

# **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.

©1997 Microchip Technology Incorporated. All rights reserved.

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

CompuServe is a registered trademark of CompuServe Incorporated.

IBM and PC/AT are registered trademarks of International Business Machines Corporation.

Intel is a registered trademark of Intel Corporation.

Windows and MS-DOS are registered trademarks of Microsoft Corporation.

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

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

Notes:



# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Table of Contents

---

---

### Part 1 – MPASM

#### Preface

Welcome .....	3
Feature List and Product Information .....	3
Migration Path .....	3

#### Chapter 1. Introduction

Product Definition .....	5
Using MPASM .....	5
Documentation Layout .....	7
Terms .....	8
Recommended Reading .....	10
System Requirements .....	10
Warranty Registration .....	10
Installation .....	10
Compatibility Issues .....	10
Customer Support .....	11

#### Chapter 2. Environment and Usage

Introduction .....	13
Highlights .....	13
Terms .....	13
Command Line Interface .....	14
DOS Shell Interface .....	16
Windows Shell Interface .....	17
Source Code Formats .....	18
Files Used by MPASM and Utility Functions .....	20
Hex File Formats .....	20
Listing File Format .....	21
Error File Format (.ERR) .....	22

#### Chapter 3. Directive Language

Introduction .....	23
Highlights .....	23

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

Terms	23
Directive Details	26
__BADRAM – Identify Unimplemented RAM	26
BANKISEL – Generate Indirect Bank Selecting Code	26
BANKSEL – Generate Bank Selecting Code	27
CBLOCK – Define a Block of Constants	28
CODE – Begin an Object File Code Section	29
__CONFIG – Set Processor Configuration Bits	29
CONSTANT – Declare Symbol Constant	30
DATA – Create Numeric and Text Data	30
DB – Declare Data of One Byte	31
DE – Declare EEPROM Data Byte	31
#DEFINE – Define a Text Substitution Label	32
DT – Define Table	32
DW – Declare Data of One Word	33
ELSE – Begin Alternative Assembly Block to IF	33
END – End Program Block	34
ENDC – End an Automatic Constant Block	34
ENDIF – End Conditional Assembly Block	35
ENDM – End a Macro Definition	35
ENDW – End a While Loop	35
EQU – Define an Assembler Constant	36
ERROR – Issue an Error Message	36
ERRORLEVEL – Set Message Level	37
EXITM – Exit from a Macro	37
EXPAND – Expand Macro Listing	38
EXTERN – Declare an Externally Defined Label	38
FILL – Specify Memory Fill Value	39
GLOBAL – Export a Label	39
IDATA – Begin an Object File Initialized Data Section	40
__IDLOCS – Set Processor ID Locations	41
IF – Begin Conditionally Assembled Code Block	41
IFDEF – Execute If Symbol has Been Defined	42
IFNDEF – Execute If Symbol has not Been Defined	42
INCLUDE – Include Additional Source File	43
LIST – Listing Options	44
LOCAL – Declare Local Macro Variable	44
MACRO – Declare Macro Definition	45

# Table of Contents

---

__MAXRAM – Define Maximum RAM Location	46
MESSG – Create User Defined Message	47
NOEXPAND – Turn off Macro Expansion	47
NOLIST – Turn off Listing Output	47
ORG – Set Program Origin	48
PAGE – Insert Listing Page Eject	48
PAGESEL – Generate Page Selecting Code	48
PROCESSOR – Set Processor Type	49
RADIX – Specify Default Radix	49
RES – Reserve Memory	50
SET – Define an Assembler Variable	50
SPACE – Insert Blank Listing Lines	51
SUBTITLE – Specify Program Subtitle	51
TITLE – Specify Program Title	51
UDATA – Begin an Object File Uninitialized Data Section	52
UDATA_OVR – Begin an Object File Overlaid Uninitialized Data Section	52
UDATA_SHR – Begin an Object File Shared Uninitialized Data Section	53
#UNDEFINE – Delete a Substitution Label	54
VARIABLE – Declare Symbol Variable	54
WHILE – Perform Loop While Condition is True	55
<b>Chapter 4. Using MPASM to Create Relocatable Objects</b>	
Introduction	57
Highlights	57
Header Files	57
Program Memory	57
Instruction Operands	58
RAM Allocation	58
Configuration Bits and ID Locations	60
Accessing Labels From Other Modules	60
Paging and Banking Issues	61
Unavailable Directives	62
Generating the Object Module	62
Example	62
<b>Chapter 5. Macro Language</b>	
Introduction	65
Highlights	65

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

Terms	65
Macro Syntax	66
Macro Directives	66
Text Substitution	67
Macro Usage	67
Examples	68
<b>Chapter 6. Expression Syntax and Operation</b>	
Introduction	71
Highlights	71
Terms	71
Text Strings	72
Numeric Constants and Radix	74
High/Low	76
Increment/Decrement	76

## **Part 2 – MPLINK**

<b>Chapter 1. Introduction</b>	
MPLINK Preview	79
Product Description	79
File Formats	80
Linker Components	80
Tools and Supported Platforms	81
<b>Chapter 2. Usage</b>	
Command Line	83
Usage Example	84
<b>Chapter 3. Command File</b>	
Directives	85
Linker Command File Example:	88
<b>Chapter 4. Linker Map File</b>	
Linker Map File	89
<b>Chapter 5. Linker Processing</b>	
Linker Allocation Algorithm	91
Relocation Example	92
Initialized Data	93

# Table of Contents

---

<b>Chapter 6. Terminology</b>	
Terminology .....	95

## Part 3 – MPLIB

<b>Chapter 1. Librarian Fundamentals</b>	
Usage .....	99
Usage Examples .....	100
Tips .....	100
Error Reporting .....	100

## Appendices

<b>Appendix A. Hex File Formats</b>	
Introduction .....	101
Highlights .....	101
Hex File Formats .....	101

<b>Appendix B. On-Line Support</b>	
Introduction .....	105
Connecting to the Microchip Internet Web Site .....	105
Connecting to the Microchip BBS .....	106
Using the Bulletin Board .....	106
Software Releases .....	107
Systems Information and Upgrade Hot Line .....	108

<b>Appendix C. MPASM Errors/Warnings/Messages</b>	
Errors .....	109
Warnings .....	113
Messages .....	116

<b>Appendix D. MPLINK Errors/Warnings</b>	
Parse Errors .....	119
Linker Errors .....	120
Library File Errors .....	122
COFF File Errors .....	123
COFF To COD Converter Errors .....	124
COFF To COD Converter Warnings .....	124

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

## **Appendix E. Quick Reference**

Key to PICmicro Family Instruction Sets .....	131
PIC16C5X Instruction Set .....	131
PIC16CXX Instruction Set .....	133
PIC17CXX Instruction Set .....	136
Hexadecimal to Decimal Conversion .....	139
ASCII Character Set .....	140

## **Appendix F. Example Initialization Code**

Initialization Code .....	141
Initialization Code for the PIC16CXX .....	141
Initialization Code for the PIC17CXX .....	147

## **Index**

Index .....	153
-------------	-----





# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## Part 1 – MPASM

---

Preface .....	3
Chapter 1. Introduction .....	5
Chapter 2. Environment and Usage .....	13
Chapter 3. Directive Language .....	23
Chapter 4. Using MPASM to Create Relocatable Objects .....	57
Chapter 5. Macro Language .....	65
Chapter 6. Expression Syntax and Operation .....	71

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:



# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Preface

---

---

### Welcome

Microchip Technology Incorporated is committed to providing useful and innovative solutions to your microcontroller designs. MPASM is the first Universal Assembler available for Microchip's entire product line of microcontrollers. MPASM will generate solid code with a directive language rich in potential.

### Feature List and Product Information

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

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

Use of the Microchip MPASM Universal Assembler requires an IBM PC/AT® or compatible computer, running MS-DOS® V5.0 or greater.

### 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.

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

**Notes:**

---

---

## Chapter 1. Introduction

---

---

### Product Definition

MPASM is a DOS or Windows-based PC application that provides a platform for developing assembly language code for Microchip's 12-, 14-, and 16-bit microcontroller families. Generically, MPASM will refer to the entire development platform including the macro assembler and utility functions.

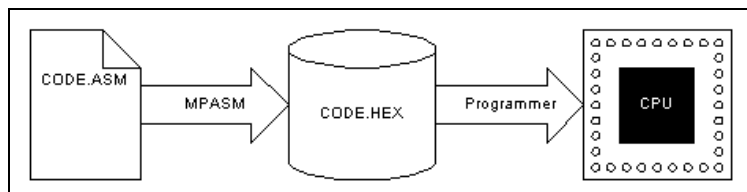
### Using MPASM

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. 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.

This process is shown below.



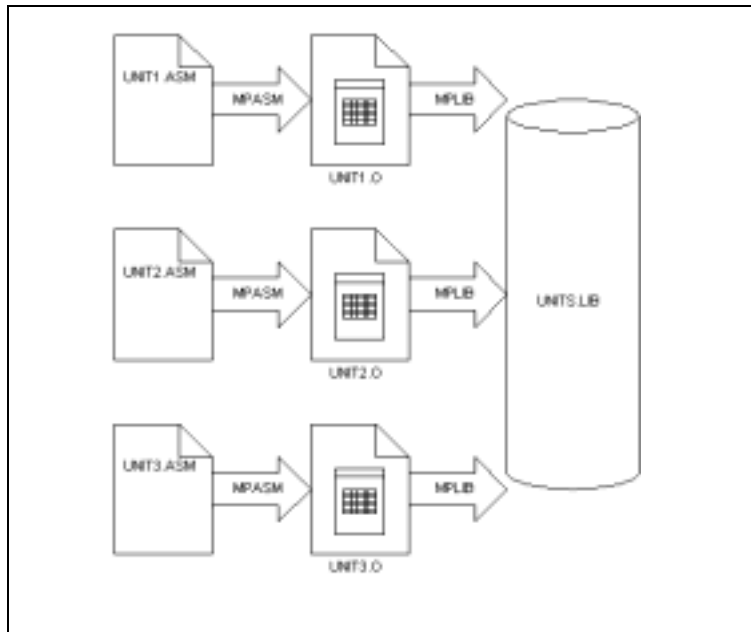
**Figure 1.1 Generating Absolute Code**

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.

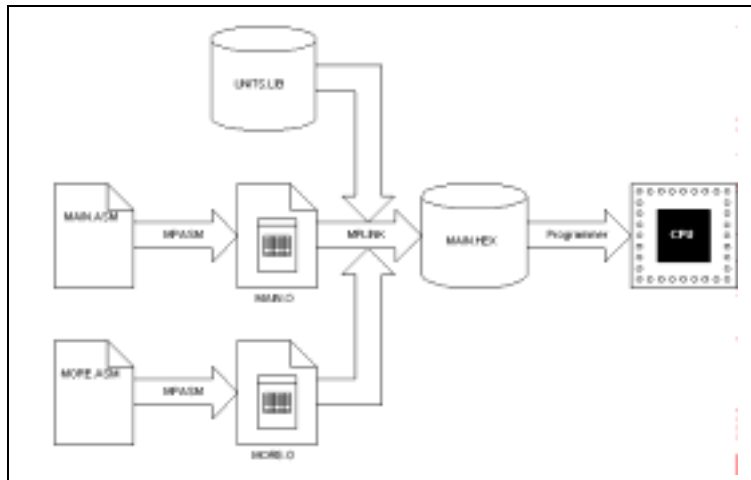
A visual representation of this process follows.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---



**Figure 1.2** Creating a Reusable Object Library



**Figure 1.3** Generating Executable Code From Object Modules

Refer to Chapter 4, "Using MPASM to Create Relocatable Objects" for more information on the differences between absolute and object assembly.

---

---

## Documentation Layout

The documentation is intended to describe how to use the assembler and its environment. It also provides some basic information about specific Microchip microcontrollers and their instruction sets, but detailed discussion of these issues is deferred to the data sheets for specific microcontrollers.

### Part 1 - MPASM

**Chapter 1: Introduction** – Introduces the user to MPASM. It describes the User's Guide layout, general conventions and terms, as well as a brief discussion of installation and platform requirements.

**Chapter 2: Environment and Usage** – This chapter describes the assembler's command line interface and shell interface. Also discussed here are the files used by MPASM, both input and output, including object file formats.

**Chapter 3: Directive Language** – This chapter describes the native directive language of MPASM.

**Chapter 4: Using MPASM to Create Relocatable Objects** – Information on the differences between absolute and object assembly.

**Chapter 5: Macro Language** – This chapter describes the macro language of MPASM. Macros are best learned by example; several will be offered for consideration.

**Chapter 6: Expression Syntax and Operation** – This chapter describes the expression syntax of MPASM, including operator precedence, radix override notation, examples and discussion.

### Part 2 - MPLINK

**Chapter 1: Introduction**

**Chapter 2: Usage**

**Chapter 3: Command File**

**Chapter 4: Linker Map File**

**Chapter 5: Linker Processing**

**Chapter 6: Terminology**

### Part 3 - MPLIB

**Chapter 1: Librarian Fundamentals**

**Appendix A: Hex File Formats** – A brief reference.

**Appendix B: On Line Support** - Information on Microchip's electronic support services.

**Appendix C: MPASM Errors/Warnings/Messages** – A list of the error messages generated by MPASM, with descriptions.

**Appendix D: MPLINK Errors/Warnings**

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

**Appendix E: Quick Reference** – A concise listing of all instructions for the MPASM Assembler.

**Appendix F: Example Initialization Code**

**Index:** A keyword cross reference to important topics and keywords.

**Table 1.1 Documentation Conventions**

Character	Represents
Square Brackets ( [ ] )	Optional Arguments
Angle Brackets ( < > )	Delimiters for special keys: <TAB>, <ESC>, or additional options.
Pipe Character (   )	Choice of mutually exclusive arguments; an OR selection.
Lowercase characters	Type of data
Courier Font	User entered code or sample code.

## Terms

In order to provide a common frame of reference, the following terms are defined:

### Assemble

The act of executing the MPASM macro assembler to translate source code to machine code.

### 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.

### Hex File

This file contains the actual machine code that can be programmed into a microcontroller or memory device. It is in a format that is readable by a device programmer.

### 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 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.

## Macro

A macro consists of a sequence of assembler commands. Passing arguments to a macro allows for flexible functionality. Macros are a form of "short hand" notation that will be expanded by MPASM.

## 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**.

## Object File

An object file can be created to provide the user with a relocatable module that can be linked with other modules. Special directives are required in the source code when generating an object file.

## PC

Any IBM or compatible Personal Computer.

## PICmicro

PICmicro refers to any Microchip microcontroller, including the representatives of the PIC12CXXX, PIC14XXX, PIC16C5X, PIC16CXX, and PIC17CXX families.

## Source Code

This is the ASCII text file 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™. It is an ASCII file that can be created using any ASCII text editor.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Recommended Reading

This manual is intended to provide a reference to using the MPASM development environment. It is not intended to replace reference material regarding specific PICmicro MCUs. Therefore, you are urged to read the Data Sheets for the PICmicro MCU specified by your application.

If this is your first microcontroller application, you are encouraged to review the Microchip *"Embedded Control Handbook."* You will find a wealth of information about applying PICmicro MCUs. The application notes described within are available from the Microchip BBS and Internet web site (see Appendix B).

All of these documents are available from your local sales office or from your Microchip Field Application Engineer (FAE).

## System Requirements

MPASM will run on any PC/AT or compatible computer, running DOS v5.0 or greater. MPASM for Windows requires Windows 3.1 or greater.

No special display or ancillary devices are required.

## Warranty Registration

<p><b>Note:</b> Upon receiving the diskette you should complete and return the Warranty Registration Card enclosed with the disk, and mail it promptly. Doing so will help to ensure that you receive product updates and notification of interim releases that become available.</p>
---

## Installation

MPASM for Windows is installed through the MPLAB installation procedure. For details, refer to the MPLAB User's Guide.

## Compatibility Issues

MPASM is compatible with all Microchip development systems currently in production. This includes MPLAB-SIM (PICmicro MCU discrete-event simulator), MPLAB PICMASTER<sup>®</sup> (PICmicro MCU Universal In-Circuit Emulator), PRO MATE<sup>®</sup> (the Microchip Universal Programmer), and PICSTART<sup>®</sup>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.

### Customer Support

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. Technical support questions should first be directed to your distributor and representative, local sales office, Field Application Engineer (FAE), or Corporate Applications Engineer (CAE).

The Microchip Internet Home Page can provide you with technical information, application notes and promotional news on Microchip products and technology. The Microchip Web address is <http://www.microchip.com>.

You can also check with the Microchip BBS (Bulletin Board System) for non-urgent support, customer forums, and the latest revisions of Microchip systems development products. Refer to Appendix B for access information.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:

---

## **Chapter 2. Environment and Usage**

---

### **Introduction**

MPASM provides a universal platform for developing code for PICmicro MCUs. This chapter is dedicated to describing the MPASM command line interface and the MPASM shells.

### **Highlights**

The points that will be highlighted in this chapter are:

- **MPASM Command Line Interface**
- **MPASM Shell Interfaces**
- **MPASM Input Files**
- **MPASM Output Files**

### **Terms**

#### **Alpha Character**

Alpha characters are those characters, regardless of case, that are normally contained in the alphabet: (a, b, ..., z, A, B, ..., Z).

#### **Alpha Numeric**

Alpha numeric characters include alpha characters and numbers: (0,1, ..., 9).

#### **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.

#### **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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Command Line Interface

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

```
MPASM [/<Option>[ ,/<Option>...]] [<file_name>]
```

or

```
MPASMWIN [/<Option>[ ,/<Option>...]] [<filename>]
```

Where

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

<file\_name> - 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 in Table ) 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 filename is omitted, the appropriate shell interface is invoked.

**Table 2.1 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.
h	N/A	Displays the MPASM Help Panel
l	On	Enable/Disable/Set Path for the listing file.
m	On	Enable/Disable macro expansion
o	Off	Enable/Disable/Set Path for the object file.

Table 2.1 Assembler Command Line Options (Continued)

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<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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## 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.

```
MPASM 01.21.15 Intermediate (c)1993-95 Microchip Technology Inc./Byte Craft Limi
Source File : *.ASM
Processor Type : None
Error File : Yes
Cross Reference File : No
Listing File : Yes
Hex Dump Type : INHX8M .HEX
Assemble to Object File : No

↑↓,Tab : Move Cursor      Esc : Quit      Type the name of your source file.
F1      : Help            F10 : Assemble
```

### 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>.

### 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>.

### 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 file name can be changed by pressing the <TAB> key to move to the shaded area and typing a new name. Wild cards are not allowed.

### Cross Reference File

Modify this field as for the Error File. It is used to optionally create a cross reference file (<sourcename>.XRF). The name may be modified as for error file and again, wild cards are not allowed.



### Listing File

Modify this field the same as for the error file. It is used to optionally disable the listing file. The output file name may be modified as for the error file.

### 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 file name, press the <TAB> key to move the shaded area, and type in the new name.

### 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 file name may be modified in the same manner as the error file.

## Windows Shell Interface

MPASM for Windows provides an interface window which can set various options. It is invoked by executing MPASMWIN . EXE while in Windows.



# 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.

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 Chapter 2, "Command Line Interface".
Save Settings on Exit	Save these settings in MPLAB.INI.

## Source Code Formats

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.

One or more spaces 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 2.2 Sample MPASM Source Code**

```
;
; 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
      goto    Start

      end
_
```

**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 alpha numeric 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.

**Mnemonics**

Assembler instruction mnemonics, assembler directives and macro calls must begin in column 2 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.

**Operands**

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

**Comments**

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Files Used by MPASM and Utility Functions

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

**Table 2.3 MPAM Default Extensions**

Extension	Purpose
.ASM	Default source file extension input to MPASM: <source_name>.ASM
.LST	Default output extension for listing files generated from MPASM: <source_name>.LST
.ERR	Output extension from MPASM for specific error files: <source_name>.ERR
.HEX	Output extension from MPAS 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

## Hex File Formats

MPASM is capable of producing different hex file formats. See Appendix A, "Hex File Formats".

## Listing File Format

Example 2.4 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

```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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, four characters wide, contains the base address in memory where the code will be placed. The second column, indented two spaces, displays the 32-bit value of any symbols created with the SET, EQU, VARIABLE, CONSTANT, or CBLOCK directives. The third column, also four characters wide, 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.

## 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.



## Chapter 3. Directive Language

### 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.

### Highlights

There are four basic types of directives provided by MPASM:

- **Control Directives**
- **Data Directives**
- **Listing Directives**
- **Macro Directives**
- **Object File Directives**

### Terms

#### 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

**Table 3.1 Directive Summary**

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



Table 3.1 Directive Summary (Continued)

Directive	Description	Syntax
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 <processor_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>"
TITLE	Specify Program Title	title "<title_text>"
UDATA	Begins uninitialized data section	[<name>] udata [<address>]
UDATA_OVR	Begins overlaid 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>

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Directive Details

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

## **\_\_BADRAM – Identify Unimplemented RAM**

### Syntax

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

### 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 precede 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`.

### Example

See the example for `__MAXRAM`.

### See Also

`__MAXRAM`

## **BANKISEL – Generate Indirect Bank Selecting Code**

### Syntax

```
bankisel <label>
```

### 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 4, “Using MPASM to Create Relocatable Objects.”

### Example

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

### See Also

PAGESEL    BANKSEL

## BANKSEL – Generate Bank Selecting Code

### Syntax

```
banksel <label>
```

### 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. If the device contains only one bank of RAM, no instructions will be generated.

For more information, refer to Chapter 4, “Using MPASM to Create Relocatable Objects.”

### Example

```
banksel Var1
movwf    Var1
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## See Also

PAGESEL    BANKISEL

## CBLOCK – Define a Block of Constants

### Syntax

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

### 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.

### 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
        QueueHeadm QueueTail
        Double1:2, Double2:2
endc
```

**See Also**

ENDC

**CODE – Begin an Object File Code Section****Syntax**

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

**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 zero if no address is specified.

For more information, refer to Chapter 4, “Using MPASM to Create Relocatable Objects.”

**Example**

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

**See Also**

IDATA UDATA UDATA\_OVR UDATA\_SHR EXTERN GLOBAL

**\_\_CONFIG – Set Processor Configuration Bits****Syntax**

```
_ _config <expr>
```

**Description**

Sets the processor's configuration bits to the value described by <expr>. 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.

**Example**

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## See Also

LIST      PROCESSOR    \_\_IDLOCS

## CONSTANT – Declare Symbol Constant

### Syntax

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

### 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.

### 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
```

## See Also

SET            VARIABLE

## DATA – Create Numeric and Text Data

### Syntax

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

### 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 with the first character in the most significant byte of the word. If an odd number of characters are given in a string, the final byte is 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.

### 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
```

### See Also

`DW`   `DB`   `DE`   `DT`   `IDATA`

## DB – Declare Data of One Byte

### Syntax

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

### 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.

### Example

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

### See Also

`DATA`   `DW`   `DT`   `DE`   `IDATA`

## DE – Declare EEPROM Data Byte

### Syntax

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

### 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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

## Example

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

## See Also

**DATA**    **DB**        **DT**        **DW**

## #DEFINE – Define a Text Substitution Label

### Syntax

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

### 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.

### 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
```

## See Also

**IFDEF**    **IFNDEF**    **#UNDEFINE**

## DT – Define Table

### Syntax

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



## 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.

## Example

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

## See Also

DATA DB DE DW

## DW – Declare Data of One Word

### Syntax

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

### Description

Reserve program memory words for data, initializing that space to specific values. 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.

### Example

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

### See Also

DATA DB IDATA

## ELSE – Begin Alternative Assembly Block to IF

### Syntax

```
else
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## 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.

## Example

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

## See Also

`IF`      `ENDIF`

## END – End Program Block

### Syntax

```
end
```

### Description

Indicates the end of the program.

### Example

```
start
.                ; executable code
.                ;
.                ;
                end                ; end of instructions
```

## ENDC – End an Automatic Constant Block

### Syntax

```
endc
```

**Description**

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

**See Also**

CBLOCK

**ENDIF – End Conditional Assembly Block****Syntax**

```
endif
```

**Description**

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

**See Also**

IF ELSE

**ENDM – End a Macro Definition****Syntax**

```
endm
```

**Description**

Terminates a macro definition begun with MACRO.

**Example**

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

**See Also**

MACRO EXITM

**ENDW – End a While Loop****Syntax**

```
endw
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## 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.

## Example

See the example for WHILE

## See Also

WHILE

## EQU – Define an Assembler Constant

### Syntax

```
<label>      equ      <expr>
```

### Description

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

### Example

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

### See Also

SET

## ERROR – Issue an Error Message

### Syntax

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

### Description

<text\_string> is printed in a format identical to any MPASM error message. <text\_string> may be from one to eighty characters.

### Example

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

```
endm
```

### See Also

MESSG

## ERRORLEVEL – Set Message Level

### Syntax

```
errorlevel 0|1|2|<+|-><msgnum>
```

### 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.

### Example

```
errorlevel 1, -202
```

### See Also

LIST

## EXITM – Exit from a Macro

### Syntax

```
exitm
```

### Description

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

### Example

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

```
endif
endm
```

## See Also

**MACRO**      **ENDM**

## EXPAND – Expand Macro Listing

### Syntax

```
expand
```

### Description

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

## See Also

**MACRO**    **NOEXPAND**

## EXTERN – Declare an Externally Defined Label

### Syntax

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

### 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. No <label> can be defined in the current module.

For more information, refer to Chapter 4, "Using MPASM to Create Relocatable Objects."

### Example

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

## See Also

**TEXT**   **IDATA**   **UDATA**   **UDATA\_OVR**   **UDATA\_SHR**   **GLOBAL**

## FILL – Specify Memory Fill Value

### Syntax

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

### Description

Generates <count> occurrences of the program word <expr>. If bounded by parentheses, <expr> can be an assembler instruction.

### Example

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

### See Also

DW ORG DATA

## GLOBAL – Export a Label

### Syntax

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

### 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 4, “Using MPASM to Create Relocatable Objects.”

### Example

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## See Also

TEXT IDATA UDATA UDATA\_OVR UDATA\_SHR EXTERN

## IDATA – Begin an Object File Initialized Data Section

### Syntax

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

### 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 zero if no address is specified. No code can be generated in this segment.

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 Appendix E 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 4, "Using MPASM to Create Relocatable Objects."

### Example

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

### See Also

TEXT UDATA UDATA\_OVR UDATA\_SHR EXTERN GLOBAL



## \_\_IDLOCS – Set Processor ID Locations

### Syntax

```
__idlocs <expr>
```

### Description

Sets the four ID locations to the hexadecimal digits of the value of `<expr>`. 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. This directive is not valid for the PIC17CXX family.

### Example

```
__idlocs H'1234'
```

### See Also

`LIST`    `PROCESSOR`    `__CONFIG`

## IF – Begin Conditionally Assembled Code Block

### Syntax

```
if <expr>
```

### 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 non-zero value, `FALSE` a value of zero.

### Example

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

```
        movwf  io_2
    endif
```

## See Also

ELSE ENDIF

## IFDEF – Execute If Symbol has Been Defined

### Syntax

```
ifdef  <label>
```

### 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.

### Example

```
#define  testing1      ; set testing "on"
        .
        .
        .
ifdef   testing
        <execute test code>; this path would
endif                                     ; be executed.
```

## See Also

#DEFINE ELSE ENDIF  
IFDEF #UNDEFINE

## IFNDEF – Execute If Symbol has not Been Defined

### Syntax

```
ifndef  <label>
```

### 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.

**Example**

```

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

```

**See Also**

**#DEFINE**      **ELSE**  
**IFDEF**        **#UNDEFINE**      **ENDIF**

**INCLUDE – Include Additional Source File****Syntax**

```

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

```

**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 placed here. 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.

**Example**

```

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

```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## LIST – Listing Options

### Syntax

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

### 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 3.2 List Directive Options**

Option	Default	Description
<code>b=nnn</code>	8	Set tab spaces.
<code>c=nnn</code>	132	Set column width.
<code>f=&lt;format&gt;</code>	INHX8M	Set the hex file output. <code>&lt;format&gt;</code> can be INHX32, INHX8M, or INHX8S.
<code>free</code>	FIXED	Use free-format parser. Provided for backward compatibility.
<code>fixed</code>	FIXED	Use fixed-format parser.
<code>mm=ON OFF</code>	On	Print memory map in list file.
<code>n=nnn</code>	60	Set lines per page.
<code>p=&lt;type&gt;</code>	None	Set processor type; for example, PIC16C54.
<code>r=&lt;radix&gt;</code>	hex	Set default radix: hex, dec, oct.
<code>st=ON OFF</code>	On	Print symbol table in list file.
<code>t=ON OFF</code>	Off	Truncate lines of listing (otherwise wrap).
<code>w=0 1 2</code>	0	Set the message level. See <code>ERRORLEVEL</code> .
<code>x=ON OFF</code>	On	Turn macro expansion on or off.

**Note:** All `LIST` options are evaluated as decimal numbers.

### Example

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

### See Also

`NOLIST`    `PROCESSOR`    `RADIX`    `ERRORLEVEL`  
`EXPAND`    `NOEXPAND`

## LOCAL – Declare Local Macro Variable

### Syntax

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

## 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.

## 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
```

## See Also

**MACRO**    **ENDM**

# MACRO – Declare Macro Definition

## Syntax

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

## 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.

## Example

```
Read   macro  device,  buffer, count
        movlw device
        movwf  ram_20
        movlw  buffer  ; buffer address
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

```
movwf    ram_21
movlw    count    ; byte count
call     sys_21   ; read file call
endm
```

## See Also

ENDM    LOCAL    IF    ELSE  
ENDIF    EXITM

## \_\_MAXRAM – Define Maximum RAM Location

### Syntax

```
__maxram <expr>
```

### 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 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.

### 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
```

## See Also

`__BADRAM`

---

---

## MESSG – Create User Defined Message

### Syntax

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

### 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.

### Example

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

### See Also

ERROR

## NOEXPAND – Turn off Macro Expansion

### Syntax

```
noexpand
```

### Description

Turns off macro expansion in the listing file.

### See Also

EXPAND

## NOLIST – Turn off Listing Output

### Syntax

```
nolist
```

### Description

Turn off listing file output.

### See Also

LIST

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## ORG – Set Program Origin

### Syntax

```
[<label>]    org    <expr>
```

### 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.

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

### Example

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

### See Also

RES FILL

## PAGE – Insert Listing Page Eject

### Syntax

```
page
```

### Description

Inserts a page eject into the listing file.

### See Also

LIST TITLE SUBTITLE

## PAGESEL – Generate Page Selecting Code

### Syntax

```
pagesel    <label>
```

### 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 more information, refer to Chapter 4, “Using MPASM to Create Relocatable Objects.”

### Example

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

### See Also

**BANKSEL** **BANKISEL**

## PROCESSOR – Set Processor Type

### Syntax

```
processor    <processor_type>
```

### Description

Sets the processor type to <processor\_type>.

### Example

```
processor    16C54
```

### See Also

**LIST**

## RADIX – Specify Default Radix

### Syntax

```
radix    <default_radix>
```

### Description

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Example

```
radix          dec
```

## See Also

LIST

## RES – Reserve Memory

### Syntax

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

### Description

Causes the program counter to be advanced from its current location by the value specified in `<mem_units>`. Note that `<label>` will be installed as an address as opposed to a constant or variable.

### Example

```
buffer        res    64    ; reserve 64 words of storage
```

### See Also

ORG FILL

## SET – Define an Assembler Variable

### Syntax

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

### Description

`<label>` assumes 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.

### Example

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

```
length      set      length + 1
```

**See Also**

EQU    VARIABLE

**SPACE – Insert Blank Listing Lines****Syntax**

```
space                    <expr>
```

**Description**

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

**Example**

```
space            3            ;Inserts three blank lines
```

**See Also**

LIST

**SUBTITLE – Specify Program Subtitle****Syntax**

```
subtitle    "<sub_text>"
```

**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.

**Example**

```
subtitle    "diagnostic section"
```

**See Also**

TITLE

**TITLE – Specify Program Title****Syntax**

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## 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.

## Example

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

## See Also

LIST      SUBTITLE

## UDATA – Begin an Object File Uninitialized Data Section

### Syntax

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

### 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 zero if no address is specified. No code can be generated in this segment. The RES directive should be used to reserve space for data.

For more information, refer to Chapter 4, "Using MPASM to Create Relocatable Objects."

### Example

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

### See Also

TEXT IDATA UDATA\_OVR UDATA\_SHR EXTERN GLOBAL

## UDATA\_OVR – Begin an Object File Overlaid Uninitialized Data Section

### Syntax

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

## 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 zero if no address is specified. The space declared by this section is overlaid by all other `udata_ovr` sections of the 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.

For more information, refer to Chapter 4, “Using MPASM to Create Relocatable Objects.”

## Example

```
Temps      udata_ovr
Temp1     res    1
Temp2     res    1
Temp3     res    1
Temps      udata_ovr
LongTemp1 res    2
LongTemp2 res    2
```

## See Also

TEXT IDATA UDATA EXTERN GLOBAL UDATA\_SHR

# UDATA\_SHR – Begin an Object File Shared Uninitialized Data Section

## Syntax

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

## 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 zero 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.

For more information, refer to Chapter 4, “Using MPASM to Create Relocatable Objects.”

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Example

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

## See Also

TEXT IDATA UDATA EXTERN GLOBAL UDATA\_OVR

## #UNDEFINE – Delete a Substitution Label

### Syntax

```
#undefine <label>
```

### 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.

### Example

```
#define          length20
.
.
.
#undefine      length
```

## See Also

#DEFINE IFDEF INCLUDE IFNDEF

## VARIABLE – Declare Symbol Variable

### Syntax

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

### 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.

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

### Example

Please refer to the `CONSTANT` example.

### See Also

`SET`    `CONSTANT`

## WHILE – Perform Loop While Condition is True

### Syntax

```
while <expr>
    .
    .
endw
```

### 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.

### Example

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

### See Also

`ENDW`    `IF`

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:





## Chapter 4. Using MPASM to Create Relocatable Objects

### Introduction

With the introduction of MPASM v2.00 and MPLINK v1.00, users have 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. Existing MPASM routines will require minor modifications to compile correctly into relocatable object modules.

### Highlights

- Use of Header Files
- Program Memory Definition
- 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 Example

### Header Files

The Microchip supplied standard header files should be used when generating object modules. These header files define the special function registers for the target processor.

### Program Memory

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

#### Absolute Code:

```
Start  CLRW  
      OPTION  
      -
```

#### Relocatable Code:

```
CODE  
Start  CLRW  
      OPTION  
      -
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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

## Example Relocatable Code:

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

```
Main      CODE  
           CLRW  
           OPTION
```

## Instruction Operands

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

```
[HIGH|LOW] (<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.

## RAM Allocation

RAM space must be allocated in a data section. Three 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\_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 look-up 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.

### Absolute Code:

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

### 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`, 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Relocatable Code:

```
                                00001      LIST      p=17C44
                                00002      IDATA
0000 01 02 03      00003  Bytes  D8      1,2,3
0003 34 12 78 56 00004  Words  DW      H'1234',H'5678'
0007 48 69 00      00005  String DB      "Hi", 0
```

## 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.

## 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.

### Relocatable Code, Defining Module:

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

                                CODE
Filter
                                GLOBAL  Filter
                                -                ; Filter code
```

### 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
```

## 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.

### 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

```

### Relocatable Code:

```

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

Var1 EQU H'10'
Var2 EQU H'30'
...
CODE
MOVLW InitialValue
BANKSEL Var1
MOVWF Var1
BANKSEL Var2
MOVWF Var2
PAGESEL Subroutine
CALL Subroutine
...
Subroutine CLRW
...
RETLW 0

```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## 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.

### Absolute Code:

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

### Relocatable Code:

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

## 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.

## Example

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

### Absolute Code:

```
          LIST      P=16C54  
          #INCLUDE  "P16C5x.INC"  
  
mulcnd   EQU      09          ; 8 bit multiplicand  
mulplr   EQU      10          ; 8 bit multiplier  
H_byte   EQU      12          ; High byte of the 16 bit result  
L_byte   EQU      13          ; Low byte of the 16 bit result  
count    EQU      14          ; loop counter  
  
mpy      clr     H_byte  
          clr     L_byte  
          movlw   8  
          movwf  count  
          movf   mulcnd,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
;*****
;                               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 8x8 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.

### Relocatable Code, Calling File

```

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

EXTERN  mulcnd, mulplr, H_byte, L_byte
EXTERN  mpy

start   CODE
        clrw
        option

main    movf    PORTB, W
        movwf   mulplr
        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

```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## 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

        clr    H_byte
        clr    L_byte
        movlw  8
        movwf  count
        movf   muland, W
loop    baf    STATUS, C    ; Clear carry bit
        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
```



---

## **Chapter 5. Macro Language**

---

### **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.

### **Highlights**

The points that will be highlighted in this chapter are:

- **Macro Syntax**
- **Text Substitution**
- **Local Symbols**
- **Recursive Macros**
- **Macro Usage**
- **Examples**

### **Terms**

#### **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 (see Chapter 3) are part of the macro definition. Labels used within the macro must be local to the macro so the macro can be called repetitively.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Local Label

A local label is one that is defined with the `LOCAL` directive (see Chapter 3). 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.

## 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.

## Nesting Depth

Macros can be nested to sixteen levels deep.

## Macro Syntax

MPASM macros are defined according to the following syntax:

```
<label>macro      [ <arg1>, <arg2> ... , <arg0> ]  
.  
.  
.  
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.

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 3. MPASM continues to process the body of the macro until an `EXITM` or `ENDM` directive is encountered.

<b>Note:</b> Forward references to macros are not permitted.
--

## Macro Directives

There are directives that are unique to macro definitions. They cannot be used out of the macro context (refer to Chapter 3 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.

## Text Substitution

String replacement and parsing patterns 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:

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

would generate:

```
entry0 dw0
entry1 dw0
entry2 dw0
entry3 dw0
```

when invoked.

## 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

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 3) 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.

## Examples

### 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.

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 `TRUE`. The entire macro is terminated by the `ENDM` directive.

### 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_jgeswitch_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 6. Expression Syntax and Operation

### Introduction

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

### Highlights

The points that will be highlighted in this chapter are:

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

### Terms

#### 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).

#### 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.

#### 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.

#### 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 (See Chapter 3) and override the default radix with certain radix override operators.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## 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.

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 6.1 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	

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## 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 they are 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.

<b>Note:</b> 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 6.2 Radix Specifications**

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

Table 6.3 Arithmetic Operators and Precedence

Operator		Example
\$	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
*	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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## High/Low

### Syntax

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

### Description

The high operators are used to return the high byte or the low byte of a 16-bit label value. This is done to handle dynamic pointer calculations as might be used with table read and write instructions.

### 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
```

## Increment/Decrement

### Syntax

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

### 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.

### Example

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



# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Part 2 – MPLINK

---

---

Chapter 1.	Introduction .....	79
Chapter 2.	Usage .....	83
Chapter 3.	Command File .....	85
Chapter 4.	Linker Map File .....	89
Chapter 5.	Linker Processing .....	91
Chapter 6.	Terminology .....	95

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:



## Chapter 1. Introduction

### MPLINK Preview

#### What is MPLINK?

MPLINK is a linker for the Microchip C compiler, MPLAB-C, and the Microchip relocatable assembler, MPASM. MPLINK is introduced with MPLAB-C v1.75.xx (pre-release and beta test versions of v2.00) and MPASM v2.00 and can only be used with these or later versions.

#### How MPLINK Helps You

MPLINK allows you to produce modular, re-usable code with MPLAB-C and MPASM. 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.

#### What you need to know to use MPLINK

MPLINK is one tool in the MPASM and MPLAB-C tool set. You should be familiar with MPASM, and with MPLAB-C if you are using the compiler. You should understand the basic architecture of the PICmicros, especially the ROM and RAM memory maps. You should also familiarize yourself with MPLIB, the librarian.

### Product Description

MPLINK combines multiple input object modules generated by MPLAB-C or MPASM, into a single executable file. The actual addresses of data and the location of functions will be assigned when MPLINK is executed. This means that you will instruct MPLINK to place code and data somewhere within named regions of memory, not to specific physical locations.

Once the linker knows about the ROM and RAM memory regions available in the target PICmicro device and it analyzes all the input files, it will 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 re-usable, 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Libraries are available for most PICmicro peripheral functions as well as for many standard C functions. The linker will only extract and link individual functions 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 the symbolic information for debugging your application with MPLAB, and produces the HEX object file for the executable PICmicro code used by device programmers.

MPLINK is executed after assembling or compiling relocatable object modules with MPASM and/or MPLAB-C.

## File Formats

MPLINK uses and generates various different file types. MPLINK accepts three types of input files: object files, library files, and linker command files. The object files and library files used by MPLINK are COFF files which support relocatable object modules and robust symbolic debugging. The linker command files used by MPLINK are ASCII text files.

MPLINK generates five types of output files:

- COFF Output File: a binary file containing the relocated input sections, no unresolved externals, robust symbolic debug info, and all the necessary information to generate a COD, listing, and hex file.
- COD File: a binary file containing debug information. This is the debug format currently supported by MPLAB.
- Map File: an ASCII text file which lists information about sections and symbols.
- Listing File: an ASCII text file which mixes the source code with disassembled code.
- Hex File: a file used for device programming.

The linker first generates the output COFF file and map file. The linker then translates the output COFF to a COD, listing, hex programming file. The COD file format is the currently supported debugging file format for MPLAB.

## Linker Components

MPLINK is actually comprised of four separate programs. The first program, 'mplink.exe' is a shell program which invokes the other three programs. The user only interacts with the 'mplink.exe' shell program.



The shell program first invokes `'_mplink.exe'`, which is the actual linker, and passes it the command line options. The linker generates an output COFF file and map file. If the linker completes successfully, the shell program then invokes `'mp2cod.exe'` to translate the output COFF file into a COD file and a listing file. If this step completes successfully, the shell program then invokes `'mp2hex.exe'` to translate the COFF file into a HEX programming file.

## Tools and Supported Platforms

MPLINK is distributed in two executable formats: a Win32 console application suitable for Windows95 and WindowsNT 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 Win32 versions. DOS-extended versions require the DOS extender program `'DOS4GW.EXE'` which is also distributed with MPLINK.

As mentioned in the section "File Formats", MPLINK is comprised of four executables: the shell program `'mplink.exe'`, the actual linker `'_mplink.exe'`, a COFF to COD file translator `'mp2cod.exe'`, and a COD to HEX file translator, `'mp2hex.exe'`.

Additional software tools are distributed with MPLINK. These tools include a librarian `'mplib.exe'`, a COFF file disassembler `'mpdis.exe'`, a COFF dump utility `'mpcoff.exe'`, and a COD dump utility `'mpcod.exe'`.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:



## Chapter 2. Usage

### Command Line

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

'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 below.

**Table 2.1 Linker Command Line Options**

Option	Description
/o <filename>	specify output file 'filename', default is 'a.out'
/m <filename>	create map file 'filename'
/L <pathlist>	semi-colon delimited list of directories to search for library/object files
/K <pathlist>	semi-colon delimited list of directories to search for linker command files
/n <length>	number of lines per listing page (0 = no pagination)
/h, /?	show this help screen
/a <hexformat>	specify format of hex programming file. Valid formats are INHX8M, INHX8S, INHX32
/q	quiet mode operation

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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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'.

## Usage Example

Suppose there are two object files to be linked, 'a.o' and 'b.o', a linker command file 'lnk17C44.lkr', a library file 'math.lib', and we want the linker to generate a mapfile 'c.map', an output COFF file 'c.out', an output COD file 'c.cod' and an output programming file 'c.hex'. If the input files are in the current directory, the following command line would produce the desired results:

```
mplink -o c.out -m c.map a.o b.o lnk17C44.lkr
```

---

## Chapter 3. Command File

---

A linker command file is an ASCII text file which is processed by the linker. Linker command files can be created to control the operation of the linker. The linker command file is used:

1. To specify an additional directory for the library/object search path
2. To specify an additional directory for the linker command file search path
3. To specify additional object files and library files for linking
4. To include additional linker command files
5. To define the target processor's memory architecture
6. To locate sections within the target processor's memory
7. To specify the size of the stack and, optionally, the target memory where it resides

Each line in a linker command file is either a directive or a comment. Any text following a '/' is ignored. The following directives are supported, each one is described below: LIBPATH, LKRPATH, FILES, INCLUDE, DATABANK, CODEPAGE, SHAREBANK, SECTION, and STACK.

### Directives

#### LIBPATH Directive:

Library and object files which do not have a path are searched for 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.

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 Directive:

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'
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

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 Directive:

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 Directive:

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
```

## DATABANK, CODEPAGE, SHAREBANK Directives:

The following directives define portions of the target's memory by specifying a name for a block of memory, its starting address, and its ending address:

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

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

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

where,

'memName' is any ASCII string used to identify a DATABANK, CODEPAGE, or SHAREBANK

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

# Chapter 3. Command File

'fillValue' is a 16 bit quantity which fills any unused portion of a memory block. Only CODEPAGE directives may have a FILL attribute.

The DATABANK and SHAREBANK directives define data memory, the CODEPAGE directive is used to define program memory.

The SHAREBANK directive identifies a region in RAM which is mapped across multiple banks. Note, a SHAREBANK directive should be given for each bank that shares a region and each of these directives should have the same NAME. Not all PICmicros have shared RAM. Check the data book and processor specific assembly files.

The PROTECTED attribute marks a memory block and prevents the linker from placing unassigned relocatable sections into the memory block. The PROTECTED marking does not prevent the linker from placing absolute sections or assigned relocatable sections into the marked memory block.

Example:

To specify a logical block of program memory named 'constants' that begins at address '0x100' and ends at address '0x1FF', the following line should be added to the linker command file:

```
CODEPAGE NAME=constants START=0x100 END=0x1FF
```

Another Example:

The following example specifies a physical block of data memory named 'sfr' which is shared across four banks. The SHAREBANK's begin at offset '0x0' in each bank and end at offset '0xF' in each bank. Each bank is 0x100 bytes long. The PROTECTED attribute prevents the linker from placing any unassigned relocatable sections into the memory block.

```
SHAREBANK NAME=sfr START=0x000 END=0x00F PROTECTED
SHAREBANK NAME=sfr START=0x100 END=0x10F PROTECTED
SHAREBANK NAME=sfr START=0x200 END=0x20F PROTECTED
SHAREBANK NAME=sfr START=0x300 END=0x30F PROTECTED
```

## SECTION Directive:

The following 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, this is the same name for the section in the COFF file

'memName' is a previously defined 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 DATABANK or SHAREBANK directive.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

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:

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

## STACK Directive:

MPLAB-C usually requires a software stack be set up. The following statement specifies the stack size and an optional target memory 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 DATABANK or SHAREBANK statement.

Example:

To set the stack size to be '0x20', the following line should be added to the linker command file:

```
STACK SIZE=0x20 RAM=gpr0
```

## Linker Command File Example:

The following is an example linker command file for the PIC17C44.

```
INCLUDE user.lkr           // Include another linker command file
LIBPATH c:\projects\current// Add a directory to the search path

CODEPAGE NAME=vectors    START=0x0END=0x20// ROM area for reset/int vectors
CODEPAGE NAME=page0      START=0x21END=0x1FFF// User program memory area

DATABANK NAME=sfr0       START=0x10END=0x17//Special function reg's in RAM bank 0
DATABANK NAME=sfr1       START=0x110END=0x117//Special function reg's in RAM bank 1
DATABANK NAME=sfr2       START=0x210END=0x217//Special function reg's in RAM bank 2
DATABANK NAME=sfr3       START=0x310END=0x317//Special function reg's in RAM bank 3

DATABANK NAME=gpr0       START=0x20END=0xFF//General purpose RAM bank 0
DATABANK NAME=gpr1       START=0x120END=0x1FF//General purpose RAM bank 1

SHAREBANK NAME=sfrShare  START=0x0END=0xF//Shared unbanked SFR's
SHAREBANK NAME=sfrShare  START=0x100END=0x10F//Shared unbanked SFR's
SHAREBANK NAME=sfrShare  START=0x200END=0x20F//Shared unbanked SFR's
SHAREBANK NAME=sfrShare  START=0x300END=0x30F//Shared unbanked SFR's

SECTION NAME=.cinit      ROM=page0    // .cinit section resides in ROM
SECTION NAME=.code       ROM=page0    // .code section resides in ROM
SECTION NAME=.udata      RAM=gpr0     // Unitialized data occupies RAM

STACK SIZE=0x20 RAM=gpr0// Specify stack size and location
```



## Chapter 4. Linker Map File

### Linker Map File

As an option, a map file can be generated by the linker. The map file contains three tables. The first table 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.

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 the section table in a map file:

Section	Section Info			
	Type	Address	Location	Size(Bytes)
Reset	code	0x0000	program	0x0002
.cinit	romdata	0x0021	program	0x0004
.code	code	0x0023	program	0x0026
.udata	udata	0x0020	data	0x0005

The second table in the map file 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	0x0026	program	static	
C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm				
loop	0x002e	programstatic		C:\MPASMV2\MUL8X8.ASM
main	0x0024	programstatic		C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
mpy	0x0028	programextern		C:\MPASMV2\MUL8X8.ASM
start	0x0023	program	static	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
H_byte	0x0022	data	extern	C:\MPASMV2\MUL8X8.ASM
L_byte	0x0023	data	extern	C:\MPASMV2\MUL8X8.ASM
count	0x0024	data	static	C:\MPASMV2\MUL8X8.ASM
mulcnd	0x0020	data	extern	C:\MPASMV2\MUL8X8.ASM
mulplr	0x0021	data	extern	C:\MPASMV2\MUL8X8.ASM

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

The third table in the map file 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 table sorted by address in a map file:

Symbols - Sorted by Address			
Name	Address	Location	Storage File
start	0x0023	programstatic	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
main	0x0024	programstatic	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
call_m	0x0026	programstatic	C:\PROGRA~1\MPLAB\ASMFOO\sampobj.asm
mpy	0x0028	programextern	C:\MPASMV2\MUL8X8.ASM
loop	0x002e	programstatic	C:\MPASMV2\MUL8X8.ASM
mulcnd	0x0020	data extern	C:\MPASMV2\MUL8X8.ASM
mulplr	0x0021	data extern	C:\MPASMV2\MUL8X8.ASM
H_byte	0x0022	data extern	C:\MPASMV2\MUL8X8.ASM
L_byte	0x0023	data extern	C:\MPASMV2\MUL8X8.ASM
count	0x0024	data static	C:\MPASMV2\MUL8X8.ASM

## Error Map File

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 5. Linker Processing

---

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 dependant 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.

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

### 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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

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

## 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.

## 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.

## 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 */
}
```

## Chapter 5. Linker Processing

Suppose this compiles into the following assembly instructions (note: this example deliberately ignores any code generated by MPLAB-C 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
```

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
```

## 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 non-volatile 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-C and MPASM) to the proper RAM location(s) at startup.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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 6. Terminology

---

### Terminology

Absolute Section:	A section with a fixed absolute address which can not be changed by the Linker.
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.
COD:	Common Object Description - a file format definition for executable files created by Byte Craft Limited of Waterloo, Canada.
COFF:	Common Object File Format - a file format definition for object/executable files.
Identifier:	A function or variable name.
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 File:	An ASCII file containing hexadecimal addresses and values suitable for programming a device.
Initialized Data:	Data which is defined with an initial value. In C, <code>int myVar=5;</code> 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

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.
Raw Data:	The binary representation of code or data associated with a section.
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.
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.
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, <code>int myVar;</code> defines a variable which will reside in an uninitialized data section.





# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Part 3 – MPLIB

---

---

Chapter 1. Librarian Fundamentals .....	99
---	----

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

Notes:



## Chapter 1. Librarian Fundamentals

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'.

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

### Usage

```
mplib [/q] /{ctdrrx} 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

---

---

## 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
```

## 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.

## Error Reporting

MPLIB detects the following sources of error and reports them:

### 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'.

### Library File Errors

Please refer to the documentation on MPLINK for error messages associated with library files.

### COFF File Errors

Please refer to the documentation on MPLINK for error messages associated with COFF files.



---

---

## Appendix A. Hex File Formats

---

---

### Introduction

MPASM is capable of generating several different hex file formats.

### Highlights

- 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)

### Hex File Formats

#### 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

<file\_name>.HEX

```
:10000000000000000000000000000000000000000000F0
:040010000000000000EC
:100032000000280040006800A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

## 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

<file\_name>.HXL

```
:0A000000000000000000000000000000F6
:1000190000284068A8E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C8080808034303E8E8FFD0
:03003900FFFF19AD
:00000001FF
```

<file\_name>.HXH

```
:0A0000000000000000000000000000F6
:1000190000000000000000010101010102020202CA
:1000290002020303030304040404050607070883
:0300390008080AAA
:00000001FF
```

---

# Appendix A. Hex File Formats

---

## 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:

```
:BBAAATTHHHH. . . .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:**





# MPASM USER'S GUIDE with MPLINK and MPLIB

---

---

## Appendix B. On-Line Support

---

---

### Introduction

Microchip provides two methods of on-line support. These are the Microchip BBS and the Microchip World Wide Web (WWW) site.

Use Microchip's Bulletin Board Service (BBS) to get current information and help about Microchip products. Microchip provides the BBS communication channel for you to use in extending your technical staff with microcontroller and memory experts.

To provide you with the most responsive service possible, the Microchip systems team monitors the BBS, posts the latest component data and software tool updates, provides technical help and embedded systems insights, and discusses how Microchip products provide project solutions.

The web site, like the BBS, 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 or Microsoft Explorer. Files are also available for FTP download from our FTP site.

### Connecting to the Microchip Internet Web Site

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

**[www.microchip.com](http://www.microchip.com)**

The file transfer site is available by using an FTP service to connect to:

**<ftp://ftp.futureone.com/pub/microchip>**

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Datasheets, 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
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## Connecting to the Microchip BBS

Connect worldwide to the Microchip BBS using either the Internet or the CompuServe® communications network.

Internet: You can telnet or ftp to the Microchip BBS at the address **mchipbbs.microchip.com**

CompuServe Communications Network: In most cases, a local call is your only expense. The Microchip BBS connection does not use CompuServe membership services, therefore

**You do not need CompuServe membership to join Microchip's BBS.**

There is **no charge** for connecting to the BBS, except for a toll charge to the CompuServe access number, where applicable. You do not need to be a CompuServe member to take advantage of this connection (you never actually log in to CompuServe).

The procedure to connect will vary slightly from country to country. Please check with your local CompuServe agent for details if you have a problem. CompuServe service allow multiple users at baud rates up to 14400 bps.

The following connect procedure applies in most locations.

1. Set your modem to 8-bit, No parity, and One stop (8N1). This is not the normal CompuServe setting which is 7E1.
2. Dial your local CompuServe access number.
3. Depress **<Enter.>** and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type +, depress **<Enter.>** and Host Name: will appear.
5. Type **MCHIPBBS**, depress **<Enter.>** and you will be connected to the Microchip BBS.

In the United States, to find the CompuServe phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600-14400 baud connection. After the system responds with `Host Name :`, type **NETWORK**, depress **<Enter.>** and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 723-1550 for your local CompuServe number.

## Using the Bulletin Board

The bulletin board is a multifaceted tool. It can provide you with information on a number of different topics.

- Special Interest Groups
- Files
- Mail
- Bug Lists

---

# Appendix B. On-Line Support

---

## Special Interest Groups

Special Interest Groups, or SIGs as they are commonly referred to, provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. SIGs may provide you with information not available by any other method because of the broad background of the PICmicro MCU user community.

There are SIGs for most Microchip systems and device families. These groups are monitored by the Microchip staff.

## Files

Microchip regularly uses the Microchip BBS to distribute technical information, application notes, source code, errata sheets, bug reports, and interim patches for Microchip systems software products. Users can contribute files for distribution on the BBS. For each SIG, a moderator monitors, scans, and approves or disapproves files submitted to the SIG. No executable files are accepted from the user community in general to limit the spread of computer viruses.

## Mail

The BBS can be used to distribute mail to other users of the service. This is one way to get answers to your questions and problems from the Microchip staff, as well as keeping in touch with fellow Microchip users worldwide.

Consider mailing the moderator of your SIG, or the SYSOP, if you have ideas or questions about Microchip products, or the operation of the BBS.

## Software Releases

Software products released by Microchip are referred to by version numbers. Version numbers use the form:

`xx.yy.zz`

Where `xx` is the major release number, `yy` is the minor number, and `zz` is the intermediate number.

## Intermediate Release

Intermediate released software represents changes to a released software system and is designated as such by adding an intermediate number to the version number. Intermediate changes are represented by:

- Bug Fixes
- Special Releases
- Feature Experiments

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

Intermediate released software does not represent our most tested and stable software. Typically, it will not have been subject to a thorough and rigorous test suite, unlike production released versions. Therefore, users should use these versions with care, and only in cases where the features provided by an intermediate release are required.

Intermediate releases are primarily available through the BBS.

## **Production Release**

Production released software is software shipped with tool products. Example products are PRO MATE II, PICSTART, and PICMASTER. The Major number is advanced when significant feature enhancements are made to the product. The minor version number is advanced for maintenance fixes and minor enhancements. Production released software represents Microchip's most stable and thoroughly tested software.

There will always be a period of time when the Production Released software is not reflected by products being shipped until stocks are rotated. You should always check the BBS or the WWW for the current production release.

## **Systems Information and Upgrade Hot Line**

The Systems Information and Upgrade 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.

These phone numbers are also listed on the "Important Information" sheet that is shipped with all development systems. The hot line message is updated whenever a new software version is added to the Microchip BBS, or when a new upgrade kit becomes available.



---

## Appendix C. MPASM Errors/Warnings/Messages

---

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.

### 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 DPML 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 DPML 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.

**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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

- 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.
- 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.

## Appendix C. MPASM Errors/Warnings/Messages

---

- 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.
- 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

- 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.
- 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.



# Appendix C. MPASM Errors/Warnings/Messages

---

**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.

**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.

## 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

- 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.
- 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 superceded 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 superceded 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.

## Appendix C. MPASM Errors/Warnings/Messages

---

- 216 Radix superceded 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.
- 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.

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

## 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 superceded 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 superceded by command line.**  
Macro expansion was specified on the command line as well as in the source file. The command line has precedence.
- 310 Superceding current maximum RAM and RAM map.**  
The `__MAXRAM` directive has been used previously.
- 311 Operand of HIGH operator was larger than H'FFFF'.**  
The HIGH operator will return the value of the bits 8 through 15 of the operator, shifted down to the lower byte. Any bits above bit 15 are ignored. The expression HIGH (H'123456') evaluates to H'34'.
- 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.

## **Appendix C. 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

---

### 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.

**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>. 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"

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

**unknown switch: 'cmdline token'.** An unrecognized command line switch was supplied. Refer to the Usage documentation for the list of supported switches.

## 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.

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.



## Appendix D. MPLINK Errors/Warnings

---

**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.

**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

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

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.

## 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

# Appendix D. MPLINK Errors/Warnings

---

'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.

## COFF File Errors

All COFF file errors indicate an internal error in the file's contents. Please contact Microchip support if any of the 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.

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

- 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.

## **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.

## **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

## Appendix E. Quick Reference

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

**Table E.1 MPASM Directive Language Summary**

Directive	Description	Syntax
<b>CONTROL DIRECTIVES</b>		
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>"
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>

# MPASM USER'S GUIDE with MPLINK and MPLIB

Table E.1 MPASM Directive Language Summary (Continued)

Directive	Description	Syntax
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>] ]
<b>CONDITIONAL ASSEMBLY</b>		
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>
<b>DATA</b>		
CBLOCK	Define a Block of Constants	cblock [<expr>]
__CONFIG	Set configuration fuses	__config<expr>
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>
RES	Reserve Memory	res <mem_units>

## Appendix E. Quick Reference

---

**Table E.1 MPASM Directive Language Summary (Continued)**

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

**Table E.2 MPASM Command Line Options**

Option	Default	Description
?	N/A	Displays the MPASM Help Panel
a	INHx8M	Generate absolute .COD and hex output directly from assembler: /a<hex-format> where <hex-format> is one of [INHx8M   INHX8S   INHX32]
c	On	Enables/Disables case sensitivity
d	N/A	Define a text string substitution: /d<label>[=<value>]
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 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 member of the PICmicro MCU family. 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	Set list file tab size
w	0	Set message level: /w<value> Where <value> is: 0: all messages 1: errors and warnings 2: errors
x	Off	enable/disable/ set path for cross reference file /x Enable /x + Enable /x - Disable /x <path>xref.file Enables/sets path



## Appendix E. Quick Reference

---

**Table E.3 Radix Types Supported**

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

# MPASM USER'S GUIDE with MPLINK and MPLIB

Table E.4 MPASM Arithmetic Operators

Operator	Description	Example
\$	Current program counter	goto \$ + 3
(	Left Parenthesis	1 + ( d * 4 )
)	Right Parenthesis	( Length + 1 ) * 256
!	Item NOT (logical complement)	if ! ( a - b )
-	Complement	flags = -flags
-	Negation (2's complement)	-1 * Length
high	Return high byte	movlw high CTR_Table
low	Return low byte	movlw low 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
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment	i ++
--	Decrement	i --

## Appendix E. Quick Reference

### 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

### PIC16C5X Instruction Set

The PIC16C5X, 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 E.5    PIC16C5X Literal and Control Operations**

Hex	Mnemonic		Description	Function
Ekk	ANDLW	k	AND literal and W	k .AND. W → W
9kk	CALL	k	Call subroutine	PC + 1 → TOS, k → PC
004	CLRWDT		Clear watchdog timer	0 → WDT (and Prescaler if assigned)
Akk	GOTO	k	Goto address (k is nine bits)	k → PC(9 bits)
Dkk	IORLW	k	Incl. OR literal and W	k .OR. W → W
Ckk	MOVLW	k	Move Literal to W	k → W
002	OPTION		Load OPTION Register	W → OPTION Register
8kk	RETLW	k	Return with literal in W	k → W, TOS → PC
003	SLEEP		Go into Standby Mode	0 → WDT, stop oscillator
00f	TRIS	f	Tristate port f	W → I/O control reg f
Fkk	XORLW	k	Exclusive OR literal and W	k .XOR. W → W

# MPASM USER'S GUIDE with MPLINK and MPLIB

Table E.6 PIC16C5X Byte Oriented File Register Operations

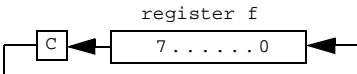
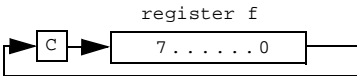
Hex	Mnemonic		Description	Function
1Cf	ADDWF	f, d	Add W and f	$W + f \rightarrow d$
14f	ANDWF	f, d	AND W and f	$W .AND. f \rightarrow d$
06f	CLRF	f	Clear f	$0 \rightarrow f$
040	CLRWF		Clear W	$0 \rightarrow W$
24f	COMF	f, d	Complement f	$.NOT. f \rightarrow d$
0Cf	DECF	f, d	Decrement f	$f - 1 \rightarrow d$
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	$W .OR. f \rightarrow d$
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	
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 E.7 PIC16C5X 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 $f(b) = 1$

## Appendix E. Quick Reference

### PIC16CXX Instruction Set

The PIC16CXX, Microchip's mid-range 8-bit microcontroller family, uses a 14-bit wide instruction set. The PIC16CXX 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 E.8 PIC16CXX 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	$0 \rightarrow WDT, \text{ stop oscillator}$
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 E.9 PIC16CXX 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	$f - 1 \rightarrow d, \text{ skip if } 0$
0Aff	INCF	f, d	Increment f	$f + 1 \rightarrow d$

# MPASM USER'S GUIDE with MPLINK and MPLIB

**Table E.9 PIC16CXX Byte Oriented File Register Operations**

Hex	Mnemonic		Description	Function
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	$f \rightarrow d$
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 E.10 PIC16CXX 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 E.11 PIC16C5X/PIC16CXX Special Instruction Mnemonics**

Mnemonic		Description	Equivalent Operation(s)	Status
ADDDCF	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	-

## Appendix E. Quick Reference

**Table E.11 PIC16C5X/PIC16CXX Special Instruction Mnemonics**

Mnemonic	Description	Equivalent Operation(s)	Status
BNC      k	Branch on No Carry	BTFSS GOTO	3,0 k      -
BNDC     k	Branch on No Digit Carry	BTFSS GOTO	3,1 k      -
BNZ      k	Branch on No Zero	BTFSS GOTO	3,2 k      -
BZ        k	Branch on Zero	BTFSC GOTO	3,2 k      -
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 INCF	f,1 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 DECF	3,0 f,d      Z
SUBDCF    f, d	Subtract Digit Carry from File	BTFSC DECF	3,1 f,d      Z
TSTF      f	Test File	MOVF	f,1      Z

# MPASM USER'S GUIDE with MPLINK and MPLIB

## PIC17CXX Instruction Set

The PIC17CXX, Microchip's high-performance 8-bit microcontroller family, uses a 16-bit wide instruction set. The PIC17CXX 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 E.12 PIC17CXX Data Movement Instructions**

Hex	Mnemonic	Description	Function
6pff	MOVFP <i>f, p</i>	Move <i>f</i> to <i>p</i>	$f \rightarrow p$
b8kk	MOVLB <i>k</i>	Move literal to BSR	$k \rightarrow \text{BSR} (3:0)$
bakx	MOVLP <i>k</i>	Move literal to RAM page select	$k \rightarrow \text{BSR} (7:4)$
4pff	MOVPF <i>p, f</i>	Move <i>p</i> to <i>f</i>	$p \rightarrow W$
01ff	MOVWF <i>f</i>	Move <i>W</i> to <i>F</i>	$W \rightarrow f$
a8ff	TABLRD <i>t, i, f</i>	Read data from table latch into file <i>f</i> , 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 <i>t, i, f</i>	Write data from file <i>f</i> 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 <i>t, f</i>	Read data from table latch into file <i>f</i> (table latch unchanged)	$\text{TBLATH} \rightarrow f$ if $t = 1$ $\text{TBLATL} \rightarrow f$ if $t = 0$
a4ff	TLWT <i>t, f</i>	Write data from file <i>f</i> into table latch	$f \rightarrow \text{TBLATH}$ if $t = 1$ $f \rightarrow \text{TBLATL}$ if $t = 0$

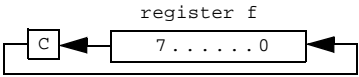
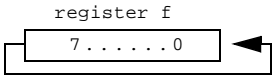
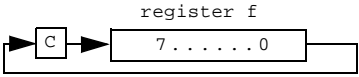
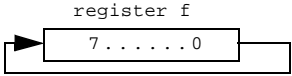
**Table E.13 PIC17CXX Arithmetic and Logical Instruction**

Hex	Mnemonic	Description	Function
b1kk	ADDLW <i>k</i>	Add literal to <i>W</i>	$(W + k) \rightarrow W$
0eff	ADDWF <i>f, d</i>	Add <i>W</i> to <i>F</i>	$(W + f) \rightarrow d$
10ff	ADDWFC <i>f, d</i>	Add <i>W</i> and Carry to <i>f</i>	$(W + f + C) \rightarrow d$



# Appendix E. Quick Reference

**Table E.13 PIC17CXX Arithmetic and Logical Instruction (Continued)**

Hex	Mnemonic		Description	Function
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	$(W \text{ .OR. } k) \rightarrow W$
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	$0xff \rightarrow f, 0xff \rightarrow d$
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	$(W \text{ .XOR. } f) \rightarrow d$

# MPASM USER'S GUIDE with MPLINK and MPLIB

Table E.14 PIC17CXX Bit Handling Instructions

Hex	Mnemonic		Description	Function
8bff	BCF	f, b	Bit clear f	0 → f(b)
8bff	BSF	f, b	Bit set f	1 → 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	.NOT. f(b) → f(b)

Table E.15 PIC17CXX 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) → f3(4:0), PC(15:13) → f3(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	(f3) → PCH:k → PCL 0 → GLINTD
b6kk	RETLW	k	Return with literal in W	k → W, TOS → PC, (f3 unchanged)
0002	RETURN		Return from subroutine	TOS → PC (f3 unchanged)
33ff	TSTFSZ	f	Test f, skip if zero	skip if f = 0

## Appendix E. Quick Reference

**Table E.16 PIC17CXX Special Control Instructions**

Hex	Mnemonic	Description	Function
0004	CLRWT	Clear watchdog timer	0 → WDT, 0 → WDT prescaler, 1 → PD, 1 → TO
0003	SLEEP	Enter Sleep Mode	Stop oscillator, power down, 0 → WDT, 0 → WDT Prescaler 1 → PD, 1 → TO

### 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

# MPASM USER'S GUIDE with MPLINK and MPLIB

## ASCII Character Set

	Most Significant Character								
	Hex	0	1	2	3	4	5	6	7
Least Significant Character	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



---

---

## Appendix F. Example Initialization Code

---

---

### Initialization Code

If the IDATA directive is used when generating an object module, the user must provide code to perform the data initialization. The following two examples may be used and modified as needed.

### Initialization Code for the PIC16CXX

```
*****
**      PIC16Cxx MPASM Initialized Data Startup File, Version 0.01  **
**      (c) Copyright 1997 Microchip Technology                    **
*****

;----- Environment variables -----;
      VARIABLE  TABLE_OFFSET = 0      ;Offset for reading from table of entries

;----- Equates -----;
;Register addresses
INDF          equ    0x00
PCL           equ    0x02
STATUS       equ    0x03
FSR          equ    0x04
PCLATH       equ    0x0A

;Bits within registers
Z            equ    0x02
C           equ    0x00
IRP         equ    0x07

;-----External variables and labels-----;
      EXTERN  _cinit      ;Start of const. data table
;-----;
; COPY_ROM_WORD_TO_RAM                                     ;
;                                                         ;
;   Reads a 16-bit word stored in program memory as a pair of;
; retlw kk instructions and stores the word in data memory  ;
; (low byte first). The macro also handles all paging and/or ;
; bank switching involved.                                 ;
;                                                         ;
; Arguments:                                             ;
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

```
; RomAddr Source address in program memory. ;
; RamAddr Destination address in data memory. ;
;-----;
COPY_ROM_WORD_TO_RAM MACRO RomAddr, RamAddr

    PAGESEL RomAddr ; Switch to correct ROM page,
    call RomAddr ; then read the low byte

    BANKSEL RamAddr ; Switch to correct RAM bank,
    movwf RamAddr ; then write the low byte

    call 1 + RomAddr ; Read the high byte from ROM
    movwf 1 + RamAddr ; and store it in RAM

    ENDM
;-----;

;*****;
VARIABLES UDATA_OVR
;-----;
; Data used for copying const. data into RAM
;
; Note: All the locations in this section can be reused
; by user programs. This can be done by declaring
; a section with the same name and attribute:
; i.e.
; VARIABLES UDATA_OVER (in MPASM)
; or
; #pragma udata overlay VARIABLES (in MPLAB-C)
;-----;
num_init RES 2 ;Number of entries in init table
init_entry_from_addr RES 2 ;ROM address to copy const. data from
init_entry_to_addr RES 2 ;RAM address to copy const. data to
init_entry_size RES 2 ;Number of bytes in each init.section
init_entry_index RES 2 ;Used to index through array of init. data
;-----;

; *****
_copy_idata_sec CODE

; *****
; * Copy initialized data from ROM to RAM *
; *****
;
```

## Appendix F. Example Initialization Code

```
; The values to be stored in initialized data are stored in
; program memory sections. The actual initialized variables are
; stored in data memory in a section defined by the IDATA directive
; in MPASM or automatically defined by MPLAB-C. There are 'num_init'
; such sections in a program. The table below has an entry for each
; section. Each entry contains the starting address in program memory
; where the data is to be copied from, the starting address in data
; memory where the data is to be copied, and the number of bytes to copy.
; The startup code below walks the table, reading those starting
; addresses and counts, and copies the data from program to data memory.
;
;
;
;      +=====+
;  _cinit | num_init (low)          |
;      +-----+
;      | num_init (high)         |
;      +=====+
;      | init_entry_from_addr (low) |      IDATA (0)
;      +-----+
;      | init_entry_from_addr (high)|
;      +-----+
;      | init_entry_to_addr (low)  |
;      +-----+
;      | init_entry_to_addr (high) |
;      +-----+
;      | init_entry_size (low)     |
;      +-----+
;      | init_entry_size (high)   |
;      +=====+
;      |          .                |      .
;      |          .                |      .
;      |          |                |
;      +=====+
;      | init_entry_from_addr (low) |      IDATA (num_init - 1)
;      +-----+
;      | init_entry_from_addr (high)|
;      +-----+
;      | init_entry_to_addr (low)  |
;      +-----+
;      | init_entry_to_addr (high) |
;      +-----+
;      | init_entry_size (low)     |
;      +-----+
;      | init_entry_size (high)   |
;      +=====+
```

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

```
; Start of code that copies initialization
; data from program to data memory
copy_init_data

; First read the count of entries (_cinit)
COPY_ROM_WORD_TO_RAM  _cinit, num_init
        TABLE_OFFSET = TABLE_OFFSET + 2

; For (num_init) do copy data for each initialization
; entry. Decrement 'num_init' every time and when it
; reaches 0 we are done copying initialization data
;
_loop_num_init
BANKSEL num_init
movf    num_init, W
iorwf   num_init+1, 0
btfss  STATUS, Z           ; If num_init is not down to 0,
goto    _copy_init_sec     ; then we have more sections to copy,
goto    _end_copy_init_data ; otherwise, we're done copying data.

; For a single initialization section, read the
; starting addresses in both program and data memory,
; as well as the number of bytes to copy.
;
_copy_init_sec
COPY_ROM_WORD_TO_RAM  TABLE_OFFSET + _cinit, init_entry_from_addr
        TABLE_OFFSET = TABLE_OFFSET + 2 ;Increment by 2 since it's a word
COPY_ROM_WORD_TO_RAM  TABLE_OFFSET + _cinit, init_entry_to_addr
        TABLE_OFFSET = TABLE_OFFSET + 2 ;Increment by 2 since it's a word
COPY_ROM_WORD_TO_RAM  TABLE_OFFSET + _cinit, init_entry_size
        TABLE_OFFSET = TABLE_OFFSET + 2 ;Increment by 2 since it's a word

; Check 'init_entry_size'. If it's 0, then go
; to the next entry in the table (if it exists).
; If 'init_entry_size' is non-zero, then go ahead
; and copy the bytes.
;
_start_copying_data
        BANKSEL init_entry_size
        movf    init_entry_size, W
        iorwf   init_entry_size+1, W
        btfsc  STATUS, Z
        goto    _dec_num_init
```



## Appendix F. Example Initialization Code

---

```
; Set up the destination address for the data in the FSR so
; we are prepared to copy data using indirect addressing
    BANKSEL init_entry_to_addr
    movf    init_entry_to_addr, W
    movwf  FSR

; Read a single data byte by doing a long jump
; into the section in program memory
    goto   _Dummy2
_Dummy1
    movf   init_entry_from_addr+1, W
    movwf  PCLATH
    movf   init_entry_from_addr, W
    movwf  PCL
_Dummy2
    call   _Dummy1                ;Puts return address on stack

; Now write the data to RAM using indirect addressing
movf   init_entry_to_addr+1, 1    ;Check if upper portion of
    btfss  STATUS, Z              ;address is non-zero. If so, then
    bsf    STATUS, IRP            ;set the IRP bit. Otherwise,
    bcf    STATUS, IRP            ;clear the IRP bit.
    movwf  INDF

; After copying one entry we need to:
; 1. Increment the program memory (source) address
; 2. Increment the data memory (destination) address
; 3. Decrement the init_entry_size

    BANKSEL init_entry_from_addr
incf   init_entry_from_addr,1
    btfsc  STATUS, C
    incf   init_entry_from_addr+1,1

    BANKSEL init_entry_to_addr    ;Increment the address
incf   init_entry_to_addr,1      ;_init_entry_to_addr
    btfsc  STATUS, C
    incf   init_entry_to_addr+1,1

    BANKSEL init_entry_size       ;Decrement the count
    movf   init_entry_size,1      ;_init_entry_size
    btfsc  STATUS, Z
    decf   init_entry_size+1,1
```

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

```
decf    init_entry_size,1

goto    _start_copying_data        ;Back to do another section

_dec_num_init
        BANKSEL num_init
        movf    num_init,1
        btfsc  STATUS, Z
decf    num_init+1,1
decf    num_init, 1

        goto    _loop_num_init

_end_copy_init_data                ;We're done copying initialized data

        return

; Must declare copy_init_data as GLOBAL to be able
; to call it from other assembly modules
GLOBAL  copy_init_data

END
```

# Appendix F. Example Initialization Code

## Initialization Code for the PIC17CXX

```
*****
**    PIC17Cxx MPASM Initialized Data Startup File, Version 0.01  **
**    (c) Copyright 1997 Microchip Technology                    **
*****

;----- Equates -----;
;Register addresses
INDF          equ    0x00
PCL           equ    0x02
STATUS       equ    0x03
FSR          equ    0x04
PCLATH       equ    0x0A

;Bits within registers
Z            equ    0x02
C            equ    0x00

;-----External variables and labels-----;
    EXTERN  _cinit      ;Start of const. data table

;*****;
VARIABLES  UDATA_OVR
;-----;
; Data used for copying const. data into RAM
;
; NOTE:  ALL THE LOCATIONS IN THIS SECTION CAN BE REUSED
;        BY USER PROGRAMS. THIS CAN BE DONE BY DECLARING
;        A SECTION WITH THE SAME NAME AND ATTRIBUTE,
;        i.e.
;
;        VARIABLES  UDATA_OVER          (in MPASM)
;
;        or
;
;        #pragma udata overlay VARIABLES (in MPLAB-C)
;-----;
num_init          RES    2  ;Number of entries in init table
init_entry_from_addr RES  2  ;ROM address to copy const. data from
init_entry_to_addr  RES  2  ;RAM address to copy const. data to
init_entry_size    RES  2  ;Number of bytes in each init.section
save_tblptrl      RES  1  ;These two variables preserve
save_tblptrh      RES  1  ;the position of TBLTRL within the entry table
;-----;

; *****
_copy_idata_sec   CODE    PROGMEM_START
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

```
; *****
; * Copy initialized data from ROM to RAM *
; *****
; The values to be stored in initialized data are stored in
; program memory sections. The actual initialized variables are
; stored in data memory in a section defined by the IDATA directive
; in MPASM or automatically defined by MPLAB-C. There are 'num_init'
; such sections in a program. The table below has an entry for each
; section. Each entry contains the starting address in program memory
; where the data is to be copied from, the starting address in data
; memory where the data is to be copied, and the number of bytes to copy.
; The startup code below walks the table, reading those starting
; addresses and counts, and copies the data from program to data memory.
;
;
;
;      +=====+
; _cinit | num_init (low) |
;      +-----+
;      | num_init (high) |
;      +=====+
;      | init_entry_from_addr (low) |          IDATA (0)
;      +-----+
;      | init_entry_from_addr (high) |
;      +-----+
;      | init_entry_to_addr (low) |
;      +-----+
;      | init_entry_to_addr (high) |
;      +-----+
;      | init_entry_size (low) |
;      +-----+
;      | init_entry_size (high) |
;      +=====+
;      | . |
;      | . |
;      | . |
;      +=====+
;      | init_entry_from_addr (low) |          IDATA (num_init - 1)
;      +-----+
;      | init_entry_from_addr (high) |
;      +-----+
;      | init_entry_to_addr (low) |
;      +-----+
;      | init_entry_to_addr (high) |
;      +-----+
;      | init_entry_size (low) |
;      +-----+
;      | init_entry_size (high) |
;      +=====+
```

## Appendix F. Example Initialization Code

---

```
; Start of code that copies initialization
; data from program to data memory
copy_init_data

; First read the count of entries (_cinit)
    movlw    HIGH _cinit
    movwf   PCLATH
    CALL    _cinit & 0x3FF
    BANKSEL num_init
    movwf   num_init
    CALL    (_cinit & 0x3FF) + 1
    movwf   num_init+1

; For (num_init) do copy data for each initialization
; entry. Decrement 'num_init' every time and when it
; reaches 0 we are done copying initialization data
;
_loop_num_init
    BANKSEL num_init
    movf    num_init, W
    iorwf   num_init+1, 0
    btfss  STATUS, Z           ; If num_init is not down to 0,
    goto   _copy_init_sec     ; then we have more sections to copy,
    goto   _end_copy_init_data ; otherwise, we're done copying data.

; For a single initialization section, read the
; starting addresses in both program and data memory,
; as well as the number of bytes to copy.
;
_copy_init_sec
; Read 'from' address in program memory
    BANKSEL init_entry_from_addr
    tablrd 0, 1, init_entry_from_addr
    tlrld  0, init_entry_from_addr
    tlrld  1, init_entry_from_addr+1

; Read 'to' address in data memory
    BANKSEL init_entry_to_addr
    tablrd 0, 1, init_entry_to_addr
    tlrld  0, init_entry_to_addr
    tlrld  1, init_entry_to_addr+1

; Read 'size' of data to be copied in BYTES
    BANKSEL init_entry_size
    tablrd 0, 1, init_entry_size
```

# MPASM USER'S GUIDE with MPLINK and MPLIB

---

```
        tlrld    0, init_entry_size
        tlrld    1, init_entry_size+1

; We must save the position of TBLPTR since TBLPTR
;is used in copying the data as well.
        movfp    TBLPTRL, WREG
        BANKSEL save_tblptrl
        movwf    save_tblptrl
        movfp    TBLPTRH, WREG
        BANKSEL save_tblptrh
        movwf    save_tblptrh

; Setup TBLPTRH:TBLPTRL to point to the ROM section
;where the initialization values of the data are stored.
        BANKSEL init_entry_from_addr
        movfp    init_entry_from_addr, WREG
        movwf    TBLPTRL
        movfp    init_entry_from_addr+1, WREG
        movwf    TBLPTRH

; We must determine whether the data section is in
;the general purpose area of RAM or in the special
;function register (SFR) area. We do this by comparing
;the address with the register memory map. We then
;determine ;whether to alter the upper or lower nibble
;of the BSR register in preparation for copying the data.
;
        BANKSEL init_entry_to_addr
        movfp    init_entry_to_addr, FSR0

; First we see if destination is GPR (>0x20)
        movlw    0x1f
        cpfslt   init_entry_to_addr    ; If it is < 0x1f continue testing,
        goto     _init_sec_gpr          ;otherwise it's 0x20 or higher (GPR)

; It's not GPR, let's see if it's SFR or unbanked between 0x18 and 0x1F
        movlw    0x18
        cpfslt   init_entry_to_addr    ;is it <=17 ?
        goto     _start_copying_data    ;No, it's between 0x18 and 0x1F

;It's <=17, let's see if it is 0x00-0x0F or 0x10-0x17
        movlw    0x0F
        cpfsgt   init_entry_to_addr
        goto     _start_copying_data    ; It's between 0x00 and 0x0F, COPY!
```

## Appendix F. Example Initialization Code

---

```
;if we fall through it's an SFR from 0x10-0x17

;OK, it's an SFR that needs MOVLB-type of bank switching!
;First mask off low nibble of BSR
    movlw    0xF0
    andwf    BSR,1                ;clear the low nibble
    movfp    init_entry_to_addr+1, WREG ;Load high portion of address
    iorwf    BSR,1                ;and paste high portion into BSR
    goto     _start_copying_data

;Well, it's a banked GPR needing MOVLR-type of bank switching!
_init_sec_gpr
;First mask off high nibble of BSR
    movlw    0x0F
    andwf    BSR,1
    BANKSEL  init_entry_to_addr+1
    swapf    (init_entry_to_addr+1), WREG ;Bank addr.in hi nibble of WREG
    iorwf    BSR,1
    goto     _start_copying_data

;Loop for # of bytes to be copied

; Since on 17Cxx we store two bytes per word we must be careful
;if the number of bytes to be copied is odd. We cannot copy word by
;word or we may end up overwriting a byte in RAM that doesn't belong
;to the initialized data section. We therefore must decrement and
;check the size for every low and high byte read from program memory.
;
_start_copying_data
    tablrd   0, 1, WREG

;*** Test ****
    movfp    init_entry_size, WREG
    iorwf    init_entry_size+1,0
    btfsc    ALUSTA, Z
    goto     _dec_num_init

;***** Copy low byte ****
    tlrld   0, WREG      ;
    movfp    WREG, INDF0 ;Low byte stored in RAM location

;*** Decrement ***
    decf     init_entry_size,1
    btfss    ALUSTA, C
    decf     init_entry_size+1,1
```

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

```
;*** Test again ***
    movfp    init_entry_size, WREG
    iorwf    init_entry_size+1,0
    btfsc    ALUSTA, Z
    goto     _dec_num_init

;**** Copy high byte ****
    tlrld   1, WREG
    movfp   WREG, INDF0

;*** Decrement ***
    decf    init_entry_size,1
    btfss   ALUSTA, C
    decf    init_entry_size+1,1

    goto   _start_copying_data

; Decrement the counter for the outermost loop (no. of init.secs.)
;
_dec_num_init
    decf    num_init,1
    btfss   ALUSTA,C
    decf    num_init+1, 1

;Now restore TBLPTRH:TBLPTRL to point to table
    movfp   save_tblptrl, WREG
    movwf   TBLPTRL
    movfp   save_tblptrh, WREG
    movwf   TBLPTRH

;Then go back to the top to do the next section, if any
    goto   _loop_num_init

;We're done copying initialized data
_end_copy_init_data

    return

;Must declare it as GLOBAL to be able to call it from other assembly modules

GLOBAL   copy_init_data

END
```





Index

Symbols

#DEFINE .. 32, 42, 54, 109, 112  
#UNDEFINE .....42, 54

A

Arithmetic Operators ..... 75, 130  
Assemble ..... 8

B

BADRAM .....26, 46, 115  
BANKSEL .....26, 113  
BANKSEL .....27, 61, 113  
BBS ..... 11  
    application notes ..... 107  
    bug reports ..... 107  
    Connecting to ..... 106  
    errata sheets ..... 107  
    Software Releases ..... 107  
    source code ..... 107  
    Special Interest Groups 107  
    Systems Information  
        and Upgrade  
        Hot Line ..... 108  
    Using the ..... 106

C

CALL ..... 110  
case sensitivity ..... 14, 18  
CBLOCK .....22, 28, 35, 110  
CODE ..... 29  
Command Line Interface 13, 14,  
    ..... 128  
Comments ..... 19  
Compatibility ..... 10  
CONFIG ..... 29  
CONSTANT .....22, 30  
cross reference file ..... 15

D

DATA .....30, 72  
DB ..... 31  
DE ..... 31  
Define ..... 14  
Directives ..... 8  
DT ..... 32  
DW ..... 33, 72

E

ELSE ..... 33, 41, 42  
END .....34  
ENDC ..... 28, 34, 112  
ENDIF ..... 35, 41, 42, 111, 112  
ENDM ..... 35, 66, 69, 112  
ENDW ..... 35, 55  
EQU ..... 22, 36, 50, 110  
ERROR .....36  
error file ..... 14, 20, 22, 37, 109,  
    .....119, 123  
ERRORLEVEL ..... 37, 115  
Escape Sequences .....73  
EXITM ..... 37, 66, 68, 112  
EXPAND .....38  
Expressions .....71  
EXTERN .....38

F

File  
    cross reference ..... 15, 16  
    error ..... 14, 16  
    listing ..... 14, 17  
    object .....14  
    source .....16  
file extensions .....20  
FILL .....39

G

GLOBAL .....39

H

Hex .....101  
Hex File ..... 8, 18, 44, 101  
hex file format ..... 14, 17, 20  
HIGH ..... 111  
High/Low .....76

I

IDATA .....40  
IDLOCS ..... 41, 111  
IF ..... 34, 41, 112  
IFDEF ..... 32, 42  
IFNDEF .....42  
INCLUDE .....43  
Increment/Decrement .....76

Initialization.....141  
Instruction Sets .....131  
    PIC16C5X .....131  
    PIC16CXX .....133  
    PIC17CXX .....136  
    Special Instructions .....134  
Internet  
    Connecting to Microchip  
        web site .....105  
Internet Home Page .....11

L

Labels .....19  
Library .....8  
Link .....9  
LIST ..... 29, 41, 44, 115  
listing .....37  
listing file 9, 14, 17, 20, 21, 23, 38,  
    ..... 44, 47, 51, 52  
LOCAL ..... 44, 68  
Local Label .....66  
LOW .....111

M

MACRO ..... 14, 35, 45  
Macro ..... 9, 65  
MAXRAM ..... 26, 46, 115, 116  
message level .... 15, 18, 37, 44  
MESSG .....47  
Migration Path .....3  
Mnemonics ..... 9, 19  
MPASM .....14  
MPASMWIN ..... 14, 17, 109  
MPLAB .....10  
MPLIB .....8  
MPLINK .....9

N

NOEXPAND ..... 38, 47  
NOLIST .....47

O

object file ..... 9, 14, 17  
Operands .....19  
Operators .....71  
ORG .....48

# **MPASM USER'S GUIDE with MPLINK and MPLIB**

---

## **P**

PAGE ..... 48  
PAGESEL ..... 48, 113  
PC ..... 9  
PICmicro ..... 9  
PICSTART ..... 10  
Precedence ..... 71  
PRO MATE ..... 10  
PROCESSOR ..... 15, 29, 41, 49  
Processor ..... 16  
processor ..... 114  
PROCESSOR directive ..... 115

## **R**

RADIX ... 10, 15, 18, 44, 49, 71,  
..... 74, 115  
RELOCATABLE OBJECTS ... 57  
RES ..... 50

## **S**

SET ..... 22, 30, 50, 54, 110  
Shell ..... 13, 14, 16  
Software Releases ..... 107  
    Intermediate Release ... 107  
    Production Release ..... 108  
Source Code ..... 9  
source file ..... 20  
SPACE ..... 51  
SUBTITLE ..... 51

## **T**

Text Strings ..... 72  
TITLE ..... 51

## **U**

UDATA ..... 52  
UDATA\_OVR ..... 52  
UDATA\_SHR ..... 53

## **V**

VARIABLE ..... 22, 30, 54

## **W**

Warnings ..... 113  
Warranty ..... 10  
Web Site  
    connecting to ..... 105  
    file transfer ..... 105  
WHILE ..... 36, 55, 69, 112  
WHILE-ENDW ..... 112

**Notes:**



**MICROCHIP**

# WORLDWIDE SALES & SERVICE

## AMERICAS

### Corporate Office

Microchip Technology Inc.  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 602-786-7200 Fax: 602-786-7277  
Technical Support: 602 786-7627  
Web: <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.  
14651 Dallas Parkway, Suite 816  
Dallas, TX 75240-8809  
Tel: 972-991-7177 Fax: 972-991-8588

### Dayton

Microchip Technology Inc.  
Two Prestige Place, Suite 150  
Miamisburg, OH 45342  
Tel: 937-291-1654 Fax: 937-291-9175

### Los Angeles

Microchip Technology Inc.  
18201 Von Karman, Suite 1090  
Irvine, CA 92612  
Tel: 714-263-1888 Fax: 714-263-1338

### New York

Microchip Technology Inc.  
150 Motor Parkway, Suite 216  
Hauppauge, NY 11788  
Tel: 516-273-5305 Fax: 516-273-5335

### San Jose

Microchip Technology Inc.  
2107 North First Street, Suite 590  
San Jose, CA 95131  
Tel: 408-436-7950 Fax: 408-436-7955

### 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  
RM 3801B, Tower Two  
Metroplaza  
223 Hing Fong Road  
Kwai Fong, N.T., Hong Kong  
Tel: 852-2-401-1200 Fax: 852-2-401-3431

### India

Microchip Technology Inc.  
India Liaison Office  
No. 6, Legacy, Convent Road  
Bangalore 560 025, India  
Tel: 91-80-229-0061 Fax: 91-80-559-9840

### 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

### Singapore

Microchip Technology Taiwan  
Singapore Branch  
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 2-717-7175 Fax: 886-2-545-0139

## EUROPE

### United Kingdom

Arizona Microchip Technology Ltd.  
Unit 6, The Courtyard  
Meadow Bank, Furlong Road  
Bourne End, Buckinghamshire SL8 5AJ  
Tel: 44-1628-851077 Fax: 44-1628-850259

### France

Arizona Microchip Technology SARL  
Zone Industrielle de la Bonde  
2 Rue du Buisson aux Fraises  
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: 39-39-6899939 Fax: 39-39-6899883

## JAPAN

Microchip Technology Intl. Inc.  
Benex S-1 6F  
3-18-20, Shinyokohama  
Kohoku-Ku, Yokohama-shi  
Kanagawa 222 Japan  
Tel: 81-45-471-6166 Fax: 81-45-471-6122

9/24/97



All rights reserved. © 1997, Microchip Technology Incorporated, USA. 10/97 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.