволит присвоить переменным указанные в этой секции значения. Обращение к данным выполняется только с использованием меток или косвенной адресацией.

В примерах показаны варианты определения переменных.

6.6.1 Абсолютный код

```
CBLOCK 0x20
   InputGain, OutputGain      ;Управление циклами
   HistoryVector              ;Должно равняться нулю
   Templ, Temp2, Temp3        ;Используются в вычислениях
ENDC
```

6.6.2 Перемещаемый код

```
                IDATA
HistoryVector   DB     0

                UDATA
InputGain       RES    1
OutputGain      RES    1

                UDATA_OVR
Templ           RES    1
Temp2           RES    1
Temp3           RES    1
```

Адрес инициализации данных в памяти можно установить, указав необязательный параметр <RAM address>. Если в программе используется тип секции данных больше одного раза, то для каждой секции необходимо указать уникальное имя. Если название секции не указано, то им присваиваются имена: .idata, .udata, .udata_acs, .udata_ovr, .udata_shr.

При инициализации данных в секции IDATA директивы DB, DW и DATA могут использоваться для определения данных. DB определяют последовательность байт в памяти. DW и DATA определяют последовательности слов данных (младший байт, старший байт).

В следующем примере показано как будут инициализированы данные:

6.6.3 Перемещаемый код

```
                        00001               LIST p=17C44
                        00002               IDATA
0000 01 02 03           00003  Bytes        DB 1,2,3
0003 34 12 78 56        00004  Words        DW H’1234’,H’5678’
0007 41 42 43 00        00005  String       DB “ABC”, 0
```

6.7 Биты конфигурации и ID

Для определения битов конфигурации и ID битов применяйте директивы _ _CONFIG и _ _IDLOCS. Только в одном объектном модуле проекта можно использовать указанные директивы до секции CODE. После использования данных директив текущая секция не определена.

6.8 Обращение к меткам других модулей

Метки, которые определены в одном модуле для использования в других объектных модулях, должны быть отмечены директивой GLOBAL после их объявления.

Модули, использующие эти метки, должны объявить их директивой EXTERN.

Пример использования директив GLOBAL и EXTERN.

6.8.1 Подключаемый модуль

```
                    UDATA
InputGain           RES 1
OutputGain          RES 1
                    GLOBAL InputGain, OutputGain

                    CODE
Filter
                    GLOBAL Filter
                    :
```

6.8.2 Основной модуль

```
                    EXTERN InputGain, OutputGain, Filter

                    UDATA
Reading             RES 1

                    CODE
...
                    MOVLW  GAIN1
                    MOVWF  InputGain
                    MOVLW  GAIN2
                    MOVWF  OutputGain
                    MOVF   Reading,W
                    CALL   Filter
```

6.9 Работа с банками и страницами памяти

В большинстве случаев используются несколько банков ОЗУ и страниц памяти программ микроконтроллера. При этом необходимо настраивать нужный рабочий банк или страницу памяти программ для обращения к метке. Т.к. при формировании объектного файла предварительно не известно размещение переменных и меток в памяти программ были разработаны две директивы BANKSEL и PAGESEL. Данные директивы указывают линкеру сгенерировать код выбора нужного банка данных или страницы памяти программ.

Примеры использования указанных директив показаны ниже.

6.9.1 Абсолютный код

```
            LIST P=12C509
#include  “P12C509.INC”

Varl        EQU H'10'
Var2        EQU H'30'
             ...
            MOVLW InitialValue
            BCF FSR, 5
            MOVWF Varl
            BSF FSR, 5
            MOVWF Var2
            BSF STATUS, PA0
            CALL Subroutine
            ...
Subroutine CLRW ;In Page 1
            ...
            RETLW 0
```

6.9.2 Перемещаемый код

```
            LIST P=12C509
     #include “P12C509.INC”

            UDATA
     Varl   RES 1
     Var2   RES 1
            ...
            CODE
            MOVLW InitialValue
            BANKSEL Varl
            MOVWF Varl
            BANKSEL Var2
            MOVWF Var2
            PAGESEL Subroutine
            CALL Subroutine
            ...
     Subroutine CLRW
            ...
            RETLW 0
```

6.10 Недопустимые директивы

Все директивы доступны для формирования объектного файла, за исключением директивы ORG. Данную директиву необходимо заменить на директиву CODE, как показано ниже.

6.10.1 Абсолютный код

```
     Reset  ORG H’01FF’
            GOTO Start
```

6.10.2 Перемещаемый код

```
     Reset  CODE H’0lFF’
            GOTO Start
```

6.11 Формирование объектного файла

Получить объектный файл можно компиляцией отредактированного исходного текста программы разными версиями MPASM:

1. При использовании MPASM для операционной системы Windows, проверьте установку флага “Object File”.
2. В случае использования DOS версии MPASM с управлением из командной строки – введите параметр /o.
3. В оконной версии MPASM для DOS установите “Yes” в строке “Assemble to Object File”.

Полученный объектный файл будет с расширением .o.

6.12 Пример программы

6.12.1 Абсолютный код

```
            LIST P=16C54
            #INCLUDE “P16C5x.INC”

            cblock H ‘020’
mulcnd      RES 1 ; 8-битный множитель
mulplr      RES 1 ; 8-битное множимое
H_byte      RES 1 ; Старший байт 16-битного результата
L_byte      RES 1 ; Младший байт 16-битного результата
count       RES 1 ; Счетчик

mpy         clrf H_byte
            clrf L_byte
            movlw 8
            movwf count
            movf mulcnd,w
            bcf STATUS,C
Loop        rrf mulplr,F
            btfsc STATUS,C
            addwf H_byte,F
            rrf H_byte,F
            rrf L_byte,F
            decfsz count,F
            goto loop

            retlw 0
;*****************************************************
; Тестовая программа
;*****************************************************
start       clrw
            option

main        movf PORTB,w
            movwf mulplr
            movf PORTB,W
            movwf mulcnd

call_m      call mpy

            goto main

            ORG 01FFh
            goto start

      END
```

Подпрограмма умножения двух 8-разрядных чисел, может использоваться и в других проектах. Создав объектный файл программы, он может быть подключен к проекту, когда это необходимо. Представленный файл может быть разделен на две части: 1 - вызывная часть, включаемая в основной текст; 2 – основная программа, которая может быть включена в библиотеку.

6.12.2 Перемещаемый код, основная часть

```
             LIST P=16C54
            #INCLUDE “P16C5x.INC”

            EXTERN mulcnd, mulplr, H_byte, L_byte
            EXTERN mpy

            CODE

start       clrw
            option

main        movf PORTB, W
            movwf mulplr
            movf PORTB, W
            movwf mulcnd

            call_m call mpy
            goto main

Reset       CODE H’0lFF’
            goto start
END
```

6.12.3 Перемещаемый код, библиотечная часть

```
                LIST P=16C54
                #INCLUDE “P16C5x.INC”

                UDATA
        mulcnd  RES l ; 8-битный множитель
        mulplr  RES 1 ; 8-битное множимое
        H_byte  RES 1 ; Старший байт 16-битного результата
        L_byte  RES 1 ; Младший байт 16-битного результата
        count   RES 1 ; Счетчик
                GLOBAL mulcnd, mulplr, H_byte, L_byte

                CODE
        mpy
                GLOBAL mpy

                clrf H_byte
                clrf L_byte
                movlw 8
                movwf count
                movf muland, W
                bcf STATUS, C
        loop    rrf mulplr, F
                btfsc STATUS, C
                addwf H_byte, F
                rrf H_byte, F
                rrf L_byte, F
                decfsz count, F
                goto loop

                retlw 0
        END
```

7. Язык макрокоманд

7.1 Введение

Макрос – определенный пользователем набор инструкций и директив, которые будут подставлены в исходный текст программы при каждом его вызове.

Макрос состоит из команд ассемблера и директив, с возможностью указания аргументов, что делает написание программ более гибким.

Основные аргументу в пользу использования макросов:

- более высокий уровень написания программы;
- текст программы удобен для чтения;
- уменьшение числа ошибок;
- упрощение возможных изменений.

Использование макросов может быть полезно при: генерации таблиц, применении часто повторяющегося кода, выполнении сложных операций.

7.2 Основные части раздела

Синтаксис макрокоманд
Директивы макрокоманд
Замена текста
Использование макросов
Примеры программ

7.3 Синтаксис макрокоманд

Макросы имеют следующий синтаксис:

```
<label> macro [<arg1>,<arg2> ..., <argn>]
:
:
endm
```

Где:

<label> - метка MPASM;

<arg> - любое количество необязательных аргументов.

При вызове макрокоманды, значение аргументов в теле макроса будут заменены везде, где встречается имя данного параметра.

В тело макроса могут быть включены любые инструкции микроконтроллера, директивы или макро-директивы MPASM (например, LOCAL). Описание директив смотрите в главе 5. MPASM продолжает обрабатывать инструкции и директивы тела макроса до тех пор, пока не встретит директиву EXITM или ENDM.

Примечание. Для вызова макроса не допускается использование команд ветвления.

7.4 Директивы макрокоманд

Часть директив используется для формирования макрокоманды, и не могут использоваться вне макроса:

MACRO
LOCAL
EXITM
ENDM

При написании макрокоманд Вы можете использовать эти и другие директивы поддерживаемые MPASM.

Примечание. Предыдущий синтаксис написания точечных макрокоманд больше не поддерживается. В целях совместимости с предыдущими версиями поддерживается формат ассемблера ASM17. Для обеспечения восходящей совместимости рекомендуется использовать только директивы MPASM, описанные в данном документе.

7.5 Замена текста

В пределах тела макроса может быть выполнена замена цепочки символов и значения выражения.

<arg> - замена имени аргумента, используется как часть макро обращения;
#v(<expr>) – возвращает значение <expr>. Используется для создания уникальной переменной с общими приставками или суффиксами. Не может использоваться в директивах условий (IDEF, WHILE).
Эти параметры могут использоваться в теле макроса, исключая его выражения.

Пример макроса

```
define_table    macro
                local a = 0
                while a < 3
entry#v(a)      dw 0
a += 1
                endw
                endm
```

Выполненные действия:

```
entry0    dw 0
entry1    dw 0
entry2    dw 0
entry3    dw 0
```

7.6 Использование макросов

Однажды определенный макрос может использоваться в любой части программы в пределах исходного модуля.
Синтаксис вызова макроса:

```
<macro_name> [<arg>, ..., <arg>]
```

Где:
<macro_name> - название предварительно определенного макроса;
<arg> - требуемые аргументы.

Макрозапрос не занимает место в памяти программ. Однако тело макроса будет расположено с текущего адреса.
Запятые могут использоваться для резервирования положения аргумента. Список аргументов заканчивается символом пробел или точка с запятой.
Директива EXITM (см. главу 5) обеспечивает дополнительный метод завершения макроса с игнорированием директивы ENDM. Если используется вложенная структура макросов, то по директиве EXITM вызовет переход на верхний уровень макрокоманд.

7.7 Примеры программ

7.7.1 Умножение двух 8-разрядных чисел

Пример макроса умножения двух 8-разрядных чисел для микроконтроллера PIC17C42 с минимизацией времени вычисления.

```
        subtitle “macro definitions”
        page

        multiply  macro arg1, arg2, dest_hi, dest_lo

                  local i = 0          ; объявление локальной переменной

                  movlw arg1           ; получение множимого
                  movwf mulplr

                  movlw arg2           ; сохранение множителя в регистре W

                  clrf dest_hi         ; очистка регистров результата
                  clrf dest_lo

                  bcf ALUSTA,C         ; сброс бита C

                  while i < 8          ; цикл, пока 8 бит не будут сдвинуты
                  rrcf mulplr
                  btfsc ALUSTA,C
                  addwf dest_hi
                  rrcf dest_hi
                  rrcf dest_lo
        i += 1
                  endw                 ; конец вычислений
                  endm                 ; конец макроса
```

В макросе объявляется четыре требуемых аргумента. Директивой LOCAL назначается местная переменная “i”, которая будет использоваться как счетчик. При инициализации значение переменной “i” равно нулю.

Макрокоманда выполняет умножение по алгоритму, в котором используется сдвиг вправо и дополнение для каждого разряда 8-битного множителя. Директива WHILE используется для организации цикла внутри макроса, пока значение переменной ‘i’ меньше 8.

Конец цикла отмечен директивой ENDW. Выполнение инструкций внутри цикла будет происходить до тех пор, пока условие WHILE не станет ложным.

Завершение макроса отмечено директивой ENDM.

7.7.2 Сравнение констант

Пример программы сравнения переменных с использованием макрокоманды.

```
include “16cxx.reg”

cfl_jge   macro file, con, jump_to
          movlw con & 0xff
          subwf file, w
          btfsc status, carry
          goto jump_to
          endm
```

Вызов процедуры макроса:

```
          cfl_jge switch_val, max_switch, switch_on
```

Подставленный текст программы:

```
          movlw max_switch & 0xff
          subwf switch_val, w
          btfsc status, carry
          goto switch_on
```

8. Синтаксис выражений и операций

8.1 Введение

В главе описываются форматы выражений, синтаксис и операторы MPASM.

8.2 Основные части раздела

Текстовые строки
Числовые константы и системы счисления
Арифметические операции
High/Low/Upper операции
Операции инкремента/декремента

8.3 Текстовые строки

Текстовые строки (длиной не более 255 символов) могут состоять из знаков ASCII (в диапазоне от 0 до 127). Если найден символ окончания строки, то определение строки считается завершено. Если символ окончания текстовой строки не найден, то окончание строки считается в конце линии.

Увеличить длину символьной строки можно дополнительным использованием директивы DW.

Директива DW сохраняет текстовую строку в последовательности слов памяти программ. Если текстовая строка имеет нечетное количество символов, то для директив DW и DATA последний байт будет равняться 00h.

Если текстовая строка должна иметь один литеральный операнд, то должен быть указан один символ, иначе произойдет ошибка.

Пример сохранения текстовой строки.

```
7465 7374 696E              dw “testing output string one\n”
6720 6F75 7470
7574 2073 7472
696E 6720 6F6E
650A
                #define     str “testing output string two”
B061                        movlw ‘a’
7465 7374 696E              data “testing first output string”
6720 6669 7273
7420 6F75 7470
7574 2073 7472
696E 6700
```

MPASM принимает некоторые ANSI ‘C’ последовательности для формирования некоторых характерных знаков.

Последовательность символов	Описание	HEX значение
\a	Предупреждение	07
\b	Возврат на один символ	08
\f	Формат	0C
\n	Новая строка	0A
\r	Перевод каретки	0D
\t	Горизонтальная табуляция	09
\v	Вертикальная табуляция	0B
\\	Наклонная черта влево	5C
\?	Вопрос	3F
\’	Апостроф	27
\”	Кавычки	22
\0OO	Восьмеричное значение	
\xHH	Шестнадцатеричное значение	

8.4 Числовые константы и системы счисления

MPASM поддерживает числа следующих систем счисления:

- шестнадцатеричные;
- десятичные;
- восьмеричные;
- двоичные;
- символы ASCII.

Если система счисления явно не определена, то при компиляции исходного текста программы по умолчанию используется шестнадцатеричная система счисления.

При определении константы, ей можно присвоить положительное или отрицательное значение (добавив знак + или – соответственно). По умолчанию считается, что значение константы положительное число.

Примечание. MPASM работает с промежуточными значениями в выражениях как с 32-разрядными числами. В случае превышения указанного предела будет сформировано соответствующее предупреждение.

В таблице представлен синтаксис числовых значений в различных системах счисления.

Тип	Синтаксис	Пример
Шестнадцатеричный	H’<hex_digits>’ 0x<hex_digits>	H’9f’ 0x9f
Десятичный	D’<digits>’	D’100’
Восьмеричный	O’<octal_digits>’	O’777’
Двоичный	B’<binary_digits>’	B’00111001’
ASCII	’<character>’ A’<character>’	’C’ A’C’

8.5 Арифметические операции

Список арифметических операций

Символ	Описание	Пример
$	Изменение счетчика программы PC	`goto $ + 3`
(	Левая скобка	`1 + (d * 4)`
)	Правая скобка	`(Length + 1) * 256`
!	НЕ с логическим дополнением	`if ! (a == b)`
-	Отрицание с дополнением двух	`-1 * Length`
~	Дополнение	`flags = ~flags`
High	Возвращает старший байт	`movlw high CTR_Table`
Low	Возвращает младший байт	`movlw low CTR_Table`
Upper	Возвращает верхний байт	`movlw upper CTR_Table`
*	Умножение	`a = b * c`
/	Деление	`a = b / c`
%	Модуль	`entry_len = tot_len % 16`
+	Сложение	`tot_len = entry_len * 8 + 1`
-	Вычитание	`entry_len = (tot - 1) / 8`
<<	Сдвинуть влево	`flags = flags << 1`
>>	Сдвинуть вправо	`flags = flags >> 1`
>=	Больше или равно	`if entry_idx >= num_entries`
>	Больше	`if entry_idx > num_entries`
<	Меньше	`if entry_idx < num_entries`
<=	Меньше или равно	`if entry_idx <= num_entries`
==	Равно	`if entry_idx == num_entries`
=	Не равно	`if entry_idx != num_entries`
&	Побитовое И	`flags = flags & ERROR_BIT`
^	Побитовое исключающее ИЛИ	`flags = flags ^ ERROR_BIT`
|	Побитовое ИЛИ	`flags = flags | ERROR_BIT`
&&	Логическое И	`if (len == 512) && (b == c)`
||	Логическое ИЛИ	`if (len == 512) || (b == c)`
=	Присвоить значение	`entry_index = 0`
+=	Сложить и присвоить значение	`entry_index += 1`
-=	Вычесть и присвоить значение	`entry_index -= 1`
*=	Умножить и присвоить значение	`entry_index *= entry_length`
/=	Разделить и присвоить значение	`entry_total /= entry_length`
%=	Взять модуль и присвоить значение	`entry_index %= 8`
<<=	Сдвинуть влево и присвоить значение	`flags <<= 3`
>>=	Сдвинуть вправо и присвоить значение	`flags >>= 3`
&=	Выполнить И и присвоить значение	`flags &= ERROR_FLAG`
|=	Выполнить ИЛИ и присвоить значение	`flags |= ERROR_FLAG`
^=	Выполнить исключающее ИЛИ и присвоить значение	`flags ^= ERROR_FLAG`
++	Инкремент	`i ++`
--	Декремент	`i --`

8.6 High/Low/Upper операции

8.6.1. Синтаксис

```
high <operand>
low <operand>
upper <operand>
```

8.6.2 Описание

Данные операнды возвращают указанный байт из многобайтной переменной. Они предназначены для динамического вычисления указателя и выполнения инструкций табличного чтения/записи.

8.6.3 Пример

```
movlw low size
movpf wreg, low size_lo
movlw high size
movpf wreg, high size_hi
```

8.7 Операции инкремента/декремента

8.7.1 Синтаксис

```
<variable>++
<variable>--
```

8.7.2 Описание

Выполняют инкремент/декремент значения переменной. Данные операнды должны использоваться на отдельной строке и не могут быть включены в выражения.

8.7.3 Пример

```
LoopCount = 4
          while  LoopCount > 0
                 rlf Reg, f
LoopCount --
          endw
```

Приложение A. Формат HEX файлов.

A.1 Введение

В данном разделе будут рассмотрены форматы генерируемых MPASM HEX файлов.

A.2 Основные части раздела

Intel HEX формат INHX8M (для стандартных программаторов)
Intel Split HEX формат INHX8S (для ODD/EVEN ROM программаторов)
Intel HEX формат INHX32 (для 16-битных программаторов)

A.3 Intel HEX формат INHX8M (.HEX)

В этом формате сохраняются 8-разрядные байты младшие байты, с поддержкой старшего байта. Т.к. по каждому адресу содержится только 8-разрядные байты, объем адресов увеличен вдвое. Данный формат используется для передачи кода программы микроконтроллеров PICmicro в программаторы PRO MATE II, PICSTART и программаторы других производителей.

Каждая строка начинается с 9 знаковой приставки и заканчивается 2-х знаковой контрольной суммой.

```
:BBAAAATTHHHH....HHHCC
```

Где:

BB	- две цифры шестнадцатеричного байта, определяет количество байт в строке;
AAAA	- четыре цифры шестнадцатеричного адреса записи данных;
TT	- две цифры шестнадцатеричного байта, указатель конца файла (00 – данные, 01 – конец файла);
HH	- две цифры шестнадцатеричного байта данных или комбинация младший/старший байт слова;
CC	- две цифры шестнадцатеричного байта, контрольная сумма, которая является дополнением ко всем предшествуемым байтам в строке.

Пример файла

```
<file_name>.HEX
:10000000000000000000000000000000000000F0
:0400100000000000EC
:10003200000280040006800A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

A.4 Intel Split HEX формат INHX8S (.HXL/.HXH)

8-разрядные данные сохраняются в двух выходных файлах: .HXL и HXH. Аналогичен формату INHX8M за исключением того, что старшие и младшие байт 16-разрядного слова сохраняются в разных файлах. Младшие байты сохраняются в файле с расширением .HXL, а старшие в файле с расширением .HXH. Например, необходимо сохранить 16-разрядные данные в двух 8-разрядных EPROM микросхемах, для этого необходимо иметь один файл для младших байт, а другой файл для старших байт данных.

Пример файлов

```
<file_name>.HXL
:0A000000000000000000000000000000F6
:10001900002840688A8E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C8080808034303E8E8FFD0
:03003900FFFF19AD
:00000001FF

<file_name>.HXH
:0A000000000000000000000000000000F6
:10001900000000000000000101010102020202CA
:10002900020203030303030404040405060707 0883
:0300390008080AAA
:00000001FF
```

A.5 Intel HEX формат INHX32 (.HEX)

Расширенный 32-разрядный шестнадцатеричный формат является подобием шестнадцатеричному 8-разрядному HEX формату, описанному выше. За исключением того, что расширен линейный отсчет адреса в памяти (указание старших 16 битов адреса). Это сделано для того, чтобы адресовать адресное пространство объемом больше 32 Кслов в 16 разрядных микроконтроллерах.

Каждая строка начинается с 9 знаковой приставки и заканчивается 2-х знаковой контрольной суммой.

```
:BBAAAATTHHHH....HHHCC
```

Где:

BB - две цифры шестнадцатеричного байта, определяет количество байт в строке;

AAAA - четыре цифры шестнадцатеричного адреса записи данных;

TT - две цифры шестнадцатеричного байта, указатель типа строки;

- 00 – данные;
- 01 – конец файла;
- 02 – адрес сегмента;
- 04 – линейный адрес;

HH - две цифры шестнадцатеричного байта данных или комбинация младший/старший байт слова;

CC - две цифры шестнадцатеричного байта, контрольная сумма, которая является дополнением ко всем предшествуемым байтам в строке.

Приложение B. Сообщения MPASM

B.1 Введение

Ниже приведен список сообщений, которые формирует MPASM. Эти сообщения всегда включаются в файл листинга программы выше строки, к которой относится сообщение.

Типы сообщения, сохраняемые в файле ошибок (.ERR) можно указать при начале компиляции исходного файла. Если указать параметр /e-, то генерации файла ошибок не будет, а сообщения будут отображены на экране. Если выбрать параметры в командной строке /q и /e-, то никакие сообщения не будут появляться на экране, в фале списка ошибок и листинга программы.

B.2 Основные части раздела

Сообщения об ошибках
Предупреждения
Информационные сообщения

B.3 Сообщения об ошибках

101 ERROR:
Ошибка пользователя при работе с директивами.

102 Out of memory.
Недостаточно памяти для макроса, #define или внутренней обработки.
Закройте открытые приложения и попробуйте выполнить компиляцию снова. Если данная ошибка произошла при выполнении компиляции MPASM DOS версии, попробуйте выполнить компиляцию MPASM Windows версией.

103 Symbol table full.
Недостаточно памяти для размещения таблицы символов.
Закройте открытые приложения и попробуйте выполнить компиляцию снова. Если данная ошибка произошла при выполнении компиляции MPASM DOS версии, попробуйте выполнить компиляцию MPASM Windows версией или MPASM_DP версией.

104 Temp file creation error.
Невозможно создать временный файл. Проверьте объем свободного места на диске.

105 Cannot open file.
Ошибка при открытии файла. Проверьте существование указанного файла. Ошибка также возникает при открытии исходного файла старой версии MPASM или защищенного от изменений файла.

106 String substitution too complex.
Слишком много вложений #define.

107 Illegal digit.
Неправильная цифра в числе.
Допускаются следующие цифры в числах:
Двоичном 0-1;
Восьмеричном 0-7;
Десятичном 0-9;
Шестнадцатеричном 0-F.

108 Illegal character.
Использование недопустимого символа в имени метки. Допускаются следующие символы в именах меток: a..z; A..Z; 0..9; _. Имя метки не должно начинаться с цифры.

109 Unmatched (
Левая скобка не имеет соответствующую правую скобку. Например, “DATA (1+2”.

110 Unmatched)
Правая скобка не имеет соответствующую левую скобку. Например, “DATA 1+2)”.

111 Missing symbol.
Отсутствие символа. Директива EQU или SET не имеет <label> для присвоения значения.

112 Missing operator.
Отсутствие арифметического оператора в выражении. Например, “DATA 1 2”.

113 Symbol not previously defined.
Использование не определенной переменной. Только метки адреса могут использоваться перед их определением. Константы и переменные должны быть сначала объявлены пользователем.

114 Divide by zero.
Обнаружение деления на нуль во время оценки выражения.

115 Duplicate label.
Объявление переменной более одного раза (например, в директивах EQU или CBLOCK).

116 Address label duplicated or different in second pass.
Определение метки адреса в памяти программ более одного раза.
Метка определена один раз, но адрес размещения изменился при втором проходе компилятора. Это может произойти когда пользователь изменяет биты указателя страницы памяти программ в макрокомандах, генерирующие различные значения на основе текущего адреса.

117 Address wrapped around 0.
Адрес выполнения программы может достигать значения FFFF, после чего будет иметь адрес 0.

118 Overwriting previous address contents.
Код был предварительно определен для данного адреса.

119 Code too fragmented.
Код программы имеет слишком много частей. Данная ошибка возникает очень редко, только при попытке обращения к памяти программ с адресом выше 32Кбайт (включая биты конфигурации).

120 Call or jump not allowed at this address.
Переход не может быть выполнен. Например, все команды CALL для микроконтроллеров PIC16C5x должны обращаться к младшей страницы памяти программ.

121 Illegal label.
Метка не может быть указана на этой строке. Разместите метку выше директивы. Так же HIGH, LOW, PAGE и BANK не допустимые имена меток.

122 Illegal opcode.
Недопустимый код инструкции.

123 Illegal directive.
Недопустимая директива для данного типа микроконтроллера. Например, директива _ _IDLOCS не может использоваться для микроконтроллеров PIC17C42.

124 Illegal argument.
Неправильный аргумент. Например, LIST STUPID.

125 Illegal condition.
Неправильный блок условий. Например, отсутствие директивы ENDIF.

126 Argument out of range.
Значение аргумента инструкции или директивы выходят за допустимые рамки. Например, TRIS 10.

127 Too many arguments.
Слишком много аргументов для вызываемого макроса.

128 Missing argument(s).
Указаны не все аргументы в вызываемом макросе или команде.

129 Expected.
Ожидался иной тип аргумента. Ожидаемый список будет указан.

130 Processor type previously defined.
Выбраны разные типы микроконтроллеров.

131 Processor type is undefined.
Тип микроконтроллера указан после части текста программы. Обратите внимание, что пока тип микроконтроллера не определен - набор поддерживаемых команд не известен.

132 Unknown processor.
Выбран неподдерживаемый тип микроконтроллера.

133 Hex file format INHX32 required.
Был определен адрес больше 32Кбайт. Например, указывая биты конфигурации для микроконтроллеров семейства PIC17CXX.

134 Illegal hex file format.
В директиве LIST был определен незнакомый формат HEX файла.

135 Macro name missing.
Попытка определения макроса без названия.

136 Duplicate macro name.
Дублирование имени макроса.

137 Macros nested too deep.
Превышен максимальный уровень вложенности макросов.

138 Include files nested too deep.
Превышен максимальный уровень вложенности файлов.

139 Maximum of 100 lines inside WHILE-ENDW.
Цикл содержит более 100 строк.

140 WHILE must terminate within 256 iterations.
Цикл имеет более 256 повторений. Предотвращение бесконечного цикла.

141 WHILEs nested too deep.
Превышен максимальный уровень вложений циклов.

142 IFs nested too deep.
Превышен максимальный уровень вложений блоков условий.

143 Illegal nesting.
Недопустимое использование директив цикла, определения макроса или блока условия. Например, если вы имеете блок условия IF внутри цикла WHILE и обнаружена директива ENDW раньше INDIF.

144 Unmatched ENDC.
Найдена директива ENDC без директивы CBLOCK.

145 Unmatched ENDM.
Найдена директива ENDM без директивы MACRO.

146 Unmatched EXITM.
Найдена директива EXITM без директивы MACRO.

147 Directive not allowed when generating an object file.
Найдена недопустимая директива ORG при генерации объектного файла. Вместо директивы ORG создайте секцию .code и если необходимо укажите адрес.

148 Expanded source line exceeded 200 characters.
Превышена максимальная длина (200 знаков) символьной строки в директиве #DEFINE или в параметре макроса. Обратите внимание, что в директиву #DEFINE комментарии не включаются в отличие от макросов.

149 Directive only allowed when generating an object file section.
Использование директив, предназначенных для формирования объектного файла (например, GLOBAL и EXTERN), при непосредственной генерации кода программы.

150 Labels must be defined in a code or data section when making an object file.
Все переменные должны быть определены внутри секции объявления данных. Не допускается использование директив EQU или SET вне секций объявления данных при генерации объектного файла.

151 Operand contains unresolvable labels or is too complex.
При генерации объектного кода операнды должны иметь следующий синтаксис:
[HIGH|LOW]([<relocatable address label>]+[<offset>]).

152 Executable code and data must be defined in an appropriate section.
При генерации объектного файла блоки исходного текста программы и определение данных должны быть помещены в соответствующих секциях.

153 Page or Bank bits cannot be evaluated for the operand.
Страница памяти программ или банк памяти не могут быть определены для операнда <relocatable address label> или <constant>.директивами PAGESEL, BANKSEL или BANKISEL.

154 Each object file section must be contiguous.

Объявленная секция данных, кроме секции UDATA_OVR, имеет уже существующее имя в данном исходном файле. Разрешить эту проблему можно, назвав каждый раздел собственным именем или определять все данные в одном разделе. Эта ошибка возникает и при присвоении двум разделам разных типов одинакового имени.

155 All overlaid sections of the same name must have the same starting address.

Объявлена секция UDATA_OVR с существующим именем, но другим адресом.

156 Operand must be an address label.

Возникает во время генерации объектного файла при попытке объявить метки созданные директивой SET или EQU (а не в секции данных) как глобальные.

157 UNKNOWN ERROR.

Произошла ошибка, которую MPASM не может распознать. Это не является ошибкой, описанном в этом приложении. Свяжитесь с инженерами компании Microchip (FAE), если Вы не можете устранить эту ошибку.

B.4 Предупреждения

201 Symbol not previously defined.
Последовательность символов указана в директиве #UNDIFINE не была предварительно определена.

202 Argument out of range. Least significant bits used.
Аргумент не соответствует месту размещения. Например, литералы должны быть 8-разрядные.

203 Found opcode in column 1.
Мнемоника инструкции была обнаружена в первой колонке, предназначенной для меток.

204 Found pseudo-op in column 1.
Псевдооператор был обнаружен в первой колонке, предназначенной для меток.

205 Found directive in column 1.
Директива была обнаружена в первой колонке, предназначенной для меток.

206 Found call to macro in column 1.
Макрокоманда была обнаружена в первой колонке, предназначенной для меток.

207 Found label after column 1.
Метка была обнаружена не в первой колонке, что часто принимается за мнемонику команды с неправильным синтаксисом.

208 Label truncated at 32 characters.
Длина метки более 32 знаков.

209 Missing quote.
Пропущен апостроф. Например, DATA ‘A.

210 Extra),
В конце строки была найдена лишняя запятая.

211 Extraneous arguments on the line.
Найдены лишние аргументы в строке. На это предупреждение следует обращать внимание, т.к. оно часто возникает из-за неправильной интерпретации записи (например, OPTION EQU 0x81 с LIST FREE).

212 Expected
Тип аргумента не соответствует ожидаемому. Предупреждение используется, когда предполагается иной тип аргумента.

213 The EXTERN directive should only be used when making a .O file.
Директива EXTERN должна использоваться только при генерации объектного файла. Предупреждение заменяет ошибку 149.

214 Unmatched (
Найдена лишняя скобка. Предупреждение используется, если скобка не влияет на значение выражения.

215 Processor superseded by command line. Verify processor symbol.
Тип микроконтроллера был определен на одной строке с инструкцией. Инструкция имеет больший приоритет.

216 Radix superseded by command line.
Система счисления по умолчанию была определена на одной строке с инструкцией. Инструкция имеет больший приоритет.

217 Hex file format specified on command line.
Тип выходного HEX файла был определен на одной строке с инструкцией. Инструкция имеет больший приоритет.

218 Expected DEC, OCT, HEX. Will use HEX.
Неизвестный тип системы счисления.

219 Invalid RAM location specified.
Неправильное распределение ОЗУ директивами _ _MAXRAM и _ _BADRAM для данного типа микроконтроллера. Обратите внимание, что подключаемые файлы описания микроконтроллеров содержат директивы _ _MAXRAM и _ _BADRAM.

220 Address exceeds maximum range for this processor.
Адресована память программ, которая не существует в данном микроконтроллере.

221 Invalid message number.
Недействительный номер сообщения, вызванного для отображения или маскирования.

222 Error messages cannot be disabled.
Сообщения об ошибках не могут быть запрещены директивой EERORLEVEL.

223 Redefining processor
Выбранный микроконтроллер повторно выбирается директивой LIST или PROCESSOR.

224 Use of this instruction is not recommended.
Используется не рекомендуемая инструкция. Например, TRIS или OPTION для микроконтроллеров PIC16CXX.

225 Invalid label in operand.
Неправильная метка в операнде. Например, когда в операнде инструкции CALL встречается имя макрокоманды.

226 UNKNOWN WARNING
Неизвестное предупреждение MPASM. Это не является предупреждением, описанном в этом приложении. Свяжитесь с инженерами компании Microchip (FAE), если Вы не можете устранить это предупреждение.

B.5 Информационные сообщения

301 MESSAGE:
Сообщение пользователя, вызванное директивой MESSG.

302 Register in operand not in bank 0. Ensure that bank bits are correct.
Адрес регистра был определен значением, в котором содержаться биты выбора банка памяти. Например, адресация памяти данных в PIC16CXXX определяется 7 битами в операнде команды и одним/двумя битами в регистре STATUS.

303 Program word too large. Truncated to core size.
Слишком большое слово программы. Для микроконтроллеров PIC12CXX и PIC16C5X 12-разрядные команды. Для микроконтроллеров PIC16CXX 14-разрядные команды.

304 ID Locations value too large. Last four hex digits used.
Слишком большое значение адреса расположения ID битов. Для определения адреса используются четыре последние цифры.

305 Using default destination of 1 (file).
Использование значения по умолчанию.

306 Crossing page boundary – ensure page bits are set.
Полученный код программы пересекает границу страницы памяти программ.

307 Setting page bits.
Страница памяти программ настраивается псевдокомандами LGOTO и LCALL.

308 Warning level superseded by command line value.
Уровень вывода предупреждений соответствует установленному в командной строке. Параметр командной строки имеет преимущество.

309 Macro expansion superseded by command line.
Разрешение включения полного текста макроса в файл листинга соответствует параметру, указанному в командной строке. Параметр командной строки имеет преимущество.

310 Superseding current maximum RAM and RAM map.
Использование директивы _ _MAXRAM.

312 Page or Bank selection not needed for this device. No code generated.
Если микроконтроллер содержит одну страницу памяти программ или один банк памяти данных, то директивы PAGESEL, BANKSEL или BANKISEL не будут приводить к генерации дополнительного кода программы.

313 CBLOCK constants will start with a value of 0.
Первая директива CBLOCK не указывает адрес размещения данных.

314 UNKNOWN MESSAGE
Неизвестное сообщение. Это не является сообщением, описанном в этом приложении. Свяжитесь с инженерами компании Microchip (FAE), если Вы не можете устранить это сообщение.

Перевод основывается на технической документации DS33014G
компании Microchip Technology Incorporated, USA.

Уважаемые господа!

ООО «Микро-Чип» поставляет полную номенклатуру комплектующих фирмы **Microchip Technology Inc** и осуществляет качественную техническую поддержку на русском языке.

С техническими вопросами Вы можете обращаться по адресу support@microchip.ru

По вопросам поставок комплектующих Вы можете обращаться к нам по телефонам:
(095) 963-9601
(095) 737-7545
и адресу sales@microchip.ru

На сайте
www.microchip.ru
Вы можете узнать последние новости нашей фирмы, найти техническую документацию и информацию по наличию комплектующих на складе.