

*MPASM
USER'S GUIDE
with MPLINK and MPLIB*

MPASM USER'S GUIDE with MPLINK and MPLIB

Information contained in this publication regarding device applications and the like is intended by way of suggestion only. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip.

© 1999 Microchip Technology Incorporated. All rights reserved.

The Microchip logo, name, PIC, PICmicro, PICMASTER, PICSTART, and PRO MATE are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries. MPLAB, and *Smart Serial* are trademarks of Microchip Technology in the U.S.A. and other countries.

All product/company trademarks mentioned herein are the property of their respective companies.

MPASM User's Guide with MPLINK and MPLIB



MPASM USER'S GUIDE with MPLINK and MPLIB

Table of Contents

General Information

Introduction	1
Highlights	1
About This Guide	1
Warranty Registration	4
Recommended Reading	4
The Microchip Internet Web Site	5
Development Systems Customer Notification Service	6
Customer Support	8

MPASM User's Guide with MPLINK and MPLIB

Part 1 – MPASM

Chapter 1. MPASM Preview

1.1	Introduction	11
1.2	Highlights	11
1.3	What MPASM Is	11
1.4	What MPASM Does	11
1.5	Migration Path	12
1.6	Compatibility Issues	12

Chapter 2. MPASM – Installation and Getting Started

2.1	Introduction	13
2.2	Highlights	13
2.3	Installation	13
2.4	Overview of Assembler	14
2.5	Assembler Input/Output Files	16

Chapter 3. Using MPASM with DOS

3.1	Introduction	21
3.2	Highlights	21
3.3	Command Line Interface	21
3.4	DOS Shell Interface	24

Chapter 4. Using MPASM with Windows and MPLAB

4.1	Introduction	27
4.2	Highlights	27
4.3	Windows Shell Interface	27
4.4	MPLAB Projects and MPASM	29
4.5	Setting Up MPLAB to use MPASM	30
4.6	Generating Output Files	32
4.7	MPLAB/MPASM Troubleshooting	32

Chapter 5. Directive Language

5.1	Introduction	35
5.2	Highlights	35

Table of Contents

5.3	Directive Summary	35
5.4	__BADRAM – Identify Unimplemented RAM	39
5.5	BANKISEL – Generate Indirect Bank Selecting Code	39
5.6	BANKSEL – Generate Bank Selecting Code	40
5.7	CBLOCK – Define a Block of Constants	41
5.8	CODE – Begin an Object File Code Section	41
5.9	__CONFIG – Set Processor Configuration Bits	42
5.10	CONSTANT – Declare Symbol Constant	43
5.11	DA – Store Strings in Program Memory	44
5.12	DATA – Create Numeric and Text Data	44
5.13	DB – Declare Data of One Byte	45
5.14	DE – Declare EEPROM Data Byte	45
5.15	#DEFINE – Define a Text Substitution Label	46
5.16	DT – Define Table	47
5.17	DW – Declare Data of One Word	47
5.18	ELSE – Begin Alternative Assembly Block to IF	48
5.19	END – End Program Block	48
5.20	ENDC – End an Automatic Constant Block	49
5.21	ENDIF – End Conditional Assembly Block	49
5.22	ENDM – End a Macro Definition	49
5.23	ENDW – End a While Loop	50
5.24	EQU – Define an Assembler Constant	50
5.25	ERROR – Issue an Error Message	51
5.26	ERRORLEVEL – Set Message Level	51
5.27	EXITM – Exit from a Macro	52
5.28	EXPAND – Expand Macro Listing	52
5.29	EXTERN – Declare an Externally Defined Label	53
5.30	FILL – Specify Memory Fill Value	53
5.31	GLOBAL – Export a Label	54
5.32	IDATA – Begin an Object File Initialized Data Section	54
5.33	__IDLOCS – Set Processor ID Locations	55

MPASM User's Guide with MPLINK and MPLIB

5.34	IF – Begin Conditionally Assembled Code Block	56
5.35	IFDEF – Execute If Symbol has Been Defined	56
5.36	IFDEF – Execute If Symbol has not Been Defined	57
5.37	INCLUDE – Include Additional Source File	58
5.38	LIST – Listing Options	58
5.39	LOCAL – Declare Local Macro Variable	59
5.40	MACRO – Declare Macro Definition	60
5.41	__MAXRAM – Define Maximum RAM Location	60
5.42	MESSG – Create User Defined Message	61
5.43	NOEXPAND – Turn off Macro Expansion	62
5.44	NOLIST – Turn off Listing Output	62
5.45	ORG – Set Program Origin	62
5.46	PAGE – Insert Listing Page Eject	63
5.47	PAGESEL – Generate Page Selecting Code	63
5.48	PROCESSOR – Set Processor Type	64
5.49	RADIX – Specify Default Radix	64
5.50	RES – Reserve Memory	65
5.51	SET – Define an Assembler Variable	65
5.52	SPACE – Insert Blank Listing Lines	66
5.53	SUBTITLE – Specify Program Subtitle	66
5.54	TITLE – Specify Program Title	66
5.55	UDATA – Begin an Object File Uninitialized Data Section	67
5.56	UDATA_ACS – Begin an Object File Access Uninitialized Data Section	68
5.57	UDATA_OVR – Begin an Object File Overlayed Uninitialized Data Section	68
5.58	UDATA_SHR – Begin an Object File Shared Uninitialized Data Section	69
5.59	#UNDEFINE – Delete a Substitution Label	70
5.60	VARIABLE – Declare Symbol Variable	70
5.61	WHILE – Perform Loop While Condition is True	71

Table of Contents

Chapter 6. Using MPASM to Create Relocatable Objects

6.1	Introduction	73
6.2	Highlights	73
6.3	Header Files	73
6.4	Program Memory	74
6.5	Instruction Operands	75
6.6	RAM Allocation	75
6.7	Configuration Bits and ID Locations	76
6.8	Accessing Labels From Other Modules	77
6.9	Paging and Banking Issues	77
6.10	Unavailable Directives	79
6.11	Generating the Object Module	79
6.12	Code Examples	79

Chapter 7. Macro Language

7.1	Introduction	83
7.2	Highlights	83
7.3	Macro Syntax	83
7.4	Macro Directives	84
7.5	Text Substitution	84
7.6	Macro Usage	85
7.7	Code Examples	86

Chapter 8. Expression Syntax and Operation

8.1	Introduction	89
8.2	Highlights	89
8.3	Text Strings	89
8.4	Numeric Constants and Radix	91
8.5	High/Low/Upper	93
8.6	Increment/Decrement (++/--)	93

MPASM User's Guide with MPLINK and MPLIB

Chapter 9. Example Initialization Code

- 9.1 Introduction95
- 9.2 Highlights95
- 9.3 Initialization Code Examples95

Table of Contents

Part 2 – MPLINK

Chapter 1. MPLINK Preview

1.1	Introduction	99
1.2	Highlights	99
1.3	What MPLINK Is	99
1.4	What MPLINK Does	99
1.5	How MPLINK Helps You	100
1.6	MPLINK Examples	100
1.7	Platforms Supported	101

Chapter 2. MPLINK – Installation and Getting Started

2.1	Introduction	103
2.2	Highlights	103
2.3	Installation	103
2.4	Overview of Linker	104
2.5	Linker Input/Output Files	105

Chapter 3. Using MPLINK with DOS

3.1	Introduction	109
3.2	Highlights	109
3.3	Linker Command Line Options	109

Chapter 4. Using MPLINK with Windows and MPLAB

4.1	Introduction	111
4.2	Highlights	111
4.3	MPLAB Projects and MPLINK	111
4.4	Setting Up MPLAB to use MPLINK	113
4.5	Generating Output Files	117
4.6	MPLAB/MPLINK Troubleshooting	117

Chapter 5. MPLINK Linker Scripts

5.1	Introduction	119
5.2	Highlights	119
5.3	Linker Scripts Defined	119

MPASM User's Guide with MPLINK and MPLIB

5.4	Command Line Information	119
5.5	Memory Region Definitions	121
5.6	Logical Section Definitions	125
5.7	STACK Definition	126
5.8	Linker Script Caveats	126

Chapter 6. Linker Processing

6.1	Introduction	127
6.2	Highlights	127
6.3	Linker Processing Overview	127
6.4	Linker Allocation Algorithm	128
6.5	Relocation Example	129
6.6	Initialized Data	130

Chapter 7. Sample Application 1

7.1	Highlights	131
7.2	Overview	131
7.3	Building the Application	132
7.4	Source Code	133

Chapter 8. Sample Application 2

8.1	Highlights	137
8.2	Overview	137
8.3	Building the Application	138
8.4	Source Code – Boot Loader	139
8.5	Source Code – Firmware	142

Chapter 9. Sample Application 3

9.1	Highlights	145
9.2	Overview	145
9.3	Building the Application	147
9.4	Source Code	148

Table of Contents

Chapter 10. Sample Application 4

10.1	Highlights	151
10.2	Overview	151
10.3	Building the Application	153
10.4	Source Code	154

MPASM User's Guide with MPLINK and MPLIB

Part 3 – MPLIB

Chapter 1. MPLIB Preview

1.1	Introduction	163
1.2	Highlights	163
1.3	What MPLIB Is	163
1.4	What MPLIB Does	163
1.5	How MPLIB Helps You	164

Chapter 2. MPLIB – Installation and Getting Started

2.1	Introduction	165
2.2	Highlights	165
2.3	Installation	165
2.4	Overview of Librarian	166

Chapter 3. Using MPLIB

3.1	Introduction	167
3.2	Highlights	167
3.3	Usage Format	167
3.4	Usage Examples	168
3.5	Tips	168

Table of Contents

Appendices

Appendix A. Hex File Formats

A.1	Introduction	171
A.2	Highlights	171
A.3	Intel Hex Format (.HEX)	171
A.4	8-Bit Split Format (.HXL/.HXH)	172
A.5	32-Bit Hex Format (.HEX)	173

Appendix B. Quick Reference

B.1	Introduction	175
B.2	Highlights	175
B.3	MPASM Quick Reference	175
B.4	Key to PICmicro Family Instruction Sets	180
B.5	12-Bit Core Instruction Set	181
B.6	14-Bit Core Instruction Set	183
B.7	16-Bit Core Instruction Set	186
B.8	Key to Enhanced 16-Bit Core Instruction Set	189
B.9	Enhanced 16-Bit Core Instruction Set	190
B.10	Hexadecimal to Decimal Conversion	194
B.11	ASCII Character Set	195

Appendix C. MPASM Errors/Warnings/Messages

C.1	Introduction	197
C.2	Highlights	197
C.3	Errors	197
C.4	Warnings	202
C.5	Messages	204

Appendix D. MPLINK Errors/Warnings

D.1	Introduction	207
D.2	Highlights	207
D.3	Parse Errors	207
D.4	Linker Errors	208

MPASM User's Guide with MPLINK and MPLIB

D.5	Linker Warnings	211
D.6	Library File Errors	211
D.7	COFF File Errors	212
D.8	COFF To COD Converter Errors	213
D.9	COFF To COD Converter Warnings	213

Appendix E. MPLIB Errors

E.1	Introduction	215
E.2	Highlights	215
E.3	Parse Errors	215
E.4	Library File Errors	215
E.5	COFF File Errors	215

Glossary

Introduction	217
Highlights	217
Terms	217

Index	225
--------------------	------------

Worldwide Sales and Service	230
--	------------

General Information

Introduction

This first chapter contains general information that will be useful to know before working with MPASM, MPLINK, and MPLIB.

Highlights

The information you will garner from this chapter:

- About This Guide
- Warranty Registration
- Recommended Reading
- The Microchip Internet Web Site
- Development Systems Customer Notification Service
- Customer Support

About This Guide

Document Layout

This document describes how to use MPASM, MPLINK, and MPLIB to develop code for PICmicro[®] microcontroller applications. All of these programs can work within the MPLAB[™] Integrated Development Environment (IDE). For a detailed discussion about basic MPLAB functions, refer to the *MPLAB User's Guide* (DS51025).

The User's Guide layout is as follows:

Part 1 – MPASM

- **Chapter 1: MPASM Preview** – defines MPASM and describes what it does and how it works with other tools.
- **Chapter 2: MPASM – Installation and Getting Started** – describes how to install MPASM and gives an overview of operation.
- **Chapter 3: Using MPASM with DOS** – describes how to use MPASM with DOS via the command line or a DOS shell interface.
- **Chapter 4: Using MPASM with Windows[®] and MPLAB** – describes how to use MPASM with Microsoft Windows via a Windows shell interface or MPLAB.
- **Chapter 5: Directive Language** – describes the MPASM programming language including statements, operators, variables, and other elements.

MPASM User's Guide with MPLINK and MPLIB

- **Chapter 6: Using MPASM to Create Relocatable Objects** – describes how to use MPASM in conjunction with MPLINK, Microchip's linker.
- **Chapter 7: Macro Language** – describes how to use MPASM's built-in macro processor.
- **Chapter 8: Expression Syntax and Operation** – provides guidelines for using complex expressions in MPASM source files.
- **Chapter 9: Example Initialization Code** – lists code examples for initializing PIC16CXX, PIC17CXX, and PIC18CXX devices.

Part 2 – MPLINK

- **Chapter 1: MPLINK Preview** – defines MPLINK and describes what it does and how it works with other tools.
- **Chapter 2: MPLINK – Installation and Getting Started** - describes how to install MPLINK and gives an overview of operation.
- **Chapter 3: Using MPLINK with DOS** – describes how to use MPLINK with DOS via the command line.
- **Chapter 4: Using MPLINK with Windows and MPLAB** – describes how to use MPLINK with Microsoft Windows via a DOS window or MPLAB.
- **Chapter 5: MPLINK Linker Scripts** – discusses how to generate and use linker scripts to control linker operation.
- **Chapter 6: Linker Processing** – describes how the linker processes files.
- **Chapter 7: Sample Application 1** – explains how to place program code in different memory regions, how to place data tables in ROM memory, and how to set configuration bits in C.
- **Chapter 8: Sample Application 2** – explains how to partition memory for a boot loader and how to compile code that will be loaded into external RAM and executed.
- **Chapter 9: Sample Application 3** – explains how to access peripherals that are memory mapped and how to create new sections.
- **Chapter 10: Sample Application 4** – explains how to manually partition RAM space for program usage.

Part 3 – MPLIB

- **Chapter 1: MPLIB Preview** – defines MPLIB and describes what it does and how it works with other tools.
- **Chapter 2: MPLIB – Installation and Getting Started** - describes how to install MPLIB and gives an overview of operation.
- **Chapter 3: Using MPLIB** – describes how to use MPLIB via the DOS command line or a DOS window in Microsoft Windows.

General Information

Appendices

- **Appendix A: Hex File Formats** – provides a description of the different hex file formats that may be used.
- **Appendix B: Quick Reference** – lists PICmicro device instruction sets, hexadecimal to decimal conversions, and ASCII Character Set.
- **Appendix C: MPASM Errors/Warnings/Messages** – contains a descriptive list of the errors, warnings, and messages generated by MPASM.
- **Appendix D: MPLINK Errors/Warnings** – contains a descriptive list of the errors and warnings generated by MPLINK.
- **Appendix E: MPLIB Errors** – contains a descriptive list of the errors generated by MPLIB.
- **Glossary** – A glossary of terms used in this guide.
- **Index** – Cross-reference listing of terms, features, and sections of this document.
- **Worldwide Sales and Service** – gives the address, telephone and fax number for Microchip Technology Inc. Sales and Service locations throughout the world.

Conventions Used in this Guide

This manual uses the following documentation conventions:

Documentation Conventions

Description	Represents	Examples
Code		
Courier Font	User-entered code or sample code	<code>#define ENIGMA</code>
Angle Brackets: <>	Variables. Text you supply	<label>, <exp>
Square Brackets: []	Optional Arguments	MPASMWIN [main.asm]
Curly Brackets and Pipe Character: { }	Choice of mutually exclusive arguments; an OR selection	errorlevel { 0 1 }
Lowercase Characters in Quotes	Type of data	"filename"
Ellipses: ...	Used to imply, but not show, additional text that is not relevant to the example.	list ["list_option" , ... "list_option"]
0xn timer	0xn timer represents a hexadecimal number where n is a hexadecimal digit	0xFFFF, 0x007A

MPASM User's Guide with MPLINK and MPLIB

Documentation Conventions

Description	Represents	Examples
Interface		
Underlined, Italics Text with Right Arrow	Defines a menu selection from the menu bar.	<i>File > Save</i>
In-text Bold Characters	Designates a button	OK, Cancel
Uppercase Characters in Angle Brackets: < >	Delimiters for special keys.	<TAB>, <ESC>
Documents		
Italic characters	Referenced books.	<i>MPLAB User's Guide</i>

Updates

All documentation becomes dated, and this user's guide is no exception. Since MPASM, MPLINK, MPLIB, and other Microchip tools are constantly evolving to meet customer needs, some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site to obtain the latest documentation available.

Warranty Registration

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in your Warranty Registration Card entitles you to receive new product updates. Interim software releases are available at the Microchip web site.

Recommended Reading

This user's guide describes how to use MPASM, MPLINK, and MPLIB. The user may also find the data sheets for specific microcontroller devices informative in developing firmware.

README.ASM, README.LKR

For the latest information on using MPASM and MPLINK, read the README files (ASCII text files) included with the MPASM software. The README files contain update information that may not be included in this document.

MPLAB User's Guide (DS51025)

Comprehensive guide that describes installation and features of Microchip's MPLAB Integrated Development Environment, as well as the editor and simulator functions in the MPLAB environment.

General Information

Technical Library CD-ROM (DS00161)

This CD-ROM contains comprehensive data sheets for Microchip PICmicro devices available at the time of print. To obtain this disk, contact the nearest Microchip Sales and Service location (see back page) or download individual data sheet files from the Microchip web site (<http://www.microchip.com>).

Embedded Control Handbook Vol.1 & 2 (DS00092 & DS00167)

These handbooks contain a wealth of information about microcontroller applications. To obtain these documents, contact the nearest Microchip Sales and Service location (see back page).

The application notes described in these manuals are also obtainable from Microchip Sales and Service locations or from the Microchip web site (<http://www.microchip.com>).

Microsoft Windows Manuals

This manual assumes that users are familiar with Microsoft Windows operating system. Many excellent references exist for this software program, and should be consulted for general operation of Windows.

The Microchip Internet Web Site

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape® Navigator or Microsoft® Internet Explorer®. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip website is available by using your favorite Internet browser to attach to:

<http://www.microchip.com>

The file transfer site is available by using an FTP program/client to connect to:

<ftp://ftp.microchip.com>

The website and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles, and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors, and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings

MPASM User's Guide with MPLINK and MPLIB

- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

Development Systems Customer Notification Service

Microchip started the customer notification service to help our customers keep current on Microchip products with the least amount of effort. Once you subscribe to one of our list servers, you will receive email notification whenever we change, update, revise or have errata related to that product family or development tool. See the Microchip WWW page for other Microchip list servers.

The Development Systems list names are:

- Compilers
- Emulators
- Programmers
- MPLAB
- Otools

Once you have determined the names of the lists that you are interested in, you can subscribe by sending a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe <listname> yourname`

Here is an example:

`subscribe mplab John Doe`

To UNSUBSCRIBE from these lists, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`unsubscribe <listname> yourname`

Here is an example:

`unsubscribe mplab John Doe`

The following sections provide descriptions of the available Development Systems lists.

General Information

Compilers

The latest information on Microchip C compilers, Linkers, and Assemblers. These include MPLAB-C17, MPLAB-C18, MPLINK, MPASM as well as the Librarian, MPLIB for MPLINK.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe compilers yourname`

Emulators

The latest information on Microchip In-Circuit Emulators. These include MPLAB-ICE and PICMASTER®.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe emulators yourname`

Programmers

The latest information on Microchip PICmicro device programmers. These include PRO MATE® II and PICSTART® Plus.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe programmers yourname`

MPLAB

The latest information on Microchip MPLAB, the Windows Integrated Development Environment for development systems tools. This list is focused on MPLAB, MPLAB-SIM, MPLAB's Project Manager and general editing and debugging features. For specific information on MPLAB compilers, linkers, and assemblers, subscribe to the COMPILERS list. For specific information on MPLAB emulators, subscribe to the EMULATORS list. For specific information on MPLAB device programmers, please subscribe to the PROGRAMMERS list.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe mplab yourname`

MPASM User's Guide with MPLINK and MPLIB

Otools

The latest information on other development system tools provided by Microchip. For specific information on MPLAB and its integrated tools refer to the other mail lists.

To SUBSCRIBE to this list, send a message to:

`listserv@mail.microchip.com`

with the following as the body:

`subscribe otools yourname`

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Corporate Applications Engineer (CAE)
- Hot line

Customers should call their distributor, representative, or field application engineer (FAE) for support. Local sales offices are also available to help customers. See the back cover for a listing of sales offices and locations.

Corporate applications engineers (CAEs) may be contacted at (602) 786-7627.

In addition, there is a Systems Information and Upgrade Line. This line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.

The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-602-786-7302 for the rest of the world.

Part 1 – MPASM

Chapter 1. MPASM Preview	11
Chapter 2. MPASM – Installation and Getting Started	13
Chapter 3. Using MPASM with DOS	21
Chapter 4. Using MPASM with Windows and MPLAB	27
Chapter 5. Directive Language	35
Chapter 6. Using MPASM to Create Relocatable Objects	73
Chapter 7. Macro Language	83
Chapter 8. Expression Syntax and Operation	89
Chapter 9. Example Initialization Code	95

MPASM User's Guide with MPLINK and MPLIB

Chapter 1. MPASM Preview

1.1 Introduction

This chapter will give you an preview of MPASM and its capabilities.

1.2 Highlights

Topics covered in this chapter:

- What MPASM Is
- What MPASM Does
- Migration Path
- Compatibility Issues

1.3 What MPASM Is

MPASM is a DOS or Windows-based PC application that provides a platform for developing assembly language code for Microchip's PICmicro microcontroller (MCU) families. Generically, MPASM will refer to the entire development platform including the macro assembler and utility functions.

Use of the Microchip MPASM Universal Assembler requires an IBM PC/AT[®] or compatible computer, running MS-DOS[®] V5.0 or greater, or Microsoft[®] Windows 95/98/NT.

MPASM supports all PICmicro, memory, and secure data products from Microchip.

1.4 What MPASM Does

MPASM provides a universal solution for developing assembly code for all of Microchip's 12-bit, 14-bit, 16-bit, and Enhanced 16-bit core PICmicro microcontrollers. Notable features include:

- All PICmicro MCU Instruction Sets
- Command Line Interface
- Command Shell Interfaces
- Rich Directive Language
- Flexible Macro Language
- MPLAB Compatibility

MPASM User's Guide with MPLINK and MPLIB

1.5 Migration Path

Since MPASM is a universal assembler for all PICmicro devices, an application developed for the PIC16C54 can be easily translated into a program for the PIC16C71. This would require changing the instruction mnemonics that are not the same between the machines (assuming that register and peripheral usage were similar). The rest of the directive and macro language will be the same.

1.6 Compatibility Issues

MPASM is compatible with all Microchip development systems currently in production. This includes MPLAB-SIM (PICmicro MCU discrete-event simulator), MPLAB-ICE (PICmicro MCU Universal In-Circuit Emulator), PRO MATE (the Microchip Universal Programmer), and PICSTART Plus (the Microchip low-cost development programmer).

MPASM supports a clean and consistent method of specifying radix (see Chapter 5). You are encouraged to develop new code using the methods described within this document, even though certain older syntaxes may be supported for compatibility reasons.

Chapter 2. MPASM – Installation and Getting Started

2.1 Introduction

This chapter provides instructions for installation of MPASM on your system, and an overview of the assembler (MPASM).

2.2 Highlights

Topics covered in this chapter:

- Installation
- Overview of Assembler
- Assembler Input/Output Files

2.3 Installation

There are three versions of MPASM:

- a DOS version, MPASM . EXE, for DOS 5.0 or greater
- a DOS-Extender version, MPASM_DP . EXE
- a Windows 3.x/95/98/NT version, MPASMWIN . EXE (Recommended)

MPASM . EXE has a command line interface. MPASM . EXE may be used with DOS or a DOS window in Windows 3.x/95/98/NT. You can use it with MPLAB, though MPASMWIN . EXE is recommended.

MPASM_DP . EXE has a DOS shell interface. MPASM_DP . EXE may be used with DOS or a DOS window in Windows 3.x. You can use it with MPLAB running under Windows 3.x, though MPASMWIN . EXE is recommended.

MPASMWIN . EXE has a Windows shell interface. MPASMWIN . EXE may be used with Windows 3.x/95/98/NT. You can use this version with MPLAB (recommended) or stand-alone.

If you are going to use MPLAB with MPASM, you do not need to install the assembler and supporting files separately. When MPLAB is installed, MPASM is also installed. To find out how to install MPLAB, please refer to the *MPLAB User's Guide*. You may obtain the MPLAB software and user's guide either from the Microchip Technical Library CD or from our website.

If you are not going to use MPLAB with MPASM, you can obtain the assembler and supporting files separately either from the Microchip Technical Library CD or from our website. MPASM will be in a zip file. To install:

- Create a directory in which to place the files
- Unzip the MPASM files using either WinZip® or PKZIP®

MPASM User's Guide with MPLINK and MPLIB

2.4 Overview of Assembler

MPASM can be used in two ways:

- To generate absolute code that can be executed directly by a microcontroller.
- To generate object code that can be linked with other separately assembled or compiled modules.

Absolute code is the default output from MPASM. This process is shown in Figure 2.1.

When a source file is assembled in this manner, all values used in the source file must be defined within that source file, or in files that have been explicitly included. If assembly proceeds without errors, a HEX file will be generated, containing the executable machine code for the target device. This file can then be used in conjunction with a device programmer to program the microcontroller.

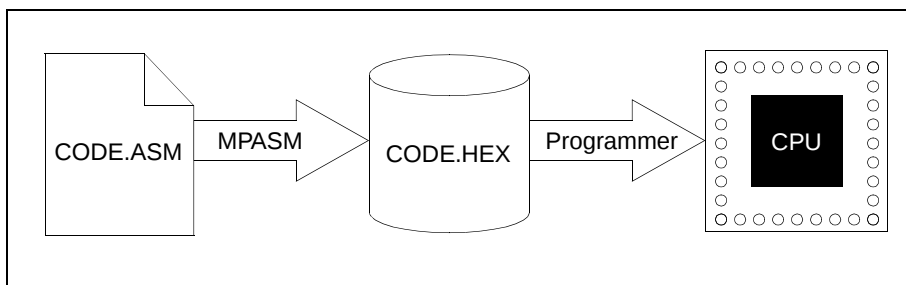


Figure 2.1: Generating Absolute Code

MPASM also has the ability to generate an object module that can be linked with other modules using Microchip's MPLINK linker to form the final executable code. This method is very useful for creating reusable modules that do not have to be retested each time they are used. Related modules can also be grouped and stored together in a library using Microchip's MPLIB Librarian. Required libraries can be specified at link time, and only the routines that are needed will be included in the final executable.

A visual representation of this process is shown in Figure 2.2 and Figure 2.3.

Refer to Chapter 6 for more information on the differences between absolute and object assembly.

MPASM – Installation and Getting Started

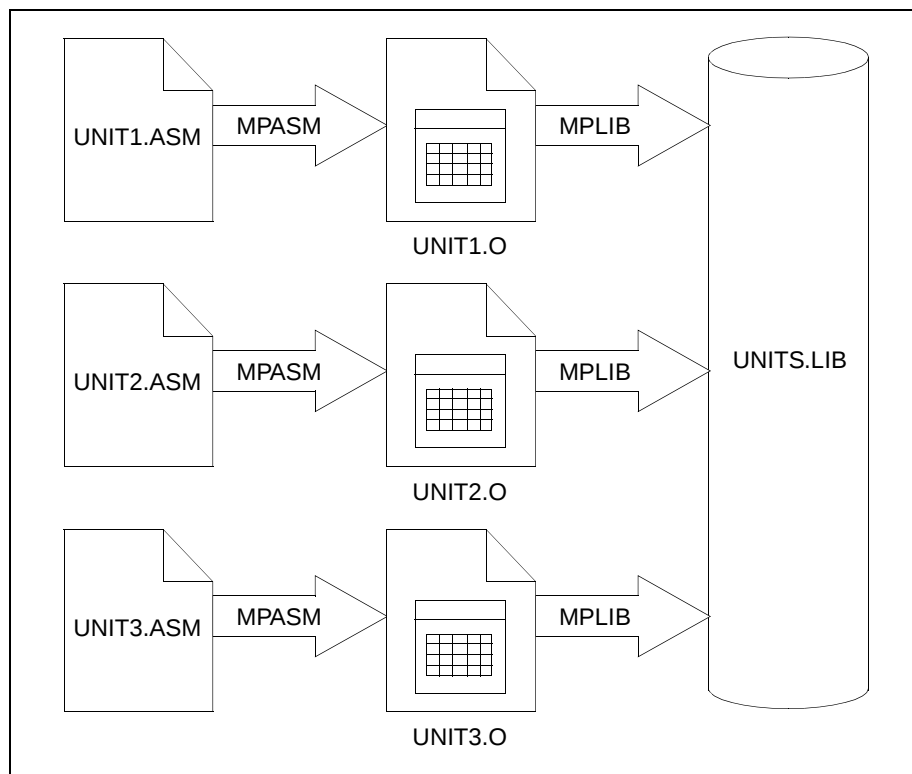


Figure 2.2: Creating a Reusable Object Library

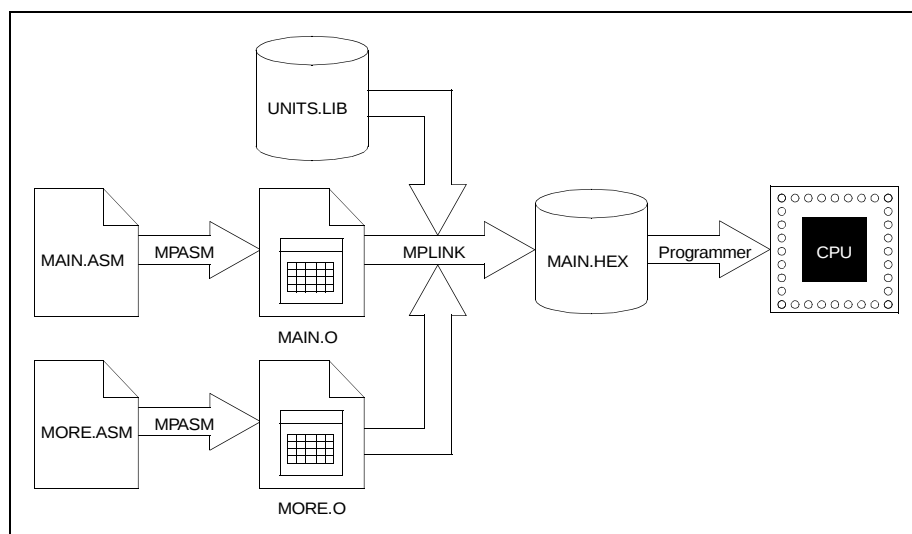


Figure 2.3: Generating Executable Code From Object Modules

MPASM User's Guide with MPLINK and MPLIB

2.5 Assembler Input/Output Files

These are the default file extensions used by MPASM and the associated utility functions.

Table 2.1: MPASM Default Extensions

Extension	Purpose
.ASM	Default source file extension input to MPASM: <source_name>.ASM
.LST	Default output extension for listing files generated by MPASM: <source_name>.LST
.ERR	Output extension from MPASM for error files: <source_name>.ERR
.HEX	Output extension from MPASM for hex files (see Appendix A), <source_name>.HEX
.HXL/ .HXH	Output extensions from MPASM for separate low byte and high byte hex files: <source_name>.HXL, <source_name>.HXH
.COD	Output extension for the symbol and debug file. This file may be output from MPASM or MPLINK: <source_name>.COD
.O	Output extension from MPASM for object files: <source_name>.O

2.5.1 Source Code Format (.ASM)

The source code file may be created using any ASCII text file editor. It should conform to the following basic guidelines.

Each line of the source file may contain up to four types of information:

- labels
- mnemonics
- operands
- comments

The order and position of these are important. Labels must start in column one. Mnemonics may start in column two or beyond. Operands follow the mnemonic. Comments may follow the operands, mnemonics or labels, and can start in any column. The maximum column width is 255 characters.

MPASM – Installation and Getting Started

Whitespace or a colon must separate the label and the mnemonic, and the mnemonic and the operand(s). Multiple operands must be separated by a comma. For example:

Example 2.1: Sample MPASM Source Code (Shows multiple operands)

```
;
; Sample MPASM Source Code. For illustration only.
;
      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 Labels

A label must start in column 1. It may be followed by a colon (:), space, tab or the end of line.

Labels must begin with an alpha character or an under bar (_) and may contain alphanumeric characters, the under bar and the question mark.

Labels may be up to 32 characters long. By default they are case sensitive, but case sensitivity may be overridden by a command line option. If a colon is used when defining a label, it is treated as a label operator and not part of the label itself.

2.5.1.2 Mnemonics

Assembler instruction mnemonics, assembler directives and macro calls must begin in column two or greater. If there is a label on the same line, instructions must be separated from that label by a colon, or by one or more spaces or tabs.

2.5.1.3 Operands

Operands must be separated from mnemonics by one or more spaces, or tabs. Multiple operands must be separated by commas.

MPASM User's Guide with MPLINK and MPLIB

2.5.1.4 Comments

MPASM treats anything after a semicolon as a comment. All characters following the semicolon are ignored through the end of the line. String constants containing a semicolon are allowed and are not confused with comments.

MPASM – Installation and Getting Started

2.5.2 Listing File Format (.LST)

Example 2.2: Sample MPASM Listing File (.LST)

```

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

LOC  OBJECT CODE  LINE SOURCE TEXT
VALUE

      00001 ;
      00002 ; Sample MPASM Source Code. For illustration only.
      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

```

The listing file format produced by MPASM is straight forward:

The product name and version, the assembly date and time, and the page number appear at the top of every page.

The first column of numbers contains the base address in memory where the code will be placed. The second column displays the 32-bit value of any symbols created with the SET, EQU, VARIABLE, CONSTANT, or CBLOCK

MPASM User's Guide with MPLINK and MPLIB

directives. The third column is reserved for the machine instruction. This is the code that will be executed by the PICmicro MCU. The fourth column lists the associated source file line number for this line. The remainder of the line is reserved for the source code line that generated the machine code.

Errors, warnings, and messages are embedded between the source lines, and pertain to the following source line.

The symbol table lists all symbols defined in the program. The memory usage map gives a graphical representation of memory usage. 'X' marks a used location and '-' marks memory that is not used by this object. The memory map is not printed if an object file is generated.

2.5.3 Error File Format (.ERR)

MPASM by default generates an error file. This file can be useful when debugging your code. The MPLAB Source Level Debugger will automatically open this file in the case of an error. The format of the messages in the error file is:

```
<type>[<number>] <file> <line> <description>
```

For example:

```
Error[113] C:\PROG.ASM 7 : Symbol not previously defined (start)
Appendix C describes the error messages generated by MPASM.
```

2.5.4 Hex File Formats (.HEX, .HXL, .HXH)

MPASM is capable of producing different hex file formats. See Appendix A for a detailed description of these formats.

2.5.5 Symbol and Debug File Format (.COD)

When MPASM is used to generate absolute code, it produces a COD file for use in MPLAB debugging of code.

2.5.6 Object File Format (.O)

Object files are the relocatable code produced from source files.

Chapter 3. Using MPASM with DOS

3.1 Introduction

This chapter is dedicated to describing the versions of MPASM for DOS (MPASM.EXE and MPASM_DP.EXE). The DOS version (MPASM.EXE) is run from a command line or a DOS shell in DOS or a DOS window in Windows. The extended DOS (MPASM_DP.EXE) version is run in the same manner but can be used when the DOS version runs out of memory.

3.2 Highlights

Topics covered in this chapter:

- Command Line Interface
- DOS Shell Interface

3.3 Command Line Interface

MPASM can be invoked through the command line interface as follows:

```
MPASM [/<Option>[ /<Option>...]] [<filename>]
```

or

```
MPASM_DP [/<Option>[ /<Option>...]] [<filename>]
```

Where

/<Option> – refers to one of the command line options

<filename> – is the file being assembled

For example, if test.asm exists in the current directory, it can be assembled with following command:

```
MPASM /e /l test
```

The assembler defaults (noted below) can be overridden with options:

- | | |
|-----------------------|---|
| • /<option> | enables the option |
| • /<option>+ | enables the option |
| • /<option>- | disables the option |
| • /<option><filename> | if appropriate, enables the option and directs the output to the specified file |

If the source filename is omitted, the appropriate shell interface is invoked.

MPASM User's Guide with MPLINK and MPLIB

Table 3.1: Assembler Command Line Options

Option	Default	Description
?	N/A	Displays the MPASM Help Panel
a	INHX8M	Set hex file format: /a<hex-format> where <hex-format> is one of [INHX8M INHX8S INHX32]
c	On	Enable/Disable case sensitivity
d	None	Define symbol: /dDebug /dMax=5 /dString="abc"
e	On	Enable/Disable/Set Path for error file /e Enable /e + Enable /e - Disable /e <path>error.file Enables/sets path
h	N/A	Displays the MPASM Help Panel
l	On	Enable/Disable/Set Path for the listing file /l Enable /l + Enable /l - Disable /l <path>list.file Enables/sets path
m	On	Enable/Disable macro expansion
o	Off	Enable/Disable/Set Path for the object file /o Enable /o + Enable /o - Disable /o <path>object.file Enables/sets path
p	None	Set the processor type: /p<processor_type> where <processor_type> is a PICmicro device. For example, PIC16C54
q	Off	Enable/Disable quiet mode (suppress screen output)
r	Hex	Defines default radix: /r<radix> where <radix> is one of [HEX DEC OCT]
t	8	List file tab size: /t<size>

Table 3.1: Assembler Command Line Options (Continued)

w	0	Set message level: /w<level> where <level> is one of [0 1 2] 0 – all messages 1 – errors and warnings 2 – errors only
x	Off	Enable/Disable/Set Path for cross reference file /x Enable /x + Enable /x - Disable /x <path>xref.file Enables/sets path

MPASM User's Guide with MPLINK and MPLIB

3.4 DOS Shell Interface

The MPASM DOS Shell interface displays a screen in Text Graphics mode. On this screen, you can fill in the name of the source file you want to assemble and other information.

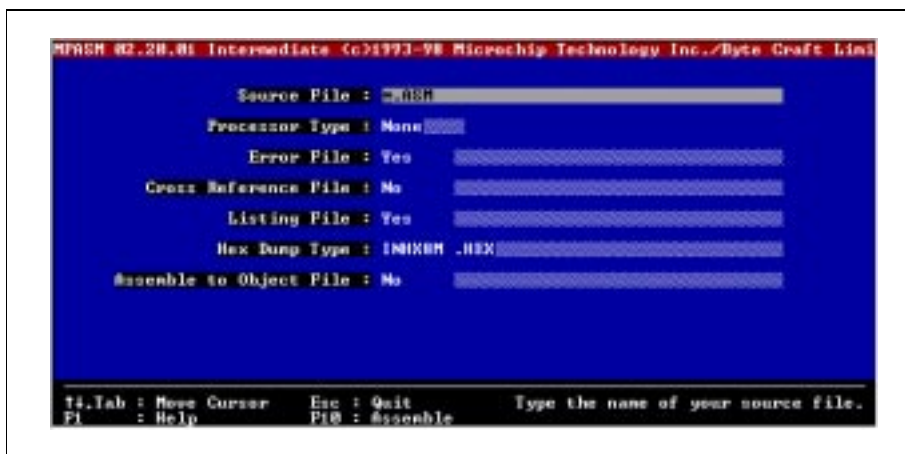


Figure 3.1: Text Graphics Mode Display

3.4.1 Source File

Type the name of your source file. The name can include a DOS path and wild cards. If you use wild cards (one of * or ?), a list of all matching files is displayed for you to select from. To automatically enter *.ASM in this field, press <TAB>.

3.4.2 Processor Type

If you do not specify the processor in your source file, use this field to select the processor. Enter the field by using the arrow keys, then toggle through the processors by pressing <RET>.

3.4.3 Error File

An error file (<sourcename>.ERR) is created by default. To turn the error file off, use the <↓> to move to the YES and press <RET> to change it to NO. The error filename can be changed by pressing the <TAB> key to move to the shaded area and typing a new name. Wild cards are not allowed in the error filename.

Using MPASM with DOS

3.4.4 Cross Reference File

A cross reference file (<sourcename>.XRF) is not generated by default. To create a cross reference file, use the keyboard arrow keys to move to the NO and press <RET> to change it to YES. The cross reference filename can be changed by pressing the <TAB> key to move to the shaded area and typing a new name. Wild cards are not allowed in the cross reference filename.

3.4.5 Listing File

A listing file (<sourcename>.LST) is created by default. To turn the listing file off, use the <↓> to move to the YES and press <RET> to change it to NO. The listing filename can be changed by pressing the <TAB> key to move to the shaded area and typing a new name. Wild cards are not allowed in the listing filename.

3.4.6 HEX Dump Type

Set this value to generate the desired hex file format. Changing this value is accomplished by moving to the field with the <↓> key and pressing the <RET> key to scroll through the available options. To change the hex filename, press the <TAB> key to move the shaded area, and type in the new name.

3.4.7 Assemble to Object File

Enabling this option will generate the relocatable object code that can be input to the linker and suppress generation of the hex file. The filename may be modified in the same manner as the error file.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 4. Using MPASM with Windows and MPLAB

4.1 Introduction

This chapter is dedicated to describing the version of MPASM for Windows (MPASMWIN.EXE). This version may be run stand-alone as a Windows shell, or within MPLAB.

4.2 Highlights

Topics covered in this chapter:

- Windows Shell Interface
- MPLAB Projects and MPASM
- Setting Up MPLAB to use MPASM
- Generating Output Files
- MPLAB/MPASM Troubleshooting

4.3 Windows Shell Interface

MPASM for Windows provides a graphical interface for setting assembler options. It is invoked by executing MPASMWIN.EXE while in Windows.

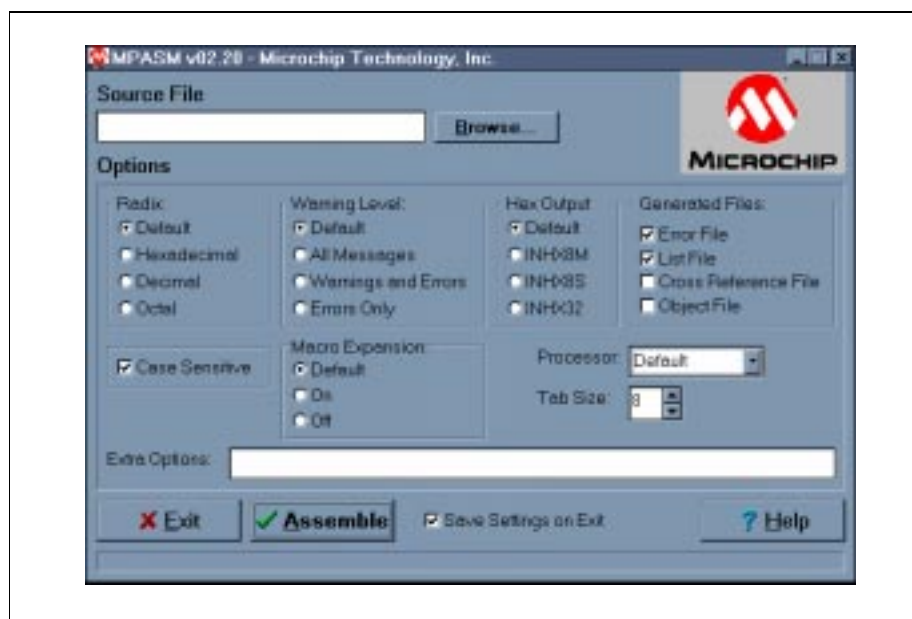


Figure 4.1: MPASM Windows Shell Interface

MPASM User's Guide with MPLINK and MPLIB

Select a source file by typing in the name or using the **Browse** button. Set the various options as described below. Then click **Assemble** to assemble the source file.

Note: When MPASM for Windows is invoked through MPLAB, the options screen is not available. Refer to the Make Setup option in the *MPLAB User's Guide* for selecting assembly options in MPLAB.

Option	Usage
Radix	Override any source file radix settings.
Warning Level	Override any source file message level settings.
Hex Output	Override any source file hex file format settings.
Generated Files	Enable/disable various output files.
Case Sensitivity	Enable/disable case sensitivity.
Macro Expansion	Override any source file macro expansion settings.
Processor	Override any source file processor settings.
Tab Size	Set the list file tab size.
Extra Options	Any additional command line options. See Section 3.3 for more details.
Save Settings on Exit	Save these settings in MPLAB.INI. They will be used the next time you run MPASMWIN.

Using MPASM with Windows and MPLAB

4.4 MPLAB Projects and MPASM

MPLAB projects are composed of nodes (Figure 4.2). These represent files used by a generated project.

- Target Node – Final Output
 - HEX File
- Project Nodes – Components
 - Assembly Source Files

In this chapter, we are concerned with the relationship between MPLAB and MPASM. For more general information about MPLAB projects, consult the *MPLAB User's Guide* (DS51025).

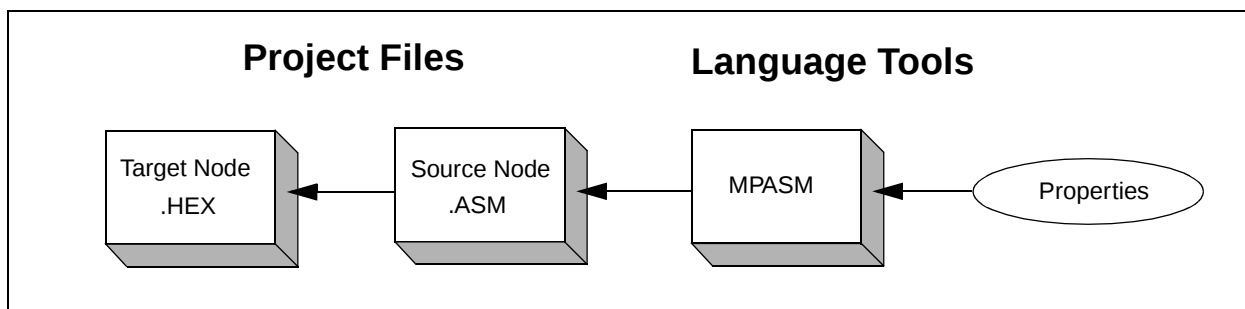


Figure 4.2: Project Relationships – MPASM

Projects are used to apply language tools, such as assemblers, compilers and linkers, to source files in order to make executable (.HEX) files. This diagram shows the relationship between the final .HEX file and the component .ASM files used to create it.

MPASM User's Guide with MPLINK and MPLIB

4.5 Setting Up MPLAB to use MPASM

Follow these steps to set up MPASM to use with MPLAB:

1. After creating your project (*Project>New Project*), the Edit Project dialog will appear. Select the name of the project (ex: tutor84.hex) in the Project Files list to activate the **Node Properties** button. Then click on **Node Properties**.



Figure 4.3: Edit Project Dialog

Using MPASM with Windows and MPLAB

- In the Node Properties dialog, select MPASM as the Language Tool and, if desired, set other assembler options.

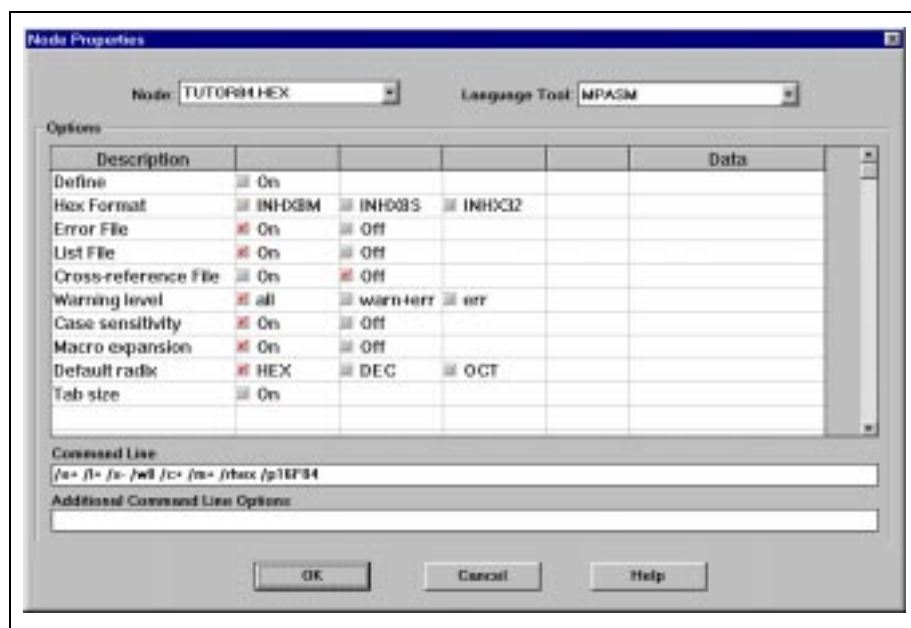


Figure 4.4: Node Properties Dialog

- Add source files to your project. Use the **Add Node** button of the Edit Project dialog to bring up the Add Node dialog.

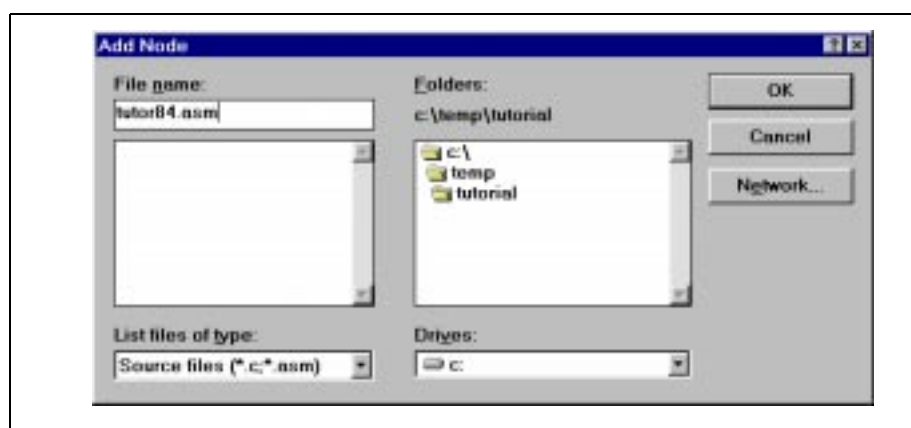


Figure 4.5: Add Node Dialog – Source Files

MPASM User's Guide with MPLINK and MPLIB

- Click **OK** on the Edit Project dialog when you are done.



Figure 4.6: Edit Project Dialog – Nodes Added

4.6 Generating Output Files

Once the MPLAB project is set up, you must build it. Select *Project>Make Project* to do this. Hex files are automatically loaded into program memory.

In addition to the hex file, MPASM generates other files to help you debug your code. See Section 2.5 for more information on these files and their specific functions.

4.7 MPLAB/MPASM Troubleshooting

If have experienced problems, check the following:

Select *Project>Install Language Tool...* and check that MPASM points to MPASMWIN.EXE in the MPLAB installation directory and that the Windowed option is selected. Alternatively, MPASM can point to MPASM.EXE, but the Command-line option should be selected.

Using MPASM with Windows and MPLAB

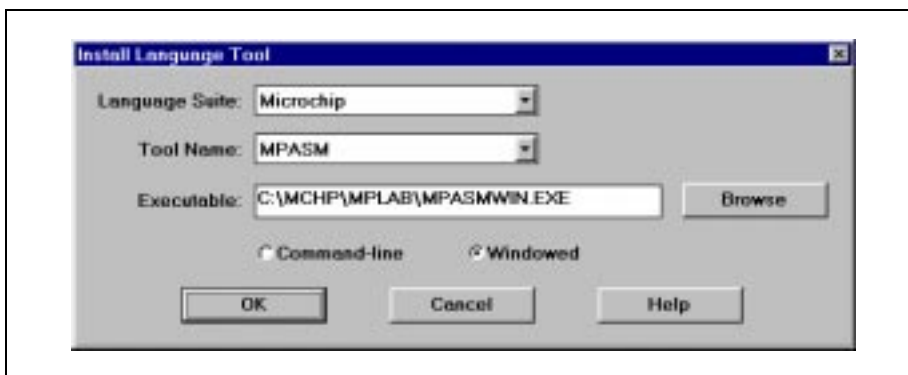


Figure 4.7: Install Language Tool Dialog - MPASM

If you are using MPASM.EXE and get a message from DOS saying that you have run out of environment space, use Microsoft Windows Internet Explorer to select the MPASM.EXE file in the MPLAB installation directory, and click on the right mouse button to bring up the Properties dialog.

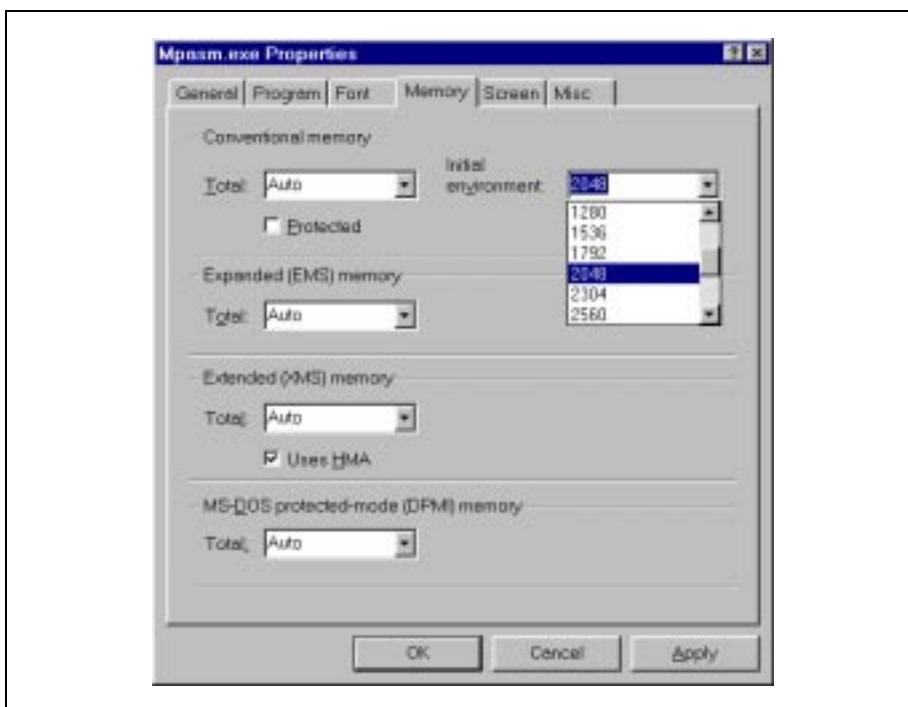


Figure 4.8: Properties Dialog - MPASM.EXE

Increase the size of the Initial Environment. Usually a setting of 2048 will suffice, but if you have a lot of applications that set variables and add to your path statement in your AUTOEXEC.BAT file, you may need to make it larger.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 5. Directive Language

5.1 Introduction

This chapter describes the MPASM directive language.

Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the assembler: its input, output, and data allocation.

Many of the assembler directives have alternate names and formats. These may exist to provide backward compatibility with previous assemblers from Microchip and to be compatible with individual programming practices. If portable code is desired, it is recommended that programs be written using the specifications contained within this document.

5.2 Highlights

There are five basic types of directives provided by MPASM:

- Control Directives – Control directives permit sections of conditionally assembled code.
- Data Directives – Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.
- Listing Directives – Listing Directives are those directives that control the MPASM listing file format. They allow the specification of titles, pagination, and other listing control.
- Macro Directives – These directives control the execution and data allocation within macro body definitions.
- Object File Directives – These directives are used only when creating an object file.

5.3 Directive Summary

Table 5.1 contains a summary of directives supported by MPASM. The remainder of this chapter is dedicated to providing a detailed description of the directives supported by MPASM. Each definition will show:

- Syntax
- Description
- Example

MPASM User's Guide with MPLINK and MPLIB

Table 5.1: Directive Summary

Directive	Description	Syntax
<code>_ _BADRAM</code>	Specify invalid RAM locations	<code>_ _badram <expr>[-<expr>][, <expr>[-<expr>]]</code>
<code>BANKISEL</code>	Generate RAM bank selecting code for indirect addressing	<code>bankisel <label></code>
<code>BANKSEL</code>	Generate RAM bank selecting code	<code>banksel <label></code>
<code>CBLOCK</code>	Define a Block of Constants	<code>cblock [<expr>]</code>
<code>CODE</code>	Begins executable code section	<code>[<name>] code [<address>]</code>
<code>_ _CONFIG</code>	Specify configuration bits	<code>_ _config <expr> OR _ _config <addr>, <expr></code>
<code>CONSTANT</code>	Declare Symbol Constant	<code>constant <label>[=<expr>, ..., <label>[=<expr>]]</code>
<code>DA</code>	Store Strings in Program Memory	<code>[<label>] da <expr> [, <expr2>, ..., <exprn>]</code>
<code>DATA</code>	Create Numeric and Text Data	<code>[<label>] data <expr>[, <expr>, ..., <expr>] [<label>] data "<text_string>"["<text_string>", ...]</code>
<code>DB</code>	Declare Data of One Byte	<code>[<label>] db <expr>[, <expr>, ..., <expr>] [<label>] db "<text_string>"["<text_string>", ...]</code>
<code>DE</code>	Define EEPROM Data	<code>[<label>] de <expr>[, <expr>, ..., <expr>] [<label>] de "<text_string>"["<text_string>", ...]</code>
<code>#DEFINE</code>	Define a Text Substitution Label	<code>define <name> [<value>] define <name> [<arg>, ..., <arg>] <value></code>
<code>DT</code>	Define Table	<code>[<label>] dt <expr>[, <expr>, ..., <expr>] [<label>] dt "<text_string>"["<text_string>", ...]</code>
<code>DW</code>	Declare Data of One Word	<code>[<label>] dw <expr>[, <expr>, ..., <expr>] [<label>] dw "<text_string>"["<text_string>", ...]</code>
<code>ELSE</code>	Begin Alternative Assembly Block to IF	<code>else</code>
<code>END</code>	End Program Block	<code>end</code>
<code>ENDC</code>	End an Automatic Constant Block	<code>endc</code>
<code>ENDIF</code>	End conditional Assembly Block	<code>endif</code>
<code>ENDM</code>	End a Macro Definition	<code>endm</code>
<code>ENDW</code>	End a While Loop	<code>endw</code>
<code>EQU</code>	Define an Assembly Constant	<code><label> equ <expr></code>
<code>ERROR</code>	Issue an Error Message	<code>error "<text_string>"</code>
<code>ERRORLEVEL</code>	Set Error Level	<code>errorlevel 0 1 2 <+ -><message number></code>

Directive Language

Table 5.1: Directive Summary (Continued)

Directive	Description	Syntax
EXITM	Exit from a Macro	exitm
EXPAND	Expand Macro Listing	expand
EXTERN	Declares an external label	extern <label>[,<label>]
FILL	Fill Memory	[<label>] fill <expr>, <count>
GLOBAL	Exports a defined label	global <label>[,<label>]
IDATA	Begins initialized data section	[<name>] idata [<address>]
_ _IDLOCS	Specify ID locations	_ _idlocs <expr>
IF	Begin Conditionally Assembled Code Block	if <expr>
IFDEF	Execute If Symbol has Been Defined	ifdef <label>
IFNDEF	Execute If Symbol has not Been Defined	ifndef <label>
#INCLUDE	Include Additional Source File	include <<include_file>> "<include_file>"
LIST	Listing Options	list [<list_option>, ..., <list_option>]
LOCAL	Declare Local Macro Variable	local <label>[,<label>]
MACRO	Declare Macro Definition	<label> macro [<arg>, ..., <arg>]
_ _MAXRAM	Specify maximum RAM address	_ _maxram <expr>
MESSG	Create User Defined Message	messg "<message_text>"
NOEXPAND	Turn off Macro Expansion	noexpand
NOLIST	Turn off Listing Output	nolist
ORG	Set Program Origin	<label> org <expr>
PAGE	Insert Listing Page Eject	page
PAGESEL	Generate ROM page selecting code	pagesel <label>
PROCESSOR	Set Processor Type	processor <processsor_type>
RADIX	Specify Default Radix	radix <default_radix>
RES	Reserve Memory	[<label>] res <mem_units>
SET	Define an Assembler Variable	<label> set <expr>
SPACE	Insert Blank Listing Lines	space <expr>
SUBTITLE	Specify Program Subtitle	subtitle "<sub_text>"

MPASM User's Guide with MPLINK and MPLIB

Table 5.1: Directive Summary (Continued)

Directive	Description	Syntax
TITLE	Specify Program Title	title "<title_text>"
UDATA	Begins uninitialized data section	[<name>] udata [<address>]
UDATA_ACS	Begins access uninitialized data section	[<name>] udata_acs [<address>]
UDATA_OVR	Begins overlayed uninitialized data section	[<name>] udata_ovr [<address>]
UDATA_SHR	Begins shared uninitialized data section	[<name>] udata_shr [<address>]
#UNDEFINE	Delete a Substitution Label	#undefine <label>
VARIABLE	Declare Symbol Variable	variable <label>[=<expr>, ..., <label>[=<expr>]]
WHILE	Perform Loop While Condition is True	while <expr>

Directive Language

5.4 `__BADRAM` – Identify Unimplemented RAM

5.4.1 Syntax

```
__badram    <expr>[-<expr>][, <expr>[-<expr>]]
```

5.4.2 Description

The `__MAXRAM` and `__BADRAM` directives together flag accesses to unimplemented registers. `__BADRAM` defines the locations of invalid RAM addresses. This directive is designed for use with the `__MAXRAM` directive. A `__MAXRAM` directive must proceed any `__BADRAM` directive. Each `<expr>` must be less than or equal to the value specified by `__MAXRAM`. Once the `__MAXRAM` directive is used, strict RAM address checking is enabled, using the RAM map specified by `__BADRAM`. To specify a range of invalid locations, use the syntax `<minloc> - <maxloc>`.

5.4.3 Example

See the example for `__MAXRAM`.

5.4.4 See Also

`__MAXRAM`

5.5 `BANKSEL` – Generate Indirect Bank Selecting Code

5.5.1 Syntax

```
bankisel <label>
```

5.5.2 Description

For use when generating an object file. This directive is an instruction to the linker to generate the appropriate bank selecting code for an indirect access of the address specified by `<label>`. Only one `<label>` should be specified. No operations can be performed on `<label>`. `<label>` must have been previously defined.

The linker will generate the appropriate bank selecting code. For 14-bit core devices, the appropriate bit set/clear instruction on the IRP bit in the STATUS register will be generated. For the 16-bit core devices, `MOVLB` or `MOVLR` will be generated. If the user can completely specify the indirect address without these instructions, no code will be generated.

For more information, refer to Chapter 6.

MPASM User's Guide with MPLINK and MPLIB

5.5.3 Example

```
movlw    Var1
movwf    FSR
bankisel Var1
...
movwf    INDF
```

5.5.4 See Also

BANKSEL PAGESEL

5.6 BANKSEL – Generate Bank Selecting Code

5.6.1 Syntax

```
banksel <label>
```

5.6.2 Description

For use when generating an object file. This directive is an instruction to the linker to generate bank selecting code to set the bank to the bank containing the designated <label>. Only one <label> should be specified. No operations can be performed on <label>. <label> must have been previously defined.

The linker will generate the appropriate bank selecting code. For 12-bit core devices, the appropriate bit set/clear instructions on the FSR will be generated. For 14-bit devices, bit set/clear instructions on the STATUS register will be generated. For the 16-bit core devices, MOVLB or MOVLR will be generated. For the enhanced 16-bit core devices, MOVLB will be generated. If the device contains only one bank of RAM, no instructions will be generated.

For more information, refer to Chapter 6.

5.6.3 Example

```
banksel Var1
movwf    Var1
```

5.6.4 See Also

BANKSEL PAGESEL

Directive Language

5.7 CBLOCK – Define a Block of Constants

5.7.1 Syntax

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

5.7.2 Description

Define a list of named constants. Each <label> is assigned a value of one higher than the previous <label>. The purpose of this directive is to assign address offsets to many labels. The list of names end when an ENDC directive is encountered.

<expr> indicates the starting value for the first name in the block. If no expression is found, the first name will receive a value one higher than the final name in the previous CBLOCK. If the first CBLOCK in the source file has no <expr>, assigned values start with zero.

If <increment> is specified, then the next <label> is assigned the value of <increment> higher than the previous <label>.

Multiple names may be given on a line, separated by commas.

cblock is useful for defining constants in program and data memory.

5.7.3 Example

```
cblock 0x20      ; name_1 will be
                  ; assigned 20
    name_1, name_2 ; name_2, 21 and so on
    name_3, name_4 ; name_4 is assigned 23.
endc
cblock 0x30
    TwoByteVar: 0, TwoByteHigh, TwoByteLow
    Queue: QUEUE_SIZE
    QueueHead, QueueTail
    Double1:2, Double2:2
endc
```

5.7.4 See Also

ENDC

5.8 CODE – Begin an Object File Code Section

5.8.1 Syntax

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

MPASM User's Guide with MPLINK and MPLIB

5.8.2 Description

For use when generating an object file. Declares the beginning of a section of program code. If <label> is not specified, the section is named .code. The starting address is initialized to the specified address or will be assigned at link time if no address is specified.

Note: Two sections in the same source file may not have the same name.

For more information, refer to Chapter 6.

5.8.3 Example

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

5.8.4 See Also

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5.9 __CONFIG – Set Processor Configuration Bits

5.9.1 Syntax

```
__config <expr> OR __config <addr>, <expr>
```

5.9.2 Description

Sets the processor's configuration bits to the value described by <expr>. For PIC18CXX devices, the address of a valid configuration byte must also be specified by <addr>. Refer to the PICmicro Microcontroller Data Book for a description of the configuration bits for each processor.

Before this directive is used, the processor must be declared through the command line, the LIST directive, or the PROCESSOR directive. If this directive is used with the PIC17CXX family, the hex file output format must be set to INHX32 through the command line or the LIST directive.

5.9.3 Example

```
list p=17c42,f=INHX32  
__config H'FFFF' ;Default configuration bits
```

5.9.4 See Also

__IDLOCS LIST PROCESSOR

Directive Language

5.10 CONSTANT – Declare Symbol Constant

5.10.1 Syntax

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

5.10.2 Description

Creates symbols for use in MPASM expressions. Constants may not be reset after having once been initialized, and the expression must be fully resolvable at the time of the assignment. This is the principal difference between symbols declared as CONSTANT and those declared as VARIABLE, or created by the SET directive. Otherwise, constants and variables may be used interchangeably in expressions.

5.10.3 Example

```
variable RecLength=64          ; Set Default
                                ;   RecLength
constant BufLength=512        ; Init BufLength
                                ; RecLength may
                                ; be reset later
                                ; in RecLength=128
                                ;
constant MaxMem=RecLength+BufLength ; CalcMaxMem
```

5.10.4 See Also

SET VARIABLE

5.11 DA – Store Strings in Program Memory

5.11.1 Syntax

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

5.11.2 Description

Generates a packed 14-bit number representing two 7-bit ASCII characters. This is useful for storing strings in memory for the PICmicro Flash ROM devices.

5.11.3 Examples

da "abcdef"

will put 30E2 31E4 32E6 3380 into program memory

da "12345678" ,0

will put 18B2 19B4 1AB6 0000 into program memory

da 0xFFFF

will put 0x3FFF into program memory

5.12 DATA – Create Numeric and Text Data

5.12.1 Syntax

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

5.12.2 Description

Initialize one or more words of program memory with data. The data may be in the form of constants, relocatable or external labels, or expressions of any of the above. The data may also consist of ASCII character strings, <text_string>, enclosed in single quotes for one character or double quotes for strings. Single character items are placed into the low byte of the word, while strings are packed two to a word. If an odd number of characters are given in a string, the final byte is zero. On all families except the PIC18CXX, the first character is in the most significant byte of the word. On the PIC18CXX, the first character is in the least significant byte of the word.

When generating an object file, this directive can also be used to declare initialized data values. Refer to the IDATA directive for more information.

Directive Language

5.12.3 Example

```
data reloc_label+10    ; constants
data 1,2,ext_label     ; constants, externals
data "testing 1,2,3"   ; text string
data 'N'               ; single character
data start_of_program ; relocatable label
```

5.12.4 See Also

DB DE DT DW IDATA

5.13 DB – Declare Data of One Byte

5.13.1 Syntax

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

5.13.2 Description

Reserve program memory words with packed 8-bit values. Multiple expressions continue to fill bytes consecutively until the end of expressions. Should there be an odd number of expressions, the last byte will be zero.

When generating an object file, this directive can also be used to declare initialized data values. Refer to the IDATA directive for more information.

5.13.3 Example

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

5.13.4 See Also

DATA DE DT DW IDATA

5.14 DE – Declare EEPROM Data Byte

5.14.1 Syntax

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

5.14.2 Description

Reserve memory words with 8-bit data. Each <expr> must evaluate to an 8-bit value. The upper bits of the program word are zeroes. Each character in a string is stored in a separate word.

Although designed for initializing EEPROM data on the PIC16C8X, the directive can be used at any location for any processor.

MPASM User's Guide with MPLINK and MPLIB

5.14.3 Example

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

5.14.4 See Also

DATA DB DT DW

5.15 #DEFINE – Define a Text Substitution Label

5.15.1 Syntax

```
#define <name> [<string>]
```

5.15.2 Description

This directive defines a text substitution string. Wherever <name> is encountered in the assembly code, <string> will be substituted.

Using the directive with no <string> causes a definition of <name> to be noted internally and may be tested for using the IFDEF directive.

This directive emulates the ANSI 'C' standard for #define. Symbols defined with this method are not available for viewing using MPLAB.

5.15.3 Example

```
#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      ; set bit 7 in f19
```

5.15.4 See Also

#UNDEFINE IFDEF IFNDEF

Directive Language

5.16 DT – Define Table

5.16.1 Syntax

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

5.16.2 Description

Generates a series of RETLW instructions, one instruction for each <expr>. Each <expr> must be an 8-bit value. Each character in a string is stored in its own RETLW instruction.

5.16.3 Example

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

5.16.4 See Also

DATA DB DE DW

5.17 DW – Declare Data of One Word

5.17.1 Syntax

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

5.17.2 Description

Reserve program memory words for data, initializing that space to specific values. For PIC18CXX devices, DW functions like DB. Values are stored into successive memory locations and the location counter is incremented by one. Expressions may be literal strings and are stored as described in the DATA directive.

When generating an object file, this directive can also be used to declare initialized data values. Refer to the IDATA directive for more information.

5.17.3 Example

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

5.17.4 See Also

DATA DB IDATA

5.18 ELSE – Begin Alternative Assembly Block to IF

5.18.1 Syntax

else

5.18.2 Description

Used in conjunction with an IF directive to provide an alternative path of assembly code should the IF evaluate to false. ELSE may be used inside a regular program block or macro.

5.18.3 Example

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

5.18.4 See Also

ENDIF IF

5.19 END – End Program Block

5.19.1 Syntax

end

5.19.2 Description

Indicates the end of the program.

5.19.3 Example

```
list p=17c42
:      ; executable code
:      ;
end     ; end of instructions
```

Directive Language

5.20 ENDC – End an Automatic Constant Block

5.20.1 Syntax

endc

5.20.2 Description

ENDC terminates the end of a CBLOCK list. It must be supplied to terminate the list.

5.20.3 See Also

CBLOCK

5.21 ENDIF – End Conditional Assembly Block

5.21.1 Syntax

endif

5.21.2 Description

This directive marks the end of a conditional assembly block. ENDIF may be used inside a regular program block or macro.

5.21.3 See Also

ELSE IF

5.22 ENDM – End a Macro Definition

5.22.1 Syntax

endm

5.22.2 Description

Terminates a macro definition begun with MACRO.

5.22.3 Example

```
make_table macro arg1, arg2
    dw arg1, 0 ; null terminate table name
    res arg2   ; reserve storage
endm
```

MPASM User's Guide with MPLINK and MPLIB

5.22.4 See Also

MACRO EXITM

5.23 ENDW – End a While Loop

5.23.1 Syntax

endw

5.23.2 Description

ENDW terminates a WHILE loop. As long as the condition specified by the WHILE directive remains true, the source code between the WHILE directive and the ENDW directive will be repeatedly expanded in the assembly source code stream. This directive may be used inside a regular program block or macro.

5.23.3 Example

See the example for WHILE

5.23.4 See Also

WHILE

5.24 EQU – Define an Assembler Constant

5.24.1 Syntax

<label> equ <expr>

5.24.2 Description

The value of <expr> is assigned to <label>.

5.24.3 Example

```
four equ 4 ; assigned the numeric value of 4
           ; to label four
```

5.24.4 See Also

SET

Directive Language

5.25 ERROR – Issue an Error Message

5.25.1 Syntax

```
error "<text_string>"
```

5.25.2 Description

<text_string> is printed in a format identical to any MPASM error message. <text_string> may be from 1 to 80 characters.

5.25.3 Example

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

5.25.4 See Also

MESSG

5.26 ERRORLEVEL – Set Message Level

5.26.1 Syntax

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

5.26.2 Description

Sets the types of messages that are printed in the listing file and error file.

Setting	Affect
0	Messages, warnings, and errors printed
1	Warnings and errors printed
2	Errors printed
-<msgnum>	Inhibits printing of message <msgnum>
+<msgnum>	Enables printing of message <msgnum>

The values for <msgnum> are in Appendix C. Error messages cannot be disabled. The setting of 0, 1, or 2 overrides individual message disabling or enabling.

5.26.3 Example

```
errorlevel 1, -202
```

MPASM User's Guide with MPLINK and MPLIB

5.26.4 See Also

LIST

5.27 EXITM – Exit from a Macro

5.27.1 Syntax

exitm

5.27.2 Description

Force immediate return from macro expansion during assembly. The effect is the same as if an ENDM directive had been encountered.

5.27.3 Example

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

5.27.4 See Also

ENDM MACRO

5.28 EXPAND – Expand Macro Listing

5.28.1 Syntax

expand

5.28.2 Description

Expand all macros in the listing file. This directive is roughly equivalent to the /m MPASM command line option, but may be disabled by the occurrence of a subsequent NOEXPAND.

5.28.3 See Also

MACRO NOEXPAND

Directive Language

5.29 EXTERN – Declare an Externally Defined Label

5.29.1 Syntax

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

5.29.2 Description

For use when generating an object file. Declares symbol names that may be used in the current module but are defined as global in a different module. The EXTERN statement must be included before the <label> is used. At least one label must be specified on the line. If <label> is defined in the current module, MPASM will generate a duplicate label error.

For more information, refer to Chapter 6.

5.29.3 Example

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

5.29.4 See Also

GLOBAL IDATA TEXT UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5.30 FILL – Specify Memory Fill Value

5.30.1 Syntax

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

5.30.2 Description

Generates <count> occurrences of the program word or byte (PIC18CXX devices), <expr>. If bounded by parentheses, <expr> can be an assembler instruction.

5.30.3 Example

```
fill 0x1009, 5 ; fill with a constant
fill (GOTO RESET_VECTOR), NEXT_BLOCK-$
```

5.30.4 See Also

DATA DW ORG

5.31 GLOBAL – Export a Label

5.31.1 Syntax

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

5.31.2 Description

For use when generating an object file. Declares symbol names that are defined in the current module and should be available to other modules. The GLOBAL statement must be after the <label> is defined. At least one label must be specified on the line.

For more information, refer to Chapter 6.

5.31.3 Example

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

5.31.4 See Also

EXTERN IDATA TEXT UDATA UDATA_ACS UDATA_OVR UDATA_SHR

5.32 IDATA – Begin an Object File Initialized Data Section

5.32.1 Syntax

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

5.32.2 Description

For use when generating an object file. Declares the beginning of a section of initialized data. If <label> is not specified, the section is named .idata. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. No code can be generated in this segment.

Directive Language

The linker will generate a look-up table entry for each byte specified in an `idata` section. The user must then link or include the appropriate initialization code. See Chapter 9 for examples of initialization codes for various PICmicro families. Note that this directive is not available for 12-bit core devices.

The `RES`, `DB` and `DW` directives may be used to reserve space for variables. `RES` will generate an initial value of zero. `DB` will initialize successive bytes of RAM. `DW` will initialize successive bytes of RAM, one word at a time, in low-byte/high-byte order.

For more information, refer to Chapter 6.

5.32.3 Example

```

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

5.32.4 See Also

`EXTERN` `GLOBAL` `TEXT` `UDATA` `UDATA_ACS` `UDATA_OVR` `UDATA_SHR`

5.33 __IDLOCS – Set Processor ID Locations

5.33.1 Syntax

`__idlocs <expr>` or `__idlocs <expr1>, <expr2>`

5.33.2 Description

For PIC12CXX, PIC14000, and PIC16CXX devices, `__idlocs` sets the four ID locations to the hexadecimal value of `<expr>`. For PIC18CXX devices, `__idlocs` sets the two-byte device ID `<expr1>` to the hexadecimal value of `<expr2>`. This directive is not valid for the PIC17CXX family.

For example, if `<expr>` evaluates to 1AF, the first (lowest address) ID location is zero, the second is one, the third is ten, and the fourth is fifteen.

Before this directive is used, the processor must be declared through the command line, the `LIST` directive, or the `PROCESSOR` directive.

5.33.3 Example

```
__idlocs H'1234'
```

5.33.4 See Also

`LIST` `PROCESSOR` `__CONFIG`

5.34 IF – Begin Conditionally Assembled Code Block

5.34.1 Syntax

`if <expr>`

5.34.2 Description

Begin execution of a conditional assembly block. If <expr> evaluates to true, the code immediately following the IF will assemble. Otherwise, subsequent code is skipped until an ELSE directive or an ENDIF directive is encountered.

An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. The IF and WHILE directives operate on the logical value of an expression. A relational TRUE expression is guaranteed to return a nonzero value, FALSE a value of zero.

5.34.3 Example

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

5.34.4 See Also

ELSE ENDIF

5.35 IFDEF – Execute If Symbol has Been Defined

5.35.1 Syntax

`ifdef <label>`

5.35.2 Description

If <label> has been previously defined, usually by issuing a #DEFINE directive or by setting the value on the MPASM command line, the conditional path is taken. Assembly will continue until a matching ELSE or ENDIF directive is encountered.

Directive Language

5.35.3 Example

```
#define testing 1          ; set testing "on"
:
:
ifdef testing
    <execute test code> ; this path would
endif                    ; be executed.
```

5.35.4 See Also

#DEFINE ELSE ENDIF IFNDEF #UNDEFINE

5.36 IFNDEF – Execute If Symbol has not Been Defined

5.36.1 Syntax

```
ifndef <label>
```

5.36.2 Description

If <label> has not been previously defined, or has been undefined by issuing an #UNDEFINE directive, then the code following the directive will be assembled. Assembly will be enabled or disabled until the next matching ELSE or ENDIF directive is encountered.

5.36.3 Example

```
#define testing1          ; set testing on
:
:
#undefine testing1        ; set testing off
ifndef testing            ; if not in testing mode
:                          ; execute
:                          ; this path
endif                     ;
:
end                        ; end of source
```

5.36.4 See Also

#DEFINE ELSE ENDIF IFDEF INE #UNDEF

MPASM User's Guide with MPLINK and MPLIB

5.37 INCLUDE – Include Additional Source File

5.37.1 Syntax

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

5.37.2 Description

The specified file is read in as source code. The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement. Upon end-of-file, source code assembly will resume from the original source file. Up to six levels of nesting are permitted. <include_file> may be enclosed in quotes or angle brackets. If a fully qualified path is specified, only that path will be searched. Otherwise, the search order is: current working directory, source file directory, MPASM executable directory.

5.37.3 Example

```
include "c:\sys\sysdefs.inc" ; system defs
include <regs.h>             ; register defs
```

5.38 LIST – Listing Options

5.38.1 Syntax

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

5.38.2 Description

Occurring on a line by itself, the LIST directive has the effect of turning listing output on, if it had been previously turned off. Otherwise, one of the following list options can be supplied to control the assembly process or format the listing file:

Table 5.2: List Directive Options

Option	Default	Description
b=nnn	8	Set tab spaces.
c=nnn	132	Set column width.
f=<format>	INHX8M	Set the hex file output. <format> can be INHX32, INHX8M, or INHX8S.
free	FIXED	Use free-format parser. Provided for backward compatibility.
fixed	FIXED	Use fixed-format parser.
mm={ON OFF}	On	Print memory map in list file.

Directive Language

Table 5.2: List Directive Options (Continued)

Option	Default	Description
n=nnn	60	Set lines per page.
p=<type>	None	Set processor type; for example, PIC16C54.
r=<radix>	hex	Set default radix: hex, dec, oct.
st={ON OFF}	On	Print symbol table in list file.
t={ON OFF}	Off	Truncate lines of listing (otherwise wrap).
w={0 1 2}	0	Set the message level. See ERRORLEVEL.
x={ON OFF}	On	Turn macro expansion on or off.

Note: All LIST options are evaluated as decimal numbers.

5.38.3 Example

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

5.38.4 See Also

ERRORLEVEL EXPAND NOEXPAND NOLIST PROCESSOR RADIX

5.39 LOCAL – Declare Local Macro Variable

5.39.1 Syntax

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

5.39.2 Description

Declares that the specified data elements are to be considered in local context to the macro. <label> may be identical to another label declared outside the macro definition; there will be no conflict between the two.

If the macro is called recursively, each invocation will have its own local copy.

5.39.3 Example

```
<main code segment>
:
:
len    equ 10          ; global version
size   equ 20          ; note that a local variable
                        ; may now be created and modified
test macro size
    local len, label  ; local len and label
len     set size       ; modify local len
label   res len        ; reserve buffer
len     set len-20     ;
endm                  ; end macro
```

MPASM User's Guide with MPLINK and MPLIB

5.39.4 See Also

ENDM MACRO

5.40 MACRO – Declare Macro Definition

5.40.1 Syntax

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

5.40.2 Description

A macro is a sequence of instructions that can be inserted in the assembly source code by using a single macro call. The macro must first be defined, then it can be referred to in subsequent source code.

A macro can call another macro, or may call itself recursively.

Please refer to Chapter 5, "Macro Language" for more information.

5.40.3 Example

```
Read macro device, buffer, count
    movlw device
    movwf ram_20
    movlw buffer ; buffer address
    movwf ram_21
    movlw count ; byte count
    call sys_21 ; read file call
endm
```

5.40.4 See Also

ELSE ENDIF ENDM EXITM IF LOCAL

5.41 __MAXRAM – Define Maximum RAM Location

5.41.1 Syntax

```
__maxram <expr>
```

5.41.2 Description

The __MAXRAM and __BADRAM directives together flag accesses to unimplemented registers. __MAXRAM defines the absolute maximum valid RAM address and initializes the map of valid RAM addresses to all addresses valid at and below <expr>. <expr> must be greater than or equal to the maximum page 0 RAM address and less than 1000H. This directive is

Directive Language

designed for use with the `_ _BADRAM` directive. Once the `_ _MAXRAM` directive is used, strict RAM address checking is enabled, using the RAM map specified by `_ _BADRAM`.

`_ _MAXRAM` can be used more than once in a source file. Each use redefines the maximum valid RAM address and resets the RAM map to all locations.

5.41.3 Example

```
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' ; Generates invalid RAM warning
movwf    H'87' ; Generates invalid RAM warning
              ; and truncation message
```

5.41.4 See Also

`_ _BADRAM`

5.42 MESSG – Create User Defined Message

5.42.1 Syntax

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

5.42.2 Description

Causes an informational message to be printed in the listing file. The message text can be up to 80 characters. Issuing a MESSG directive does not set any error return codes.

5.42.3 Example

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

5.42.4 See Also

ERROR

5.43 NOEXPAND – Turn off Macro Expansion

5.43.1 Syntax

noexpand

5.43.2 Description

Turns off macro expansion in the listing file.

5.43.3 See Also

EXPAND

5.44 NOLIST – Turn off Listing Output

5.44.1 Syntax

nolist

5.44.2 Description

Turn off listing file output.

5.44.3 See Also

LIST

5.45 ORG – Set Program Origin

5.45.1 Syntax

[<label>] org <expr>

5.45.2 Description

Set the program origin for subsequent code at the address defined in <expr>. If <label> is specified, it will be given the value of the <expr>. If no ORG is specified, code generation will begin at address zero.

For PIC18CXX devices, only even <expr> values are allowed.

This directive may not be used when generating an object file.

Directive Language

5.45.3 Example

```
int_1 org 0x20
      ; Vector 20 code goes here
int_2 org int_1+0x10
      ; Vector 30 code goes here
```

5.45.4 See Also

FILL RES

5.46 PAGE – Insert Listing Page Eject

5.46.1 Syntax

page

5.46.2 Description

Inserts a page eject into the listing file.

5.46.3 See Also

LIST SUBTITLE TITLE

5.47 PAGESEL – Generate Page Selecting Code

5.47.1 Syntax

pagesel <label>

5.47.2 Description

For use when generating an object file. An instruction to the linker to generate page selecting code to set the page bits to the page containing the designated <label>. Only one <label> should be specified. No operations can be performed on <label>. <label> must have been previously defined.

The linker will generate the appropriate page selecting code. For 12-bit core devices, the appropriate bit set/clear instructions on the STATUS register will be generated. For 14-bit and 16-bit core devices, MOVLW and MOVWF instructions will be generated to modify the PCLATH. If the device contains only one page of program memory, no code will be generated.

For PIC18CXX devices, this command will do nothing.

For more information, refer to Chapter 6.

MPASM User's Guide with MPLINK and MPLIB

5.47.3 Example

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

5.47.4 See Also

BANKISEL BANKSEL

5.48 PROCESSOR – Set Processor Type

5.48.1 Syntax

```
processor <processor_type>
```

5.48.2 Description

Sets the processor type to <processor_type>.

5.48.3 Example

```
processor 16C54
```

5.48.4 See Also

LIST

5.49 RADIX – Specify Default Radix

5.49.1 Syntax

```
radix <default_radix>
```

5.49.2 Description

Sets the default radix for data expressions. The default radix is hex. Valid radix values are: hex, dec, or oct.

5.49.3 Example

```
radix dec
```

5.49.4 See Also

LIST

Directive Language

5.50 RES – Reserve Memory

5.50.1 Syntax

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

5.50.2 Description

Causes the memory location pointer to be advanced from its current location by the value specified in <mem_units>. In non-relocatable code, <label> is assumed to be a program memory address. In relocatable code (using MPLINK), res can also be used to reserve data storage.

Address locations are defined in words for 12-, 14- and 16-bit PICmicros, and bytes for enhanced 16-bit PICmicros.

5.50.3 Example

```
buffer res 64 ; reserve 64 address locations of storage
```

5.50.4 See Also

FILL ORG

5.51 SET – Define an Assembler Variable

5.51.1 Syntax

```
<label> set <expr>
```

5.51.2 Description

<label> is assigned the value of the valid MPASM expression specified by <expr>. The SET directive is functionally equivalent to the EQU directive except that SET values may be subsequently altered by other SET directives.

5.51.3 Example

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

5.51.4 See Also

EQU VARIABLE

5.52 SPACE – Insert Blank Listing Lines

5.52.1 Syntax

space <expr>

5.52.2 Description

Insert <expr> number of blank lines into the listing file.

5.52.3 Example

space 3 ;Inserts three blank lines

5.52.4 See Also

LIST

5.53 SUBTITLE – Specify Program Subtitle

5.53.1 Syntax

subtitle "<sub_text>"

5.53.2 Description

<sub_text> is an ASCII string enclosed in double quotes, 60 characters or less in length. This directive establishes a second program header line for use as a subtitle in the listing output.

5.53.3 Example

subtitle "diagnostic section"

5.53.4 See Also

TITLE

5.54 TITLE – Specify Program Title

5.54.1 Syntax

title "<title_text>"

Directive Language

5.54.2 Description

<title_text> is a printable ASCII string enclosed in double quotes. It must be 60 characters or less. This directive establishes the text to be used in the top line of each page in the listing file.

5.54.3 Example

```
title "operational code, rev 5.0"
```

5.54.4 See Also

LIST SUBTITLE

5.55 UDATA – Begin an Object File Uninitialized Data Section

5.55.1 Syntax

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

5.55.2 Description

For use when generating an object file. Declares the beginning of a section of uninitialized data. If <label> is not specified, the section is named .udata. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. No code can be generated in this segment. The RES directive should be used to reserve space for data.

Note: Two sections in the same source file may not have the same name.

For more information, refer to Chapter 6.

5.55.3 Example

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

5.55.4 See Also

EXTERN GLOBAL IDATA UDATA_ACS UDATA_OVR UDATA_SHR TEXT

5.56 UDATA_ACS – Begin an Object File Access Uninitialized Data Section

5.56.1 Syntax

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

5.56.2 Description

For use when generating an object file. Declares the beginning of a section of access uninitialized data. If <label> is not specified, the section is named `.udata_acs`. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. This directive is used to declare variables that are allocated in access RAM of PIC18CXX devices. No code can be generated in this segment. The RES directive should be used to reserve space for data.

Note: Two sections in the same source file may not have the same name.

For more information, refer to Chapter 6.

5.56.3 Example

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

5.56.4 See Also

EXTERN GLOBAL IDATA UDATA UDATA_OVR UDATA_SHR TEXT

5.57 UDATA_OVR – Begin an Object File Overlaid Uninitialized Data Section

5.57.1 Syntax

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

5.57.2 Description

For use when generating an object file. Declares the beginning of a section of overlaid uninitialized data. If <label> is not specified, the section is named `.udata_ovr`. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. The space declared by this section is overlaid by all other `udata_ovr` sections of the

Directive Language

same name. It is an ideal way of declaring temporary variables since it allows multiple variables to be declared at the same memory location. No code can be generated in this segment. The `res` directive should be used to reserve space for data.

Note: This is the exception to the rule that two sections in the same source file may not have the same name.

For more information, refer to Chapter 6.

5.57.3 Example

```
Temps      udata_ovr
Temp1      res 1
Temp2      res 1
Temp3      res 1
Temps      udata_ovr
LongTemp1  res 2 ; this will be a variable at the
                  ; same location as Temp1 and Temp2
LongTemp2  res 2 ; this will be a variable at the
                  ; same location as Temp3
```

5.57.4 See Also

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_SHR TEXT

5.58 UDATA_SHR – Begin an Object File Shared Uninitialized Data Section

5.58.1 Syntax

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

5.58.2 Description

For use when generating an object file. Declares the beginning of a section of shared uninitialized data. If `<label>` is not specified, the section is named `.udata_shr`. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. This directive is used to declare variables that are allocated in RAM that is shared across all RAM banks (i.e. unbanked RAM). No code can be generated in this segment. The `RES` directive should be used to reserve space for data.

Note: Two sections in the same source file may not have the same name.

For more information, refer to Chapter 6.

MPASM User's Guide with MPLINK and MPLIB

5.58.3 Example

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

5.58.4 See Also

EXTERN GLOBAL IDATA UDATA UDATA_ACS UDATA_OVR TEXT

5.59 #UNDEFINE – Delete a Substitution Label

5.59.1 Syntax

```
#undefine <label>
```

5.59.2 Description

<label> is an identifier previously defined with the #DEFINE directive. It must be a valid MPASM label. The symbol named is removed from the symbol table.

5.59.3 Example

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

5.59.4 See Also

#DEFINE IFDEF INCLUDE IFNDEF

5.60 VARIABLE – Declare Symbol Variable

5.60.1 Syntax

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

5.60.2 Description

Creates symbols for use in MPASM expressions. Variables and constants may be used interchangeably in expressions.

The VARIABLE directive creates a symbol that is functionally equivalent to those created by the SET directive. The difference is that the VARIABLE directive does not require that symbols be initialized when they are declared.

Directive Language

Note that variable values cannot be updated within an operand. You must place variable assignments, increments, and decrements on separate lines.

5.60.3 Example

Please refer to the example given for the CONSTANT directive.

5.60.4 See Also

CONSTANT SET

5.61 WHILE – Perform Loop While Condition is True

5.61.1 Syntax

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

5.61.2 Description

The lines between the WHILE and the ENDW are assembled as long as <expr> evaluates to TRUE. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. A relational TRUE expression is guaranteed to return a non-zero value; FALSE a value of zero. A WHILE loop can contain at most 100 lines and be repeated a maximum of 256 times.

5.61.3 Example

```
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 See Also

ENDW IF

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 6. Using MPASM to Create Relocatable Objects

6.1 Introduction

Since the introduction of MPASM v2.00 and MPLINK v1.00, users have had the ability to generate and link precompiled object modules. Writing source code that will be assembled to an object module is slightly different from generating executable code directly to a hex file. MPASM routines designed for absolute address assembly will require minor modifications to compile correctly into relocatable object modules.

6.2 Highlights

Topics covered in this chapter:

- Header Files
- Program Memory
- Instruction Operands
- RAM Allocation
- Configuration Bits and ID Locations
- Accessing Labels from Other Modules
- Paging and Banking Issues
- Unavailable Directives
- Generating the Object Module
- Code Examples

6.3 Header Files

The Microchip supplied standard header files (e.g., p17c756.inc) should be used when generating object modules. These header files define the special function registers for the target processor.

6.4 Program Memory

Program memory code must be preceded by a CODE section declaration.

6.4.1 Absolute Code:

```
Start  CLRW
      OPTION
      :
```

6.4.2 Relocatable Code:

```
CODE
Start  CLRW
      OPTION
      :
```

If more than one CODE section is defined in a source file, each section must have a unique name. If the name is not specified, it will be given the default name .code.

Each program memory section must be contiguous within a single source file. A section may not be broken into pieces within a single source file.

The physical address of the code can be fixed by supplying the optional address parameter of the CODE directive. Situations where this might be necessary are:

- Specifying interrupt vectors
- Ensuring that a code segment does not overlap page boundaries

6.4.3 Example Relocatable Code:

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

```
Main  CODE
      CLRW
      OPTION
```

Using MPASM to Create Relocatable Objects

6.5 Instruction Operands

There are some restrictions involving instruction operands. Instruction operands must be of the form:

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

where:

- <relocatable symbol> is any label that defines a program or data memory address
- <constant offset> is an expression that is resolvable at assembly time to a value between -32768 and 32767

Either <relocatable symbol> or <constant offset> may be omitted.

Operands of the form:

<relocatable symbol> - <relocatable symbol>

will be reduced to a constant value if both symbols are defined in the same code or data section.

If HIGH is used, only bits 8 through 15 of the expression will be used. If LOW is used, only bits 0 through 7 of the expression will be used. If UPPER is used, only bits 16 through 21 of the expression will be used.

6.6 RAM Allocation

RAM space must be allocated in a data section. Five types of data sections are available:

- UDATA – Uninitialized data. This is the most common type of data section. Locations reserved in this section are not initialized and can be accessed only by the labels defined in this section or by indirect accesses.
- UDATA_ACS – Uninitialized access data. This data section is used for variables that will be placed in access RAM of PIC18CXX devices. Access RAM is used as quick data access for specified instructions.
- UDATA_OVR – Uninitialized overlaid data. This data section is used for variables that can be declared at the same address as other variables in the same module or in other linked modules. A typical use of this section is for temporary variables.
- UDATA_SHR – Uninitialized shared data. This data section is used for variables that will be placed in RAM that is unbanked or shared across all banks.
- IDATA – Initialized data. The linker will generate a lookup table that can be used to initialize the variables in this section to the specified values. The locations reserved by this section can be accessed only by the labels defined in this section or by indirect accesses.

The following example shows how a data declaration might be created.

MPASM User's Guide with MPLINK and MPLIB

6.6.1 Absolute Code:

```
CBLOCK    0x20
    InputGain, OutputGain    ;Control loop gains
    HistoryVector            ;Must be initialized to 0
    Temp1, Temp2, Temp3      ;Used for internal calculations
ENDC
```

6.6.2 Relocatable Code:

```
                                IDATA
HistoryVector    DB    0

                                UDATA
InputGain        RES    1
OutputGain       RES    1

                                UDATA_OVR
Temp1            RES    1
Temp2            RES    1
Temp3            RES    1
```

If necessary, the location of the section may be fixed in memory by supplying the optional address parameter. If more than one of each section type is specified, each section must have a unique name. If a name is not provided, the default section names are: `.idata`, `.udata`, `.udata_acs`, `.udata_shr`, and `.udata_ovr`.

When defining initialized data in an IDATA section, the directives `DB`, `DW`, and `DATA` can be used. `DB` will define successive bytes of data memory. `DW` and `DATA` will define successive words of data memory in low-byte/high-byte order. The following example shows how data will be initialized.

6.6.3 Relocatable Code:

```
                                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 Configuration Bits and ID Locations

Configuration bits and ID locations can still be defined in a relocatable object using the `_ _CONFIG` and `_ _IDLOCS` directives. Only one linked module can specify these directives. They should be used prior to declaring any `CODE` sections. After using these directives, the current section is undefined.

Using MPASM to Create Relocatable Objects

6.8 Accessing Labels From Other Modules

Labels that are defined in one module for use in other modules must be exported using the GLOBAL directive. Labels must be defined before they are declared GLOBAL. Modules that use these labels must use the EXTERN directive to declare the existence of these labels. An example of using the GLOBAL and EXTERN directives is shown below.

6.8.1 Relocatable Code, Defining Module:

```

                                UDATA
InputGain    RES 1
OutputGain   RES 1
                                GLOBAL InputGain, OutputGain

                                CODE
Filter
                                GLOBAL Filter
                                :          ; Filter code

```

6.8.2 Relocatable Code, Referencing Module:

```

                                EXTERN InputGain, OutputGain, Filter

                                UDATA
Reading      RES 1

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

```

6.9 Paging and Banking Issues

In many cases, RAM allocation will span multiple banks, and executable code will span multiple pages. In these cases, it is necessary to perform proper bank and page set-up to properly access the labels. However, since the absolute addresses of these variable and address labels are not known at assembly time, it is not always possible to place the proper code in the source file. For these situations, two new directives, BANKSEL and PAGESEL have been added. These directives instruct the linker to generate the correct bank or page selecting code for a specified label. An example of how code should be converted is shown below.

MPASM User's Guide with MPLINK and MPLIB

6.9.1 Absolute Code:

```
LIST P=12C509
#include "P12C509.INC"

Var1 EQU H'10'
Var2 EQU H'30'
...
MOVLW InitialValue
BCF FSR, 5
MOVWF Var1
BSF FSR, 5
MOVWF Var2
BSF STATUS, PA0
CALL Subroutine
...
Subroutine CLRW ;In Page 1
...
RETLW 0
```

6.9.2 Relocatable Code:

```
LIST P=12C509
#include "P12C509.INC"

UDATA
Var1 RES 1
Var2 RES 1
...
CODE
MOVLW InitialValue
BANKSEL Var1
MOVWF Var1
BANKSEL Var2
MOVWF Var2
PAGESEL Subroutine
CALL Subroutine
...
Subroutine CLRW
...
RETLW 0
```

Using MPASM to Create Relocatable Objects

6.10 Unavailable Directives

Macro capability and nearly all directives are available when generating an object file. The only directive that is not allowed is the ORG directive. This can be replaced by specifying an absolute CODE segment, as shown below.

6.10.1 Absolute Code:

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

6.10.2 Relocatable Code:

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

6.11 Generating the Object Module

Once the code conversion is complete, the object module is generated by requesting an object file on the command line or in the shell interface. When using MPASM for Windows, check the checkbox labeled "Object File." When using the DOS command line interface, specify the /o option. When using the DOS shell interface, toggle "Assemble to Object File" to "Yes." The output file will have a .o extension.

6.12 Code Examples

The following is extracted from the example multiply routines given as a sample with MPASM. Most of the comments have been stripped for brevity.

6.12.1 Absolute Code:

```
LIST P=16C54
#include "P16C5x.INC"

cblock H'020'
mulcnd RES 1 ; 8 bit multiplicand
mulplr RES 1 ; 8 bit multiplier
H_byte RES 1 ; High byte of the 16 bit result
L_byte RES 1 ; Low byte of the 16 bit result
count RES 1 ; loop counter

mpy      clrf H_byte
         clrf L_byte
         movlw 8
         movwf count
         movf mulcnd,w
         bcf STATUS,C ;Clear carry bit
```

MPASM User's Guide with MPLINK and MPLIB

```
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
;*****
;          Test Program
;*****
start   clrw
        option
main    movf PORTB,W
        movwf mulplr ; multiplier (in mulplr) = 05
        movf PORTB,W
        movwf mulcnd

call_m  call mpy      ; The result is in F12 & F13
                        ; H_byte & L_byte

        goto main

        ORG 01FFh
        goto start

END
```

Since a eight-by-eight bit multiply is a useful, generic routine, it would be handy to break this off into a separate object file that can be linked in when required. The above file can be broken into two files, a calling file representing an application and a generic routine that could be incorporated in a library.

6.12.2 Relocatable Code, Calling File

```
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
```

Using MPASM to Create Relocatable Objects

```

        movf  PORTB, W
        movwf mulcnd

call_m  call  mpy      ; The result is in H_byte & L_byte
        goto main

Reset   CODE  H'01FF'
        goto start

END

```

6.12.3 Relocatable Code, Library Routine:

```

        LIST  P=16C54
        #INCLUDE "P16C5x.INC"

        UDATA
mulcnd  RES 1 ; 8 bit multiplicand
mulplr  RES 1 ; 8 bit multiplier
H_byte  RES 1 ; High byte of the 16 bit result
L_byte  RES 1 ; Low byte of the 16 bit result
count   RES 1 ; loop counter
        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 ; Clear carry bit
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

```

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 7. Macro Language

7.1 Introduction

Macros are user defined sets of instructions and directives that will be evaluated in-line with the assembler source code whenever the macro is invoked.

Macros consist of sequences of assembler instructions and directives. They can be written to accept arguments, making them quite flexible. Their advantages are:

- Higher levels of abstraction, improving readability and reliability.
- Consistent solutions to frequently performed functions.
- Simplified changes.
- Improved testability.

Applications might include creating complex tables, frequently used code, and complex operations.

7.2 Highlights

Topics covered in this chapter:

- Macro Syntax
- Macro Directives
- Text Substitution
- Macro Usage
- Code Examples

7.3 Macro Syntax

MPASM macros are defined according to the following syntax:

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

where <label> is a valid MPASM label and <arg> is any number of optional arguments supplied to the macro. The values assigned to these arguments at the time the macro is invoked will be substituted wherever the argument name occurs in the body of the macro.

MPASM User's Guide with MPLINK and MPLIB

The body of a macro may be comprised of MPASM directives, PICmicro MCU assembly instructions, or MPASM Macro Directives (LOCAL for example). Refer to Chapter 5. MPASM continues to process the body of the macro until an EXITM or ENDM directive is encountered.

Note: Forward references to macros are not permitted.

7.4 Macro Directives

There are directives that are unique to macro definitions. They cannot be used out of the macro context (refer to Chapter 7 for details concerning these directives):

- MACRO
- LOCAL
- EXITM
- ENDM

When writing macros, you can use any of these directives PLUS any other directives supported by MPASM.

Note: The previous syntax of the "dot" format for macro specific directives is no longer supported. For compatibility reasons, old ASM17 code that use this format will assemble by MPASM, but as mentioned before, you are encouraged to write new code based on the constructs defined within this document to ensure upward compatibility with MPASM.

7.5 Text Substitution

String replacement and expression evaluation may appear within the body of a macro.

Command	Description
<arg>	Substitute the argument text supplied as part of the macro invocation.
#v(<expr>)	Return the integer value of <expr>. Typically, used to create unique variable names with common prefixes or suffixes. Cannot be used in conditional assembly directives (e.g. IFDEF, WHILE).

Arguments may be used anywhere within the body of the macro, except as part of normal expression. For example, the following macro:

Macro Language

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

when invoked, would generate:

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

7.6 Macro Usage

Once the macro has been defined, it can be invoked at any point within the source module by using a macro call, as described below:

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

where `<macro_name>` is the name of a previously defined macro and arguments are supplied as required.

The macro call itself will not occupy any locations in memory. However, the macro expansion will begin at the current memory location. Commas may be used to reserve an argument position. In this case, the argument will be an empty string. The argument list is terminated by white space or a semicolon.

The EXITM directive (see Chapter 5) provides an alternate method for terminating a macro expansion. During a macro expansion, this directive causes expansion of the current macro to stop and all code between the EXITM and the ENDM directives for this macro to be ignored. If macros are nested, EXITM causes code generation to return to the previous level of macro expansion.

7.7 Code Examples

7.7.1 Eight-by-Eight Multiply

```
    subtitle "macro definitions"
    page
;
; multiply - eight by eight multiply macro, executing
; in program memory. optimized for speed, straight
; line code.
;
; written for the PIC17C42.
;
multiply macro arg1, arg2, dest_hi, dest_lo
    local i = 0 ; establish local index variable
                ; and initialize it.
                ;
    movlw arg1 ; setup multiplier
    movwf mulplr ;
                ;
    movlw arg2 ; setup multiplicand in w reg
                ;
    clrf dest_hi ; clear the destination regs
    clrf dest_lo ;
                ;
    bcf _carry ; clear carry for test
                ;
    while i < 8 ; do all eight bits
    addwf dest_hi ; then add multiplicand
    rrcf dest_hi ; shift right through carry
    rrcf dest_lo ; shift right again, snag carry
                ; if set by previous rotate
i += 1 ; increment loop counter
    endw ; break after eight iterations
    endm ; end of macro.
```

The macro declares all of the required arguments. In this case, there are four. The LOCAL directive then establishes a local variable "i" that will be used as an index counter. It is initialized to zero.

A number of assembler instructions are then included. When the macro is executed, these instructions will be written in line with the rest of the assembler source code.

The macro writes the multiplication code using an algorithm that uses right shifts and adds for each bit set in the eight bits of the multiplier. The WHILE directive is used for this function, continuing the loop until "i" is greater than or equal to eight.

Macro Language

The end of the loop is noted by the ENDW directive. Execution continues with the statement immediately following the ENDW when the WHILE condition becomes FALSE. The entire macro is terminated by the ENDM directive.

7.7.2 Constant Compare

As another example, if the following macro were written:

```
include "16cxx.reg"
;
; compare file to constant and jump if file
; >= constant.
;
cfl_jge macro file, con, jump_to
    movlw con & 0xff
    subwf file, w
    btfsc status, carry
    goto jump_to
endm
```

and invoked by:

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

it would produce:

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

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 8. Expression Syntax and Operation

8.1 Introduction

This chapter describes various expression formats, syntax, and operations used by MPASM.

8.2 Highlights

Topics covered in this chapter:

- Text Strings
- Numeric Constants and Radix
- Arithmetic Operators and Precedence
- High/Low/Upper and Increment/Decrement Operators

8.3 Text Strings

A "string" is a sequence of any valid ASCII character (of the decimal range of 0 to 127) enclosed by double quotes.

Strings may be of any length that will fit within a 255 column source line. If a matching quote mark is found, the string ends. If none is found before the end of the line, the string will end at the end of the line. While there is no direct provision for continuation onto a second line, it is generally no problem to use a second DW directive for the next line.

The DW directive will store the entire string into successive words. If a string has an odd number of characters (bytes), the DW and DATA directives will pad the end of the string with one byte of zero (00).

If a string is used as a literal operand, it must be exactly one character long, or an error will occur.

MPASM User's Guide with MPLINK and MPLIB

See the examples below for the object code generated by different statements involving strings.

```
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
```

The assembler accepts the ANSI 'C' escape sequences to represent certain special control characters:

Table 8.1: ANSI 'C' Escape Sequences

Escape Character	Description	Hex Value
\a	Bell (alert) character	07
\b	Backspace character	08
\f	Form feed character	0C
\n	New line character	0A
\r	Carriage return character	0D
\t	Horizontal tab character	09
\v	Vertical tab character	0B
\\	Backslash	5C
\?	Question mark character	3F
\'	Single quote (apostrophe)	27
\"	Double quote character	22
\000	Octal number (zero, Octal digit, Octal digit)	
\xHH	Hexadecimal number	

Expression Syntax and Operation

8.4 Numeric Constants and Radix

MPASM supports the following radix forms: hexadecimal, decimal, octal, binary, and ASCII. The default radix is hexadecimal; the default radix determines what value will be assigned to constants in the object file when a radix is not explicitly specified by a base descriptor.

Constants can be optionally preceded by a plus or minus sign. If unsigned, the value is assumed to be positive.

Note: Intermediate values in constant expressions are treated as 32-bit unsigned integers. Whenever an attempt is made to place a constant in a field for which it is too large, a truncation warning will be issued.

The following table presents the various radix specifications:

Table 8.2: Radix Specifications

Type	Syntax	Example
Decimal	D'<digits>'	D'100'
Hexadecimal	H'<hex_digits>' 0x<hex_digits>	H'9f' 0x9f
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001'
ASCII	'<character>' A'<character>'	'C' A'C'

Table 8.3: Arithmetic Operators and Precedence

Operator		Example
\$	Current/Return program counter	goto \$ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags
high	Return high byte	movlw high CTR_Table
low	Return low byte	movlw low CTR_Table
upper	Return upper byte	movlw upper CTR_Table

MPASM User's Guide with MPLINK and MPLIB

Table 8.3: Arithmetic Operators and Precedence

Operator		Example
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
	Bitwise inclusive OR	flags = flags ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
	Logical OR	if (len == 512) (b == c)
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
=	Inclusive OR, set equal	flags = ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment	i ++
--	Decrement	i --

Expression Syntax and Operation

8.5 High/Low/Upper

8.5.1 Syntax

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

8.5.2 Description

These operators are used to return one byte of a multi-byte label value. This is done to handle dynamic pointer calculations as might be used with table read and write instructions.

8.5.3 Example

```
movlw low size ; handle the lsb's
movpf wreg, low size_lo
movlw high size ; handle the msb's
movpf wreg, high size_hi
```

8.6 Increment/Decrement (++/--)

8.6.1 Syntax

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

8.6.2 Description

Increments or decrements a variable value. These operators can only be used on a line by themselves; they cannot be embedded within other expression evaluation.

8.6.3 Example

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

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 9. Example Initialization Code

9.1 Introduction

This chapter discusses initialization code for various PICmicro devices.

9.2 Highlights

Topic covered in this chapter:

- Initialization Code Examples

9.3 Initialization Code Examples

If the IDATA directive is used when generating an object module, the user must provide code to perform the data initialization. Examples of initialization code that may be used and modified as needed may be found with MPLINK sample application examples on our web site (<http://www.microchip.com>).

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Part 2 – MPLINK

Chapter 1. MPLINK Preview	99
Chapter 2. MPLINK – Installation and Getting Started	103
Chapter 3. Using MPLINK with DOS	109
Chapter 4. Using MPLINK with Windows and MPLAB	111
Chapter 5. MPLINK Linker Scripts	119
Chapter 6. Linker Processing	127
Chapter 7. Sample Application 1	131
Chapter 8. Sample Application 2	137
Chapter 9. Sample Application 3	145
Chapter 10. Sample Application 4	151

MPASM User's Guide with MPLINK and MPLIB

Chapter 1. MPLINK Preview

1.1 Introduction

This chapter will define what MPLINK is and how it can help you develop firmware for PICmicro devices.

1.2 Highlights

Topics covered in this chapter:

- What MPLINK Is
- What MPLINK Does
- How MPLINK Helps You
- MPLINK Examples
- Tool Versions and Platforms Supported

1.3 What MPLINK Is

MPLINK is a linker for the Microchip relocatable assembler, MPASM, and the Microchip C compiler, MPLAB-C17/C18. For more information on MPASM, see Part 1 of this document. For more information on MPLAB-C17/C18, see the *MPLAB-C17/C18 Compiler User's Guide* (DS51112).

MPLINK also may be used with the Microchip librarian, MPLIB. See Part 3 of this document for more information on MPLIB.

MPLINK is designed to be used with MPLAB, though it does not have to be. However, once an application is built, it is convenient to use MPLAB to test and debug the code using the simulator or an emulator, and to use a device programmer to program a part. For more information on MPLAB, see the *MPLAB User's Guide* (DS51025).

1.4 What MPLINK Does

MPLINK performs many functions:

- **Locates Code and Data.** Takes as input relocatable source files. Using the linker script, it decides where the code will be placed in program memory and where variables will be placed in RAM.
- **Resolves Addresses.** External references in a source file generate relocation entries in the object file. After MPLINK locates code and data, it uses this relocation information to update all external references with the actual addresses.

MPASM User's Guide with MPLINK and MPLIB

- **Generates an Executable.** Produces a .HEX file that can be programmed into a PICmicro or loaded into an emulator or simulator to be executed.
- **Configures Stack Size and Location.** Allows MPLAB-C17/C18 to set aside RAM space for dynamic stack usage.
- **Identifies Address Conflicts.** Checks to ensure that program/data do not get assigned to space that has already been assigned or reserved.
- **Provides Symbolic Debug Information.** Outputs a file that MPLAB uses to track address labels, variable locations, and line/file information for source level debugging.

1.5 How MPLINK Helps You

MPLINK allows you to produce modular, reusable code. Control over the linking process is accomplished through a linker “script” file and with command line options. MPLINK ensures that all symbolic references are resolved and that code and data fit into the available PICmicro device.

MPLINK can help you with:

- **Reusable Source Code.** You can build up your application in small, reusable modules.
- **Libraries.** You can make libraries of related functions which can be used to make efficient, readily compilable applications.
- **MPLAB-C17/C18.** The Microchip compiler for PIC17CXX and PIC18CXX devices requires the use of MPLINK and can be used with pre-compiled libraries and MPASM.
- **Centralized Memory Allocation.** By using application-specific linker scripts, precompiled objects and libraries can be combined with new source modules and placed efficiently into available memory at link time rather than at compile/assemble time.
- **Accelerated Development.** Since tested modules and libraries don't have to be recompiled each time a change is made in your code, compilation time may be reduced.

1.6 MPLINK Examples

You can learn how to use MPLINK from the four sample applications at the end of this part. These sample applications can be used as templates for your own application.

- Sample Application 1
 - How to place program code in different memory regions
 - How to place data tables in ROM memory
 - How to set configuration bits in C

- Sample Application 2
 - How to partition memory for a boot loader
 - How to compile code that will be loaded into external RAM and executed
- Sample Application 3
 - How to access peripherals that are memory mapped
 - How to create new sections
- Sample Application 4
 - How to manually partition RAM space for program usage

1.7 Platforms Supported

MPLINK is distributed in two executable formats: a Windows 32 console application suitable for Windows 95/98 and Windows NT platforms and a DOS-extended DPML application suitable for Windows 3.x and DOS platforms. Executables which are DOS-extended applications have names which end with 'd' to distinguish them from the Windows 32 versions. DOS-extended versions require the DOS extender program 'DOS4GW.EXE' which is also distributed with MPLINK.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 2. MPLINK – Installation and Getting Started

2.1 Introduction

This chapter will tell you how to install MPLINK on your system and give you an overview of how the linker (MPLINK) works.

2.2 Highlights

Topics covered in this chapter:

- Installation
- Overview of Linker
- Linker Input/Output Files

2.3 Installation

There are two versions of MPLINK. MPLINKD.EXE is a DOS extender version and is only recommended for DOS or Windows 3.x systems. MPLINK.EXE is a Windows 32-bit console application and is for Windows 95/98/NT. MPLINKD.EXE and MPLINK.EXE are command-line applications.

MPLINKD.EXE may be used with DOS or a DOS window in Windows 3.x.

MPLINK.EXE may be used with a DOS window in Windows 95/98/NT. You can use it with MPLAB (recommended).

If you are going to use MPLAB with MPLINK, you do not need to install the linker and supporting files separately. When MPLAB is installed, MPLINK is also installed. To find out how to install MPLAB, please refer to the *MPLAB User's Guide* (DS51025). You may obtain the MPLAB software and user's guide either from the Microchip Technical Library CD or from our web site.

If you are not going to use MPLAB with MPLINK, you can obtain the linker and supporting files separately either from the Microchip Technical Library CD or from our web site. MPLINK is bundled with MPLIB into a zip file. To install:

- Create a directory in which to place the files
- Unzip the MPLINK and MPLIB files using either WinZip or PKZIP

MPASM User's Guide with MPLINK and MPLIB

2.4 Overview of Linker

Figure 2.1 is a diagram of how MPLINK works with other Microchip tools.

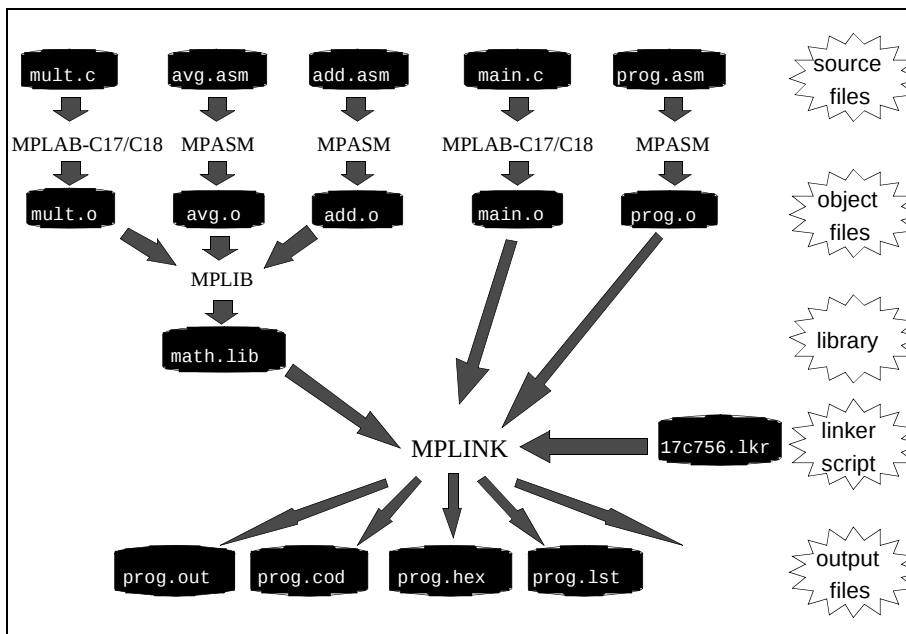


Figure 2.1: Microchip Tool Overview

MPLINK combines multiple input object modules, generated by MPASM or MPLAB-C17/C18, into a single output executable (hex) file. A linker script tells the linker how to combine these modules.

MPLINK is executed after assembling or compiling relocatable object modules with MPASM and/or MPLAB-C17/C18. The actual addresses of data and the location of functions will be assigned when MPLINK is executed. This means that you will instruct MPLINK, via a linker script, to place code and data somewhere within named regions of memory. These named regions may be specific physical locations or not.

The linker script must also tell MPLINK about the ROM and RAM memory regions available in the target PICmicro device. Then, it can analyze all the input files and try to fit the application's routines into ROM and assign its data variables into available RAM. If there is too much code or too many variables to fit, MPLINK will give an error message.

MPLINK also provides flexibility for specifying that certain blocks of data memory are reusable, so that different routines (which never call each other and which don't depend upon this data to be retained between execution) can share limited RAM space.

MPLINK – Installation and Getting Started

Libraries are available for most PICmicro peripheral functions as well as for many standard C functions (See Chapter 2 for more information on MPLIB). The linker will only extract and link individual object files that are needed for the current application from the included libraries. This means that relatively large libraries can be used in a highly efficient manner.

MPLINK combines all input files to generate the executable output and ensures that all addresses are resolved. Any function in the various input modules that attempts to access data or call a routine that has not been allocated or created will cause MPLINK to generate an error.

MPLINK also generates symbolic information for debugging your application with MPLAB (.cod, .lst, and .map files).

2.5 Linker Input/Output Files

MPLINK is a linker which combines multiple object files into one executable hex file. File types used as linker inputs are:

- **Object files (.o).** Relocatable code produced from source files.
- **Library files (.lib).** A collection of object files grouped together for convenience.
- **Linker script files (.lkr).** Description of memory layout for a particular processor / project.

Note: Source files (C or assembly code) are used when building the listing file.

File types used as linker outputs are:

- **COFF file (.out, .cof).** Intermediate file used by MPLINK to generate Code file, HEX file, and Listing file.
- **Code file (.cod).** Debug file used by MPLAB.
- **HEX file (.hex).** Binary executable with no debug information.
- **Listing file (.lst).** Original source code, side-by-side with final binary code.
- **Map file (.map).** Shows the memory layout after linking. Indicates used and unused memory regions.

2.5.1 Object File (.O)

Object files are the relocatable code produced from source files. It is the linker's job to assemble source files and library files, according to a linker script, into an object file.

MPASM User's Guide with MPLINK and MPLIB

2.5.2 Library File (.LIB)

A library file may be created from object files by MPLIB or may be an existing standard library. For more information on library files, see Part 3.

2.5.3 Linker Script File (.LKR)

Linker script files are the command files of MPLINK. For more information on linker scripts, see Chapter 5.

2.5.4 COFF Object Module File (.OUT, .COF)

MPLINK generates a COFF file which serves as an intermediate file. MP2COD.EXE generates the COD files and List files from this, and MP2HEX.EXE generates the HEX file.

2.5.5 Symbol and Debug File (.COD)

MPLINK produces a COD file for use in MPLAB debugging of code.

2.5.6 Hex File (.HEX)

MPLINK is capable of producing different HEX file formats. See Appendix A for a detailed description of these formats.

2.5.7 Absolute Listing File (.LST)

The absolute listing file generated by MPLINK can be viewed by selecting Window>Absolute Listing. It provides a mapping of source code to machine instructions. This window is automatically reloaded each time the project is rebuilt. For more information on the format of this file, see Part 1, Section 2.5.2.

2.5.8 Map File (.MAP)

The map file generated by MPLINK can be viewed by selecting File>Open and choosing the file you specified in the MPLINK options. It provides information on the absolute location of source code symbols in the final output. It also provides information on memory use, indicating used/unused memory. This window is automatically reloaded after each rebuild.

The map file contains three tables. The first table (Section Info) displays information about each section. The information includes the name of the section, its type, beginning address, whether the section resides in program or data memory, and its size in bytes.

MPLINK – Installation and Getting Started

There are four types of sections:

- code
- initialized data (idata)
- uninitialized data (udata)
- initialized ROM data (romdata)

The following table is an example of the section table in a map file:

Section	Type	Section Info		
		Address	Location	Size(Bytes)
Reset	code	0x000000	program	0x000002
.cinit	romdata	0x000021	program	0x000004
.code	code	0x000023	program	0x000026
.udata	udata	0x000020	data	0x000005

The second table in the map file (Symbols – Sorted by Name) provides information about the symbols in the output module. The table is sorted by the symbol name and includes the address of the symbol, whether the symbol resides in program or data memory, whether the symbol has external or static linkage, and the name of the file where defined. The following table is an example of the symbol table sorted by symbol name in a map file:

Symbols - Sorted by Name				
Name	Address	Location	Storage	File
call_m	0x000026	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
loop	0x00002e	program	static	C:\MPASMV2\MUL8X8.ASM
main	0x000024	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
mpy	0x000028	program	extern	C:\MPASMV2\MUL8X8.ASM
start	0x000023	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
H_byte	0x000022	data	extern	C:\MPASMV2\MUL8X8.ASM
L_byte	0x000023	data	extern	C:\MPASMV2\MUL8X8.ASM
count	0x000024	data	static	C:\MPASMV2\MUL8X8.ASM
mulcnd	0x000020	data	extern	C:\MPASMV2\MUL8X8.ASM
mulplr	0x000021	data	extern	C:\MPASMV2\MUL8X8.ASM

MPASM User's Guide with MPLINK and MPLIB

The third table in the map file (Symbols – Sorted by Address) provides the same information that the second table provides, but it is sorted by symbol address rather than symbol name. The following is an example of the symbol table sorted by address in a map file:

Symbols - Sorted by Address				
Name	Address	Location	Storage	File
-----	-----	-----	-----	-----
start	0x000023	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
main	0x000024	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
call_m	0x000026	program	static	C:\PROGRA~1\MPLAB\ASMF00\sampobj.asm
mpy	0x000028	program	extern	C:\MPASMV2\MUL8X8.ASM
loop	0x00002e	program	static	C:\MPASMV2\MUL8X8.ASM
mulcnd	0x000020	data	extern	C:\MPASMV2\MUL8X8.ASM
mulplr	0x000021	data	extern	C:\MPASMV2\MUL8X8.ASM
H_byte	0x000022	data	extern	C:\MPASMV2\MUL8X8.ASM
L_byte	0x000023	data	extern	C:\MPASMV2\MUL8X8.ASM
count	0x000024	data	static	C:\MPASMV2\MUL8X8.ASM

If a linker error is generated, a complete map file can not be created. However, if the `-m` option was supplied, an error map file will be created. The error map file contains only section information --no symbol information is provided. The error map file in conjunction with the error message should provide enough context to determine why a section could not be allocated.

Chapter 3. Using MPLINK with DOS

3.1 Introduction

This chapter will describe how to use MPLINK on a DOS command line.

3.2 Highlights

Topic covered in this chapter:

- Linker Command Line Options

3.3 Linker Command Line Options

When used in MPLAB, all of MPLINK's options are available through the Node Properties dialog, accessed from the Edit Project dialog. See the next chapter for more on using MPLINK with MPLAB.

When using MPLINK in a batch file, or directly from DOS, MPLINK is invoked with the following syntax:

```
mplink { cmdfile | [objfile] | [libfile] | [option] } . . .
```

You must have one command file (cmdfile) and at least one object file (objfile).

'cmdfile' is the name of a linker command file. All linker command files must have the extension '.lkr'.

'objfile' is the name of an assembler or compiler generated object file. All object files must have the extension '.o'.

'libfile' is the name of a librarian created library file. All library files must have the extension '.lib'.

'option' is a linker command line option described in Table 3.1.

Table 3.1: MPLINK Command Line Options

Option	Description
/o filename	Specify output file 'filename'. Default is a.out.
/m filename	Create map file 'filename'.
/l pathlist	Add directories to library search path.
/k pathlist	Add directories to linker script search path.
/n length	Specify number of lines per listing page.
/h, /?	Display help screen.
/a hexformat	Specify format of hex output file.
/q	Quiet mode.
/d	Don't create an absolute listing file.

MPASM User's Guide with MPLINK and MPLIB

There is no required order for the command line arguments; however, changing the order can affect the operation of the linker. Specifically, additions to the library/object directory search path are appended to the end of the current library/object directory search path as they are encountered on the command line and in command files.

Library and object files are searched for in the order in which directories occur in the library/object directory search path. Therefore, changing the order of directories may change which file is selected.

The /o option is used to supply the name of the generated output COFF file. The linker also generates a COD file for MPLAB debugging, and an Intel[®] format HEX programming. Both of these files have the same name as the output COFF file but with the file extensions '.cod' and '.hex' respectively. If the /o option is not supplied, the default output COFF file is named 'a.out' and the corresponding COD and HEX files are named 'a.cod' and 'a.hex'.

An example of an MPLINK command line is shown in Example 3.1.

Example 3.1:

<code>MPLINK 17C756.LKR MAIN.O FUNCT.O ADC17.LIB /m MAIN.MAP /o MAIN.OUT</code>

This instructs MPLINK to use the 17C756.LKR linker script file to link the input modules MAIN.O, FUNCT.O, and the precompiled library ADC17.LIB. It also instructs MPLINK to produce a map file named MAIN.MAP. MAIN.O and FUNCT.O must have been previously compiled or assembled with MPLAB-C17/C18 or MPASM. The output files MAIN.HEX and MAIN.COD file will be produced if no errors occur during the link process.

Chapter 4. Using MPLINK with Windows and MPLAB

4.1 Introduction

This chapter will describe how to use MPLINK with MPLAB projects.

4.2 Highlights

Topics contained in this chapter:

- MPLAB Projects and MPLINK
- Setting Up MPLAB to Use MPLINK
- Generating Output Files
- MPLAB/MPLINK Troubleshooting

4.3 MPLAB Projects and MPLINK

MPLAB projects are composed of nodes (Figure 4.1). These represent files used by a generated project.

- Target Node – Final Output
 - HEX File
- Project Nodes – Components
 - C Source Files
 - Assembly Source Files
 - Library Files
 - Pre-Compiled Object Files
 - Linker Script Files

In this chapter, we are concerned with the relationship between MPLAB and MPLINK. For more general information about MPLAB projects, consult the *MPLAB User's Guide* (DS51025).

MPASM User's Guide with MPLINK and MPLIB

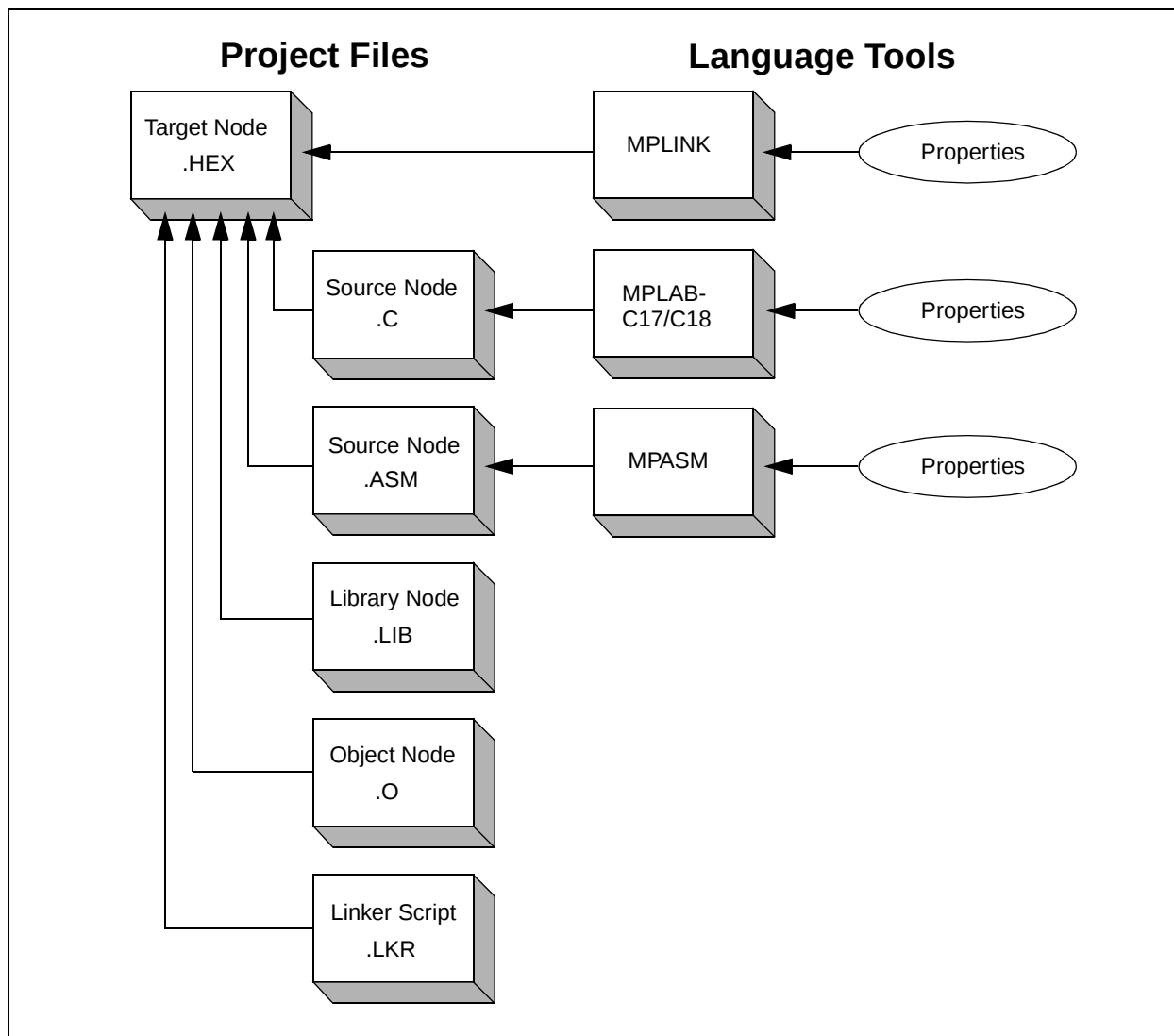


Figure 4.1: Project Relationships – MPLINK

Projects are used to apply language tools, such as assemblers, compilers and linkers, to source files in order to make executable (.HEX) files. This diagram shows the relationship between the final .HEX file and the components used to create it.

Using MPLINK with Windows and MPLAB

4.4 Setting Up MPLAB to use MPLINK

Follow these steps to set up MPLINK to use with MPLAB:

1. After creating your project (*Project>New Project*), the Edit Project dialog will appear. Select the name of the project (e.g., AD.HEX) in the Project Files list to activate the **Node Properties** button. Then click on **Node Properties**.



Figure 4.2: Edit Project Dialog

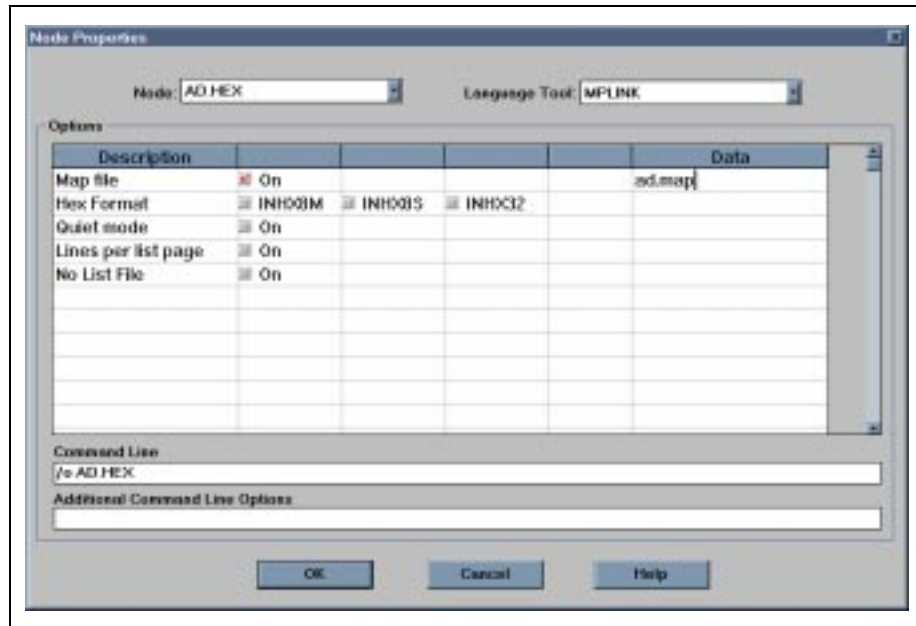


Figure 4.3: Node Properties Dialog

Using MPLINK with Windows and MPLAB

3. Add source files to your project. Use the **Add Node** button of the Edit Project dialog to bring up the Add Node dialog. Specify the tool that will be used for compiling each into an object file by selecting the source file name and clicking **Node Properties**.

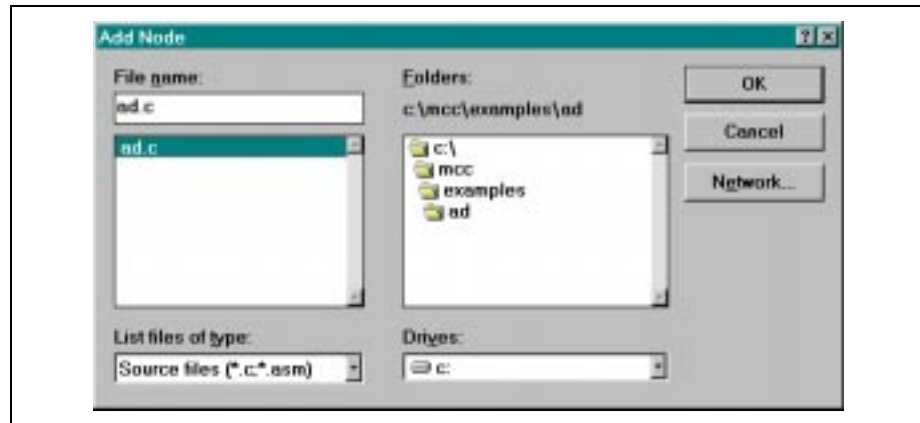


Figure 4.4: Add Node Dialog – Source Files

4. Add precompiled object files and libraries (such as those included with MPLAB-C17/C18). You can add more than one file by holding down the CTRL key while clicking on several files.

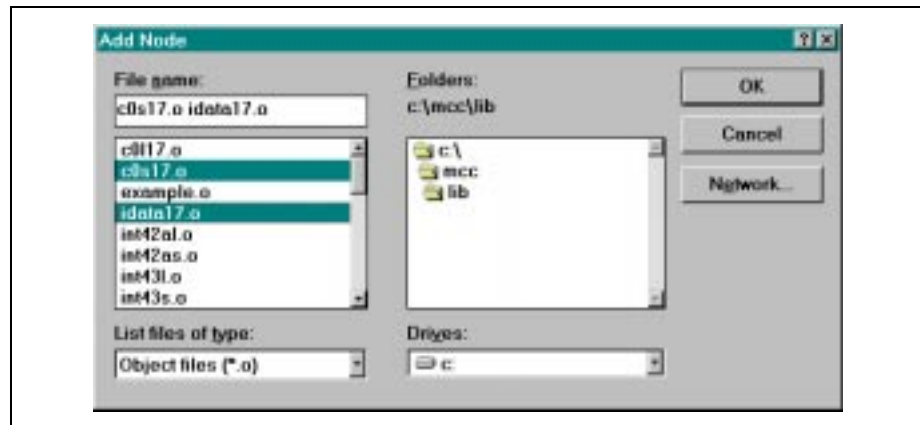


Figure 4.5: Add Node Dialog – Object Files

MPASM User's Guide with MPLINK and MPLIB

5. Add the linker script file as a node to your project.

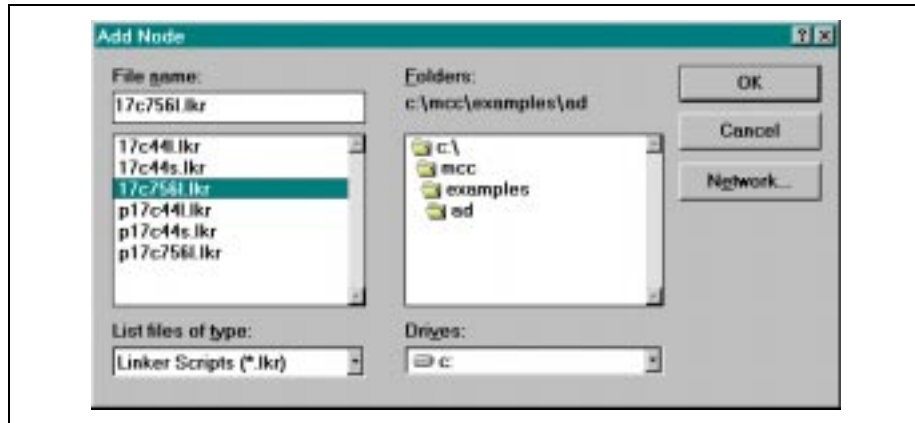


Figure 4.6: Add Node Dialog – Linker Script Files

6. Click **OK** on the Edit Project dialog when you are done.

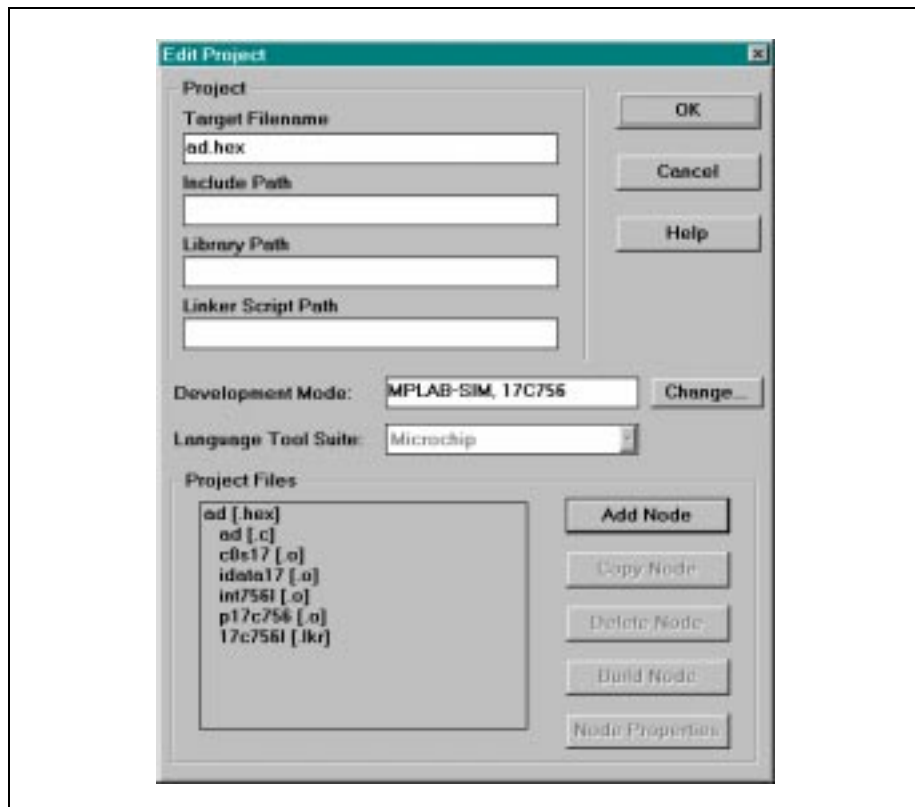


Figure 4.7: Edit Project Dialog – Nodes Added

Using MPLINK with Windows and MPLAB

4.5 Generating Output Files

Once the MPLAB project is set up, you must build it. Select *Project>Make Project* to do this. Hex files are automatically loaded into program memory.

In addition to the hex file, MPLINK generates other files to help you debug your code. See Section 2.5 for more information on these files and their specific functions.

4.6 MPLAB/MPLINK Troubleshooting

If have experienced problems, check the following:

Select *Project>Install Language Tool...* and check that MPLINK points to MPLINK.EXE in the MPLAB installation directory and that the Command-line option is selected.

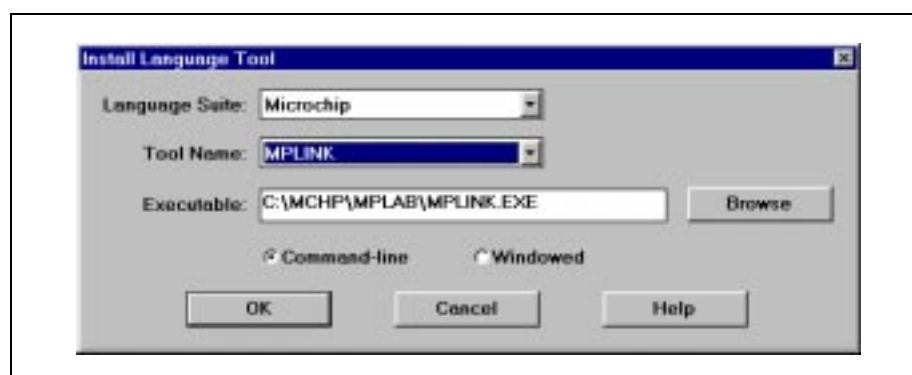


Figure 4.8: Install Language Tool Dialog – MPLINK

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 5. MPLINK Linker Scripts

5.1 Introduction

This chapter will describe how to make and modify linker scripts for use with MPLINK.

5.2 Highlights

Topics covered in this chapter:

- Linker Scripts Defined
- Command Line Information
- Memory Region Definitions
- Logical Section Definitions
- Stack Definitions
- Linker Script Caveats

5.3 Linker Scripts Defined

Linker script files are the command files of MPLINK. They specify:

- Program and data memory for the target part
- Stack size and location (for MPLAB-C17/C18)
- Logical sections used in source code to place code and data

Linker script directives form the command language that controls the linker's behavior. There are four basic categories of linker script directives. Each of these directives, plus some useful linker script caveats, are discussed in the following sections.

Linker script comments are specified by '//', i.e., any text between a '/' and the end of a line is ignored.

5.4 Command Line Information

The MPLAB Project Manager can set this information directly. You probably only need to use these if you are linking from the command line.

LIBPATH

Library and object files which do not have a path are searched using the library/object search path. The following directive appends additional search directories to the library/object search path:

```
LIBPATH 'libpath'
```

where, 'libpath' is a semicolon delimited list of directories.

MPASM User's Guide with MPLINK and MPLIB

Example:

To append the current directory and the directory 'C:\PROJECTS\INCLUDE' to the library/object search path, the following line should be added to the linker command file:

```
LIBPATH .;C:\PROJECTS\INCLUDE
```

LKRPATH

Linker command files that do not have a path are searched for using the linker command file search path. The following directive appends additional search directories to the linker command file search path:

```
LKRPATH 'lkrpath'
```

where, 'lkrpath' is a semicolon delimited list of directories.

Example:

To append the current directory's parent and the directory 'C:\PROJECTS\SCRIPTS' to the linker command file search path, the following line should be added to the linker command file:

```
LKRPATH ..;C:\PROJECTS\SCRIPTS
```

FILES

The following directive specifies object or library files for linking:

```
FILES 'objfile/libfile' ['objfile/libfile'...]
```

where, 'objfile/libfile' is either an object or library file. Note, more than one object or library file can be specified in a single FILES directive.

Example:

To specify that the object module 'main.o' be linked with the library file 'math.lib', the following line should be added to the linker command file:

```
FILES main.o math.lib
```

INCLUDE

The following directive includes an additional linker command file:

```
INCLUDE 'cmdfile'
```

where, 'cmdfile' is the name of the linker command file to include.

Example:

To include the linker command file named 'mylink.lkr', the following line should be added to the linker command file:

```
INCLUDE mylink.lkr
```

5.5 Memory Region Definitions

The linker script describes the memory architecture of the PICmicro. This allows the linker to place code in available ROM space and variables in available RAM space. Regions that are marked PROTECTED will not be used for general allocation of program or data. These regions are marked for special usage and you can only put code or data in these regions by specifically marking code and data in your source code with #pragma code/udata type directives in MPLAB-C17/C18 or with code/udata type directives in MPASM.

5.5.1 Defining ROM Memory Regions

The CODEPAGE directive is used for program code, initialized data values, constant data values and external memory for PIC17CXX devices. It has the following format:

```
CODEPAGE NAME=memName START=addr END=addr [PROTECTED] [FILL=fillvalue]
```

where:

'memName' is any ASCII string used to identify a CODEPAGE.

'addr' is a decimal or hexadecimal number specifying an address.

'fillValue' is a value which fills any unused portion of a memory block. If this value is in decimal notation, it is assumed to be a 16-bit quantity. If it is in hexadecimal notation (e.g., 0x2346), it may be any length divisible by full words (16 bits).

The optional keyword PROTECTED indicates a region of memory that only can be used by program code that specifically requests it.

5.5.2 Defining ROM Memory Regions – Example

Figure 5.1 shows the program memory layout for a PIC17C42A microcontroller. Based on this map, the CODEPAGE declarations are shown in Example 5.1.

MPASM User's Guide with MPLINK and MPLIB

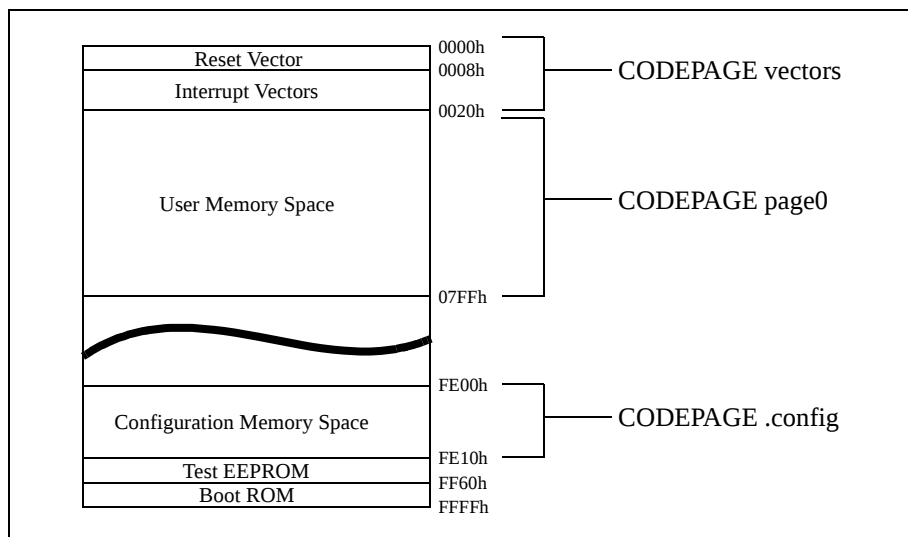


Figure 5.1: PIC17C42A Program Memory Map (Microcontroller Mode)

Example 5.1: ROM Memory Declarations for PIC17C42A

```
CODEPAGE NAME=vectors START=0x0    END=0x27    PROTECTED
CODEPAGE NAME=page0    START=0x28    END=0x7FF
CODEPAGE NAME=.config  START=0xFE00   END=0xFE0F PROTECTED
```

5.5.3 Defining RAM Memory Regions

The DATABANK, SHAREBANK and ACCESSBANK directives are used for variable data in internal RAM. The formats for these directives are as follows.

- Banked registers

DATABANK NAME=memName START=addr END=addr [PROTECTED]

- Unbanked registers

SHAREBANK NAME=memName START=addr END=addr [PROTECTED]

- Access registers (PIC18CXX only)

ACCESSBANK NAME=memName START=addr END=addr [PROTECTED]

where:

'memName' is any ASCII string used to identify an area in RAM.

'addr' is a decimal or hexadecimal number specifying an address.

The optional keyword PROTECTED indicates a region of memory that only can be assigned to variables that are specifically identified in the source code.

5.5.4 Defining RAM Memory Regions – Example

Based on the RAM memory layout shown in Figure 5.2, the DATABANK and SHAREBANK entries in the linker script file would appear as shown in examples Example 5.2 and Example 5.3.

Address	Bank 0	Bank 1	Bank 2	Bank 3
00h	INDF0			
01h	FSR0			
:	:			
0Eh	TBLPTRH			
0Fh	BSR			
10h	PORTA	DDRC	TMR1	PW1DCL
11h	DDRB	PORTC	TMR2	PW2DCL
:	:			
16h	TXREG	PIR	PR3LCA1L	TCON1
17h	SPBRG	PIE	PR3HCA1H	TCON2
18h	PRODL			
19h	PRODH			
1Ah	General Purpose RAM (Unbanked)			
:				
1Fh				
20h	General Purpose RAM	General Purpose RAM	Not Used	
:				
FFh				

Figure 5.2: PIC17C42A Register File Map

MPASM User's Guide with MPLINK and MPLIB

Example 5.2: RAM Memory Declarations for PIC17C42A - Banked Memory

```
DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED //Special Function Registers in Bank0
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED //Special Function Registers in Bank1
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED //Special Function Registers in Bank2
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED //Special Function Registers in Bank3

DATABANK NAME=gpr0 START=0x20 END=0xFF //General Purpose RAM in Bank0
DATABANK NAME=gpr1 START=0x120 END=0x1FF //General Purpose RAM in Bank1
```

Example 5.3: RAM Memory Declarations for PIC17C42A - Unbanked Memory

```
SHAREBANK NAME=sfrnobnk START=0x0 END=0xF PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED //INDF0 thru BSR - available in all banks
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED //INDF0 thru BSR - available in all banks

SHAREBANK NAME=sfrprod START=0x18 END=0x19 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED //PRODL, PRODH - available in all banks
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED //PRODL, PRODH - available in all banks

SHAREBANK NAME=gpr2 START=0x1A END=0x1F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x11A END=0x11F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x21A END=0x21F PROTECTED //1A thru 1F - available in all banks
SHAREBANK NAME=gpr2 START=0x31A END=0x31F PROTECTED //1A thru 1F - available in all banks
```

5.6 Logical Section Definitions

Logical sections are used to specify which of the defined memory regions should be used for a portion of source code. To use logical sections, define the section in the linker script file with the `SECTION` directive and then reference that name in the source file using that language's built-in mechanism (e.g., `#pragma section` for MPLAB-C17/C18).

The section directive defines a section by specifying its name, and either the block of program memory in ROM or the block of data memory in RAM which contains the section:

```
SECTION NAME='secName' { ROM='memName' | RAM='memName' }
```

where:

'secName' is an ASCII string used to identify a SECTION.

'memName' is a previously defined ACCESSBANK, SHAREBANK, DATABANK, or CODEPAGE.

The ROM attribute must always refer to program memory previously defined using a CODEPAGE directive. The RAM attribute must always refer to data memory previously defined with a ACCESSBANK, DATABANK or SHAREBANK directive.

Example:

To specify that a section whose name is 'filter_coeffs' be loaded into the previously defined logical block of program memory named 'constants', the following line should be added to the linker command file:

Example 5.4: Logical Section Definition

```
SECTION NAME=filter_coeffs ROM=constants
```

To place MPASM source code into the `filter_coeffs` section, use the following line prior to the desired source code:

Example 5.5: Logical Section Usage

```
filter_coeffs CODE
```

MPASM User's Guide with MPLINK and MPLIB

5.7 STACK Definition

MPLAB-C17/C18 requires a software stack be set up. The following statement specifies the stack size and an optional DATABANK where the stack is to be allocated:

```
STACK SIZE='allocSize' [RAM='memName']
```

where:

'allocSize' is the size in bytes of the stack and 'memName' is the name of a memory previously declared using a ACCESSBANK, DATABANK or SHAREBANK statement.

Example:

To set the stack size to be '0x20' in the RAM area previously defined by gpr0, the following line should be added to the linker command file:

Example 5.6: Stack Definition

<pre>STACK SIZE=0x20 RAM=gpr0</pre>

5.8 Linker Script Caveats

Some linker script caveats:

- You will likely need to modify the linker script files included with MPLINK before using them.
- You will need to set stack size to use MPLAB-C17/C18 with MPLINK.
- You will need to split up memory pages if your code contains goto or call instructions without PAGESEL pseudo-instructions.
- You cannot, in MPASM, switch back and forth to a section in a single file, i.e., you cannot do this:

Example 5.7: MPASM Limitations

<pre>CODE MY_ROM : (instructions) : UDATA MY_VARS : (variables) : CODE MY_ROM :</pre>

Chapter 6. Linker Processing

6.1 Introduction

This chapter discusses how linker processing works.

6.2 Highlights

Topics covered in this chapter:

- Linker Processing Overview
- Linker Allocation Algorithm
- Relocation Example
- Initialized Data

6.3 Linker Processing Overview

A linker combines multiple input object modules into a single executable output module. The input object modules may contain relocatable or absolute sections of code or data which the linker will allocate into target memory. The target memory architecture is described in a linker command file. This linker command file provides a flexible mechanism for specifying blocks of target memory and maps sections to the specified memory blocks. If the linker can not find a block of target memory in which to allocate a section, an error is generated. The linker combines like-named input sections into a single output section. The linker allocation algorithm is described below.

Once the linker has allocated all sections from all input modules into target memory it begins the process of symbol relocation. The symbols defined in each input section have addresses dependent upon the beginning of their sections. The linker adjusts the symbol addresses based upon the ultimate location of their allocated sections.

After the linker has relocated the symbols defined in each input section, it resolves external symbols. The linker attempts to match all external symbol references with a corresponding symbol definition. If any external symbol references do not have a corresponding symbol definition, an attempt is made to locate the corresponding symbol definition in the input library files. If the corresponding symbol definition is not found, an error is generated.

If the resolution of external symbols was successful, the linker then proceeds to patch each section's raw data. Each section contains a list of relocation entries which associate locations in a section's raw data with relocatable symbols. The addresses of the relocatable symbols are patched into the raw data. The process of relocating symbols and patching sections is described below.

MPASM User's Guide with MPLINK and MPLIB

After the linker has processed all relocation entries, it generates the executable output module.

6.4 Linker Allocation Algorithm

The linker allocates sections to allow maximal control over the location of code and data, called "sections," in target memory. There are four kinds of allocations that the linker handles. Sections can be absolute or relocatable (non-absolute), and they can be assigned target memory blocks in the linker command file or they may be left unassigned. So, the following types of allocations exist:

1. Absolute Assigned
2. Absolute Unassigned
3. Relocatable Assigned
4. Relocatable Unassigned

The linker performs allocation of absolute sections first, followed by relocatable assigned sections, followed by relocatable unassigned sections.

6.4.1 Absolute Allocation

Absolute sections may be assigned to target memory blocks in the linker command file. But, since the absolute section's address is fixed, the linker can only verify that if there is an assigned target memory block for an absolute section, the target memory block has enough space and the absolute section does not overlap other sections. If no target memory block is assigned to an absolute section, the linker tries to find the one for it. If one can not be located, an error is generated. Since absolute sections can only be allocated at a fixed address, assigned and unassigned sections are performed in no particular order.

6.4.2 Relocatable Allocation

Once all absolute sections have been allocated, the linker allocates relocatable assigned sections. For relocatable assigned sections, the linker checks the assigned target memory block to verify that there is space available, otherwise it's an error. The allocation of relocatable assigned sections occurs in the order in which they were specified in the linker command file.

After all relocatable assigned sections have been allocated, the linker allocates relocatable unassigned sections. The linker starts with the largest relocatable unassigned section and works its way down to the smallest relocatable unassigned section. For each allocation, it chooses the target memory block with the smallest available space that can accommodate the section. By starting with the largest section and choosing the smallest accommodating space, the linker increases the chances of being able to allocate all the relocatable unassigned sections.

The stack is not a section but gets allocated along with the sections. The linker command file may or may not assign the stack to a specific target memory block. If the stack is assigned a target memory block, it gets allocated just before the relocatable assigned sections are allocated. If the stack is unassigned, then it gets allocated after the relocatable assigned sections and before the other relocatable unassigned sections are allocated.

6.5 Relocation Example

The following example illustrates how the linker relocates sections. Suppose the following source code fragment occurred in a file:

```
/* File: ref.c */
char var1;                /* Line 1 */
void setVar1(void) {      /* Line 2 */
    var1 = 0xFF;          /* Line 3 */
}
```

Suppose this compiles into the following assembly instructions (note: this example deliberately ignores any code generated by MPLAB-C17/C18 to handle the function's entry and exit):

```
0x0000 MOVLW 0xFF
0x0001 MOVLR ??      ; Need to patch with var1's bank
0x0002 MOVWF ??      ; Need to patch with var1's offset
```

When the compiler processes source line 1, it creates a symbol table entry for the identifier var1 which has the following information:

```
Symbol[index] => name=var1, value=0, section=.data,
class=extern
```

When the compiler processes source line 3, it generates two relocation entries in the code section for the identifier symbol var1 since its final address is unknown until link time. The relocation entries have the following information:

```
Reloc[index] => address=0x0001 symbol=var1 type=bank
Reloc[index] => address=0x0002 symbol=var1 type=offset
```

Once the linker has placed every section into target memory, the final addresses are known. Once all identifier symbols have their final addresses assigned, the linker must patch all references to these symbols using the relocation entries. In the example above, the updated symbol might now be at location 0x125:

```
Symbol[index] => name=var1, value=0x125, section=.data,
class=extern
```

If the code section above were relocated to begin at address 0x50, the updated relocation entries would now begin at location 0x51:

```
Reloc[index] => address=0x0051 symbol=var1 type=bank
Reloc[index] => address=0x0052 symbol=var1 type=offset
```

MPASM User's Guide with MPLINK and MPLIB

The linker will step through the relocation entries and patch their corresponding sections. The final assembly equivalent output for the above example would be:

```
0x0050 MOVLW 0xFF
0x0051 MOVLR 0x1 ; Patched with var1's bank
0x0052 MOVWF 0x25 ; Patched with var1's offset
```

6.6 Initialized Data

MPLINK performs special processing for input sections with initialized data. Initialized data sections contain initial values (initializers) for the variables and constants defined within them. Because the variables and constants within an initialized data section reside in RAM, their data must be stored in nonvolatile program memory (ROM). For each initialized data section, the linker creates a section in program memory. The data is moved by initializing code (supplied with MPLAB-C17/C18 and MPASM) to the proper RAM location(s) at start-up.

The names of the initializer sections created by the linker are the same as the initialized data sections with a "_i" appended. For example, if an input object module contains an initialized data section named ".idata_main.o" the linker will create a section in program memory with the name ".idata_main.o_i" which contains the data.

In addition to creating initializer sections, the linker creates a section named ".cinit" in program memory. The ".cinit" section contains a table with entries for each initialized data section. Each entry is a triple which specifies where in program memory the initializer section begins, where in data memory the initialized data section begins, and how many bytes are in the initialized data section. The boot code accesses this table and copies the data from ROM to RAM.

Chapter 7. Sample Application 1

7.1 Highlights

Topics covered in this chapter:

- How to place program code in different memory regions
- How to place data tables in ROM memory
- How to set configuration bits in C

7.2 Overview

This example is for the PIC17C44 in extended microcontroller mode. It places interrupt handling code written in assembly at 0x8000. The assembly code directive, `INTHAND CODE`, places the code that follows into the `INTHAND` section. The linker script maps the `INTHAND` section to the `CODE` region that begins at 0x8000.

It also has a 0x1000 element data table in program memory in the same code page with the interrupt handlers.

Note: A code page in the PIC17CXXX family is 8K words, i.e., 0000 – 1FFF, 2000 – 3FFF, etc.

The data table is defined in C using the `#pragma romdata` directive to place the table in a section called `DATTBL`. The linker script maps the `DATTBL` section to the `CODE` region that begins at 0x8000.

The main function in the C file is placed in the default `CODE` section because there is no section directive explicitly assigned.

MPASM User's Guide with MPLINK and MPLIB

0x0000	Vectors	<u>Code Regions</u> 0-28 – vectors 29-1FFF – code, on-chip 2000-3FFF – code, external 8000-9FFF – protected Interrupt handlers: ASM Main code: C
0x0028		
0x0029	Main Application Code (on-chip)	
0x1FFF		
0x2000	Main Application Code (external)	
0x3FFF		
0x4000		
0x7FFF		
0x8000	Interrupt Handlers	
0x87FE		
0x87FF	Data Tables	
0x9FFF		
0xA000		
0xFFFF		

Figure 7.1: Program Memory Map – Sample Application 1

7.3 Building the Application

The source files for this sample application may be compiled by the included batch file or through MPLAB.

The batch file `app1.bat` compiles this application. This file assumes that `MCC17`, `MPASM`, and `MPLINK` can be found in your `PATH` statement. Also, the `MCC_INCLUDE` environment variable should be set to point to `MCC\H`. You can set these in your `AUTOEXEC.BAT` file and check them by going to the MS-DOS prompt and typing `SET`. `MPLINK` sets the library path to `C:\MCC\LIB`. If `MPLAB-C17/C18` is installed somewhere else, change the path in the `MPLINK` line of the batch file.

To build this application in an MPLAB project

1. Name new project APP1
 - `APP1.HEX` – Becomes the top node of project
2. Set `MPLINK` as the language tool for `APP1.HEX`
3. Add these nodes to the project and set the language tools.
 - `eeeprom.asm` – set `MPASMWIN` as the language tool
 - `eeeprom1.c` – set `MPLAB-C17/C18` as the language tool
 - `eeeprom.lkr`
 - `p17C44.o` – this is in `mcc\lib`
 - `C0L17.0` – this is in `mcc\lib`
 - `IDATA17.0` – this is in `mcc\lib`

4. Edit the APP1 MPLAB Project, since one of the files includes the P17C44.H header file:

- Include path = c:\mcc\h

Note: When linking, you may get the following message:
"Warning – Could not open source file '<filename>'.
This file will not be present in the list file."
This comes from using precompiled libraries, where the source for these libraries is not in the default directory.

7.4 Source Code

EEPROM.ASM – contains a list directive, a code directive with section INTHAND, and a place for putting interrupt handler code.

EEPROM1.C – contains a 0x1000 data array placed in romdata section DATTBL, and a main function in the default code section.

EEPROM.LKR – contains the PIC17C44 description with a protected CODE region from 0x8000 to 0x9FFF. The sections INTHAND and DATTBL are mapped into the protected CODE region.

7.4.1 eeprom.asm

```
; EEPROM.ASM

LIST p=17c44

#include "p17c44.inc"

INTHAND CODE

; place interrupt handling code in here

END
```

MPASM User's Guide with MPLINK and MPLIB

7.4.2 eeprom1.c

```
/* EEPROM1.C */

#include "p17c44.h"

#define DATA_SIZE 0x1000

#pragma romdata DATTBL // put following romdata into section DATTBL
unsigned rom data[DATA_SIZE];

#pragma romdata // set back to default romdata section
/* Main application code for default CODE section */
void main( void )
{
    while( 1 )
    {

    } /* end while */
} /* end main */
```

7.4.3 eeprom.lkr

```
// File: EEPROM.LKR PIC17C44

LIBPATH .

CODEPAGE NAME=vectors START=0x0 END=0x27 PROTECTED
CODEPAGE NAME=page START=0x28 END=0x1FFF
CODEPAGE NAME=eeprom START=0x8000 END=0x9FFF PROTECTED

SHAREBANK NAME=gprshare START=0x1A END=0x1F
SHAREBANK NAME=gprshare START=0x11A END=0x11F
SHAREBANK NAME=gprshare START=0x21A END=0x21F
SHAREBANK NAME=gprshare START=0x31A END=0x31F

DATABANK NAME=gpr0 START=0x20 END=0xFF
DATABANK NAME=gpr1 START=0x120 END=0x1FF

SHAREBANK NAME=sfrnobnk START=0x0 END=0xF PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED

SHAREBANK NAME=sfrprod START=0x18 END=0x19 PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED

DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED

SECTION NAME=STARTUP ROM=vectors // Reset and interrupt vectors
SECTION NAME=PROG ROM=page // main application code space
SECTION NAME=INTHAND ROM=eeprom // Interrupt handlers
SECTION NAME=DATTBL ROM=eeprom // Data tables

STACK SIZE=0x3F RAM=gpr0
```

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 8. Sample Application 2

8.1 Highlights

Topics covered in this chapter:

- How to partition memory for a boot loader
- How to compile code that will be loaded into external RAM and executed

8.2 Overview

Many applications require field-updateable firmware. The PIC17CXXX family of processors supports this in a straightforward manner in extended microcontroller mode. This example uses a PIC17C44.

A simple boot loader is located in on-chip program memory, possibly with one or more support functions that may be called by the firmware. The primary firmware is located in off-chip program memory. The program memory data tables are located in the appropriate memory region as an assigned section.

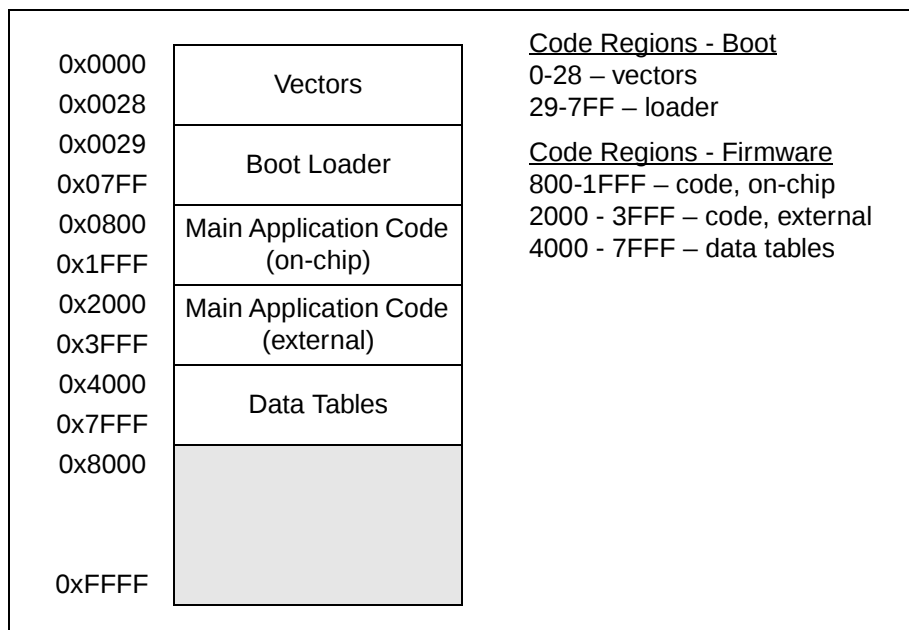


Figure 8.1: Program Memory Map – Sample Application 2

8.3 Building the Application

The source files for this sample application may be compiled by the included batch file or through MPLAB.

The batch file `app2.bat` compiles this application. This file assumes that `MCC17`, `MPASM`, and `MPLINK` can be found in your `PATH` statement. Also, the `MCC_INCLUDE` environment variable should be set to point to `MCC\H`. You can set these in your `AUTOEXEC.BAT` file and check them by going to the MS-DOS prompt and typing `SET`. `MPLINK` sets the library path to `C:\MCC\LIB`. If `MPLAB-C17/C18` is installed somewhere else, change the path in the `MPLINK` line of the batch file.

This application consists of two separate object files, one that will be programmed into a `PIC17C44`, and another that will be downloadable firmware that can be loaded into external program memory.

To build this application in an MPLAB project, two projects need to be created and built. Set up the first project to build the boot loader:

1. Name new project `APP2BOOT`
 - `APP2BOOT.HEX` – Becomes the top node of project
2. Set `MPLINK` as the language tool for `APP1.HEX`
3. Add these nodes to the project and set the language tools.
 - `boot.c` – set `MPLAB-C17/C18` as the language tool
 - `loader.lkr`
 - `C0S17.0` – this is in `mcc\lib`
4. Edit the `APP2BOOT` MPLAB Project
 - Library path = `c:\mcc\lib`
5. Build project, then save and close.

Set up a second project to compile the firmware:

1. Name new project `APP2`
 - `APP2.HEX` – Becomes the top node of project
2. Set `MPLINK` as the language tool for `APP1.HEX`
3. Add these nodes to the project and set the language tools.
 - `fwentry.asm` – set `MPASMWIN` as the language tool
 - `firmware.c` – set `MPLAB-C17/C18` as the language tool
 - `fwtables.c` – set `MPLAB-C17/C18` as the language tool
 - `firmware.lkr`
4. Edit the `APP1` MPLAB Project:
 - Library path = `c:\mcc\lib` (or where your `mcc\lib` is located)
5. Build project, save and close.

Note: When linking, you may get the following message:
“Warning – Could not open source file '<filename>'.
This file will not be present in the list file.”
This comes from using precompiled libraries, where the source for
these libraries is not in the default directory.

8.4 Source Code – Boot Loader

boot.c – The boot loader code copies the firmware into external program memory and jumps to a pre-defined address which is the entry point of the firmware. A firmware-callable support function is supplied at a fixed address as well.

loader.lkr – The boot loader linker script defines only those regions of program memory which are on-chip.

MPASM User's Guide with MPLINK and MPLIB

8.4.1 boot.c

```
#include <p17c44.h>

/* The firmware entry point is defined to be at 0x4000 */
#define FW_ENTRY    0x4000

#define LGOTO(x) { _asm movlw high (x) movwf PCLATH movlw (x) movwf PCL _endasm }

void load_firmware( void );

void load_firmware( void )
{
    /* Do whatever needs to be done to load in the firmware
     * into external memory.
     */
}

void main( void )
{
    load_firmware();
    LGOTO( FW_ENTRY );
}

/* We'll provide a function for the firmware to call to output a character
 * to an LCD display.
 *
 * We need the function at a hard address so we can define the entry
 * point in the firmware.
 */
#pragma code out_lcd=0x1000

void out_lcd( unsigned char );

void out_lcd( unsigned char ch )
{
    /* Do whatever needs doing */
}

#pragma romdata CONFIG
// The following config bit definitions are from the P17C44.INC header file:
#define _PMC_MODE 0x7FAF
#define _XMC_MODE 0xFFBF
#define _MC_MODE 0xFFEF
#define _MP_MODE 0xFFFF

#define _WDT_NORM 0xFFF3
#define _WDT_OFF 0xFFF3
#define _WDT_64 0xFFF7
#define _WDT_256 0xFFF7
#define _WDT_1 0xFFFF

#define _LF_OSC 0xFFFC
#define _RC_OSC 0xFFFD
#define _XT_OSC 0xFFFE
#define _EC_OSC 0xFFFF

rom const unsigned int my_config = _MC_MODE & _WDT_OFF & _XT_OSC ;
```

8.4.2 loader.lkr

```
LIBPATH .           // Add a directory to the library search path
LIBPATH c:\work\mcc\cc\install\lib
FILES    p17c44.o

CODEPAGE    NAME=reset_vector    START=0x0000    END=0x0027    // Reset Vector
CODEPAGE    NAME=page0           START=0x0028    END=0x1FFF    // On chip memory
CODEPAGE    NAME=page1           START=0x2000    END=0x3FFF    // On chip memory
CODEPAGE    NAME=config          START=0xFE00    END=0xFE0F    PROTECTED

SHAREBANK   NAME=sharesfr        START=0x00     END=0x0f     PROTECTED
DATABANK    NAME=sfr0            START=0x10     END=0x17     PROTECTED
DATABANK    NAME=sfr1            START=0x110    END=0x117    PROTECTED
DATABANK    NAME=sfr2            START=0x210    END=0x217    PROTECTED
DATABANK    NAME=sfr3            START=0x310    END=0x317    PROTECTED
DATABANK    NAME=sfr4            START=0x410    END=0x417    PROTECTED
DATABANK    NAME=sfr5            START=0x510    END=0x517    PROTECTED
DATABANK    NAME=sfr6            START=0x610    END=0x617    PROTECTED
DATABANK    NAME=sfr7            START=0x710    END=0x717    PROTECTED
SHAREBANK   NAME=registers       START=0x18     END=0x1f     PROTECTED
DATABANK    NAME=gpr0            START=0x20     END=0xEF     // General purpose RAM bank 0
DATABANK    NAME=gpr1            START=0x120    END=0x1FB    // General purpose RAM bank 1

DATABANK    NAME=stack           START=0xF0     END=0xFF     PROTECTED // Stack RAM

SECTION     NAME=CONFIG          ROM=config// configuration bits location

STACK SIZE=0x0F RAM=stack
```

MPASM User's Guide with MPLINK and MPLIB

8.5 Source Code – Firmware

firmware.c – The entry function is located in an absolute section at the pre-determined firmware entry point. The rest of the firmware is located by the linker.

fwtables.c – The program memory data tables are defined in a data section which will be located by the linker.

fwentry.h – The support functions are prototyped just as any other C callable function.

fwentry.asm – For the firmware, the linker needs a symbol at the entry point address for each function.

firmware.lkr – The firmware linker script defines protected regions from the program memory data tables and the on-chip support functions. The rest of external memory is defined as available for use.

8.5.1 firmware.c

```
#include <p17c44.h>
#include "fwentry.h"

void fw_entry_func( void );
void signon( void );

#pragma code fw_entry = 0x4000
void fw_entry_func( void )
{
    /* signon message */
    signon();

    /* do whatever needs doing... */

    while(1);
}
#pragma code // we don't care where the rest of stuff goes

void signon( void )
{
    static const rom char msg[] = "FW v1.00";
    rom char *p;
    for( p=msg ; *p != 0 ; p++ )
        out_lcd( *p );
}
```

8.5.2 fwtables.c

```
#include <p17c44.h>

#pragma romdata fw_tables

rom unsigned datatable[] = {
    0x0000,
    0x0001,
    0x0002,
    0x0003,
    0x0004,
    0x0005,
    0x0006,
    0x0007,
    0x0008,
};
```

8.5.3 fwentry.h

```
#ifndef FWENTRY_H
#define FWENTRY_H

void out_lcd( unsigned char ch );

#endif
```

8.5.4 fwentry.asm

```
; Provide symbols at the appropriate addresses for the system routines
; in ROM which the firmware can call. The actual code for these routines
; is in the bootloader in ROM, these are just place-holders.

LIST P=17c44

GLOBAL out_lcd

OUTLCDENTRY CODE 0x1000
out_lcd
END
```

8.5.5 firmware.lkr

```
LIBPATH . // Add a directory to the library search path
LIBPATH c:\work\mcc\cc\install\lib
FILES p17c44.o

//CODEPAGE NAME=reset_vector START=0x0000 END=0x0027 // Reset Vector
// For off-chip firmware, we don't want anything located in the
// on-chip address space.
//
// We make some of the on-chip stuff PROTECTED because we put
// the entry points to the bootloader which the firmware calls
// in there. No code will actually go there, but we need to
// locate empty sections there to get the relocations.
CODEPAGE NAME=entrypoints START=0x1000 END=0x3fff PROTECTED

CODEPAGE NAME=firmware START=0x4000 END=0x5FFF
CODEPAGE NAME=tables START=0x6000 END=0x7FFF PROTECTED

SHAREBANK NAME=sharesfr START=0x00 END=0x0f PROTECTED
DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED
DATABANK NAME=sfr4 START=0x410 END=0x417 PROTECTED
DATABANK NAME=sfr5 START=0x510 END=0x517 PROTECTED
DATABANK NAME=sfr6 START=0x610 END=0x617 PROTECTED
DATABANK NAME=sfr7 START=0x710 END=0x717 PROTECTED
SHAREBANK NAME=registers START=0x18 END=0x1f PROTECTED
DATABANK NAME=gpr0 START=0x20 END=0xef // General purpose RAM bank 0
DATABANK NAME=gpr1 START=0x120 END=0x1ff // General purpose RAM bank 1

DATABANK NAME=stack START=0xf0 END=0xff PROTECTED // Stack RAM

STACK SIZE=0x0f RAM=stack
```

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 9. Sample Application 3

9.1 Highlights

Topics covered in this chapter:

- How to access peripherals that are memory mapped
- How to create new sections

9.2 Overview

The example has two external devices; a DAC at 0x8000 and an IRD at 0x9000.

The code declares multiple variables at each of the two locations 0x8000 and 0x9000. When one of these variables is read or written, the device at the location of the variable is read from or written to. The directive `#pragma romdata` creates an absolute section at the specified memory location.

The linker script file declares code pages for each device and for the program memory. The code page for each device is set to `PROTECTED` to prevent the linker from placing unassigned relocatable sections into this memory block. The linker script file also has sections declared for the `code0` and `code1` code pages. The section names are used with the directive `#pragma code` so the linker will place the code following the `#pragma` in that section's memory location.

MPASM User's Guide with MPLINK and MPLIB

0x0000	Vectors	<u>Code Regions</u> 0-28 – vectors 29-1FFF – code page 0 2000-3FFF – code page 1 8000 - 800F – device 1 (protected) 9000 - 9001 – device 2 (protected) Code Page 0 & 1: C
0x0028		
0x0029	Code Page 0	
0x1FFF		
0x2000	Code Page 1	
0x3FFF		
0x8000	Device 1 - DAC	
0x800F		
0x9000	Device 2 - IRD	
0x9001		
0xFFFF		

Figure 9.1: Program Memory Map – Sample Application 3

9.3 Building the Application

The source files for this sample application may be compiled by the included batch file or through MPLAB.

The batch file `app3.bat` compiles this application. This file assumes that `MCC17`, `MPASM`, and `MPLINK` can be found in your `PATH` statement. Also, the `MCC_INCLUDE` environment variable should be set to point to `MCC\H`. You can set these in your `AUTOEXEC.BAT` file and check them by going to the MS-DOS prompt and typing `SET`. `MPLINK` sets the library path to `C:\MCC\LIB`. If `MPLAB-C17/C18` is installed somewhere else, change the path in the `MPLINK` line of the batch file.

To build this application in an MPLAB project

1. Name new project `APP3`
 - `APP3.HEX` – Becomes the top node of project
2. Set `MPLINK` as the language tool for `APP1.HEX`
3. Add these nodes to the project and set the language tools.
 - `memmapio.c` – set `MPLAB-C17/C18` as the language tool
 - `memmapio.lkr`
 - `p17C756.o` – this is in `mcc\lib`
4. Edit the `APP1` MPLAB Project:
 - Library path = `c:\mcc\lib`

Note: When linking, you may get the following message:
"Warning – Could not open source file '<filename>'.
This file will not be present in the list file."
This comes from using precompiled libraries, where the source for these libraries is not in the default directory.

Note: Additionally, you will get a message:
"Warning[2003]...\MEMMAPIO.C 25 :
'rom' and 'volatile' in same declaration"
This message is normal and is just reminding you that the compiler will treat the variable as the application requires.

MPASM User's Guide with MPLINK and MPLIB

9.4 Source Code

9.4.1 memmapio.c

```
#include <P17C756.H>
void main(void);

#pragma romdata dev1 = 0x8000
unsigned rom loc0;
unsigned rom loc1;
unsigned rom loc2;
unsigned rom loc3;
unsigned rom loc4;
unsigned rom loc5;
unsigned rom loc6;
unsigned rom loc7;
unsigned rom loc8;
unsigned rom loc9;
unsigned rom loca;
unsigned rom locb;
unsigned rom locc;
unsigned rom locd;
unsigned rom loce;
unsigned rom locf;

#pragma romdata dev2 = 0x9000
const volatile unsigned rom inp;
unsigned rom outp;

#pragma code code0
void main(void)
{
    unsigned int input;

    loc0 = 0;
    loc1 = 1;

    input = inp;
    outp = 0x7;
}
```

9.4.2 memmapio.lkr

```
// File: memmapio.lkr
// Sample linker command file for 17C756, 17C756A, 17C766
// 12/16/97

LIBPATH .

CODEPAGE NAME=vectors START=0x0 END=0x27 PROTECTED
CODEPAGE NAME=code0 START=0x28 END=0x1fff
CODEPAGE NAME=code1 START=0x2000 END=0x3FFF
CODEPAGE NAME=device1 START=0x8000 END=0x800F PROTECTED
CODEPAGE NAME=device2 START=0x9000 END=0x9001 PROTECTED
CODEPAGE NAME=config START=0xFE00 END=0xFE0F PROTECTED

SHAREBANK NAME=sfrnobnk START=0x0 END=0xF PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x400 END=0x40F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x500 END=0x50F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x600 END=0x60F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x700 END=0x70F PROTECTED

DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED
DATABANK NAME=sfr4 START=0x410 END=0x417 PROTECTED
DATABANK NAME=sfr5 START=0x510 END=0x517 PROTECTED
DATABANK NAME=sfr6 START=0x610 END=0x617 PROTECTED
DATABANK NAME=sfr7 START=0x710 END=0x717 PROTECTED

SHAREBANK NAME=sfrprod START=0x18 END=0x19 PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED
SHAREBANK NAME=sfrprod START=0x418 END=0x419 PROTECTED
SHAREBANK NAME=sfrprod START=0x518 END=0x519 PROTECTED
SHAREBANK NAME=sfrprod START=0x618 END=0x619 PROTECTED
SHAREBANK NAME=sfrprod START=0x718 END=0x719 PROTECTED

SHAREBANK NAME=registers START=0x1A END=0x1F PROTECTED
SHAREBANK NAME=registers START=0x11A END=0x11F PROTECTED
SHAREBANK NAME=registers START=0x21A END=0x21F PROTECTED
SHAREBANK NAME=registers START=0x31A END=0x31F PROTECTED
SHAREBANK NAME=registers START=0x41A END=0x41F PROTECTED
SHAREBANK NAME=registers START=0x51A END=0x51F PROTECTED
SHAREBANK NAME=registers START=0x61A END=0x61F PROTECTED
SHAREBANK NAME=registers START=0x71A END=0x71F PROTECTED

DATABANK NAME=gpr0 START=0x20 END=0xFF
DATABANK NAME=gpr1 START=0x120 END=0x1FF
DATABANK NAME=gpr2 START=0x220 END=0x2FF
DATABANK NAME=gpr3 START=0x320 END=0x3FF

SECTION NAME=code0 ROM=code0
SECTION NAME=code1 ROM=code1

// The stack directive below is for use with MPLAB-C17/C18
STACK SIZE=0x20
```

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Chapter 10. Sample Application 4

10.1 Highlights

Topic covered in this chapter:

- How to manually partition RAM space for program usage

10.2 Overview

You are given a PIC17C756 small-model, microcontroller-mode project, `morse.pjt`, that retrieves a Morse code pattern from USART1 and outputs that pattern to PORTB. It consists of the following source files:

<code>main.c</code>	main program to output a Morse code pattern to PORTB
<code>morse.c</code>	subroutines that output a Morse code pattern to PORTB
<code>morse.h</code>	header file for <code>morse.c</code> – defines function prototypes and defines important constants (e.g., input pattern representation)
<code>usart.c</code>	subroutines that handle getting input from USART1
<code>usart.h</code>	header file for <code>usart.c</code> – defines function prototypes
<code>portb.c</code>	subroutines that handle getting output to PORTB
<code>portb.h</code>	header file for <code>portb.c</code> – defines function prototypes
<code>delayms.asm</code>	subroutine that performs a time delay (calculations are based on a 4 Mhz clock)
<code>delayms.h</code>	header file for <code>delayms.asm</code> – defines function prototypes
<code>morse756.lkr</code>	linker script file containing no sections, and configured for the small memory model

The linker script file and source code locate the data memory as described in Figure 10.1.

MPASM User's Guide with MPLINK and MPLIB

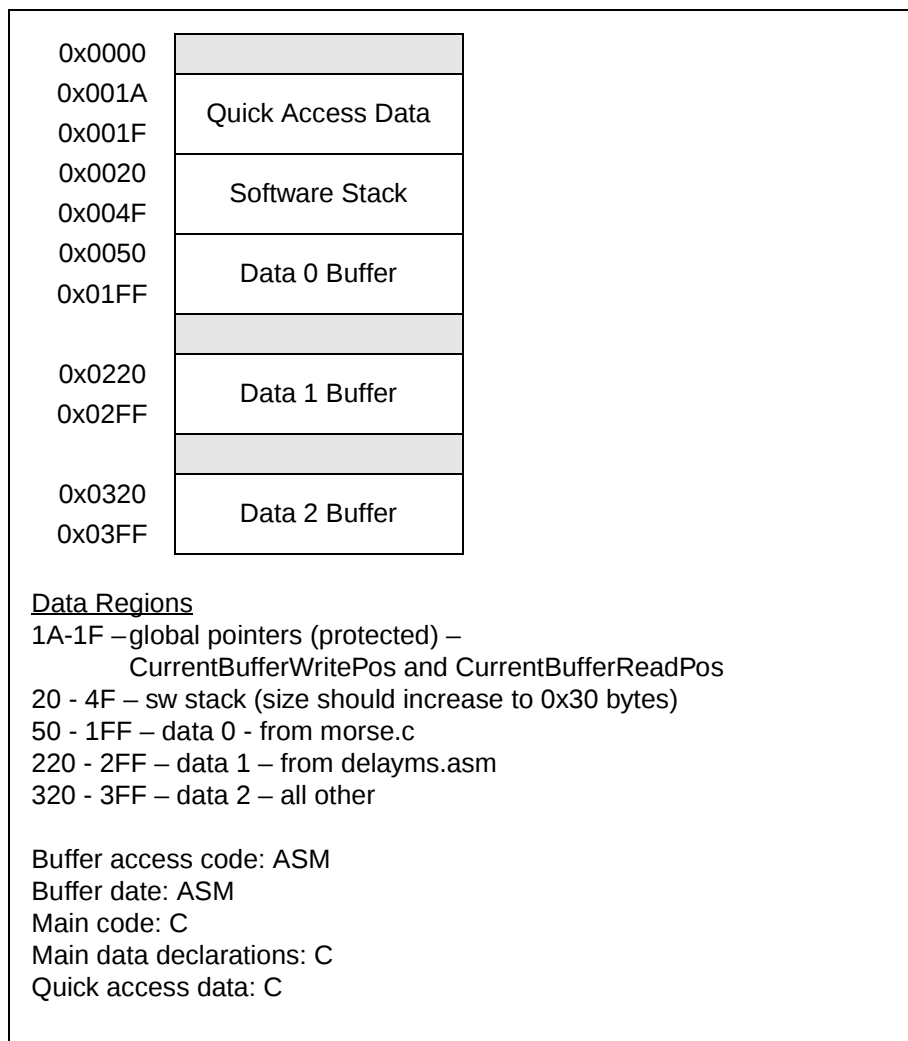


Figure 10.1: Register File Map – Sample Application 4

10.3 Building the Application

The source files for this sample application may be compiled by the included batch file or through MPLAB.

The batch file `app4.bat` compiles this application. This file assumes that MCC17, MPASM, and MPLINK can be found in your PATH statement. Also, the MCC_INCLUDE environment variable should be set to point to MCC\H. You can set these in your AUTOEXEC.BAT file and check them by going to the MS-DOS prompt and typing SET. MPLINK sets the library path to C:\MCC\LIB. If MPLAB-C17/C18 is installed somewhere else, change the path in the MPLINK line of the batch file.

To build this application in an MPLAB project

1. Name new project APP1
 - APP4.HEX – Becomes the top node of project
2. Set MPLINK as the language tool for APP4.HEX
3. Add these nodes to the project and set the language tools:
 - delayms.asm – set MPASM as the language tool
 - main.c – set MPLAB-C17/C18 as the language tool
 - morse.c – set MPLAB-C17/C18 as the language tool
 - portb.c – set MPLAB-C17/C18 as the language tool
 - usart.c – set MPLAB-C17/C18 as the language tool
 - morse756.lkr
 - p17C756.o – this is in mcc\lib
 - C0L17.O – this is in mcc\lib
 - IDATA17.O – this is in mcc\lib
 - PMC756L.LIB – this is in mcc\lib
 - INT756L.LIB – this is in mcc\lib

Note: When linking, you may get the following message:
"Warning – Could not open source file '<filename>'.
This file will not be present in the list file."
This comes from using precompiled libraries, where the source for these libraries is not in the default directory.

MPASM User's Guide with MPLINK and MPLIB

10.4 Source Code

10.4.1 morse756.lkr

```
// File: morse756.lkr

LIBPATH .

CODEPAGE NAME=vectors START=0x0 END=0x27 PROTECTED
CODEPAGE NAME=page0 START=0x28 END=0x1FFF
CODEPAGE NAME=page1 START=0x2000 END=0x3FFF PROTECTED
CODEPAGE NAME=config START=0xFE00 END=0xFE0F PROTECTED

SHAREBANK NAME=sfrnobnk START=0x0 END=0xF PROTECTED
SHAREBANK NAME=sfrnobnk START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x300 END=0x30F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x400 END=0x40F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x500 END=0x50F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x600 END=0x60F PROTECTED
SHAREBANK NAME=sfrnobnk START=0x700 END=0x70F PROTECTED

DATABANK NAME=sfr0 START=0x10 END=0x17 PROTECTED
DATABANK NAME=sfr1 START=0x110 END=0x117 PROTECTED
DATABANK NAME=sfr2 START=0x210 END=0x217 PROTECTED
DATABANK NAME=sfr3 START=0x310 END=0x317 PROTECTED
DATABANK NAME=sfr4 START=0x410 END=0x417 PROTECTED
DATABANK NAME=sfr5 START=0x510 END=0x517 PROTECTED
DATABANK NAME=sfr6 START=0x610 END=0x617 PROTECTED
DATABANK NAME=sfr7 START=0x710 END=0x717 PROTECTED

SHAREBANK NAME=sfrprod START=0x18 END=0x19 PROTECTED
SHAREBANK NAME=sfrprod START=0x118 END=0x119 PROTECTED
SHAREBANK NAME=sfrprod START=0x218 END=0x219 PROTECTED
SHAREBANK NAME=sfrprod START=0x318 END=0x319 PROTECTED
SHAREBANK NAME=sfrprod START=0x418 END=0x419 PROTECTED
SHAREBANK NAME=sfrprod START=0x518 END=0x519 PROTECTED
SHAREBANK NAME=sfrprod START=0x618 END=0x619 PROTECTED
SHAREBANK NAME=sfrprod START=0x718 END=0x719 PROTECTED

SHAREBANK NAME=share0 START=0x1A END=0x1F PROTECTED
SHAREBANK NAME=share0 START=0x11A END=0x11F PROTECTED
SHAREBANK NAME=share0 START=0x21A END=0x21F PROTECTED
SHAREBANK NAME=share0 START=0x31A END=0x31F PROTECTED
SHAREBANK NAME=share0 START=0x41A END=0x41F PROTECTED
SHAREBANK NAME=share0 START=0x51A END=0x51F PROTECTED
SHAREBANK NAME=share0 START=0x61A END=0x61F PROTECTED
SHAREBANK NAME=share0 START=0x71A END=0x71F PROTECTED

DATABANK NAME=stackram START=0x20 END=0x4F PROTECTED
DATABANK NAME=gpr0 START=0x50 END=0xFF PROTECTED
DATABANK NAME=gpr1 START=0x120 END=0x1FF PROTECTED
DATABANK NAME=gpr2 START=0x220 END=0x2FF PROTECTED
DATABANK NAME=gpr3 START=0x320 END=0x3FF

// Put your sections here
SECTION NAME=global_vars RAM=share0
SECTION NAME=morse_dat RAM=gpr0
SECTION NAME=morse_buf RAM=gpr1
SECTION NAME=delay_dat RAM=gpr2

// The stack directive below is for use with MPLAB-C17/C18
STACK SIZE=0x30 RAM=stackram
```

10.4.2 main.c

```
#include "usart.h"
#include "morse.h"
#include "delayms.h"
#include <p17c756.h>

void main( void )
{
    InitializeMorseTable();
    Install_INT( GetMorseFromUSART );
    InitializeUSART();

    while( DisplayNextMorseVal() )
    {
        DelayXMS( 100 );
    }

    TerminateUSARTInput();
}
```

10.4.3 morse.h

```
#ifndef _MORSE_H_
#define _MORSE_H_

#define MORSE_BUFFER_SIZE    220

#define DOT          1
#define DASH         2
#define TERMINATE    3

#define DOT_LENGTH    100
#define DASH_LENGTH   250

void InitializeMorseTable( void );
unsigned char DisplayNextMorseVal( void );

#endif
```

MPASM User's Guide with MPLINK and MPLIB

10.4.4 morse.c

```
#include "portb.h"
#include "morse.h"
#include "delayms.h"

#pragma udata shared global_vars
unsigned char *CurrentBufferWritePos;
unsigned char *CurrentBufferReadPos;

#pragma udata morse_buf
unsigned char USARTBuffer[MORSE_BUFFER_SIZE];

#pragma udata morse_dat
void InitializeMorseTable( void )
{
    CurrentBufferWritePos = USARTBuffer;
    CurrentBufferReadPos = USARTBuffer;
    PortBInitialize();
}

unsigned char DisplayNextMorseVal( void )
{
    unsigned int HoldTime;

    if( CurrentBufferReadPos == CurrentBufferWritePos )
        return 1;

    if( CurrentBufferReadPos == USARTBuffer )
        CurrentBufferReadPos = USARTBuffer + MORSE_BUFFER_SIZE - 1;
    else
        CurrentBufferReadPos--;

    switch( *CurrentBufferReadPos )
    {
        case TERMINATE:
            return 0;

        case DOT:
            HoldTime = DOT_LENGTH;
            break;

        case DASH:
            HoldTime = DASH_LENGTH;
            break;

        default:
            return 1;
    }

    PortBOutput( 0xFF );
    DelayXMS( HoldTime );
    PortBOutput( 0x00 );
    DelayXMS( HoldTime );
    return 1;
}
```

10.4.5 portb.h

```
#ifndef _PORTB_H_
#define _PORTB_H_

void PortBInitialize( void );
void PortBOutput( unsigned char OutputValue );

#endif
```

10.4.6 portb.c

```
#include "portb.h"
#include <p17C756.h>

void PortBInitialize( void )
{
    PORTB = 0xff;    // Clear Port B register
    DDRB = 0x00;     // Set Port B as output
}

void PortBOutput( unsigned char OutputValue )
{
    PORTB = 0xff;    // Clear Port B register
    PORTB = OutputValue; // Write value out
}
```

10.4.7 usart.h

```
#ifndef _USART_H_
#define _USART_H_

void InitializeUSART( void );
void TerminateUSARTInput( void );
void GetMorseFromUSART( void );

#endif
```

MPASM User's Guide with MPLINK and MPLIB

10.4.8 **usart.c**

```
#include "usart.h"
#include "morse.h"
#include <usart16.h>
#include <p17C756.h>

extern unsigned char USARTBuffer[];
extern unsigned char *CurrentBufferWritePos;
extern unsigned char *CurrentBufferReadPos;

void InitializeUSART( void )
{
    OpenUSART1( USART_TX_INT_ON & USART_ASYNC_MODE &
                USART_EIGHT_BIT & USART_CONT_RX, 1 );
}

void TerminateUSARTInput( void )
{
    CloseUSART1();
}

void GetMorseFromUSART( void )
{
    /* Wrap if we are at the end of the buffer */
    if( CurrentBufferWritePos >= (USARTBuffer + MORSE_BUFFER_SIZE - 1) )
    {
        /* Unless buffer is full, increment position */
        if( CurrentBufferReadPos != USARTBuffer )
            CurrentBufferWritePos = USARTBuffer;
    }
    else
    {
        /* Unless buffer is full, increment position */
        if( CurrentBufferReadPos != (CurrentBufferWritePos + 1) )
        {
            CurrentBufferWritePos++;
        }
    }

    *CurrentBufferWritePos = ReadUSART1();
}
```

10.4.9 **delayms.h**

```
#ifndef _DELAYMS_H_
#define _DELAYMS_H_

void DelayXMS( unsigned int time );

#endif
```

10.4.10 delayms.asm

```
list p=17c756
#include <p17c756.inc>

EXTERN _stack
GLOBAL DelayXMS

delay_dat    UDATA
MSB          RES    1
LSB          RES    1

CODE
DelayXMS:
    banksel _stack
    movfp   _stack,FSR0
    decf    FSR0,f
    movfp   INDF0,MSB
    decf    FSR0,f
    movfp   INDF0,LSB

dly1:
    nop
    nop
    decfsz  LSB,1
    goto    dly1
    decfsz  MSB,1
    goto    dly1
    return

end
```

MPASM User's Guide with MPLINK and MPLIB

NOTES:



MPASM USER'S GUIDE with MPLINK and MPLIB

Part 3 – MPLIB

Chapter 1. MPLIB Preview	163
Chapter 2. MPLIB – Installation and Getting Started	165
Chapter 3. Using MPLIB	167

MPASM User's Guide with MPLINK and MPLIB

Chapter 1. MPLIB Preview

1.1 Introduction

Topics covered in this chapter.

1.2 Highlights

Topics covered in this chapter:

- What MPLIB Is
- What MPLIB Does
- How MPLIB Helps You

1.3 What MPLIB Is

MPLIB is a librarian for use with COFF object modules (`filename.o`) created using either MPASM v2.0, MPASMWIN v2.0, or MPLAB-C v2.0 or later.

1.4 What MPLIB Does

A librarian manages the creation and modification of library files. A library file is simply a collection of object modules that are stored in a single file. There are several reasons for creating library files:

- Libraries make linking easier. Since library files can contain many object files, the name of a library file can be used instead of the names of many separate object files when linking.
- Libraries help keep code small. Since a linker only uses the required object files contained in a library, not all object files which are contained in the library necessarily wind up in the linker's output module.
- Libraries make projects more maintainable. If a library is included in a project, the addition or removal of calls to that library will not require a change to the link process.
- Libraries help to convey the purpose of a group of object modules. Since libraries can group together several related object modules, the purpose of a library file is usually more understandable than the purpose of its individual object modules. For example, the purpose of a file named 'math.lib' is more apparent than the purpose of 'power.o', 'ceiling.o', and 'floor.o'.

1.5 How MPLIB Helps You

MPLIB can help you in the following ways:

- MPLIB makes linking easier because single libraries can be included instead of many smaller files.
- MPLIB helps keep code maintainable by grouping related modules together.
- MPLIB commands allow libraries to be created and modules to be added, listed, replaced, deleted, or extracted.

Chapter 2. MPLIB – Installation and Getting Started

2.1 Introduction

This chapter discussed how to install MPLIB and an overview of the librarian (MPLIB).

2.2 Highlights

Topics covered in this chapter:

- Installation
- Overview of Librarian

2.3 Installation

There are two versions of MPLIB. MPLIBD.EXE is a DOS extender version and is only recommended for DOS or Windows 3.x systems. MPLIB.EXE is a Windows 32-bit console application and is for Windows 95/98/NT.

MPLIBD.EXE and MPLIB.EXE are command-line applications. MPLIBD.EXE may be used with DOS or a DOS window in Windows 3.x. MPLIB.EXE may be used with a DOS window in Windows 95/98/NT.

If you are going to use MPLAB, you do not need to install the librarian separately. When MPLAB is installed, MPLIB is also installed. To find out how to install MPLAB, please refer to the *MPLAB User's Guide* (DS51025). You may obtain the MPLAB software and user's guide either from the Microchip Technical Library CD or from our web site.

If you are not going to use MPLAB, you can obtain the librarian separately either from the Microchip Technical Library CD or from our web site. MPLINK is bundled with MPLIB into a zip file. To install:

- Create a directory in which to place the files
- Unzip the MPLINK and MPLIB files using either WinZip or PKZIP

MPASM User's Guide with MPLINK and MPLIB

2.4 Overview of Librarian

Figure 2.1 is a diagram of how MPLIB works with other Microchip tools.

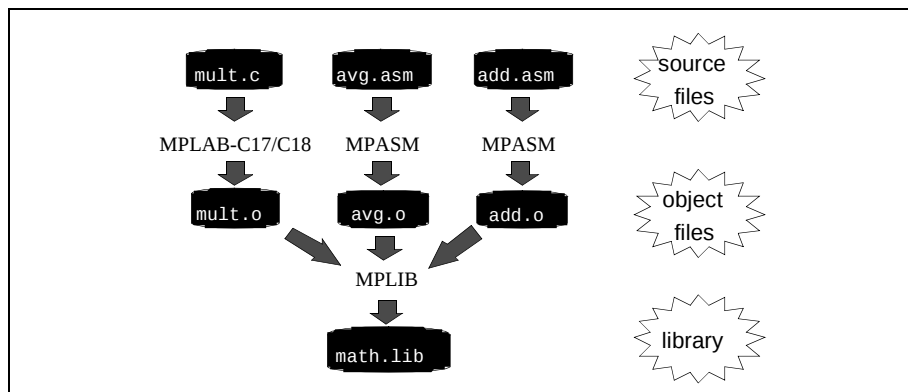


Figure 2.1: MPLIB and Other Microchip Tools

MPLIB combines multiple input object modules, generated by MPASM or MPLAB-C17/C18, into a single output library (lib) file. Library files are used in conjunction with a linker (See Chapter 2, “MPLINK – Installation and Getting Started” for more information on MPLINK) to produce executable code.

Chapter 3. Using MPLIB

3.1 Introduction

This chapter discussed how to use the librarian (MPLIB).

3.2 Highlights

Topics covered in this chapter:

- Usage Format
- Usage Examples
- Tips

3.3 Usage Format

MPLIB is invoked with the following syntax:

```
mplib [/q] [{ctdrx} LIBRARY [MEMBER...]
```

options:

/c create library;	creates a new LIBRARY with the listed MEMBER(s)
/t list members;	prints a table showing the names of the members in the LIBRARY
/d delete member;	deletes MEMBER(s) from the LIBRARY; if no MEMBER is specified the LIBRARY is not altered
/r add/replace member;	if MEMBER(s) exist in the LIBRARY, then they are replaced, otherwise MEMBER is appended to the end of the LIBRARY
/x extract member;	if MEMBER(s) exist in the LIBRARY, then they are extracted. If no MEMBER is specified, all members will be extracted
/q quiet mode;	no output is displayed

MPASM User's Guide with MPLINK and MPLIB

3.4 Usage Examples

Suppose a library named `dsp.lib` is to be created from three object modules named `fft.o`, `fir.o`, and `iir.o`. The following command line would produce the desired results:

```
mplib /c dsp.lib fft.o fir.o iir.o
```

To display the names of the object modules contained in a library file named `dsp.lib`, the following command line would be appropriate:

```
mplib /t dsp.lib
```

3.5 Tips

MPLIB creates library files that may contain only a single external definition for any symbol. Therefore, if two object modules define the same external symbol, MPLIB will generate an error if both object modules are added to the same library file.

To minimize the code and data space which results from linking with a library file, the library's member object modules should be as small as possible. Creating object modules that contain only a single function can significantly reduce code space.

Appendices

Appendix A. Hex File Formats	171
Appendix B. Quick Reference	175
Appendix C. MPASM Errors/Warnings/Messages	197
Appendix D. MPLINK Errors/Warnings	207
Appendix E. MPLIB Errors	215

MPASM User's Guide with MPLINK and MPLIB

Appendix A. Hex File Formats

A.1 Introduction

MPASM is capable of generating several different hex file formats.

A.2 Highlights

Topics covered in this appendix:

- Intel® Hex Format (INHX8M) (for standard programmers)
- Intel Split Hex Format (INHX8S) (for ODD/EVEN ROM programmers)
- Intel Hex 32 Format (INHX32) (for 16-bit core programmers)

A.3 Intel Hex Format (.HEX)

This format produces one 8-bit hex file with a low byte, high byte combination. Since each address can only contain 8 bits in this format, all addresses are doubled. This file format is useful for transferring PICmicro series code to PRO MATE II, PICSTART and third party PICmicro programmers.

Each data record begins with a 9 character prefix and ends with a 2-character checksum. Each record has the following format:

:BBAAAATTHHHH . . . HHHCC

where:

BB – is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line.

AAAA – is a four digit hexadecimal address representing the starting address of the data record.

TT – is a two digit record type record type that will always be '00' except for the end-of-file record, which will be '01'.

HH – is a two digit hexadecimal data byte, presented in low-byte/high-byte combinations.

CC – is a two digit hexadecimal checksum that is the two's complement of the sum of all preceding bytes in the record.

MPASM User's Guide with MPLINK and MPLIB

Example A.1:

```
<file_name>.HEX
:100000000000000000000000000000000000F0
:0400100000000000EC
:100032000000280040006800A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

A.4 8-Bit Split Format (.HXL/.HXH)

The split 8-bit file format produces two output files: .HXL and .HXH. The format is the same as the normal 8-bit format, except that the low bytes of the data word are stored in the .HXL file, and the high bytes of the data word are stored in the .HXH file, and the addresses are divided by two. This is used to program 16-bit words into pairs of 8-bit EPROMs, one file for Low Byte, one file for High Byte.

Example A.2:

```
<file_name>.HXL
:0A0000000000000000000000000000F6
:1000190000284068A8E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C8080808034303E8E8FFD0
:03003900FFFF19AD
:00000001FF
<file_name>.HXH
:0A0000000000000000000000000000F6
:10001900000000000000000010101010102020202CA
:1000290002020303030303040404050607070883
:0300390008080AAA
:00000001FF
```

A.5 32-Bit Hex Format (.HEX)

The extended 32-bit address hex format is similar to the hex 8 format described above, except that the extended linear address record is output also to establish the upper 16 bits of the data address. This is mainly used for 16-bit core devices since their addressable program memory exceeds 32 k words.

Each data record begins with a 9 character prefix and ends with a 2 character checksum. Each record has the following format:

:BBAAAATTHHHH . . . HHHCC

where:

BB – is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line.

AAAA – is a four digit hexadecimal address representing the starting address of the data record.

TT – is a two digit record type:

- 00 – Data record
- 01 – End of File record
- 02 – Segment address record
- 04 – Linear address record

HH – is a two digit hexadecimal data byte, presented in low byte, high byte combinations.

CC – is a two digit hexadecimal checksum that is the two's complement of the sum of all preceding bytes in the record.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Appendix B. Quick Reference

B.1 Introduction

This appendix lists abbreviated information on MPASM and PICmicro instruction sets for use in developing applications using MPASM, MPLINK and MPLIB.

B.2 Highlights

Topics covered in this appendix:

- MPASM Quick Reference
- Key to PICmicro Instruction Sets
- 12-Bit Core Instruction Set
- 14-Bit Core Instruction Set
- 16-Bit Core Instruction Set
- Key to Enhanced 16-Bit Core Instruction Set
- Enhanced 16-Bit Core Instruction Set
- Hexadecimal to Decimal Conversion
- ASCII Character Set

B.3 MPASM Quick Reference

The following Quick Reference Guide gives all the instructions, directives, and command line options for the Microchip MPASM Assembler.

Table B.1: MPASM Directive Language Summary

Directive	Description	Syntax
CONTROL DIRECTIVES		
CONSTANT	Declare Symbol Constant	constant <label> [= <expr>, ..., <label> [= <expr>]]
#DEFINE	Define a Text Substitution Label	#define <name> [[(<arg>, ..., <arg>)]<value>]
END	End Program Block	end
EQU	Define an Assembly Constant	<label> equ <expr>
ERROR	Issue an Error Message	error "<text_string>"
ERRORLEVEL	Set Messge Level	errorlevel 0 1 2 <+-><msg>
#INCLUDE	Include Additional Source File	include <<include_file>> include "<include_file>"

MPASM User's Guide with MPLINK and MPLIB

Table B.1: MPASM Directive Language Summary (Continued)

Directive	Description	Syntax
LIST	Listing Options	list [<option>[, ..., <option>]]
MESSG	Create User Defined Message	messg "<message_text>"
NOLIST	Turn off Listing Output	nolist
ORG	Set Program Origin	<label> org <expr>
PAGE	Insert Listing Page Eject	page
PROCESSOR	Set Processor Type	processor <processor_type>
RADIX	Specify Default Radix	radix <default_radix>
SET	Define an Assembler Variable	<label> set <expr>
SPACE	Insert Blank Listing Lines	space [<expr>]
SUBTITLE	Specify Program Subtitle	subtitl "<sub_text>"
TITLE	Specify Program Title	title "<title_text>"
#UNDEFINE	Delete a Substitution Label	#undefine <label>
VARIABLE	Declare Symbol Variable	variable <label> [= <expr>, ..., <label> [= <expr>]]
CONDITIONAL ASSEMBLY		
ELSE	Begin Alternative Assembly Block to IF	else
ENDIF	End Conditional Assembly Block	endif
ENDW	End a While Loop	endw
IF	Begin Conditionally Assembled Code Block	if <expr>
IFDEF	Execute If Symbol is Defined	ifdef <label>
IFNDEF	Execute If Symbol is Not Defined	ifndef <label>
WHILE	Perform Loop While Condition is True	while <expr>
DATA		
_ _BADRAM	Specify invalid RAM locations	_ _badram <expr>
CBLOCK	Define a Block of Constants	cblock [<expr>]
_ _CONFIG	Set configuration fuses	_ _config <expr> OR _ _config <addr>, <expr>
DA	Store Strings in Program Memory	[<label>] da <expr> [, <expr2>, ..., <exprn>]
DATA	Create Numeric and Text Data	data <expr> [, <expr>, ..., <expr>] data "<text_string>" [, "<text_string>", ...]
DB	Declare Data of One Byte	db <expr> [, <expr>, ..., <expr>]
DE	Declare EEPROM Data	de <expr> [, <expr>, ..., <expr>]
DT	Define Table	dt <expr> [, <expr>, ..., <expr>]
DW	Declare Data of One Word	dw <expr> [, <expr>, ..., <expr>]
ENDC	End an Automatic Constant Block	endc
FILL	Specify Memory Fill Value	fill <expr>, <count>
_ _ IDLOCS	Set ID locations	_ _idlocs <expr>

Table B.1: MPASM Directive Language Summary (Continued)

Directive	Description	Syntax
<code>_ _MAXRAM</code>	Specify maximum RAM address	<code>_ _maxram <expr></code>
<code>RES</code>	Reserve Memory	<code>res <mem_units></code>
MACROS		
<code>ENDM</code>	End a Macro Definition	<code>endm</code>
<code>EXITM</code>	Exit from a Macro	<code>exitm</code>
<code>EXPAND</code>	Expand Macro Listing	<code>expand</code>
<code>LOCAL</code>	Declare Local Macro Variable	<code>local <label> [, <label>]</code>
<code>MACRO</code>	Declare Macro Definition	<code><label> macro [<arg>, ..., <arg>]</code>
<code>NOEXPAND</code>	Turn off Macro Expansion	<code>noexpand</code>
OBJECT FILE DIRECTIVES		
<code>BANKISEL</code>	Generate RAM bank selecting code for indirect addressing	<code>bankisel <label></code>
<code>BANKSEL</code>	Generate RAM bank selecting code	<code>banksel <label></code>
<code>CODE</code>	Begins executable code section	<code>[<name>] code [<address>]</code>
<code>EXTERN</code>	Declares an external label	<code>extern <label> [, <label>]</code>
<code>GLOBAL</code>	Exports a defined label	<code>extern <label> [.<label>]</code>
<code>IDATA</code>	Begins initialized data section	<code>[<name>] idata [<address>]</code>
<code>PAGESEL</code>	Generate ROM page selecting code	<code>pagesel <label></code>
<code>UDATA</code>	Begins uninitialized data section	<code>[<name>] udata [<address>]</code>
<code>UDATA_ACS</code>	Begins access uninitialized data section	<code>[<name>] udata_acs [<address>]</code>
<code>UDATA_OVR</code>	Begins overlayed uninitialized data section	<code>[<name>] udata_ovr [<address>]</code>
<code>UDATA_SHR</code>	Begins shared uninitialized data section	<code>[<name>] udata_shr [<address>]</code>

Table B.2: MPASM Command Line Options

Option	Default	Description
<code>?</code>	N/A	Displays the MPASM Help Panel
<code>a</code>	INHX8M	Generate absolute .COD and hex output directly from assembler: /a<hex-format> where <hex-format> is one of [INHX8M INHX8S INHX32]
<code>c</code>	On	Enables/Disables case sensitivity
<code>d</code>	N/A	Define a text string substitution: /d<label>[=<value>]
<code>e</code>	On	Enable/Disable/Set Path for error file /e Enable /e + Enable /e - Disable /e <path>error.file Enables/sets path
<code>h</code>	N/A	Displays the MPASM Help Panel

MPASM User's Guide with MPLINK and MPLIB

Table B.2: MPASM Command Line Options (Continued)

Option	Default	Description
l	On	Enable/Disable/ Set Path for list file /l Enable /l + Enable /l - Disable /l <path>list.file Enables/sets path
m	On	Enable/Disable macro expansion
o	Off	Enable/Disable/Set Path for object file /o Enable /o + Enable /o - Disable /o <path>object.file Enables/sets path
p	None	Set the processor type: /p<processor_type> where <processor_type> is a PICmicro device. For example, PIC16C54.
q	Off	Enable/Disable quiet mode (suppress screen output)
r	Hex	Defines default radix: /r<radix> where <radix> is one of [HEX DEC OCT]
t	8	List file tab size: /t<size>
w	0	Set message level: /w<value> where <level> is one of [0 1 2] 0 – all messages 1 – errors and warnings 2 – errors only
x	Off	Enable/Disable/ Set Path for cross reference file /x Enable /x + Enable /x - Disable /x <path>xref.file Enables/sets path

Table B.3: Radix Types Supported

Radix	Syntax	Example
Decimal	D'<digits>'	D'100'
Hexadecimal (default)	H'<hex_digits>' 0x<hex_digits>	H'9f' 0x9f
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001'
Character (ASCII)	'<character>' A'<Character>'	'C' A'C'

Table B.4: MPASM Arithmetic Operators

Operator	Description	Example
\$	Current/Return program counter	goto \$ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a - b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags
high	Return high byte	movlw high CTR_Table
low	Return low byte	movlw low CTR_Table
upper	Return upper byte	movlw upper CTR_Table
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	val = flags << 1
>>	Right shift	val = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
	Bitwise inclusive OR	flags = flags ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
	Logical OR	if (len == 512) (b == c)
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
=	Inclusive OR, set equal	flags = ERROR_FLAG

MPASM User's Guide with MPLINK and MPLIB

Table B.4: MPASM Arithmetic Operators

Operator	Description	Example
<code>^=</code>	Exclusive OR, set equal	<code>flags ^= ERROR_FLAG</code>
<code>++</code>	Increment	<code>i ++</code>
<code>--</code>	Decrement	<code>i --</code>

B.4 Key to PICmicro Family Instruction Sets

Field	Description
b	Bit address within an 8 bit file register
d	Destination select; d = 0 Store result in W (f0A). d = 1 Store result in file register f. Default is d = 1.
f	Register file address (0x00 to 0xFF)
k	Literal field, constant data or label
W	Working register (accumulator)
x	Don't care location
i	Table pointer control; i = 0 Do not change. i = 1 Increment after instruction execution.
p	Peripheral register file address (0x00 to 0x1f)
t	Table byte select; t = 0 Perform operation on lower byte. t = 1 Perform operation on upper byte.
PH:PL	Multiplication results registers

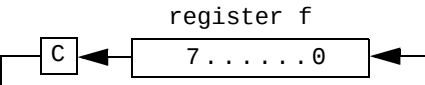
B.5 12-Bit Core Instruction Set

Microchip's base-line 8-bit microcontroller family uses a 12-bit wide instruction set. All instructions execute in a single instruction cycle unless otherwise noted. Any unused opcode is executed as a NOP. The instruction set is grouped into the following categories:

Table B.5: 12-Bit Core Literal and Control Operations

Hex	Mnemonic		Description	Function
Ekk	ANDLW	k	AND literal and W	$k \text{ .AND. } W \rightarrow W$
9kk	CALL	k	Call subroutine	$PC + 1 \rightarrow TOS, k \rightarrow PC$
004	CLRWDT		Clear watchdog timer	$0 \rightarrow WDT$ (and Prescaler if assigned)
Akk	GOTO	k	Goto address (k is nine bits)	$k \rightarrow PC(9 \text{ bits})$
Dkk	IORLW	k	Incl. OR literal and W	$k \text{ .OR. } W \rightarrow W$
Ckk	MOVLW	k	Move Literal to W	$k \rightarrow W$
002	OPTION		Load OPTION Register	$W \rightarrow \text{OPTION Register}$
8kk	RETLW	k	Return with literal in W	$k \rightarrow W, TOS \rightarrow PC$
003	SLEEP		Go into Standby Mode	$0 \rightarrow WDT, \text{ stop oscillator}$
00f	TRIS	f	Tristate port f	$W \rightarrow \text{I/O control reg } f$
Fkk	XORLW	k	Exclusive OR literal and W	$k \text{ .XOR. } W \rightarrow W$

Table B.6: 12-Bit Core Byte Oriented File Register Operations

Hex	Mnemonic		Description	Function
1Cf	ADDWF	f, d	Add W and f	$W + f \rightarrow d$
14f	ANDWF	f, d	AND W and f	$W \text{ .AND. } f \rightarrow d$
06f	CLRF	f	Clear f	$0 \rightarrow f$
040	CLRW		Clear W	$0 \rightarrow W$
24f	COMF	f, d	Complement f	$\text{.NOT. } f \rightarrow d$
0Cf	DECF	f, d	Decrement f	$f - 1 \rightarrow d$
2Cf	DECFSZ	f, d	Decrement f, skip if zero	$f - 1 \rightarrow d, \text{ skip if zero}$
28f	INCF	f, d	Increment f	$f + 1 \rightarrow d$
3Cf	INCFSZ	f, d	Increment f, skip if zero	$f + 1 \rightarrow d, \text{ skip if zero}$
10f	IORWF	f, d	Inclusive OR W and f	$W \text{ .OR. } f \rightarrow d$
20f	MOVF	f, d	Move f	$f \rightarrow d$
02f	MOVWF	f	Move W to f	$W \rightarrow f$
000	NOP		No operation	
34f	RLF	f, d	Rotate left f	

MPASM User's Guide with MPLINK and MPLIB

Table B.6: 12-Bit Core Byte Oriented File Register Operations (Continued)

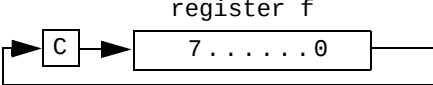
Hex	Mnemonic	Description	Function
30f	RRF f, d	Rotate right f	
08f	SUBWF f, d	Subtract W from f	$f - W \rightarrow d$
38f	SWAPF f, d	Swap halves f	$f(0:3) \leftrightarrow f(4:7) \rightarrow d$
18f	XORWF f, d	Exclusive OR W and f	$W .XOR. f \rightarrow d$

Table B.7: 12-Bit Core Bit Oriented File Register Operations

Hex	Mnemonic	Description	Function
4bf	BCF f, b	Bit clear f	$0 \rightarrow f(b)$
5bf	BSF f, b	Bit set f	$1 \rightarrow f(b)$
6bf	BTFSC f, b	Bit test, skip if clear	skip if $f(b) = 0$
7bf	BTFSS f, b	Bit test, skip if set	skip if $f(b) = 1$

B.6 14-Bit Core Instruction Set

Microchip's mid-range 8-bit microcontroller family uses a 14-bit wide instruction set. This instruction set consists of 36 instructions, each a single 14-bit wide word. Most instructions operate on a file register, f, and the working register, W (accumulator). The result can be directed either to the file register or the W register or to both in the case of some instructions. A few instructions operate solely on a file register (BSF for example). The instruction set is grouped into the following categories:

Table B.8: 14-Bit Core Literal and Control Operations

Hex	Mnemonic		Description	Function
3Ekk	ADDLW	k	Add literal to W	$k + W \rightarrow W$
39kk	ANDLW	k	AND literal and W	$k .AND. W \rightarrow W$
2kkk	CALL	k	Call subroutine	$PC + 1 \rightarrow TOS, k \rightarrow PC$
0064	CLRWDT	T	Clear watchdog timer	$0 \rightarrow WDT$ (and Prescaler if assigned)
2kkk	GOTO	k	Goto address (k is nine bits)	$k \rightarrow PC(9 \text{ bits})$
38kk	IORLW	k	Incl. OR literal and W	$k .OR. W \rightarrow W$
30kk	MOVLW	k	Move Literal to W	$k \rightarrow W$
0062	OPTION		Load OPTION register	$W \rightarrow OPTION \text{ Register}$
0009	RETFIE		Return from Interrupt	$TOS \rightarrow PC, 1 \rightarrow GIE$
34kk	RETLW	k	Return with literal in W	$k \rightarrow W, TOS \rightarrow PC$
0008	RETURN		Return from subroutine	$TOS \rightarrow PC$
0063	SLEEP		Go into Standby Mode	$0 \rightarrow WDT$, stop oscillator
3Ckk	SUBLW	k	Subtract W from literal	$k - W \rightarrow W$
006f	TRIS	f	Tristate port f	$W \rightarrow I/O \text{ control reg } f$
3Akk	XORLW	k	Exclusive OR literal and W	$k .XOR. W \rightarrow W$

Table B.9: 14-Bit Core Byte Oriented File Register Operations

Hex	Mnemonic		Description	Function
07ff	ADDWF	f, d	Add W and f	$W + f \rightarrow d$
05ff	ANDWF	f, d	AND W and f	$W .AND. f \rightarrow d$
018f	CLRF	f	Clear f	$0 \rightarrow f$
0100	CLRW		Clear W	$0 \rightarrow W$
09ff	COMF	f, d	Complement f	$.NOT. f \rightarrow d$
03ff	DECF	f, d	Decrement f	$f - 1 \rightarrow d$
0Bff	DECFSZ	f, d	Decrement f, skip if zero	$f - 1 \rightarrow d$, skip if 0
0Aff	INCF	f, d	Increment f	$f + 1 \rightarrow d$
0Fff	INCFSZ	f, d	Increment f, skip if zero	$f + 1 \rightarrow d$, skip if 0
04ff	IORWF	f, d	Inclusive OR W and f	$W .OR. f \rightarrow d$
08ff	MOVF	f, d	Move f	$f \rightarrow d$

MPASM User's Guide with MPLINK and MPLIB

Table B.9: 14-Bit Core Byte Oriented File Register Operations (Continued)

Hex	Mnemonic	Description	Function
008f	MOVWF f	Move W to f	$W \rightarrow f$
0000	NOP	No operation	
0Dff	RLF f, d	Rotate left f	
0Cff	RRF f, d	Rotate right f	
02ff	SUBWF f, d	Subtract W from f	$f - W \rightarrow d$
0Eff	SWAPF f, d	Swap halves f	$f(0:3) \leftrightarrow f(4:7) \rightarrow d$
06ff	XORWF f, d	Exclusive OR W and f	$W .XOR. f \rightarrow d$

Table B.10: 14-Bit Core Bit Oriented File Register Operations

Hex	Mnemonic	Description	Function
1bff	BCF f, b	Bit clear f	$0 \rightarrow f(b)$
1bff	BSF f, b	Bit set f	$1 \rightarrow f(b)$
1bff	BTFSC f, b	Bit test, skip if clear	skip if $f(b) = 0$
1bff	BTFSS f, b	Bit test, skip if set	skip if $f(b) = 1$

Table B.11: 12-Bit/14-Bit Core Special Instruction Mnemonics

Mnemonic	Description	Equivalent Operation(s)	Status
ADDCF f, d	Add Carry to File	BTFSC 3,0 INCF f,d	Z
ADDDCF f, d	Add Digit Carry to File	BTFSC 3,1 INCF f,d	Z
B k	Branch	GOTO k	-
BC k	Branch on Carry	BTFSC 3,0 GOTO k	-
BDC k	Branch on Digit Carry	BTFSC 3,1 GOTO k	-
BNC k	Branch on No Carry	BTFSS 3,0 GOTO k	-
BNDC k	Branch on No Digit Carry	BTFSS 3,1 GOTO k	-
BNZ k	Branch on No Zero	BTFSS 3,2 GOTO k	-
BZ k	Branch on Zero	BTFSC 3,2 GOTO k	-

Table B.11: 12-Bit/14-Bit Core Special Instruction Mnemonics (Continued)

Mnemonic	Description	Equivalent Operation(s)	Status
CLRC	Clear Carry	BCF 3,0	-
CLRDC	Clear Digit Carry	BCF 3,1	-
CLRZ	Clear Zero	BCF 3,2	-
LCALL k			
LGOTO k			
MOVFW f	Move File to W	MOVF f,0	Z
NEGF f, d	Negate File	COMF f,1 INCF f,d	Z
SETC	Set Carry	BSF 3,0	-
SETDC	Set Digit Carry	BSF 3,1	-
SETZ	Set Zero	BSF 3,2	-
SKPC	Skip on Carry	BTFSS 3,0	-
SKPDC	Skip on Digit Carry	BTFSS 3,1	-
SKPNC	Skip on No Carry	BTFSC 3,0	-
SKPNDC	Skip on No Digit Carry	BTFSC 3,1	-
SKPNZ	Skip on Non Zero	BTFSC 3,2	-
SKPZ	Skip on Zero	BTFSS 3,2	-
SUBCF f, d	Subtract Carry from File	BTFSC 3,0 DECF f,d	Z
SUBDCF f, d	Subtract Digit Carry from File	BTFSC 3,1 DECF f,d	Z
TSTF f	Test File	MOVF f,1	Z

MPASM User's Guide with MPLINK and MPLIB

B.7 16-Bit Core Instruction Set

Microchip's high-performance 8-bit microcontroller family uses a 16-bit wide instruction set. This instruction set consists of 55 instructions, each a single 16-bit wide word. Most instructions operate on a file register, f, and the working register, W (accumulator). The result can be directed either to the file register or the W register or to both in the case of some instructions. Some devices in this family also include hardware multiply instructions. A few instructions operate solely on a file register (BSF for example).

Table B.12: 16-Bit Core Data Movement Instructions

Hex	Mnemonic	Description	Function
6pff	MOVFP f, p	Move f to p	$f \rightarrow p$
b8kk	MOVLB k	Move literal to BSR	$k \rightarrow \text{BSR} (3:0)$
bakx	MOVLP k	Move literal to RAM page select	$k \rightarrow \text{BSR} (7:4)$
4pff	MOVFP p, f	Move p to f	$p \rightarrow W$
01ff	MOVWF f	Move W to F	$W \rightarrow f$
a8ff	TABLRD t, i, f	Read data from table latch into file f, then update table latch with 16-bit contents of memory location addressed by table pointer	$\text{TBLATH} \rightarrow f$ if $t=1$, $\text{TBLATL} \rightarrow f$ if $t=0$; $\text{ProgMem}(\text{TBLPTR}) \rightarrow \text{TBLAT}$; $\text{TBLPTR} + 1 \rightarrow \text{TBLPTR}$ if $i=1$
acff	TABLWT t, i, f	Write data from file f to table latch and then write 16-bit table latch to program memory location addressed by table pointer	$f \rightarrow \text{TBLATH}$ if $t = 1$, $f \rightarrow \text{TBLATL}$ if $t = 0$; $\text{TBLAT} \rightarrow \text{ProgMem}(\text{TBLPTR})$; $\text{TBLPTR} + 1 \rightarrow \text{TBLPTR}$ if $i=1$
a0ff	TLRD t, f	Read data from table latch into file f (table latch unchanged)	$\text{TBLATH} \rightarrow f$ if $t = 1$ $\text{TBLATL} \rightarrow f$ if $t = 0$
a4ff	TLWT t, f	Write data from file f into table latch	$f \rightarrow \text{TBLATH}$ if $t = 1$ $f \rightarrow \text{TBLATL}$ if $t = 0$

Table B.13: 16-Bit Core Arithmetic and Logical Instruction

Hex	Mnemonic	Description	Function
b1kk	ADDLW k	Add literal to W	$(W + k) \rightarrow W$
0eff	ADDWF f, d	Add W to F	$(W + f) \rightarrow d$
10ff	ADDWFC f, d	Add W and Carry to f	$(W + f + C) \rightarrow d$
b5kk	ANDLW k	AND Literal and W	$(W .\text{AND.} k) \rightarrow W$
0aff	ANDWF f, d	AND W with f	$(W .\text{AND.} f) \rightarrow d$
28ff	CLRF f, d	Clear f and Clear d	$0x00 \rightarrow f, 0x00 \rightarrow d$
12ff	COMF f, d	Complement f	$.\text{NOT.} f \rightarrow d$
2eff	DAW f, d	Dec. adjust W, store in f,d	$W \text{ adjusted} \rightarrow f \text{ and } d$
06ff	DECF f, d	Decrement f	$(f - 1) \rightarrow f \text{ and } d$
14ff	INCF f, d	Increment f	$(f + 1) \rightarrow f \text{ and } d$
b3kk	IORLW k	Inclusive OR literal with W	$(W .\text{OR.} k) \rightarrow W$

Table B.13: 16-Bit Core Arithmetic and Logical Instruction (Continued)

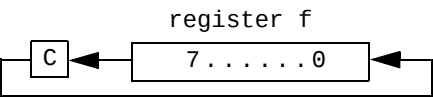
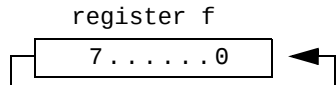
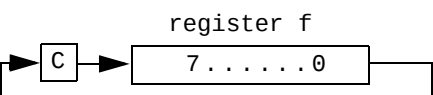
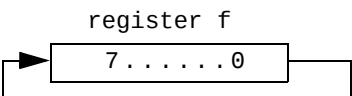
Hex	Mnemonic	Description	Function
08ff	IORWF f, d	Inclusive or W with f	$(W \text{ .OR. } f) \rightarrow d$
b0kk	MOVLW k	Move literal to W	$k \rightarrow W$
bckk	MULLW k	Multiply literal and W	$(k \times W) \rightarrow \text{PH:PL}$
34ff	MULWF f	Multiply W and f	$(W \times f) \rightarrow \text{PH:PL}$
2cff	NEGW f, d	Negate W, store in f and d	$(W + 1) \rightarrow f, (W + 1) \rightarrow d$
1aff	RLCF f, d	Rotate left through carry	
22ff	RLNCF f, d	Rotate left (no carry)	
18ff	RRCF f, d	Rotate right through carry	
20ff	RRNCF f, d	Rotate right (no carry)	
2aff	SETF f, d	Set f and Set d	$0\text{xff} \rightarrow f, 0\text{xff} \rightarrow d$
b2kk	SUBLW k	Subtract W from literal	$(k - W) \rightarrow W$
04ff	SUBWF f, d	Subtract W from f	$(f - W) \rightarrow d$
02ff	SUBWFB f, d	Subtract from f with borrow	$(f - W - c) \rightarrow d$
1cff	SWAPF f, d	Swap f	$f(0:3) \rightarrow d(4:7),$ $f(4:7) \rightarrow d(0:3)$
b4kk	XORLW k	Exclusive OR literal with W	$(W \text{ .XOR. } k) \rightarrow W$
0cff	XORWF f, d	Exclusive OR W with f	$(W \text{ .XOR. } f) \rightarrow d$

Table B.14: 16-Bit Core Bit Handling Instructions

Hex	Mnemonic	Description	Function
8bff	BCF f, b	Bit clear f	$0 \rightarrow f(b)$
8bff	BSF f, b	Bit set f	$1 \rightarrow f(b)$
9bff	BTFSC f, b	Bit test, skip if clear	skip if $f(b) = 0$
9bff	BTFSS f, b	Bit test, skip if set	skip if $f(b) = 1$
3bff	BTG f, b	Bit toggle f	$\text{.NOT. } f(b) \rightarrow f(b)$

MPASM User's Guide with MPLINK and MPLIB

Table B.15: 16-Bit Core Program Control Instructions

Hex	Mnemonic		Description	Function
ekkk	CALL	k	Subroutine call (within 8k page)	PC+1 → TOS, k → PC(12:0), k(12:8) → PCLATH(4:0), PC(15:13) → PCLATH(7:5)
31ff	CPFSEQ	f	Compare f/w, skip if f = w	f-W, skip if f = w
32ff	CPFSGT	f	Compare f/w, skip if f > w	f-W, skip if f > w
30ff	CPFSLT	f	Compare f/w, skip if f < w	f-W, skip if f < w
16ff	DECFSZ	f, d	Decrement f, skip if 0	(f-1) → d, skip if 0
26ff	DCFSNZ	f, d	Decrement f, skip if not 0	(f-1) → d, skip if not 0
ckkk	GOTO	k	Unconditional branch (within 8k)	k → PC(12:0) k(12:8) → PCLATH(4:0), PC(15:13) → PCLATH(7:5)
1eff	INCFSZ	f, d	Increment f, skip if zero	(f+1) → d, skip if 0
24ff	INFSNZ	f, d	Increment f, skip if not zero	(f+1) → d, skip if not 0
b7kk	LCALL	k	Long Call (within 64k)	(PC+1) → TOS; k → PCL, (PCLATH) → PCH
0005	RETFIE		Return from interrupt, enable interrupt	(PCLATH) → PCH; k → PCL 0 → GLINTD
b6kk	RETLW	k	Return with literal in W	k → W, TOS → PC, (PCLATH unchanged)
0002	RETURN		Return from subroutine	TOS → PC (PCLATH unchanged)
33ff	TSTFSZ	f	Test f, skip if zero	skip if f = 0

Table B.16: 16-Bit Core Special Control Instructions

Hex	Mnemonic	Description	Function
0004	CLRWT	Clear watchdog timer	0 → WDT, 0 → WDT prescaler, 1 → PD, 1 → T0
0003	SLEEP	Enter Sleep Mode	Stop oscillator, power down, 0 → WDT, 0 → WDT Prescaler 1 → PD, 1 → T0

B.8 Key to Enhanced 16-Bit Core Instruction Set

Field	Description
File Addresses	
f	8-bit file register address
fs	12-bit source file register address
fd	12-bit destination file register address
dest	W register if d = 0; file register if d = 1
r	0, 1, or 2 for FSR number
Literals	
kk	8-bit literal value
kb	4-bit literal value
kc	bits 8-11 of 12-bit literal value
kd	bits 0-7 of 12-bit literal value
Offsets, Addresses	
nn	8-bit relative offset (signed, 2's complement)
nd	11-bit relative offset (signed, 2's complement)
ml	bits 0-7 of 20-bit program memory address
mm	bits 8-19 of 20-bit program memory address
xx	any 12-bit value
Bits	
b	bits 9-11; bit address
d	bit 9; 0=W destination; 1=f destination
a	bit 8; 0=common block; 1=BSR selects bank
s	bit 0 (bit 8 for CALL); 0=no update; 1(fast)=update/save W, STATUS, BSR

MPASM User's Guide with MPLINK and MPLIB

B.9 Enhanced 16-Bit Core Instruction Set

Microchip's new high-performance 8-bit microcontroller family uses a 16-bit wide instruction set. This instruction set consists of 76 instructions, each a single 16-bit wide word (2 bytes). Most instructions operate on a file register, f, and the working register, W (accumulator). The result can be directed either to the file register or the W register or to both in the case of some instructions. A few instructions operate solely on a file register (BSF for example)

Table B.17: Enhanced 16-Bit Core Literal Operations

Hex	Mnemonic		Description	Function
0Fkk	ADDLW	kk	ADD literal to WREG	W+kk → W
0Bkk	ANDLW	kk	AND literal with WREG	W .AND. kk → W
0004	CLRWDT		Clear Watchdog Timer	0 → WDT, 0 → WDT postscaler, 1 → TO, 1 → PD
0007	DAW		Decimal Adjust WREG	if W<3:0> >9 or DC=1, W<3:0>+6→W<3:0>; else W<3:0> → W<3:0>; if W<7:4> >9 or C=1, W<7:4>+6→W<7:4>; else W<7:4> → W<7:4>;
09kk	IORLW	kk	Inclusive OR literal with WREG	W .OR. kk → W
EFkc F0kd	LFSR	r,kd:kc	Load 12-bit Literal to FSR (second word)	kd:kc → FSRr
01kb	MOVLB	kb	Move literal to low nibble in BSR	kb → BSR
0Ekk	MOVLW	kk	Move literal to WREG	kk → W
0Dkk	MULLW	kk	Multiply literal with WREG	W * kk → PRODH:PRODL
08kk	SUBLW	kk	Subtract W from literal	kk-W → W
0Akk	XORLW	kk	Exclusive OR literal with WREG	W .XOR. kk → W

Table B.18: Enhanced 16-Bit Core Memory Operations

Hex	Mnemonic		Description	Function
0008	TBLRD *		Table Read (no change to TBLPTR)	Prog Mem (TBLPTR) → TABLAT
0009	TBLRD *+		Table Read (post-increment TBLPTR)	Prog Mem (TBLPTR) → TABLAT TBLPTR +1 → TBLPTR
000A	TBLRD *-		Table Read (post-decrement TBLPTR)	Prog Mem (TBLPTR) → TABLAT TBLPTR -1 → TBLPTR
000B	TBLRD +*		Table Read (pre-increment TBLPTR)	TBLPTR +1 → TBLPTR Prog Mem (TBLPTR) → TABLAT
000C	TBLWT *		Table Write (no change to TBLPTR)	TABLAT → Prog Mem(TBLPTR)
000D	TBLWT *+		Table Write (post-increment TBLPTR)	TABLAT → Prog Mem(TBLPTR) TBLPTR +1 → TBLPTR

Quick Reference

Table B.18: Enhanced 16-Bit Core Memory Operations (Continued)

Hex	Mnemonic		Description	Function
000E	TBLWT *-		Table Write (post-decrement TBLPTR)	TABLAT → Prog Mem(TBLPTR) TBLPTR -1 → TBLPTR
000F	TBLWT +*		Table Write (pre-increment TBLPTR)	TBLPTR +1 → TBLPTR TABLAT → Prog Mem(TBLPTR)

Table B.19: Enhanced 16-Bit Core Control Operations

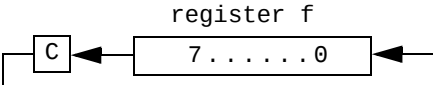
Hex	Mnemonic		Description	Function
E2nn	BC	nn	Relative Branch if Carry	if C=1, PC+2+2*nn→PC, else PC+2→PC
E6nn	BN	nn	Relative Branch if Negative	if N=1, PC+2+2*nn→PC, else PC+2→PC
E3nn	BNC	nn	Relative Branch if Not Carry	if C=0, PC+2+2*nn→PC, else PC+2→PC
E7nn	BNN	nn	Relative Branch if Not Negative	if N=0, PC+2+2*nn→PC, else PC+2→PC
E5nn	BNOV	nn	Relative Branch if Not Overflow	if OV=0, PC+2+2*nn→PC, else PC+2→PC
E1nn	BNZ	nn	Relative Branch if Not Zero	if Z=0, PC+2+2*nn→PC, else PC+2→PC
E4nn	BOV	nn	Relative Branch if Overflow	if OV=1, PC+2+2*nn→PC, else PC+2→PC
E0nd	BRA	nd	Unconditional relative branch	PC+2+2*nd→PC
E0nn	BZ	nn	Relative Branch if Zero	if Z=1, PC+2+2*nn→PC, else PC+2→PC
ECml Fmm	CALL	mm:ml,s	Absolute Subroutine Call (second word)	PC+4 → TOS, mm:ml → PC<20:1>, if s=1, W → WS, STATUS → STATUSS, BSR → BSRS
EFml Fmm	GOTO	mm:ml	Absolute Branch (second word)	mm:ml → PC<20:1>
0000	NOP		No Operation	No operation
0006	POP		Pop Top of return stack	TOS-1 → TOS
0005	PUSH		Push Top of return stack	PC +2→ TOS
D8nd	RCALL	nd	Relative Subroutine Call	PC+2 → TOS, PC+2+2*nd→PC
00FF	RESET		Generate a Reset (same as MCR reset)	same as MCLR reset
0010	RETFIE	s	Return from interrupt (and enable interrupts)	TOS → PC, 1 → GIE/GIEH or PEIE/GIEL, if s=1, WS → W, STATUSS → STATUS, BSRS → BSR, PCLATU/PCLATH unchngd.
0Ckk	RETLW	kk	Return from subroutine, literal in W	kk → W,
0012	RETURN	s	Return from subroutine	TOS → PC, if s=1, WS → W, STATUSS → STATUS, BSRS → BSR, PCLATU/PCLATH are unchanged
0003	SLEEP		Enter SLEEP Mode	0 → WDT, 0 → WDT postscaler, 1 → TO, 0 → PD

MPASM User's Guide with MPLINK and MPLIB

Table B.20: Enhanced 16-Bit Core Bit Operations

Hex	Mnemonic		Description	Function
9bf	BCF	f,b,a	Bit Clear f	$0 \rightarrow f$
8bf	BSF	f,b,a	Bit Set f	$1 \rightarrow f$
Bbf	BTFSC	f,b,a	Bit test f, skip if clear	if $f=0$, $PC+4 \rightarrow PC$, else $PC+2 \rightarrow PC$
Abf	BTFSS	f,b,a	Bit test f, skip if set	if $f=1$, $PC+4 \rightarrow PC$, else $PC+2 \rightarrow PC$
7bf	BTG	f,b,a	Bit Toggle f	$f \rightarrow f$

Table B.21: Enhanced 16-Bit Core File Register Operations

Hex	Mnemonic		Description	Function
24f	ADDWF	f,d,a	ADD WREG to f	$W+f \rightarrow \text{dest}$
20f	ADDWFC	f,d,a	ADD WREG and Carry bit to f	$W+f+C \rightarrow \text{dest}$
14f	ANDWF	f,d,a	AND WREG with f	$W \text{ .AND. } f \rightarrow \text{dest}$
6Af	CLRF	f,a	Clear f	$0 \rightarrow f$
1Cf	COMF	f,d,a	Complement f	$f \rightarrow \text{dest}$
62f	CPFSEQ	f,a	Compare f with WREG, skip if f=WREG	f=W, if $f=W$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
64f	CPFSGT	f,a	Compare f with WREG, skip if f > WREG	f>W, if $f > W$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
60f	CPFSLT	f,a	Compare f with WREG, skip if f < WREG	f<W, if $f < W$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
04f	DECF	f,d,a	Decrement f	$f-1 \rightarrow \text{dest}$
2Cf	DECFSZ	f,d,a	Decrement f, skip if 0	$f-1 \rightarrow \text{dest}$, if $\text{dest}=0$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
4Cf	DCFSNZ	f,d,a	Decrement f, skip if not 0	$f-1 \rightarrow \text{dest}$, if $\text{dest} \neq 0$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
28f	INCF	f,d,a	Increment f	$f+1 \rightarrow \text{dest}$
3Cf	INCFSZ	f,d,a	Increment f, skip if 0	$f+1 \rightarrow \text{dest}$, if $\text{dest}=0$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
48f	INFSNZ	f,d,a	Increment f, skip if not 0	$f+1 \rightarrow \text{dest}$, if $\text{dest} \neq 0$, $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
10f	IORWF	f,d,a	Inclusive OR WREG with f	$W \text{ .OR. } f \rightarrow \text{dest}$
50f	MOVF	f,d,a	Move f	$f \rightarrow \text{dest}$
Cfs Ffd	MOVFF	fs,fd	Move fs to fd (second word)	$fs \rightarrow fd$
6Ef	MOVWF	f,a	Move WREG to f	$W \rightarrow f$
02f	MULWF	f,a	Multiply WREG with f	$W * f \rightarrow \text{PRODH:PRODL}$
6Cf	NEGF	f,a	Negate f	$f+1 \rightarrow f$
34f	RLCF	f,d,a	Rotate left f through Carry	

Quick Reference

Table B.21: Enhanced 16-Bit Core File Register Operations (Continued)

Hex	Mnemonic		Description	Function
44f	RLNCF	f,d,a	Rotate left f (no carry)	
30f	RRCF	f,d,a	Rotate right f through Carry	
40f	RRNCF	f,d,a	Rotate right f (no carry)	
48f	SETF	f,a	Set f	0xFF → f
54f	SUBFWB	f,d,a	Subtract f from WREG with Borrow	W - f - C → dest
5Cf	SUBWF	f,d,a	Subtract WREG from f	f - W → dest
58f	SUBWFB	f,d,a	Subtract WREG from f with Borrow	f - W - C → dest
38f	SWAPF	f,d,a	Swap nibbles of f	f<3:0> → dest<7:4>, f<7:4> → dest<3:0>
66f	TSTFSZ	f,a	Test f, skip if 0	PC+4 → PC, if f=0, else PC+2 → PC
18f	XORWF	f,d,a	Exclusive OR WREG with f	W .XOR. f → dest

MPASM User's Guide with MPLINK and MPLIB

B.10 Hexadecimal to Decimal Conversion

Byte				Byte			
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

Using This Table: For each Hex digit, find the associated decimal value. Add the numbers together. For example, Hex A38F converts to 41871 as follows:

Hex 1000's Digit		Hex 100's Digit		Hex 10's Digit		Hex 1's Digit		Result
40960	+	768	+	128	+	15	=	41871 Decimal

B.11 ASCII Character Set

Least Significant Character	Most Significant Character								
	Hex	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	'	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	—	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Appendix C. MPASM Errors/Warnings/Messages

C.1 Introduction

The following messages are produced by MPASM. These messages always appear in the listing file directly above each line in which the error occurred.

The messages are stored in the error file (.ERR) if no MPASM options are specified. If the /e- option is used (turns error file off), then the messages will appear on the screen. If the /q (quiet mode) option is used with the /e-, then the messages will not display on the screen or in an error file. The messages will still appear in the listing file.

C.2 Highlights

Topics covered in this appendix:

- Errors
- Warnings
- Messages

C.3 Errors

101 ERROR:

User error, invoked with the ERROR directive.

102 Out of memory.

Not enough memory for macros, #defines or internal processing. Eliminate any TSR's, close any open applications, and try assembling the file again. If this error was obtained using the Real Mode DOS executable, try using either the Windows version (MPASMWIN) or DPMI version (MPASM_DP)

103 Symbol table full.

No more memory available for the symbol table. Eliminate any TSR's, close any open applications, and try assembling the file again. If this error was obtained using the Real Mode DOS executable, try using either the Windows version (MPASMWIN) or DPMI version (MPASM_DP)

104 Temp file creation error.

Could not create a temporary file. Check the available disk space.

105 Cannot open file.

Could not open a file. If it is a source file, the file may not exist. If it is an output file, the old version may be write protected.

MPASM User's Guide with MPLINK and MPLIB

106 String substitution too complex.

Too much nesting of #defines.

107 Illegal digit.

An illegal digit in a number. Valid digits are 0-1 for binary, 0-7 for octal, 0-9 for decimal, and 0-9, a-f, and A-F for hexadecimal.

108 Illegal character.

An illegal character in a label. Valid characters for labels are alphabetic (a..f, A..F), numeric (0-9), the underscore (_), and the question mark (?). Labels may not begin with a numeric.

109 Unmatched (

An open parenthesis did not have a matching close parenthesis. For example, "DATA (1+2".

110 Unmatched)

An close parenthesis did not have a matching open parenthesis. For example, DATA 1+2).

111 Missing symbol.

An EQU or SET statement did not have a symbol to assign the value to.

112 Missing operator.

An arithmetic operator was missing from an expression. For example, DATA 1 2.

113 Symbol not previously defined.

A symbol was referenced that has not yet been defined. Only addresses may be used as forward references. Constants and variables must be declared before they are used.

114 Divide by zero.

Division by zero encountered during an expression evaluation.

115 Duplicate label.

A label was declared as a constant (e.g., with the EQU or CBLOCK directive) in more than one location.

116 Address label duplicated or different in second pass.

The same label was used in two locations. Alternately, the label was used only once but evaluated to a different location on the second pass. This often happens when users try to write page-bit setting macros that generate different numbers of instructions based on the destination.

117 Address wrapped around 0.

The location counter can only advance to FFFF. After that, it wraps back to 0.

118 Overwriting previous address contents.

Code was previously generated for this address.

MPASM Errors/Warnings/Messages

119 Code too fragmented.

The code is broken into too many pieces. This error is very rare, and will only occur in source code that references addresses above 32K (including configuration bits).

120 Call or jump not allowed at this address.

A call or jump cannot be made to this address. For example, CALL destinations on the PIC16C5x family must be in the lower half of the page.

121 Illegal label.

Labels are not allowed on certain directive lines. Simply put the label on its own line, above the directive. Also, HIGH, LOW, PAGE, and BANK are not allowed as labels.

122 Illegal opcode.

Token is not a valid opcode.

123 Illegal directive.

Directive is not allowed for the selected processor; for example, the `_IDLOCS` directive on the PIC17C42.

124 Illegal argument.

An illegal directive argument; for example, LIST STUPID.

125 Illegal condition.

A bad conditional assembly. For example, an unmatched ENDIF.

126 Argument out of range.

Opcode or directive argument out of the valid range; for example, TRIS 10.

127 Too many arguments.

Too many arguments specified for a macro call.

128 Missing argument(s).

Not enough arguments for a macro call or an opcode.

129 Expected.

Expected a certain type of argument. The expected list will be provided.

130 Processor type previously defined.

A different family of processor is being selected.

131 Processor type is undefined.

Code is being generated before the processor has been defined. Note that until the processor is defined, the opcode set is not known.

132 Unknown processor.

The selected processor is not a valid processor.

MPASM User's Guide with MPLINK and MPLIB

133 Hex file format INHX32 required.

An address above 32K was specified. For example, specifying the configuration bits on the PIC17CXX family.

134 Illegal hex file format.

An illegal hex file format was specified in the LIST directive.

135 Macro name missing.

A macro was defined without a name.

136 Duplicate macro name.

A macro name was duplicated.

137 Macros nested too deep.

The maximum macro nesting level was exceeded.

138 Include files nested too deep.

The maximum include file nesting level was exceeded.

139 Maximum of 100 lines inside WHILE-ENDW.

A WHILE-ENDW can contain at most 100 lines.

140 WHILE must terminate within 256 iterations.

A WHILE-ENDW loop must terminate within 256 iterations. This is to prevent infinite assembly.

141 WHILEs nested too deep.

The maximum WHILE-ENDW nesting level was exceeded.

142 IFs nested too deep.

The maximum IF nesting level was exceeded.

143 Illegal nesting.

Macros, IF's and WHILE's must be completely nested; they cannot overlap. If you have an IF within a WHILE loop, the ENDIF must come before the ENDW.

144 Unmatched ENDC.

ENDC found without a CBLOCK.

145 Unmatched ENDM.

ENDM found without a MACRO definition.

146 Unmatched EXITM.

EXITM found without a MACRO definition.

147 Directive not allowed when generating an object file.

The ORG directive is not allowed when generating an object file. Instead, declare a data or code section, specifying the address if necessary.

MPASM Errors/Warnings/Messages

148 Expanded source line exceeded 200 characters.

The maximum length of a source line, after #DEFINE and macro parameter substitution, is 200 characters. Note that #DEFINE substitution does not include comments, but macro parameter substitution does.

149 Directive only allowed when generating an object file section.

Certain directives, such as GLOBAL and EXTERN, only have meaning when an object file is generated. They cannot be used when generating absolute code.

150 Labels must be defined in a code or data section when making an object file.

When generating an object file, all data and code address labels must be defined inside a data or code section. Symbols defined by the EQU and SET directives can be defined outside of a section.

151 Operand contains unresolvable labels or is too complex.

When generating an object file, operands must be of the form [HIGH|LOW]([<relocatable address label>]+[<offset>]).

152 Executable code and data must be defined in an appropriate section.

When generating an object file, all executable code and data declarations must be placed within appropriate sections.

153 Page or Bank bits cannot be evaluated for the operand.

The operand of a PAGESEL, BANKSEL or BANKISEL directive must be of the form <relocatable address label> or <constant>.

154 Each object file section must be contiguous.

Object file sections, except UDATA_OVR sections, cannot be stopped and restarted within a single source file. To resolve this problem, either name each section with its own name or move the code and data declarations such that each section is contiguous. This error will also be generated if two sections of different types are given the same name.

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

If multiple UDATA_OVR sections with the same name are declared, they must all have the same starting address.

156 Operand must be an address label.

When generating object files, only address labels in code or data sections may be declared global. Variables declared by the SET or EQU directives may not be exported.

MPASM User's Guide with MPLINK and MPLIB

157 UNKNOWN ERROR.

An error has occurred which MPASM cannot understand. It is not any of the errors described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this error.

C.4 Warnings

201 Symbol not previously defined.

Symbol being #undefined was not previously defined.

202 Argument out of range. Least significant bits used.

Argument did not fit in the allocated space. For example, literals must be 8 bits.

203 Found opcode in column 1.

An opcode was found in column one, which is reserved for labels.

204 Found pseudo-op in column 1.

A pseudo-op was found in column one, which is reserved for labels.

205 Found directive in column 1.

A directive was found in column one, which is reserved for labels.

206 Found call to macro in column 1.

A macro call was found in column one, which is reserved for labels.

207 Found label after column 1.

A label was found after column one, which is often due to a misspelled opcode.

208 Label truncated at 32 characters.

Maximum label length is 32 characters.

209 Missing quote.

A text string or character was missing a quote. For example, DATA 'a.

210 Extra),

An extra comma was found at the end of the line.

211 Extraneous arguments on the line.

Extra arguments were found on the line. These warnings should be investigated, since they are often indications of the free-format parser interpreting something in a manner other than was intended (try assembling OPTION EQU 0x81 with LIST FREE).

212 Expected

Expected a certain type of argument. A description should be provided. For the warning, an assumption is made about the argument.

MPASM Errors/Warnings/Messages

- 213 The EXTERN directive should only be used when making a .O file.**
The EXTERN directive only has meaning if an object file is being created. This warning has been superseded by Error 149.
- 214 Unmatched (**
An unmatched parenthesis was found. The warning is used if the parenthesis is not used for indicating order of evaluation.
- 215 Processor superseded by command line. Verify processor symbol.**
The processor was specified on the command line as well as in the source file. The command line has precedence.
- 216 Radix superseded by command line.**
The radix was specified on the command line as well as in the source file. The command line has precedence.
- 217 Hex file format specified on command line.**
The hex file format was specified on the command line as well as in the source file. The command line has precedence.
- 218 Expected DEC, OCT, HEX. Will use HEX.**
Bad radix specification.
- 219 Invalid RAM location specified.**
If the `__MAXRAM` and `__BADRAM` directives are used, this warning flags use of any RAM locations declared as invalid by these directives. Note that the provided header files include `__MAXRAM` and `__BADRAM` for each processor.
- 220 Address exceeds maximum range for this processor.**
A ROM location was specified that exceeds the processor's memory size.
- 221 Invalid message number.**
The message number specified for displaying or hiding is not a valid message number.
- 222 Error messages cannot be disabled.**
Error messages cannot be disabled with the `ERRORLEVEL` command.
- 223 Redefining processor**
The selected processor is being reselected by the `LIST` or `PROCESSOR` directive.
- 224 Use of this instruction is not recommended.**
Use of the `TRIS` and `OPTION` instructions is not recommended for a PIC16CXX device.
- 225 Invalid label in operand.**
Operand was not a valid address. For example, if the user tried to issue a `CALL` to a MACRO name.

MPASM User's Guide with MPLINK and MPLIB

226 UNKNOWN WARNING

A warning has occurred which MPASM cannot understand. It is not any of the warnings described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this warning.

C.5 Messages

301 MESSAGE:

User message, invoked with the MESSG directive.

302 Register in operand not in bank 0. Ensure that bank bits are correct.

Register address was specified by a value that included the bank bits. For example, RAM locations in the PIC16CXX are specified with 7 bits in the instruction and one or two bank bits.

303 Program word too large. Truncated to core size.

Program words for the PIC16C5X may only be 12-bits; program words for the PIC16CXX may only be 14-bits.

304 ID Locations value too large. Last four hex digits used.

Only four hex digits are allowed for the ID locations.

305 Using default destination of 1 (file).

If no destination bit is specified, the default is used.

306 Crossing page boundary – ensure page bits are set.

Generated code is crossing a page boundary.

307 Setting page bits.

Page bits are being set with the LCALL or LGOTO pseudo-op.

308 Warning level superseded by command line value.

The warning level was specified on the command line as well as in the source file. The command line has precedence.

309 Macro expansion superseded by command line.

Macro expansion was specified on the command line as well as in the source file. The command line has precedence.

310 Superseding current maximum RAM and RAM map.

The `__MAXRAM` directive has been used previously.

312 Page or Bank selection not needed for this device. No code generated.

If a device contains only one ROM page or RAM bank, no page or bank selection is required, and any `PAGESEL`, `BANKSEL`, or `BANKISEL` directives will not generate any code.

MPASM Errors/Warnings/Messages

313 CBLOCK constants will start with a value of 0.

If the first CBLOCK in the source file has no starting value specified, this message will be generated.

314 UNKNOWN MESSAGE

A message has occurred which MPASM cannot understand. It is not any of the messages described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this message.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Appendix D. MPLINK Errors/Warnings

D.1 Introduction

MPLINK produces the following errors and warnings.

D.2 Highlights

Topics covered in this appendix:

- Parse Errors
- Linker Errors
- Linker Warnings
- Library File Errors
- COFF File Errors
- COFF to COD Converter Errors
- COFF to COD Converter Warnings

D.3 Parse Errors

Invalid attributes for memory in 'cmdfile:line'. A CODEPAGE, DATABANK, or SHAREBANK directive does not specify a NAME, START, or END attribute; or another attribute is specified which is not valid.

Invalid attributes for STACK in 'cmdfile:line'. A STACK directive does not specify a SIZE attribute, or another attribute is specified which is not valid.

Invalid attributes for SECTION in 'cmdfile:line'. A SECTION directive must have a NAME and either a RAM or ROM attribute.

Could not open 'cmdfile'. A linker command file could not be opened. Check that the file exists, is in the current search path, and is readable.

Multiple inclusion of linker command file 'cmdfile'. A linker command file can only be included once. Remove multiple INCLUDE directives to the referenced linker command file.

Illegal <libpath> for LIBPATH in 'cmdfile:line'. The 'libpath' must be a semicolon delimited list of directories. Enclose directory name which have embedded spaces in double quotes.

Illegal <lkrpath> for LKRPATH in 'cmdfile:line'. The 'lkrpath' must be a semicolon delimited list of directories. Enclose directory names which have embedded spaces in double quotes.

Illegal <filename> for FILES in 'cmdfile:line'. An object or library filename must end with '.o' or '.lib' respectively.

MPASM User's Guide with MPLINK and MPLIB

Illegal <filename> for INCLUDE in 'cmdfile:line'. A linker command filename must end with '.lkr'.

Unrecognized input in 'cmdfile:line'. All statements in a linker command file must begin with a directive keyword or the comment Delimiter '//'.

-o switch requires <filename>. A COFF output filename must be specified. For example: -o main.out

-m switch requires <filename>. A map filename must be specified. For example: -m main.map

-n switch requires <length>. The number of source lines per listing file page must be specified. A 'length' of zero will suppress pagination of the listing file.

-L switch requires <pathlist>. A semicolon delimited path must be specified. Enclose directory names containing embedded spaces with double quotes. For example: -L ..;c:\mplab\lib;"c:\program files\mplink"

-K switch requires <pathlist>. A semicolon delimited path must be specified. Enclose directory names containing embedded spaces with double quotes. For example: -L ..;c:\mplab\lib;"c:\program files\mplink"

unknown switch: 'cmdline token'. An unrecognized command line switch was supplied. Refer to the Usage documentation for the list of supported switches.

D.4 Linker Errors

Memory 'memName' overlaps memory 'memName'. All CODEPAGE blocks must specify unique memory ranges which do not overlap. Similarly DATABANK and SHAREBANK blocks may not overlap.

Duplicate definition of memory 'memName'. All CODEPAGE and DATABANK directives must have unique NAME attributes.

Multiple map files declared: 'File1', 'File2'. The -m <mapfile> switch was specified more than once.

Multiple output files declared: 'File1', 'File2'. The -o <outfile> switch was specified more than once.

Multiple inclusion of object file 'File1', 'File2'. An object file has been included multiple times either on the command line or with a FILES directive in a linker command file. Remove the multiple references.

Overlapping definitions of SHAREBANK 'memName'. A SHAREBANK directive specifies a range of addresses that overlap a previous definition. Overlaps are not permitted.

Inconsistent length definitions of SHAREBANK 'memName'. All SHAREBANK definitions which have the same NAME attribute must be of equal length.

Multiple STACK definitions. A STACK directive occurs more than once in the linker command file or included linker command files. Remove the multiple STACK directives.

MPLINK Errors/Warnings

Undefined DATABANK/SHAREBANK 'memName' for STACK.

Duplicate definitions of SECTION 'secName'. Each SECTION directive must have unique NAME attributes. Remove duplicate definitions.

Undefined CODEPAGE 'memName' for SECTION 'secName'. A SECTION directive with a ROM attribute refers to a memory block which has not been defined. Add a CODEPAGE directive to the linker command file for the undefined memory block.

Undefined DATABANK/SHAREBANK 'memName' for SECTION 'secName'. A SECTION directive with a RAM attribute refers to a memory block that has not been defined. Add a DATABANK or SHAREBANK directive to the linker command file for the undefined memory block.

No input object files specified. At least one object module must be specified either on the command line or in the linker command file using the FILES directive.

Could not find file 'File'. An input object or library file was specified which does not exist, or cannot be found in the linker path.

Processor types do not agree across all input files. Each object module and library file specifies a processor type or a processor family. All input modules processor types or families must match.

ROM width of 'xx' not supported. An input module specifies a processor whose ROM width is not 12, 14, or 16 bits wide.

Unknown section type for 'secName' in file 'File'. An input object or library module is not of the proper file type or it may be corrupted.

Section types for 'secName' do not match across input files. A section with the name 'secName' may occur in more than one input file. All input files which have this section must also have the same section type.

Section 'secName' is absolute but occurs in more than one input file. An absolute section with the name 'secName' may only occur in a single input file. Relocatable sections with the same name may occur in multiple input files. Either remove the multiple absolute sections in the source files or use relocatable sections instead.

Section share types for 'secName' do not match across input files. A section with the name 'secName' occurs in more than one input file, however, in some it is marked as a shared section and in some it is not. Change the section's share type in the source files and rebuild the object modules.

Section 'secName' contains code and can not have a 'RAM' memory attribute specified in the linker command file. Use only the ROM attribute when defining the section in the linker command file.

Section 'secName' contains uninitialized data and can not have a 'ROM' memory attribute specified in the linker command file. Use only the RAM attribute when defining the section in the linker command file.

MPASM User's Guide with MPLINK and MPLIB

Section 'secName' contains initialized data and can not have a 'ROM' memory attribute specified in the linker command file. Use only the RAM attribute when defining the section in the linker command file.

Section 'secName' contains initialized rom data and can not have a 'RAM' memory attribute specified in the linker command file. Use only the ROM attribute when defining the section in the linker command file.

Section 'secName' has a memory 'memName' which can not fit the section. Section 'secName' length='0xHHHH'. The memory which was assigned to the section in the linker command file either does not have space to fit the section, or the section will overlap another section. Use the -m <mapfile> switch to generate an error map file. The error map will show the sections which were allocated prior to the error.

Section 'secName' has a memory 'memName' which is not defined in the linker command file. Add a CODEPAGE, DATABANK, or SHAREBANK directive for the undefined memory to the linker command file.

Section 'secName' can not fit the section. Section 'secName' length='0xHHHH'. A section which has not been assigned to a memory in the linker command file can not be allocated. Use the -m <mapfile> switch to generate an error map file. The error map will show the sections which were allocated prior to the error. More memory must be made available by adding a CODEPAGE, SHAREBANK, or DATABANK directive, or by removing the PROTECTED attribute, or the number of input sections must be reduced.

Section 'secName' has a memory 'memName' which can not fit the absolute section. Section 'secName' start=0xHHHH, length=0xHHHH. The memory which was assigned to the section in the linker command file either does not have space to fit the section, or the section will overlap another section. Use the -m <mapfile> switch to generate an error map file. The error map will show the sections which were allocated prior to the error.

Section 'secName' can not fit the absolute section. Section 'secName' start=0xHHHH, length=0xHHHH. A section which has not been assigned to a memory in the linker command file can not be allocated. Use the -m <mapfile> switch to generate an error map file. The error map will show the sections which were allocated prior to the error. More memory must be made available by adding a CODEPAGE, SHAREBANK, or DATABANK directive, or by removing the PROTECTED attribute, or the number of input sections must be reduced.

Symbol 'symName' has multiple definitions. A symbol may only be defined in a single input module.

Could not resolve symbol 'symName' in file 'File'. The symbol 'symName' is an external reference. No input module defines this symbol. If the symbol is defined in a library module, ensure that the library module is included on the command line or in the linker command file using the FILES directive.

Could not open map file 'File' for writing. Verify that if 'File' exists, it is not a read-only file.

MPLINK Errors/Warnings

Symbol 'symName' out of range of relative branch instruction. A relative branch instruction had 'symName' as its target, but a 2s complement encoding of the offset to 'symName' wouldn't fit in the limited number of instruction bits used for the target of a branch instruction.

Symbol 'symName' is not word-aligned. It cannot be used as the target of a {branch | call or goto} instruction. The target of a branch, call, or goto instruction was at an odd address, but the instruction encoding cannot reference addresses that are not word-aligned.

{PCL | TOSH | TOSU | TOSL} cannot be used as the destination of a MOVFF instruction. The MOVFF instruction has unpredictable results when its destination is the PCL, TOSH, TOSU, or TOSL registers. MPLINK will not allow the destination of a MOVFF instruction to be replaced with any of these addresses.

Absolute code section 'secName' must start at a word-aligned address. Program code sections will only be allocated at word-aligned addresses. MPLINK will give this error message if an absolute code section address is specified that is not word-aligned.

D.5 Linker Warnings

Fill pattern for memory 'memName' doesn't divide evenly into unused section locations. Last value was truncated. If a fill pattern is specified for a ROM section, but the free space in that section isn't evenly divisible by the fill pattern size, this warning will be issued to warn of incomplete patterns.

D.6 Library File Errors

Symbol 'name' has multiple external definitions. A symbol may only be defined once in a library file.

Could not open library file 'filename' for reading. Verify that 'filename' exists and can be read.

Could not read archive magic string in library file 'filename'. The file is not a valid library file or it may be corrupted.

File 'filename' is not a valid library file. Library files must end with '.lib'.

Library file 'filename' has a missing member object file. The file not a valid object file or it may be corrupted.

Could not build member 'memberName' in library file 'filename'. The file is not a valid library file or it is corrupted.

Could not open library file 'filename' for writing. Verify that if 'filename' exists, it is not read-only.

Could not write archive magic string in library file 'filename'. The file may be corrupted.

Could not write member header for 'memberName' in library file 'filename'. The file may be corrupted.

MPASM User's Guide with MPLINK and MPLIB

'memberName' is not a member of 'filename'. 'memberName' can not be extracted or deleted from a library unless it is a member of the library.

D.7 COFF File Errors

All COFF file errors indicate an internal error in the file's contents. Please contact Microchip support if any of the following errors are generated:

- Unable to find section name in string table.
- Unable to find symbol name in string table.
- Unable to find aux_file name in string table.
- Could not find section name 'secName' in string table.
- Could not find symbol name 'symName' in string table.
- Coff file 'filename' symbol['xx'] has an invalid n_scnum.
- Coff file 'filename' symbol['xx'] has an invalid n_offset.
- Coff file 'filename' section['xx'] has an invalid s_offset.
- Coff file 'filename' has relocation entries but an empty symbol table.
- Coff file 'filename', section 'secName' reloc['xx'] has an invalid r_symndx.
- Coff file 'filename', symbol['xx'] has an invalid x_tagndx or x_endndx.
- Coff file 'filename', section 'secName' line['xx'] has an invalid l_srcndx.
- Coff file 'filename', section 'secName' line['xx'] has an invalid l_fcndx.
- Coff file 'filename', cScnHdr.size() != cScnNum.size().
- Could not open Coff file 'filename' for reading.
- Coff file 'filename' could not read file header.
- Coff file 'filename' could not read optional file header.
- Coff file 'filename' missing optional file header.
- Coff file 'filename' could not read string table length.
- Coff file 'filename' could not read string table.
- Coff file 'filename' could not read symbol table.
- Coff file 'filename' could not read section header.
- Coff file 'filename' could not read raw data.
- Coff file 'filename' could not read line numbers.
- Coff file 'filename' could not read relocation info.
- Could not open Coff file 'filename' for writing.
- Coff file 'filename' could not write file header.
- Coff file 'filename' could not write optional file header.
- Coff file 'filename' could not write section header.

- Coff file 'filename' could not write raw data.
- Coff file 'filename' could not write reloc.
- Coff file 'filename' could not write lineinfo.
- Coff file 'filename' could not write symbol.
- Coff file 'filename' could not write string table length.
- Coff file 'filename' could not write string.

D.8 COFF To COD Converter Errors

Coff file 'filename' must contain at least one 'code' or 'romdata' section. In order to convert a COFF file to a COD file, the COFF file must have either a code or a romdata section.

Could not open list file 'filename' for writing. Verify that if 'filename' exists and that it is not a read-only file.

D.9 COFF To COD Converter Warnings

Could not open source file 'filename'. This file will not be present in the list file. The referenced source file could not be opened. This can happen if an input object/library module was built on a machine with a different directory structure. If source level debugging for the file is desired, rebuild the object or library on the current machine.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Appendix E. MPLIB Errors

E.1 Introduction

MPLIB detects the following sources of error and reports them.

E.2 Highlights

Topics covered in this appendix:

- Parse Errors
- Library File Errors
- COFF File Errors

E.3 Parse Errors

invalid switch. An unsupported switch was specified. Refer to Usage for a list of supported command line options.

library filename is required. All commands require a library filename. All library filenames must end with '.lib'.

invalid object filename. All object filenames must end with '.o'.

E.4 Library File Errors

Please refer to the documentation on MPLINK for error messages associated with library files.

E.5 COFF File Errors

Please refer to the documentation on MPLINK for error messages associated with COFF files.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Glossary

Introduction

To provide a common frame of reference, this glossary defines the terms that are used in this document.

Highlights

This glossary contains definitions for the terms used in the MPASM, MPLINK and MPLIB sections. Most terms are used in all three, though some may be section-specific.

Terms

Absolute Section

A section with a fixed absolute address which can not be changed by the Linker.

Alpha Character

Alpha characters are those characters, regardless of case, that are normally contained in the alphabet: (a, b, ..., z, A, B, ..., Z).

Alphanumeric

Alphanumeric characters include alpha characters and numbers: (0,1, ..., 9).

Application

A set of software and hardware developed by the user, usually designed to be a product controlled by a PICmicro microcontroller.

Assemble

The act of executing the MPASM macro assembler to translate source code to machine code.

Assembler Source Code

A text file that is processed by an assembler to produce a one-to-one correspondence between assembler instructions and PICmicro machine code.

Assigned Section

A section which has been assigned to a target memory block in the linker command file. The Linker allocates an assigned section into its assigned target memory block.

Build

A function that recompiles all the source files for an application.

MPASM User's Guide with MPLINK and MPLIB

COFF

Common Object File Format - a file format definition for object/executable files. Used as a relocatable object file from MPASM or MPLAB-C17/18.

Command Line Interface

Command Line Interface refers to executing a program with options. Executing MPASM with any command line options or just the file name will invoke the assembler. In the absence of any command line options, a prompted input interface (shell) will be executed.

Control Directives

Control directives permit sections of conditionally assembled code.

Data Directives

Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.

Data RAM

General purpose file registers from RAM on the PICmicro device being emulated. The File Register window displays data RAM.

Directives

Directives provide control of the assembler's operation by telling MPASM how to treat mnemonics, define data, and format the listing file. Directives make coding easier and provide custom output according to specific needs.

Expressions

Expressions are used in the operand field of the source line and may contain constants, symbols, or any combination of constants and symbols separated by arithmetic operators. Each constant or symbol may be preceded by a plus or minus to indicate a positive or negative expression.

Note: Expressions are evaluated in 32 bit integer math (floating point is not currently supported).
--

External Linkage

A function or variable has external linkage if it can be accessed from outside the module in which it is defined.

External Symbol

A symbol for an identifier which has external linkage.

External Symbol Definition

An external symbol for a function or variable defined in the current module.

External Symbol Reference

An external symbol which references a function or variable defined outside the current module.

External Symbol Resolution

A process performed by the linker in which external symbol definitions from all input modules are collected in an attempt to update all external symbol references. Any external symbol references which do not get updated cause a linker error to be reported.

Hex Code

Executable instructions assembled or compiled from source code into standard hexadecimal format code. Hex code can be directly converted to object code. Hex code is contained in a hex file.

Hex File

An ASCII file containing hexadecimal addresses and values (hex code) suitable for programming a device. This format is readable by a device programmer.

Identifier

A function or variable name.

Initialized Data

Data which is defined with an initial value. In C, `int myVar=5;` defines a variable which will reside in an initialized data section.

Internal Linkage

A function or variable has internal linkage if it can not be accessed from outside the module in which it is defined.

Library

A library is a collection of relocatable object modules. It is created by assembling multiple source files to object files, and then using the librarian to combine the object files into one library file. A library can be linked with object modules and other libraries to create executable code. Libraries are created and manipulated with Microchip's librarian, MPLIB.

Link

Linking is the process of combining object files and libraries to create executable code. Linking is performed by Microchip's linker, MPLINK.

Listing Directives

Listing Directives are those directives that control the MPASM listing file format. They allow the specification of titles, pagination and other listing control.

Listing File

A listing file is an ASCII text file that shows the machine code generated for each assembly instruction, MPASM directive, or macro encountered in a source file.

MPASM User's Guide with MPLINK and MPLIB

Local Label

A local label is one that is defined with the LOCAL directive. These labels are particular to a given instance of the macro's instantiation. In other words, the symbols and labels that are declared as local are purged from the symbol table when the ENDM macro is encountered.

Macro

A macro is a collection of assembler instructions that are included in the assembly code when the macro name is encountered in the source code. Macros must be defined before they are used; forward references to macros are not allowed.

All statements following the MACRO directive are part of the macro definition. Labels used within the macro must be local to the macro so the macro can be called repetitively.

Macro Directives

These directives control the execution and data allocation within macro body definitions.

Make Project

A command that rebuilds an application, re-compiling only those source files that have changed since the last complete compilation.

Mnemonics

These are instructions that are translated directly into machine code. These are used to perform arithmetic and logical operations on data residing in program or data memory of a microcontroller. They also have the ability to move data in and out of registers and memory as well as change the flow of program execution. Also referred to as **Opcodes**.

MPASM

Microchip Technology's Relocatable macro assembler.

MPLAB

The name of the main executable program that supports the IDE with an Editor, Project Manager, and Emulator/Simulator Debugger. The MPLAB Software resides on the PC host. The executable file name is MPLAB.EXE. MPLAB.EXE calls many other files.

MPLAB-C17/C18

Microchip Technology's C compiler for PIC17CXX and PIC18CXX products.

MPLIB

Microchip Technology's Librarian for library files used with MPLINK.

MPLINK

Microchip Technology's Linker.

Nesting Depth

Macros can be nested to sixteen levels deep.

Node

MPLAB project component.

Object Code

The machine code that is produced from the source code after it is processed by an assembler or compiler. This code will be the memory-resident code that will run on the PICmicro in the user's application. Relocatable code is code produced by MPASM or MPLAB-C17/C18 that can be run through MPLINK. Object code is contained in an object file.

Object File

A module which may contain relocatable code or data and references to external code or data. Typically, multiple object modules are linked to form a single executable output. Special directives are required in the source code when generating an object file. The object file contains object code.

Object File Directives

These directives are used only when creating an object file.

Operators

Operators are arithmetic symbols, like the plus sign '+' and the minus sign '-', that are used when forming well-defined expressions. Each operator has an assigned precedence.

PC

Any IBM® or compatible Personal Computer. MPLAB needs a 486X or better machine.

PC Host

The computer running Windows 3.1x or Windows 95/98.

PICmicro

PICmicro refers to the PIC12CXX, PIC14000, PIC16C5X, PIC16CXX, PIC17CXX and PIC18CXX Microchip microcontroller families.

Precedence

Precedence is the concept that some elements of an expression get evaluated before others. Operators of the same precedence are evaluated from left to right.

Program Memory

Memory in the emulator or simulator containing the downloaded target application firmware.

Project

A set of source files and instructions to build the object code for an application.

MPASM User's Guide with MPLINK and MPLIB

Radix

Radix is the base-numbering system that the assembler uses when evaluating expressions. The default radix is hexadecimal (base 16). You can change the default radix and override the default radix with certain radix override operators.

RAM

Random Access Memory (Data Memory).

Raw Data

The binary representation of code or data associated with a section.

Recursion

This is the concept that a macro, having been defined, can call itself. Great care should be taken when writing recursive macros; it is easy to get caught in an infinite loop where there will be no exit from the recursion.

Relocatable Section

A section whose address is not fixed. The linker assigns addresses to relocatable sections through a process called relocation.

Relocation

A process performed by the linker in which absolute addresses are assigned to relocatable sections and all identifier symbol definitions within the relocatable sections are updated to their new addresses.

ROM

Read-only Memory (Program Memory).

Script

Linker script files are the command files of MPLINK.

Section

An aggregate of code or data which has a name, size, and address.

Shared Section

A section which resides in a shared (non-banked) region of data RAM.

Shell

The MPASM shell is a prompted input interface to the macro assembler. There are two MPASM shells, one for the DOS version and one for the Windows version.

Software Stack

MPLAB-C17/C18 software stack.

Source Code

Source code consists of PICmicro instructions and MPASM directives and macros that will be translated into machine code. This code is suitable for use by a PICmicro or Microchip development system product like MPLAB™.

Source File

The ASCII text file of PICmicro instructions and MPASM directives and macros (source code) that will be translated into machine code. It is an ASCII file that can be created using any ASCII text editor.

Stack

An area in data memory where function arguments, return values, local variables, and return addresses are stored.

Symbol

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, section names, file names, struct/enum/union tag names, etc.

Unassigned Section

A section which has not been assigned to a specific target memory block in the linker command file. The linker must find a target memory block in which to allocate an unassigned section.

Uninitialized Data

Data which is defined without an initial value. In C, `int myVar;` defines a variable which will reside in an uninitialized data section.

MPASM User's Guide with MPLINK and MPLIB

NOTES:

Index

Symbols

#DEFINE 46, 56, 70, 197, 201
#UNDEFINE 57, 70
__BADRAM 39
__CONFIG 42

A

ACCESSBANK 122, 125, 126
Accessing Labels From Other Modules 77
Application Notes 5
Arithmetic Operators 91, 179
ASCII Character Set 195

B

BADRAM 60, 203
BANKSEL 39, 201
BANKSEL 40, 77, 201
Build 217

C

CALL 199
Case Sensitivity 22
CBLOCK 19, 41, 49, 198
CD-ROM 5
CODE 41
CODEPAGE 121, 122, 125
Command Line Interface 21, 177, 218
Comments 18
Compatibility 12
Configuration Bits 76
CONSTANT 19, 43
Conventions 3
Conversion, Hexadecimal to Decimal 194
Cross Reference File 23
Customer Support 8

D

DA 44
DATA 44, 89
DATABANK 122, 123, 125, 126
DB 45
DE 45
Define 22
Directive Summary 35
Directives 35, 218
Document Layout 1

DOS Shell Interface 24
DT 47
DW 47, 89

E

ELSE 48, 56
END 48
ENDC 41, 49, 200
ENDIF 49, 56, 199, 200
ENDM 49, 84, 87, 200
ENDW 50, 71
EQU 19, 50, 65, 198
ERROR 51
Error File 16, 20, 22, 51, 197
ERRORLEVEL 51, 203
Errors 197, 207, 208, 211, 212, 213, 215
Escape Sequences 90
EXITM 52, 84, 85, 200
EXPAND 52
Expressions 218
EXTERN 53

F

File

Cross Reference 23
Error 22, 51, 197
Listing 22, 35, 51, 52, 58, 62, 66, 67, 219
Object 22, 221
File Extensions 16
FILL 53

G

GLOBAL 54
Glossary 217

H

Header Files 73
Hex File Format 20, 22, 25, 58, 171
HIGH 199
High/Low 93

I

ID Locations 76
IDATA 54
IDLOCS 55, 199
IF 48, 56, 200
IFDEF 46, 56

MPASM User's Guide with MPLINK and MPLIB

IFNDEF	57	MPASMWIN	21, 197
INCLUDE	58	MPLAB	220
Increment/Decrement	93	MPLAB User's Guide	4
Initialization	95	MPLAB-ICE	12
Instruction Operands	75	MPLIB	163, 219
Instruction Sets	180, 189	COFF File Errors	215
12-Bit Core	181	Installation	165
14-Bit Core	183	Library File Errors	215
16-Bit Core	186	Overview	166
Special Instructions	184	Parse Errors	215
Internet Address	5	Usage	167
L		MPLINK	99, 219
Labels	17	COFF File Errors	212
Library	219	COFF To COD Converter Errors	213
Link	219	COFF To COD Converter Warnings	213
Linker Options	109	Command Line Information	119
Linker Processing	127	Input/Output Files	105
Linker Scripts	119	Installation	103
LIST	42, 55, 58, 203	Library File Errors	211
Listing File 16, 19, 22, 35, 51, 52, 58, 62, 66, 67, 219		Linker Errors	208
LOCAL	59, 86	Linker Scripts	119
Local Label	220	Linker Warnings	211
Logical Section Definitions	125	MPLAB Projects	111
LOW	199	Overview	104
M		Parce Errors	207
MACRO	22, 49, 60	Sample Application 1	131
Macro Language	83	Sample Application 2	137
MAXRAM	39, 60, 203, 204	Sample Application 3	145
Memory Region Definitions	121	Sample Application 4	151
Message Level	23, 51, 59	N	
Messages	204	NOEXPAND	52, 62
MESSG	61	NOLIST	62
Microchip Internet Web Site	5	O	
Microcontroller Mode	221	Object File	22, 25, 221
Migration Path	12	Objects, Relocatable	73
Mnemonics	17, 220	Off-Chip Memory	221
MPASM	11, 21	Operands	17
Directive Summary	35	Operators	221
Directives	35	ORG	62
Errors	197	P	
Input/Output Files	16	PAGE	63
Installation	13	PAGESEL	63, 201
Messages	204	Paging and Banking Issues	77
MPLAB Projects	29	PICmicro	221, 222, 223
Overview	14	PICSTART	12
Quick Reference	175	Precedence	221
Warnings	202	PRO MATE	12

PROCESSOR 42, 55, 64, 203
Processor Type 22, 203
Program Memory 74
Project 29, 111, 221

R

RADIX 64
Radix 12, 22, 59, 91, 203, 222
RAM Allocation 75
README Files 4
Relocatable Objects 73
RES 65

S

SET 19, 43, 65, 70, 198
SHAREBANK 122, 123, 125, 126
Shell 21, 222
Source Code 222
Source File 16
SPACE 66
STACK Definition 126
SUBTITLE 66

T

Text Strings 89
TITLE 66

U

UDATA 67
UDATA_ACS 68, 75
UDATA_OVR 68
UDATA_SHR 69
UPPER 75

V

VARIABLE 19, 43, 70

W

Warnings 202, 211, 213
WHILE 50, 71, 86, 200
WHILE-ENDW 200
Windows Shell Interface 27
WWW Address 5

MPASM User's Guide with MPLINK and MPLIB

NOTES:

NOTES:



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: <http://www.microchip.com>

Atlanta

Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton

Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit

Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto

Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong

Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing

Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India

Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai

Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C

Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark

Microchip Technology Denmark ApS
Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France

Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

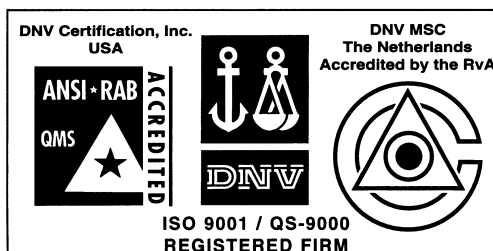
Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELoc® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

All rights reserved. © 1999 Microchip Technology Incorporated. Printed in the USA. 11/99 Printed on recycled paper.

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.



MICROCHIP

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199

Tel: 602.786.7200 Fax: 602.899.9210

© 1999 Microchip Technology Inc., Printed in the U.S.A. 3/99
DS33014