“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. The Microchip logo and name are trademarks of Microchip Technology Incorporated. All rights reserved. All other trademarks mentioned herein are the property of their respective companies”.

©1994 Microchip Technology Incorporated. All rights Reserved.

The Microchip logo and name are trademarks of Microchip Technology Inc.

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

ALLPRO is a registered trademark of Logical Devices Incorporated.

CompuServe is a registered trademark of CompuServe Incorporated.

Data I/O is a registered trademark of Data I/O Corporation.

Excel and Windows are registered trademarks of Microsoft Corporation.

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

Intel is a registered trademark of Intel Corporation.

MS-DOS is a registered trademark of Microsoft Corporation.

Unisite is a registered trademark of UNIX System Labs./Novell, Inc.
All product/company trademarks mentioned herein are the property of their respective companies.
# Table of Contents

Preface:
- Welcome ................................................................. 1
- Feature List and Product Information .................. 1
- Migration Path ......................................................... 1

Chapter 1: Introduction .................................................. 3
- Product Definition ..................................................... 3
- Documentation Layout ............................................... 3
- Terms ........................................................................ 5
  - PIC16/17 .................................................................. 5
  - Source Code ............................................................ 5
  - Assemble .................................................................... 5
  - Mnemonics ............................................................... 5
  - Directives ................................................................... 6
  - Macro ......................................................................... 6
  - Relocatable Object .................................................. 6
  - Linking ....................................................................... 6
  - Listing ....................................................................... 6
  - PC ............................................................................... 6
  - DOS ........................................................................... 6

Recommended Reading .................................................. 7
- System Requirements .................................................. 7
- Warranty Registration .................................................. 7
- Installation ................................................................. 7
- Compatibility Issues ................................................... 8

Chapter 2: Environment and Usage ..................................... 9
- Introduction ............................................................... 9
- Highlights .................................................................... 9
- Terms .......................................................................... 9
  - Command Line Interface ........................................ 9
  - Shell ......................................................................... 9
  - Alpha Character ....................................................... 9
  - Alpha Numeric ........................................................ 9

Command Line Interface ................................................ 10
- Shell Interface ........................................................... 12
  - Source File ............................................................. 12
  - Error File ............................................................... 12
  - Cross Reference File ............................................. 12
  - Listing File ............................................................. 13
  - HEX Dump Type .................................................... 13
  - Assemble to Object File ........................................... 13

Source Code Formats ..................................................... 13
- Labels ......................................................................... 14
- Mnemonics ............................................................... 14
- Operands .................................................................... 14
- Comments ................................................................... 15

Files Used by MPASM and Utility Functions .................. 15
- Object Code Formats ................................................ 16
- Listing File Format .................................................... 16
Chapter 3: Directive Language ........................................................... 19

Introduction ............................................................................... 19

Highlights .................................................................................. 19

Terms ........................................................................................ 19

Data Directives ............................................................. 19
Listing Directives .......................................................... 19
Control Directives ......................................................... 19
Macro Directives .......................................................... 19

Directive Details ........................................................................ 21

CBANK - Future Feature ........................................................... 21
CBLOCK - Define a Block of Constants .................................... 22
CONSTANT - Declare Symbol Constant .................................. 22
DATA - Create Numeric and Text Data ..................................... 23
DB - Declare Data of One Byte ................................................. 24
#DEFINE - Define a Text Substitution Label ............................. 25
DW - Declare Data of One Word ............................................... 26
ELSE - Begin Alternative Assembly Block to IF ...................... 26
END - End Program Block ........................................................ 27
ENDC - End an Automatic Constant Block ............................... 27
ENDIF - End Conditional Assembly Block ................................. 28
ENDM - End a Macro Definition ................................................ 28
ENDW - End a While Loop ........................................................ 29
EQU - Define an Assembler Constant... .................................... 29
ERROR - Issue an Error Message ............................................ 30
EXITM - Exit from a Macro ........................................................ 30
EXPAND - Expand Macro Listing .............................................. 31
FILL - Specify Memory Fill Value .............................................. 31
IF - Begin Conditionally Assembled Code Block ...................... 32
IFDEF - Execute If Symbol has Been Defined ........................... 33
IFNDEF - Execute If Symbol has not Been Defined ................. 34
INCLUDE - Include Additional Source File ................................ 35
LIST - Listing Options ................................................................ 36
LOCAL - Declare Local Macro Variable .................................... 37
MACRO - Declare Macro Definition .......................................... 38
MESSG - Create User Defined Message .................................... 38
NOEXPAND - Turn off Macro Expansion .................................. 39
NOLIST - Turn off Listing Output .............................................. 39
ORG - Set Program Origin ....................................................... .40
PAGE - Insert Listing Page Eject .............................................. 40
PROCESSOR - Set Processor Type ............................................ 41
RADIX - Specify Default Radix ............................................... 41
RES - Reserve Memory ............................................................ 42
SET - Define an Assembler Variable ........................................ 42
SPACE - Insert Blank Listing Lines ........................................... 43
SUBTITLE - Specify Program Subtitle ...................................... 43
TITLE - Specify Program Title ................................................... 44
#UNDEFINE - Delete a Substitution Label ................................ 44
VARIABLE - Declare Symbol Variable ...................................... 45
WHILE - Perform Loop While Condition is True ....................... 46
# Table of Contents

Chapter 4: Macro Language ................................................................. 47  
  Introduction ................................................................................. 47  
  Highlights ................................................................................... 47  
  Terms .......................................................................................... 47  
    Macro ....................................................................................... 47  
    Local Label ................................................................----------- 48  
    Recursion ................................................................................ 48  
  Macro Syntax ................................................................................ 48  
  Macro Directives .......................................................................... 49  
  Text Substitution ......................................................................... 49  
  Recursive Macros ........................................................................ 50  
  Macro Usage ................................................................................ 50  
  Examples ...................................................................................... 51  
    Eight by Eight Multiply .......................................................... 51  
    Constant Compare ..................................................................... 52  

Chapter 5: Expression Syntax and Operation .............................................. 53  
  Introduction .................................................................................. 53  
  Highlights ................................................................................... 53  
  Terms .......................................................................................... 53  
    Expressions .............................................................................. 53  
    Operators ................................................................................ 53  
    Precedence .............................................................................. 53  
    Radix ....................................................................................... 53  
  Text Strings ................................................................................ 54  
  Numeric Constant and Radix ....................................................... 56  
  High / Low ................................................................................ 58  

Chapter 6: MPLIB - MPASM Librarian ...................................................... 59  
  Introduction .................................................................................. 59  

Chapter 7: MPLINK - MPASM Linker ....................................................... 61  
  Introduction .................................................................................. 61  

Appendix A: Object Code Formats .......................................................... 63  
  Introduction .................................................................................. 63  
  Highlights ................................................................................... 63  
  Object Code Formats ................................................................... 63  
    Intel Hex Format (.HEX) .......................................................... 63  
    Example ................................................................................... 64  
    8-Bit Split Format (.HXU,.HXH) ............................................. 64  
    Example ................................................................................... 64  
    32-Bit Hex Format ................................................................. 65  

Appendix B: Customer Support ............................................................... 67  
  Keeping Current with Microchip Systems ....................................... 67  
  Highlights ................................................................................... 67  
  Bulletin Board Access .................................................................. 68  
  Bulletin Board Usage ................................................................... 68  
  Special Interest Groups ............................................................... 68  
  Files ......................................................................................... 69  
  Mail ......................................................................................... 69  
  Software Revisions ..................................................................... 70
# Table of Contents

Symbol table full ........................................................... 81  
Temp file creation error ................................................ 82  
Too many arguments ................................................... 82  
Undefined argument ..................................................... 82  
Unknown error .............................................................. 82  
WHILE failed to terminate within 256 iterations ............. 82

**Warning Messages** ............................................................. 82
Addresses above 32K not currently supported. ...
Using MaxRom. ............................................................ 82  
Argument out of range, least significant bits used ...... 82  
Crossing page boundary—ensure page bits are set ... 83  
...is not currently supported ........................................ 83  
LCALL should only be used for multi-paged program ...
memory ............................................................... 83  
... May not be handled as preprocessor directive ...... 83  
... Not a single byte quantity ..................................... 83  
This number is being treated as a binary ...
representation ......................................................... 83

**Quick Reference Guide** .......................................................... 85
Highlights ........................................................................ 85
Terms ............................................................................. 85  
PIC16C5X ................................................................ 85  
PIC16CXX ................................................................ 85  
PIC17CXX ................................................................ 85  
PIC16C5X Instruction Set .................................................. 87  
PIC16C5X Notes .......................................................... 89  
PIC16CXX Instruction Set ................................................. 90  
PIC16CXX Notes .......................................................... 94  
PIC17C42 Instruction Set .................................................. 95  
PIC17C42 Notes .......................................................... 99

Index ................................................................................ 101
Reader Response ............................................................ 103
Sales and Service ............................................................. 104
TABLE OF EXAMPLES
1 Sample MPASM Source Code ......................................................... 14
2 Sample MPASM Listing File (.LST) .................................................. 16

TABLE OF TABLES
1 Documentation Conventions ............................................................... 5
2 Assembler Command Line Options .................................................. 11
3 MPASM Default File Extensions ....................................................... 15
4 Directive Summary ........................................................................... 20
5 List Directive Options ...................................................................... 36
6 ANSI 'C' Escape Sequences ............................................................ 55
7 Radix Specifications ....................................................................... 56
8 Arithmetic Operators and Precedence .............................................. 57
9 Directive Summary ........................................................................... 86
10 PIC16C5X Operand Codes .............................................................. 87
11 PIC16C5X Byte Oriented File Register Operations ......................... 88
12 PIC16C5X Bit Oriented Register Operations .................................... 88
13 PIC16C5X Literal and Control Operations ....................................... 89
14 PIC16CXX Byte Oriented File Register Operations ......................... 91
15 PIC16CXX Bit Oriented File Register Operations ........................... 91
16 PIC16CXX Literal and Control Operations ...................................... 92
17 PIC16CXX Special Instruction Mnemonics ....................................... 93
18 PIC17C42 Operand Codes ............................................................... 95
19 PIC17C42 Data Move Instructions .................................................. 96
20 PIC17C42 Arithmetic and Logical Instructions ................................. 97
21 PIC17C42 Program Control Instructions ......................................... 98
22 PIC17C42 Bit Handling Instructions ................................................ 98
23 PIC17C42 Special Control Instructions ............................................ 99
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 the PIC16C5X, PIC16CXX, PIC17CXX, and future microcontroller offerings. Notable features include:

- PIC16C5X, PIC16CXX, and PIC17CXX Instruction Set
- Command Line Interface
- Command Shell Interface
- Rich Directive Language
- Flexible Macro Language
- PICMASTER™ Compatibility
- MPSIM Compatibility

Use of the Microchip MPASM Universal Assembler requires an IBM PC/AT® or compatible computer, running MS-DOS® V4.1 or greater.

Migration Path

Users of MPALC and ASM17 will find that much of their existing code will assemble with MPASM with little or no editing. This provides a simple migration path. Appendices C and D describe some of the simple changes you may need to implement to assemble existing code.

But more importantly, because the program is universal, an application developed for the PIC16C54 can be easily translated into a program for the PIC16C71. This would simply 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 was developed in conjunction with Byte Craft Limited, recognized as a world leader in microcontroller language tools.
Chapter 1. Introduction

Product Definition

MPASM is a DOS based PC application that provides a platform for developing assembly language code for Microchip microcontrollers including the PIC16C5X, PIC16CXX and PIC17CXX families. Generically, MPASM will refer to the entire development platform including the macro assembler and utility functions. Specifically:

MPASM - refers to the macro assembler that generates relocatable object code from assembly source code.

MPLINK - refers to the linker that translates relocatable objects to executable binary code at absolute memory locations.

MPLIB - refers to the librarian utility that allows relocatable objects to be grouped together in one file, or library, for convenience. These libraries can be referenced via MPLINK as an object file output from MPASM.

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. In particular:

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 (CLI), 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 native directive language of MPASM. This language should be familiar to previous users of either Microchip or Byte Craft development system.

Chapter 4: Macro Language - This chapter describes the macro language of MPASM. Macros are best learned by example; several will be offered for consideration.

Chapter 5: Expression Syntax and Operation - This chapter describes the expression syntax of MPASM, including operator precedence, radix override notation, examples and discussion.
Chapter 6: MPLIB - MPASM Librarian - This chapter describes the purpose, use and CLI of the librarian utility provided as part of the MPASM development environment. It will also provide examples and discussion.

Chapter 7: MPLINK - MPASM Linker - This chapter describes the purpose, use and CLI of the linker utility provided with MPASM. It also describes the script language provided to specify where relocated objects should be placed in memory.

This document offers the following General Reference sections:

Appendix A: Object code formats, a brief overview.

Appendix B: Customer Support - Provides information about accessing the Microchip Bulletin Board for the latest revisions of products, user forums and non-urgent questions about applying Microchip products.

Appendix C: MPALC Conversion Guide. A short description designed to assist users of MPALC to move their code to MPASM.

Appendix D: ASM17 Conversion Guide. A short description designed to assist users of ASM17 to move their code to MPASM.

Appendix E: Error Messages. A list of the error messages generated by MPASM, with descriptions.

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

Quick Reference Guide: This section provides a quick reference to the instruction set for each family of microprocessors as well as quick references to the directive and macro language, possibly for "tear-out."
TABLE 1: DOCUMENTATION CONVENTIONS

<table>
<thead>
<tr>
<th>Character</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square Brackets ([])</td>
<td>Optional Arguments</td>
</tr>
<tr>
<td>Angle Brackets (&lt;&gt;)</td>
<td>Delimiters for special keys: &lt;TAB&gt;, &lt;ESC&gt;, or additional options.</td>
</tr>
<tr>
<td>Pipe Character (!)</td>
<td>Choice of mutually exclusive arguments; an OR selection.</td>
</tr>
<tr>
<td>Lowercase characters</td>
<td>Type of data</td>
</tr>
<tr>
<td><em>Italic characters</em></td>
<td>A variable argument; it can be either a type of data (in lowercase characters or a specific example (in uppercase characters)</td>
</tr>
<tr>
<td>Courier Font</td>
<td>User entered code or sample code.</td>
</tr>
</tbody>
</table>

Terms

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

**PIC16/17**

PIC16/17 refers to any Microchip microcontroller, including the representatives of the PIC16C5X, PIC16CXX, and PIC17CXX families.

**Source Code**

This is the file of PIC16/17 instructions and MPASM directives and macros that will be translated into executable code. This code is suitable for use by a PIC16/17 or Microchip development system product like an emulator, a simulator or a programmer. It is an ASCII file that can be created using any ASCII text editor.

**Assemble**

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

**Mnemonics**

These are instructions that are translated directly into machine code. These are used to perform arithmetic and logical operations on data residing program or data memory of a PIC16/17. They also have the ability to move data in and out of registers and memory as well as conditionally branch to specified program addresses.
Directives

Directives provide control of the assembler's operation by telling MPASM how to treat mnemonics, data references and format the listing file. Directives make coding easier and provide custom output according to specific needs.

Macro

A macro consists of a sequence of assembler commands. Passing arguments to a macro allows for flexible functionality.

Relocatable Object

A unit of intermediate code that may not have an absolute base address in PIC16/17 memory. This base address may be assigned at link time.

Linking

Linking, or to link, refers to the translation of relocatable objects to machine code suitable for execution by a PIC16/17. Absolute addresses may be assigned to relocated objects at this time.

Listing

A listing is an ASCII text file that shows the machine code generated for each assembly instruction, MPASM directive, or macro encountered in a source file. An absolute listing file shows the collection of relocated objects, together with their absolute addresses in PIC16/17 memory (relative addresses will be shown in listings output directly from the macro assembler).

PC

Any IBM or compatible Personal Computer.

DOS

Disk Operating System that provides the basis for most applications that run on PCs.
Chapter 1: Introduction

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 PIC16/17 microcontrollers. Therefore, you are urged to read the Data Sheets for the PIC16/17 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 PIC16/17s. The application notes described within are available from the Microchip BBS (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 V4.1 or greater. The distribution is provided on 3.5", double density (720k) floppy diskettes.

No special display or ancillary devices are required.

Warranty Registration

NOTE: 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.

Installation

Never use the original diskette as your working copy. Make a backup copy of the MPASM distribution disk using the DOS "DISKCOPY" program, then label the new copy and store the original in a safe place.

It is recommended that you execute MPASM from your hard disk. To do this, create a new directory (MKDIR) for the assembler and copy all files from the backup distribution diskette to that directory (MPASM, and its accompanying utility programs and source examples are distributed at the root level of the distribution diskette).

If you want to be able to run MPASM from any directory (without fully qualifying the path to the executable program), you must add the new directory to the DOS PATH environment variable.

For information on using DISKCOPY or any DOS command, and DOS environment variables, refer to your IBM DOS User's Guide.
Compatibility Issues

MPASM is compatible with all Microchip development systems currently in production. This includes MPSIM (PIC16/17 software core simulator), PICMASTER, PRO MATE™ (the Microchip Universal Programmer), and the Microchip low-cost development programmer.

It is not compatible with certain older Microchip In-Circuit Emulators.

Microchip Technology is sensitive to your investment in PIC16/17 firmware. Whenever possible, we endeavor to protect that investment and remain backward compatible as new products are developed and released.

MPASM is intended to be backward compatible with source code developed with MPALC and ASM17. In order to provide the largest coverage in backward compatibility, you may encounter small discrepancies in the directive and macro syntax. Whenever practical, MPASM will attempt to make a rational decision as to your coding intent, and flag older syntax as warnings. Unfortunately, this is not always possible.

Specifically, MPASM no longer supports the "." (dot) directives within the macro language. This is offset by the fact that almost all directives can be executed within or without a macro.

As with any software product, users may have questions about the MPASM Assembler. These questions can be posed to Microchip in the following ways:

- Contact your regional sales office. The locations, phone and fax numbers are listed at the end of this manual.
- Contact your local distributor or representative.
- Connect worldwide to the Microchip BBS using the 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 use the Microchip BBS.**

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 services allow multiple users at baud rates up to 9600.

- Contact the factory Applications Group.
Chapter 2. Environment and Usage

Introduction

MPASM provides a universal platform for developing code for PIC16/17s. The product is represented by several programs: MPASM, MPLINK, and MPLIB. Each of these programs has its own Command Line Interface; the former two can be accessed through the MPASM shell while MPLIB can only be accessed through its CLI. This chapter is dedicated to describing the MPASM CLI and the MPASM shell.

Highlights

The points that will be highlighted in this chapter are:

- MPASM Command Line Interface
- MPASM Shell Interface
- MPASM Input Files
- MPASM and Associated Output Files

Terms

Command Line Interface

Command Line Interface or CLI refers to executing a program with options. In the case of MPASM, 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 and linker. It is a DOS Text Graphics screen where the user fills in the appropriate assembly and linker options.

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

MPASM can be invoked through the CLI as follows:

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

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 2) can be overridden with options supplied to the CLI:

- /<option> enables the option
- /<option>+ enables the option
- /<option>- disables the option
## TABLE 2: ASSEMBLER COMMAND LINE OPTIONS

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>N/A</td>
<td>Displays the MPASM Help Panel</td>
</tr>
<tr>
<td>c</td>
<td>On</td>
<td>Enables/Disables case sensitivity</td>
</tr>
<tr>
<td>e</td>
<td>On</td>
<td>Enable/Disable Error File</td>
</tr>
<tr>
<td>h</td>
<td>N/A</td>
<td>Displays the MPASM Help Panel</td>
</tr>
<tr>
<td>l</td>
<td>On</td>
<td>Enables/Disables the listing file generated from the macro assembler. This would include relative addresses in the case of relocatable objects.</td>
</tr>
<tr>
<td>m</td>
<td>Off</td>
<td>Enables/Disable macro expansion</td>
</tr>
</tbody>
</table>
| o      | N/A     | Sets the path for object files: 
\[ /o<path>\object.file \] 
where <path> describes the output directory, and <object.file> to be created. For example: 
\[ /Oc:\temp\file.obj \] |
| p      | None    | Set the processor type: 
\[ /p<processor_type> \] 
Where <processor_type> is one of [ PIC16C54 | PIC16C55 | PIC16C56 | PIC16C57 | PIC16C71 | PIC16C84 | PIC17C42 | PIC16C58 | PIC16C64 ] |
| q      | Off     | Enable/Disable quiet mode (suppress screen output) |
| r      | Hex     | Defines default radix: 
\[ /r<radix> \] 
where <radix> is one of [ HEX | DEC | OCT ] |
| x      | Off     | Enable/Disable cross reference in listing file. |
| a      | INHX8M  | Generate absolute .COD and hex output directly from assembler: 
\[ /a<hex-format> \] 
where <hex-format> is one of [INHX8M | INHX8S | INHX32] |
Shell Interface

The MPASM 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.

### 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. A binary code file (<sourcename>.COD) is automatically created.

### Error File

An error file (<sourcename>.ERR) is created by default. To turn the error field 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.
Chapter 2: Environment and Usage

Listing File

Modify this field as for the Error File. It is used to optionally disable the listing file. This may be a relative or absolute listing file, depending on whether or not the Linker is invoked. The output file name may be modified as for the Error file.

HEX Dump Type

Set this value to generate the desired output format from the Linker. 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

Changing this option will generate the relocatable object code that can be input to the linker. It is modified as for the Error File. Turning it off will have the effect of generating no object file at all.

Source Code Formats

Code written for previous Microchip assemblers (MPALC and ASM17) need not be rewritten according to these standards.

The source code file is created using any ASCII Text File editor (the editor included with the PICMASTER Source Level Debugger was designed for this purpose). 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, or can start in any column if the first non space character is either an asterisk (*) or a semi-colon (;). The maximum column width is 255 characters.
One or more spaces must separate the label and the mnemonic, or the mnemonic and the operand(s). Operands may be separated by a comma. For example:

**EXAMPLE 1: SAMPLE MPASM SOURCE CODE**

```plaintext
; ; Sample MPASM Source Code. It is for illustration only.
; list p=16C54,r=HEX
org 0x1ff ; Reset Vector
          ; Go back to the beginning
org 0x000 ; The main line code
starts here
Start
  movlw 0x0a ; Perform some PIC16/17 code
  movlw 0x0b ;
goto Start ; do it forever...
end
```

**Labels**

All labels must start in column 1. It may be followed by a colon (:), space, tab or the end of line. Comments may also start in column 1 if one of the valid comment denotations is used.

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

Labels may be up to 31 characters long. By default they are case sensitive, but case sensitivity may be overridden by command line or directive options. 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 at least column 2. If there is a label on the same line, they 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. Operand lists must be separated by commas. If the operand requires a fixed number of operands, anything on the line after the operands is ignored. Comments are allowed at the end of the line. If the mnemonics permits a variable number of operands, the end of the operand list is determined by the end of the line or the comment.
Chapter 2: Environment and Usage

Comments

Comments which are on a line by themselves must start with either of the
comment characters (* or ;). Comments at the end of a source line must
be separated from the rest of the line by one or more spaces or tabs.
Anything encountered on the line following the comment character is
ignored until the end of line.

Files Used by MPASM and Utility Functions

There are a number of default file extensions used by MPASM and the
associated utility functions.

TABLE 3: MPASM DEFAULT FILE EXTENSIONS

<table>
<thead>
<tr>
<th>Extension</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASM</td>
<td>Default source code file extension input to MPASM: &lt;source_name&gt;.ASM</td>
</tr>
<tr>
<td>.OBJ</td>
<td>Default output extension for relocatable objects from MPASM: &lt;source_name&gt;.OBJ</td>
</tr>
</tbody>
</table>
| .LST      | Default output extension for listing files generated from either the
           assembl er or MPASM or MPLINK: <source_name>.LST |
| .ERR      | Default output extension from MPASM for specific error files: <source_name>.ERR |
| .MAP      | Default output extension from MPLINK for map output: <source_name>.MAP |
| .HEX      | Default output extension from MPASM or MPLINK for Intel Hex object code (see Appendix A)
           <source_name>.HEX |
| .HXL/.HXH | Default output extensions from MPASM or MPLINK for separate
           low byte and high byte Intel Hex format files:
           <source_name>.HXL, <source_name>.HXH |
| .LIB      | Default extension for library files created by MPLIB, and
           referenced by MPLINK: <source_name>.LIB |
| .LNK      | Default extension for linker script files: <source_name>.LNK |
| .COD      | Default output extension for the symbol and debug file. This file
           may be output from MPASM or MPLINK: <source_name>.COD |
Object Code Formats

MPLINK and MPASM (with absolute addresses) are capable of producing a number of different output formats. See Appendix A.

Listing File Format

Sample MPASM Listing File (.LST)

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT CODE</th>
<th>LINE SOURCE TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td></td>
<td>Sample MPASM Source Code. It is for illustration only.</td>
</tr>
<tr>
<td>0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td></td>
<td>list p=16c54,r=HEX</td>
</tr>
<tr>
<td>0005</td>
<td></td>
<td>org 0x1ff ; Reset Vector</td>
</tr>
<tr>
<td>0006</td>
<td></td>
<td>goto Start ; Go back to the beginning</td>
</tr>
<tr>
<td>0007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td></td>
<td>org 0x000 ; The main line code starts</td>
</tr>
<tr>
<td>0009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td>Start movlw 0x0a ; Perform some PIC16/17 code</td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td>movlw 0x0b ;</td>
</tr>
<tr>
<td>0012</td>
<td></td>
<td>goto Start ; do it forever...</td>
</tr>
<tr>
<td>0013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>0016</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SYMBOl TABLE

<table>
<thead>
<tr>
<th>LABEL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>XXX</td>
</tr>
<tr>
<td>0040</td>
<td>-----</td>
</tr>
<tr>
<td>0180</td>
<td>-----</td>
</tr>
<tr>
<td>01C0</td>
<td>----- X</td>
</tr>
</tbody>
</table>

All other memory blocks unused.

Errors : 0
Warnings : 0
Chapter 2: Environment and Usage

The listing file format produced by MPASM is straightforward:

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, also four characters wide, is reserved for the machine instruction. This is the code that will be executed by the PIC16/17. The third 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.

The symbol table lists all symbols in the program, and where they are defined. 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.

Error File Format (.ERR)

MPASM can generate an error file by supplying the /e option. This file can be used to provide useful information when debugging your code. The file name is followed by the line number of the offending line. A description of the error encountered follows. (The PICMASTER Source Level Debugger will automatically open this file in the case of an error). The error file looks like this:

Error EXAMPLE.ASM 7 :Undefined argument (start in get arg)

Appendix E alphabetically 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.

There are four basic types of directives provided by MPASM.

Highlights

The points that will be highlighted in this chapter are:

- Data Directives
- Listing Directives
- Control Directives
- Macro Directives

Terms

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 and format. They allow the specification of titles, page ejects and other listing control.

Control Directives

Control directives permit sections of conditionally assembled code.

Macro Directives

These directives control the execution and data allocation within macro body definitions.
### TABLE 4: DIRECTIVE SUMMARY

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBANK</td>
<td>Future Feature</td>
<td></td>
</tr>
<tr>
<td>CBL</td>
<td>Define a Block of Constants</td>
<td>cblock [&lt;expr&gt;]</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>Declare Symbol Constant</td>
<td>constant &lt;label&gt;[=expr&gt;, ...,&lt;label&gt;[=expr&gt;] ]</td>
</tr>
<tr>
<td>DATA</td>
<td>Create Numeric and Text Data</td>
<td>data &lt;expr&gt;[,&lt;expr&gt;, ...,&lt;expr&gt;]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA “text_string [,&quot;text_string “, ...]</td>
</tr>
<tr>
<td>DB</td>
<td>Declare Data of One Byte</td>
<td>db &lt;expr&gt;[,&lt;expr&gt;, ...,&lt;expr&gt;]</td>
</tr>
<tr>
<td>#DEFINE</td>
<td>Define a Text Substitution Label</td>
<td>define &lt;name&gt; [&lt;value&gt;] define &lt;name&gt; [=&lt;arg&gt;,...,&lt;arg&gt;] &lt;value&gt;</td>
</tr>
<tr>
<td>DW</td>
<td>Declare Data of One Word</td>
<td>dw &lt;expr&gt;[,&lt;expr&gt;, ...,&lt;expr&gt;]</td>
</tr>
<tr>
<td>ELSE</td>
<td>Begin Alternative Assembly Block to IF</td>
<td>else</td>
</tr>
<tr>
<td>END</td>
<td>End Program Block</td>
<td>end</td>
</tr>
<tr>
<td>ENDC</td>
<td>End an Automatic Constant Block</td>
<td>endc</td>
</tr>
<tr>
<td>ENDIF</td>
<td>End conditional Assembly Block</td>
<td>endif</td>
</tr>
<tr>
<td>ENDM</td>
<td>End a Macro Definition</td>
<td>endm</td>
</tr>
<tr>
<td>ENDW</td>
<td>End a While Loop</td>
<td>endw</td>
</tr>
<tr>
<td>EQU</td>
<td>Define an Assembly Constant</td>
<td>&lt;label&gt; equ &lt;expr&gt;</td>
</tr>
<tr>
<td>ERROR</td>
<td>Issue an Error Message</td>
<td>error “&lt;text_string&gt;”</td>
</tr>
<tr>
<td>EXITM</td>
<td>Exit from a Macro</td>
<td>exitm</td>
</tr>
<tr>
<td>EXPAND</td>
<td>Expand Macro Listing</td>
<td>expand</td>
</tr>
<tr>
<td>FILL</td>
<td>Specify Memory Fill Value</td>
<td>fill &lt;expr&gt;</td>
</tr>
<tr>
<td>IF</td>
<td>Begin Conditionally Assembled Code Block</td>
<td>if &lt;expr&gt;</td>
</tr>
<tr>
<td>IFDEF</td>
<td>Execute If Symbol has Been Defined</td>
<td>ifdef &lt;label&gt;</td>
</tr>
<tr>
<td>IFNDEF</td>
<td>Execute If Symbol has not Been Defined</td>
<td>ifndef &lt;label&gt;</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Include Additional Source File</td>
<td>include &lt;&lt;include_file&gt;&gt; &lt;include_file&gt;</td>
</tr>
<tr>
<td>LIST</td>
<td>Listing Options</td>
<td>list [&lt;list_option&gt;,...,&lt;list_option&gt;]</td>
</tr>
<tr>
<td>LOCAL</td>
<td>Declare Local Macro Variable</td>
<td>local &lt;label&gt;[,&lt;local&gt;]</td>
</tr>
<tr>
<td>MACRO</td>
<td>Declare Macro Definition</td>
<td>label macro [&lt;arg&gt;, ...,&lt;arg&gt;]</td>
</tr>
<tr>
<td>MESSG</td>
<td>Create User Defined Message</td>
<td>messg “&lt;message_text&gt;”</td>
</tr>
<tr>
<td>NOEXPAND</td>
<td>Turn off Macro Expansion</td>
<td>noexpand</td>
</tr>
</tbody>
</table>

(Cont.)
## 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

A table of the MPASM directives is provided as a quick reference at the end of this document.

### CBANK - Future Feature

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>while &lt;expr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>endw</td>
</tr>
</tbody>
</table>

© 1994 Microchip Technology Inc.
MPASM USER'S GUIDE

CBLOCK - Define a Block of Constants

Syntax

cblock [<expr>]

Description

Define a list of named constants. Each is assigned a value of one higher than the last one. The purpose of this directive is to assign address offsets to many labels. The list of names end when and 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 or the current program counter.

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

See Also

ENDC

CONSTANT - Declare Symbol Constant

Syntax

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

Description

<label> is a valid MPASM label, and <expr> is a valid MPASM expression. The expression must be fully resolvable at the time of the assignment.

The CONSTANT directive creates symbols for use in MPASM expressions. Constants may not be reset after having once been initialized. 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.
Chapter 3: Directive Language

Example

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

See Also
SET
VARIABLE

DATA - Create Numeric and Text Data

Syntax

data <expr>, [, <expr>, ... , <expr>]
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 right justified into a whole word, while
strings are packed two to a word with the first character in the most signifi­
cant byte of the word. If an odd number of characters are given in a string,
the final byte is zero filled.

All of the ANSI escape characters may be used in either of the latter two
data formats.

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
DB - Declare Data of One Byte

Syntax

\[ \text{db } <\text{expr}>[,<\text{expr}>,...,<\text{expr}>] \]

Description

Reserve memory bytes, 8-bits of value expression. 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 null filled.

Example

\[ \text{db} \quad \text{'t', 0x0f, 'e', 0x0f, 's', 0x0f, 't', '\n'} \]

See Also

DATA  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 and evaluated if possible.

Using the directive with no <value> 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 the PICMASTER or MPSIM.

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
DW - Declare Data of One Word

Syntax

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

Description

Reserve memory words for data, initializing that space to specific values. 
<expr> is a variable number of valid MPASM expressions. 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.

Example

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

See Also

DATA DB

ELSE - Begin Alternative Assembly Block to IF

Syntax

else

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
Chapter 3: Directive Language

END - End Program Block

Syntax
end

Description
Indicates the end of the program. After program termination, the symbol table is dumped to the listing file.

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

See Also
N/A

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. ENDIF may be used inside a regular program block or macro.

See Also
IF ELSE

ENDM - End a Macro Definition

Syntax
endm

Description
Macro definitions begin with a MACRO directive, and are terminated by the ENDM directive.

Example
make_table macro arg1, arg2
  dw "arg1", 0 ; null terminate table name
  resv arg2 ; reserve storage
  endm

See Also
MACRO EXITM
ENDW - End a While Loop

Syntax

endw

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

<expr> is a valid MPASM expression. The value of the expression is assigned to <label>.

Example

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

See Also

SET #DEFINE
ERROR - Issue an Error Message

Syntax

error "<text_string>"

Description

When conditions dictate that the MPASM assembler encounters an ERROR directive, the <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

EXITM - Exit from a Macro

Syntax

exitm

Description

Forces 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"
endm

See Also

MACRO ENDM
Chapter 3: Directive Language

EXPAND - Expand Macro Listing

Syntax
expand

Description
Causes all macros to be fully expanded 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

FILL - Specify Memory Fill Value

Syntax
fill <expr>

Description
The purpose of the FILL directive is to control the value placed in unused code locations of PROMs and ROMs. The FILL directive enables the fill function and specifies the fill value. This means that the linker output code files will contain record values for these locations. Unused code gaps are created by specifying address advances with ORG and RES directives.

The fill directive can be invoked multiple times to cause different values to be used. If no FILL directives are encountered, then no data records are generated for the unused code locations.

Example

fill 0x1009 ; fill with a constant

See Also
DW ORG RES
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.

Other conditions that may be checked:

- IFABS - If <label> is absolute
- IFNDEF - If <label> is not defined

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 value of one; FALSE a value of zero.

Example

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

See Also

ELSE     ENDIF
Chapter 3: Directive Language

IFDEF - Execute If Symbol has Been Defined

Syntax
ifdef <label>

Description
<label> is a valid MPASM label. If the 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(s)
#define testing 1 ; set testing "on"
ifdef testing
<execute test code> ; this path would be executed.
endif

See Also
#define ELSE ENDIF
ifndef #UNDEFINE
IFNDEF - Execute If Symbol has not Been Defined

Syntax

ifdef <label>

Description

<label> is a valid MPASM label. If the 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 ENDF directive is encountered.

Example

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

See Also

#define ELSE
ifdef #undef #endif
INCLUDE - Include Additional Source File

Syntax

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

Description

The specified file is read in as source code. Upon end-of-file, source code assembly will resume from the original source file. Up to six levels of nesting is permitted. <include_file> may be enclosed in quotes or angle brackets. In either case, only the current working directory will be searched, unless a fully qualified path is specified.

Example

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

See Also

N/A
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=nnn</td>
<td>80</td>
<td>Set column width.</td>
</tr>
<tr>
<td>n=nnn</td>
<td>59</td>
<td>Set lines per page.</td>
</tr>
<tr>
<td>t=ON</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>p=&lt;type&gt;</td>
<td>None</td>
<td>Set processor type: PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC16C84, PIC17C42, PIC16C58, PIC16C64.</td>
</tr>
<tr>
<td>r=&lt;radix&gt;</td>
<td>hex</td>
<td>Set default radix: hex, dec, oct.</td>
</tr>
<tr>
<td>x=ON</td>
<td>OFF</td>
<td>Off</td>
</tr>
</tbody>
</table>

See Also

NOLIST
LOCAL - Declare Local Macro Variable

Syntax
local <label>[,<local>]

Description
<label> is a valid MPASM label. It may be a label that exists outside the context of the macro definition. The directive declares that the specified data elements are to be considered in local context to the macro.

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 the chapter "Macro Language" for more information.

Example

Read macro device, buffer, count
movlw device
movwf ram 20

movlw buffer ; buffer address
movwf ram 21
movlw count ; byte count
callsys_21 ; read file call
endm

See Also

ENDM LOCAL
IF ELSE
ENDIF EXITM

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 255 characters. Issuing a MESSG directive does not set any error return codes.
Example

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

See Also
ERROR

NOEXPAND - Turn off Macro Expansion

Syntax
noexpand

Description
Turns off macro expansion.

See Also
EXPAND

NOLIST - Turn off Listing Output

Syntax
NOLIST

Description
Turn off listing file output.

See Also
LIST
ORG - Set Program Origin

Syntax

<label> org <expr>

Description

Set the program origin for subsequent code at the address defined in <expr>. MPASM outputs relocatable object code, while MPLINK will place the code at the specified address. If <label> is specified, it will be given the value of the <expr>. The default origin is zero.

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
Chapter 3: Directive Language

PROCESSOR - Set Processor Type

Syntax

processor <processor_type>

Description

Set the processor type to <processor_type>:
[16C54 | 16C55 | 16C56 | 16C57 | 16C71 | 16C84 | 17C42]

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 are: hex, dec, or oct.

Example

radix dec

See Also

LIST
RES - Reserve Memory

Syntax
res <mem_units>

Description
The RES directive is a relative org command. The command causes the program counter to be advanced from its current location by the value specified in <mem_units>.

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 altered by SET directives.

Example
area set 0
width set 0x12
length set 0x14
Area set length * width
length set length + 1

See Also
EQU
Chapter 3: Directive Language

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

```
subtitl "<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

```
subtitl "diagnostic section"
```

See Also

TITLE
TITLE - Specify Program Title

Syntax

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

Description

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

Example

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

See Also

LIST, SUBTITLE

#UNDEFINE - Delete a Substitution Label

Syntax

```markdown
#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

```plaintext
#define length 20
...
#undefine length
```

See Also

#DEFINE, IFDEF, IFNDEF
VARIABLE - Declare Symbol Variable

Syntax

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

Description

<label> is a valid MPASM label, and <expr> is a valid MPASM expression. The expression must be fully resolvable at the time of the assignment.

The VARIABLE directive creates symbols for use in MPASM expressions. Variables differ from 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 being that the VARIABLE directive does not require that symbols be initialized when they are declared.

Example

Please refer to the CONSTANT example.

See Also

SET CONSTANT
WHILE - Perform Loop While Condition is True

Syntax

while <expr>
.
.
endw

Description

<expr> is a valid MPASM expression that controls the number of times the loop is performed. 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 value of one; FALSE a value of zero.

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
Chapter 4. Macro Language

Introduction

Macros are user defined sets of instructions and directives that will be included 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 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

As define before, a macro is a collection of assembler instructions that are included in the assembly code when the macro is invoked by the source code. Macros must be defined before their first invocation; forward references to macros are not allowed.

All statements following the MACRO directive (see Chapter 5) are part of the macro definition. Lines consisting of a comment only are not saved in the macro definition. Labels used within the macro must be local to the macro so the macro can be called repetitively.
Local Label

A local label is one that is defined with the \texttt{LOCAL} directive (see Chapter 5). 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 \texttt{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.

Macro Syntax

MPASM macros are defined according to the following syntax:

\begin{verbatim}
<label> macro [<arg>, ..., <arg>]
  ...
  ...
endm
\end{verbatim}

Where \texttt{<label>} is a valid MPASM label and \texttt{<arg>} are 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, PIC16/17 assembly instructions, or MPASM Macro Directives (\texttt{LOCAL} for example). Refer back to Chapter 5. MPASM continues to process the body of the macro until a \texttt{EXITM} or \texttt{ENDM} directive is encountered.

\textbf{NOTE:} Once again, forward references to macros are not permitted.
Chapter 4: Macro Language

Macro Directives

As noted in Chapter 5, there are a few directives that are unique to macro definitions. They make no sense out of the macro context (refer to Chapter 5 for details concerning these directives):

- MACRO
- LOCAL
- 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 uses 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

A variety of string replacement and parsing patterns may appear within the body of a macro. They may be used only within the body of a macro.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;arg&gt;</td>
<td>Substitute the argument text supplied as part of the macro invocation.</td>
</tr>
<tr>
<td>#v(&lt;label&gt;)</td>
<td>Returns the integer value of the simple &lt;label&gt;. Typically, used to create unique variable names with common prefixes or suffixes.</td>
</tr>
</tbody>
</table>
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
    variable a = 0
    while a < 3
        entry#v( a ) dw 0
        a += 1
    endw
endm
```

would generate:
```
entry0 dw 0
entry1 dw 0
entry2 dw 0
entry3 dw 0
```
when invoked.

**Recursive Macros**

Macros may invoke themselves. This is known as recursion. Care should be exercised, as in all cases of recursion, to avoid infinite loops. Macros called recursively will generate their own local variables if the LOCAL directive is used (see Chapter 3).

**Macro Usage**

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

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

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

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

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 previously level of macro expansion.
Chapter 4: Macro Language

Examples

Eight by Eight Multiply

subtitl "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 ; establish local index variable
variable i = 0 ; 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**

For further example, if the following macro were written:

```mpasm
include "16cxx.reg"

; compare file to constant and jump if file
; >= constant.
;
cfl_jge macro file, con, jump_to
   movlw con & Oxff
   subwf file, w
   btfsc status, carry
   goto jump_to
endm
```

and invoked by:

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

it would produce:

```mpasm
movlw max_switch & 0xff
subwf switch_val, w
btfsc status, carr
goto switch_on
```
Chapter 5. 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 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. In general, precedence is established from left to right, and expressions within parentheses are always evaluated first.

Radix

Radix is the base numbering system that the assembler is supposed to use when evaluating expressions. The default radix is hexadecimal (base 16). You can change the default radix (See Chapter 5) and override the default with certain radix override operators. These are described in this chapter.
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 132 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 zeros (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
one
6720 6F75 7470
 movlw 7470
7574 2073 7472
 data 7472
696E 6720 6F6E
650A
  "testing output string"

7465 7374 696E
two
B061
  #define str
7465 7374 696E
  movlw "a"
string
7420 6F75 7470
  data "testing first output"
6720 6669 7273
7420 6F75 7470
7574 2073 7472
696E 6700
```
Chapter 5: Expression Syntax and Operation

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

**TABLE 6: ANSI 'C' ESCAPE SEQUENCES**

<table>
<thead>
<tr>
<th>Escape Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>Bell (alert) character</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace character</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed character</td>
</tr>
<tr>
<td>\n</td>
<td>New line character</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return character</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab character</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical tab character</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>?</td>
<td>Question mark character</td>
</tr>
<tr>
<td>'</td>
<td>Single quote (apostrophe)</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote character</td>
</tr>
<tr>
<td>\000</td>
<td>Octal number (zero, Octal digit, Octal digit)</td>
</tr>
<tr>
<td>\xHH</td>
<td>Hexadecimal number</td>
</tr>
</tbody>
</table>
MPASM USER'S GUIDE

Numeric Constants and Radix

MPASM supports the following radix forms: hexadecimal, decimal, octal, binary, and character. 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.

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

The following table presents the various radix specifications:

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>D'&lt;digits&gt;'</td>
<td>D'100'</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>H'&lt;hex_digits&gt;'</td>
<td>H'9f</td>
</tr>
<tr>
<td>Octal</td>
<td>O'&lt;octal_digits&gt;'</td>
<td>O'777'</td>
</tr>
<tr>
<td>Binary</td>
<td>B'&lt;binary_digits&gt;'</td>
<td>B'00111001'</td>
</tr>
<tr>
<td>Character</td>
<td>'&lt;character&gt;'</td>
<td>'C'</td>
</tr>
<tr>
<td></td>
<td>A'&lt;Character&gt;'</td>
<td>A'C'</td>
</tr>
</tbody>
</table>
## Chapter 5: Expression Syntax and Operation

### TABLE 8: ARITHMETIC OPERATORS AND PRECEDENCE

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>1 + ( d * 4 )</td>
</tr>
<tr>
<td>)</td>
<td>( Length + 1 ) * 256</td>
</tr>
<tr>
<td>!</td>
<td>if ! ( a - b )</td>
</tr>
<tr>
<td>-</td>
<td>-1 * Length</td>
</tr>
<tr>
<td>high</td>
<td>movlw high CTR Table</td>
</tr>
<tr>
<td>low</td>
<td>movlw low CTR Table</td>
</tr>
<tr>
<td>*</td>
<td>a = b * c</td>
</tr>
<tr>
<td>/</td>
<td>a = b / c</td>
</tr>
<tr>
<td>%</td>
<td>entry_len = tot_len % 16</td>
</tr>
<tr>
<td>+</td>
<td>tot_len = entry_len * 8 + 1</td>
</tr>
<tr>
<td>-</td>
<td>entry_len = ( tot - 1 ) / 8</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>&lt;&lt; flags</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>&gt;&gt; flags</td>
</tr>
<tr>
<td>&gt;=</td>
<td>if entry_idx &gt;= num_entries</td>
</tr>
<tr>
<td>&gt;</td>
<td>if entry_idx &gt; num_entries</td>
</tr>
<tr>
<td>&lt;</td>
<td>if entry_idx &lt; num_entries</td>
</tr>
<tr>
<td>&lt;=</td>
<td>if entry_idx &lt;= num_entries</td>
</tr>
<tr>
<td>==</td>
<td>if entry_idx == num_entries</td>
</tr>
<tr>
<td>!=</td>
<td>if entry_idx != num_entries</td>
</tr>
<tr>
<td>&amp;</td>
<td>flags = flags &amp; ERROR_BIT</td>
</tr>
<tr>
<td>^</td>
<td>flags = flags ^ ERROR_BIT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>flags = flags ~ ERROR_BIT</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>if ( len == 512 ) &amp;&amp; b == c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>entry_index = 0</td>
</tr>
<tr>
<td>+=</td>
<td>entry_index += 1</td>
</tr>
<tr>
<td>-=</td>
<td>entry_index -= 1</td>
</tr>
<tr>
<td>*=</td>
<td>entry_index *= entry_length</td>
</tr>
<tr>
<td>/=</td>
<td>entry_total /= entry_length</td>
</tr>
<tr>
<td>%=</td>
<td>entry_index %= 8</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>flags &lt;&lt;= 8</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
<td>flags &gt;&gt;= 3</td>
</tr>
<tr>
<td>&amp;=</td>
<td>flags &amp;= ERROR_FLAG</td>
</tr>
<tr>
<td></td>
<td>=</td>
</tr>
<tr>
<td>^=</td>
<td>flags ^= ERROR_FLAG</td>
</tr>
<tr>
<td>$</td>
<td>goto $ + 3</td>
</tr>
</tbody>
</table>
High / Low

Syntax

<instruction> high <operand>
<instruction> low <operand>

Description

Where <instruction> is an appropriate MPASM assembler instruction and <operand> is an appropriate argument list for that instruction.

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
Chapter 6. MPLIB - MPASM Librarian

Introduction

A librarian is a tool that allows several different objects to be grouped into one logical collection, or library. MPLIB combines several object modules, created with MPASM, into a single file. When a library file is linked with other modules, only those library functions that are referenced by the other modules are linked into the final binary code.

Object modules can be added, deleted, or replaced from the MPLIB Command Line Interface.
Chapter 7. MPLINK - MPASM Linker

Introduction

MPLINK is the MPASM relocatable object linker. It joins any number of object files, together with any library modules, into an executable binary file that is fixed in the PIC16/17's memory. MPLINK can also generate an absolute listing file of the PIC16/17 application. This file can be invaluable when debugging your design; it is in the same general format as the listing file generated for relocatable objects by MPASM.

MPLINK accepts options from either its Command Line Interface or a script file.
Appendix A. Object Code Formats

Introduction

MPASM and MPLINK are capable of outputting several different object file formats, suitable for a variety of programmer and emulator applications.

Highlights

- Intel® HEX Format (INHX8M)
- Intel Split HEX Format (INHX8S)
- Intel HEX 32 Format (INHX32)

Object Code Formats

MPLINK and MPASM (with absolute addresses) are capable of producing a number of different output 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 PIC16/17 series code to third party EPROM programmers (for example, Data I/O® Unisite™ and Logical Devices ALLPRO™).

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 word, 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 including the prefix.
Example

<file_name>.HEX

:100000000000000000000000000000000000000000F0
:0400100000000000EC
:10003200000028006800A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:10006200E8040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF

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

The Intellec 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.

Example

<file_name>.HXL

:0A0000000000000000000000000000000F6
:1000190000284068A8E8C82868A989E28086ABFAA
:10002900E0E82868BFE8C80808034303EBE8FFD0
:03003900FFFD19AD
:0000001FF

<file_name>.HXH

:0A0000000000000000000000000000000F6
:1000190000000000000000010101010102020202CA
:10002900020203030303030404040404050607070883
:030039008080AAA
:0000001FF
Appendix A: Object Code 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 Intel extended linear address record is output also to establish the upper 16 bits of the data address.

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:

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

HH - is a two digit hexadecimal data word.

CC - is a two digit hexadecimal checksum that is the two's compliment of the sum of all preceding bytes in the record including the prefix.
Keeping Current with Microchip Systems

Microchip Technology endeavors at all times to provide the best service and responsiveness possible to its users. The Microchip Technology Systems BBS is one mechanism to facilitate this process.

The BBS is supported as a service to its customers. This is where all of the most recent information regarding systems products can be found. The BBS is monitored several times a week for questions. Truly urgent issues should not be left with the BBS, but referred to your local distributor, or Microchip sales office.

The BBS is an evolving product. Details of its operation will not be found here. This chapter provides a brief discussion of the general services available.

This chapter also describes the Microchip Systems software numbering scheme.

Highlights

The points that will be highlighted in this chapter are:

- Access to the BBS
- Special Interest Groups
- Files
- Mail
- Software Releases
Bulletin Board Access

Access to the bulletin board is 24 hours per day, barring technical or mechanical difficulties. Access is gained by calling your local CompuServe access number. Your modem should be set to 8-bits, No parity, 1 stop bit (8-1-N). The service supports baud rates from 300 to 9600 baud. To access the BBS, follow these steps:

1. Dial your local CompuServe access number.
2. Press <ret> and a garbage string will appear.
3. Enter +<ret> and Host Name: will appear.
4. Enter mchipbbs<ret> and you will be connected to the Microchip BBS.

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

Bulletin Board Usage

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
- Technical Assistance

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. They may be able to provide you with information not available by any other method because of the broad background of the PIC16/17 user community.
Appendix B: Customer Support

There are SIGs for most Microchip systems, including:

- PRO MATE™
- PICMASTER™
- MPASM
- Utilities
- Bugs

These groups are monitored by the Microchip staff.

Files

The Microchip BBS is used regularly 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. These files will be monitored, scanned and approved or disapproved by the moderator of the SIG to which the file is submitted. 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 to keep in touch with fellow Microchip users worldwide.

Consider mailing the moderator of your SIG, or SYSOP, if you have ideas or questions about Microchip products, or the operation of the BBS. Be aware, though, that the SIGs are moderated only about once per day. Truly urgent questions should be referred to your local distributor, sales representative, or FAE. They are your first line of defense.
Software Revisions

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

xx.yy.zz <status>

Where xx is the major release number, yy is the minor number, and zz is the intermediate number. The status field displays one of the following categories:

- Alpha
- Intermediate
- Beta
- Released

Production releases are numbered with major, and minor version numbers like:

3.04 Released

Alpha, Beta and Intermediate releases are numbered with the major, minor and intermediate numbers:

3.04.01 Alpha

Alpha Release

Alpha designated software is engineering software that has not been submitted to any quality assurance testing. In general, this grade of software is intended for software development team access only, but may be sent to selected individuals for conceptual evaluation. Once Alpha grade software has passed quality assurance testing, it may be upgraded to Beta or Intermediate status.
Appendix B: Customer Support

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

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.

Beta Release

Preproduction software is designated as Beta. Beta software is sent to Applications Engineers and Consultants, FAES, and select customers. The Beta Test period is limited to a few weeks. Software that passes Beta testing without having significant flaws, will be production released. Flawed software will be evaluated, repaired, and updated with a new revision number for a subsequent Beta trial.

Production Release

Production released software is software shipped with tool products. Example products are PRO MATE™, 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 for the current production release.
Appendix C. MPALC Conversion Guide

Introduction

MPASM attempts to be backward compatible with MPALC.

It is very possible that your source code will assemble as is. There are, however, a number of inconsistencies between the two assemblers that require some simple changes. In addition, MPASM attempts to provide a clean and simple assembler solution for the future. To that end, there are also a number of changes that are recommended to encourage compatibility going forward.

Highlights

- Required Source Code Updates
- Recommended Source Code Updates

Required Source Code Updates

- Specify processor type at the very top of the first source file in programs assembled with MPASM using either the PROCESSOR or LIST directives, or specify the processor on the command line. MPASM will not assemble your source without this information.
Recommended Source Code Updates

- You are encouraged to move all of your labels to column one, and assembler directives (like LIST and TITLE) to at least column two. An example would be:

```plaintext
list              p=16c54, r=hex
title             Sample Code
#include           "c:\tools\regs.h"
#define             zero     0
#define             one      1
alabel            set      zero
blabel            set      one
org               0x00
goto              Start
org               0x28
Start              goto    Start
end
```

and so on.

- Specify the radix you are assuming using either the RADIX or LIST directives, or the command line option.

- Change all radix overrides currently included in your code to one of those specified in the MPASM User's Guide. For example:

  Change:  `movlw 5D`
  to:      `movlw D'5'`

- Fully qualify moves of label addresses by using the HIGH or LOW directives.
Appendix D. ASM17 Conversion Guide

Introduction

MPASM attempts to be backward compatible with ASM17.

It is very possible that your source code will assemble as is. There are, however, a number of inconsistencies between the two assemblers that require some simple changes. In addition, MPASM attempts to provide a clean and simple assembler solution for the future. To that end, there are also a number of changes that are recommended to encourage compatibility going forward.

Highlights

- Required Source Code Updates
- Recommended Source Code Updates

Required Source Code Updates

- Specify processor type at the very top of the first source file in programs assembled with MPASM using either the PROCESSOR or LIST directives, or specify the processor on the command line. MPASM will not assemble your source without this information.

- Change DOS paths for ASM17 source code from using two backslashes to one.

  Example: Change: 
  
  #include c:\tools\regs.h
  
  to: 
  
  #include c:\tools\regs.h

- Recode any macros that use a variable number of arguments to call out specific arguments. This would only be appropriate for ASM17 code, and is a feature that will be included at some point in the future.

- If you are using the ASM17 HALT directive, recode this as a macro or remove it. This feature will be included at some point in the future.

- If you are using the ASM17 FILL directive, temporarily remove it. This feature will be included at some point in the future.
Recommended Source Code Updates

- You are encouraged to move all of your labels to column one, and assembler directives (like LIST and TITLE) to at least column two. An example would be:

```
list          p=17c42, r=dec
title         Sample Code
#include        "c:\tools\regs.h"
#define          zero     0
#define          one      1
alabel        set       zero
blabel        set       one
org           0x00
goto          Start
org           0x28
Start          goto     Start
end
```

and so on.

- Specify the radix you are assuming using either the RADIX or LIST directives, or the command line option.

- Change all radix overrides currently included in your code to one of those specified in the MPASM User's Guide. For example:

  Change: movlw 5D
  to:      movlw D'5'

- Fully qualify moves of label addresses by using the HIGH or LOW directives.
Appendix E. Error Messages

The following error and warning messages are produced by MPASM. These messages always appear in the listing file directly above each line in which the error occurred.

The error and warning 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.

Error Messages

Address exceeds maximum limit available

You are trying to access memory that is not supported. The current program counter is greater than the maximum program memory limit for the specified processor type. Please refer to the data sheet for this processor to find the valid memory range.

Attempt to redefine reserved word

The words "END", "ERROR", "HIGH", "LOW" and PAGE are reserved words in MPASM. You must not use these words as labels or symbols. Remove or rename any occurrences of these words and then reassemble.

NOTE: It is too costly with regard to both time and space to treat all directives, opcodes and operators as reserved words. MPASM reserves only the above minimal list of words to avoid the most common misuses of directives.

Branch or jump out of range

A branch or jump statement is addressing the last half of a program memory page. This is not allowed. Any instruction which writes to the Program Counter (CALL, JUMP, BRANCH or GOTO) is limited to the first 256 locations of any program memory page.

Call or jump not allowed at this address

A call or computed jump statement is addressing the last half of a program memory page. This is not allowed. Any Instruction which writes to the Program Counter (CALL, JUMP, BRANCH or GOTO) is limited to the first 256 locations of any program memory page.
Couldn’t open . . .

MPASM couldn’t open the specified object file, memory map file, code file, error file, listing file, or cross-reference file. Either the file already exists and is read/write protected, or there is not enough disk space to create or write to the file.

Couldn’t open source file . . .

The source file specified on the command line, or through the interactive menu, does not exist. Check the current directory for the desired file, and verify the spelling of the source file name.

Duplicate label or redefining symbol that cannot be redefined

You have either used the same label name twice in your program, or a constant, a #DEFINE’d symbol, or an EQU’d symbol has been used on the left-hand side of an equation. MPASM does not know which definition to use. This ERROR message appears before BOTH definitions of the symbol, when appropriate.

Error in parameter

One of the options used on the MPASM command line was not a valid option, or an option was incorrectly formatted. Type mpasm /h or mpasm /? at the DOS prompt to see a usage message showing the valid command line options.

Expected . . .

The syntax of the source line is incorrect. MPASM expected to see one thing, but got something different. Check the syntax of the directive or opcode in error in the MPASM User’s Guide.

File not found

The file specified in the shell screen’s source file field does not exist in the current directory. This error appears when the shell interface is used to invoke MPASM, rather than the command line interface. Check the spelling of the file name, and verify that you are in the desired directory. Press any key to continue.

Illegal argument

The radix specified with either a LIST directive or the RADIX directive is not one of the valid radix choices. Change the radix to: DEC for decimal, OCT for octal, or HEX for hexadecimal radix.
Appendix E: Error Messages

Illegal condition

An IF statement is using an illegal comparison operator. The valid conditions which can be checked by an IF statement are:

== (equal to)

!= (not equal to)

> (greater than)

< (less than)

>= (greater than or equal to)

<= (less than or equal to).

Illegal condition, EOF encountered before END or conditional end directive

The END directive is missing, or a CBLOCK, an IF, a WHILE or a MACRO statement is missing an ENDC, ENDIF, ENDW or ENDM respectively.

Illegal conditional compile

There is a problem with the construction of the indicated IF / ELSE / ENDPF statements.

Illegal character . . . in label . . .

The specified label contains an illegal character. Legal characters are: underscore (_), period (.), capital letters (A through Z), lower case letters (a through z), or decimal digits (0 through 9).

Illegal digit

The specified digit is illegal in the context used. The digit is either incorrect for the radix specified in the source file, or is an unsupported ANSI escape sequence. Check the LIST directive used in the source code to verify the specified radix. See Chapter 5: Expression Syntax and Operation in the MPASM User's Guide for valid radix specifications and ANSI 'C' escape sequences.

Illegal opcode

The indicated opcode or directive is not recognized by MPASM. The opcode may be misspelled, or is no longer supported. Or, an otherwise legal opcode may be used in an illegal context. For example, a valid directive, such as LIST, prepended with a pound sign (#LIST) will generate this "Illegal opcode" error message. Other examples are: using an ELSE without an associated IF, or using an INCLUDE directive inside a macro definition.
Include file not found

The file to be included does not exist in the current directory. Check the spelling of the include file name, and verify that you are in the desired directory. If necessary, specify the complete DOS path (for example: 
C:\SOURCE\INCLUDE\FILENAME.H).

Include files nested too deep

The current include file cannot include another file, because you have reached the maximum level of include file nesting. The maximum number of include files nested within each other is five (5).

Macro name missing

The term "macro" has been encountered without an associated name for the macro. Macro names can be any legal, unique MPASM label.

Macros nested too deep

The current macro definition cannot call another macro, because you have reached the maximum level of macro nesting. The maximum number of macros nested within each other is eight (8).

Missing argument(s)

This opcode, directive, or macro call requires at least one more operand (argument) than is provided. Check the instruction set for the proper syntax. This error can also occur if a #DEFINE'd label is missing a value. In this case, the error message won't appear until the label is used as an operand.

Missing terminator

There is an open parenthesis, curly bracket or square bracket without a matching closed parenthesis, curly bracket or square bracket, respectively. This error message can also occur when a comma or blank is expected, but not found.

Nested forward reference not allowed

The indicated label has not been defined yet, and is not allowed to be used before it is defined. Specifically, forward references to macros are not permitted. If a macro call is generating the error, move the call to a point in the code below the macro definition.

This error message also occurs when MPASM cannot tell the type of a given label: variable, constant, address, local variable, or reserved word. The label may be defined more than once.
Appendix E: Error Messages

Out of memory

All the PC's available memory has been used to create code segments, macros and forward references. Try reducing the number of macros in your source file(s). Also, close out of any terminate-and-stay-resident programs (TSR's) and close any applications, then try assembling the file again. If you are running Windows and assembling from a DOS prompt, try exiting Windows and assembling directly from DOS.

Overwriting previous address contents

The location for which MPASM is trying to generate object code has already been used by this program. Usually, an ORG directive for this address occurs prior to the source line that produces this error message. The only time MPASM will allow you to overwrite a previously-used address is if it was reserved with an RES directive.

Processor type is undefined

No processor type has been specified. Use the LIST or PROCESSOR directive in your source code, or use the /p option on the command line, to define a processor type (PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C64, PIC16C71, PIC16C84, or PIC17C42).

Processor type previously defined

A processor type has already been specified. You cannot change processor types in the middle of a program. Check the LIST or PROCESSOR directive in your source code to see which processor type is defined. If you specify the same processor type two or more times in the same program, no error will occur.

Symbol table full

All the PC's available memory has been used to create symbols. MPASM requires more memory to create all the symbols defined in this source file. Eliminate any TSR's, and close any applications, then try assembling the file again. If you are running Windows and assembling from a DOS prompt, try exiting Windows and assembling directly from DOS.

You may get the qualifier "Out of macro space (#define)" appended to this message. If this is the case, eliminate #DEFINE symbols in your code by hardcoding as many values as possible.

Or, you may see this error message qualified with "more than 8 locals in a macro." The maximum number of local variables allowed per macro is eight (8). Eliminate local variables from the macro definition.
**Temp file creation error**

A temporary file could not be created as needed. MPASM uses temporary files when building the symbol table. This error could be caused by a DOS disk write error, if the disk is full.

**Too many arguments**

An opcode has been given too many operands, or a macro has been invoked with too many arguments. Check the processor's instruction set for the proper opcode syntax, or verify the number of arguments in the macro definition.

**Undefined argument**

A label is being used that has not yet been defined. The label might be used as an operand or as a macro argument. If there is a mistake in the argument list of a macro definition, this error message will appear when the macro is invoked, since the arguments were never properly defined.

**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 unknown error.

**WHILE failed to terminate within 256 iterations**

The end condition of a WHILE loop was never met. This is flagged as an "Unknown error" because MPASM doesn't know why the WHILE loop didn't terminate. Check your condition statement for proper syntax and logic.

**Warning Messages**

**Addresses above 32K not currently supported. Using MaxRom.**

MPASM does not currently allow you to access memory above 0x8000 (32K). Eventually, addresses up to 64K will be supported, when the linker and librarian are implemented.

**Argument out of range, least significant bits used**

The operand is not between the maximum and minimum values allowed for this opcode in this processor family. Most "Argument out of range" errors are WARNING messages. The out-of-range argument is truncated to the maximum value allowed. However, any argument that can produce unexpected object code (for example, TRIS 0 would evaluate to a NOP) generates an ERROR message rather than a WARNING.
Appendix E: Error Messages

Crossing page boundary — ensure page bits are set

MPASM is informing you that a page boundary has been crossed, and is recommending that you check to see that you have properly set the page bits. Please refer to the data sheet for your specific processor to find its memory boundaries.

... Is not currently supported

Any directive that is not currently supported generates this WARNING. Please look in the User's Guide for alternative directives, or contact your Microchip Field Applications Engineer for options.

LCALL should only be used for multi-paged program memory

You are using the LCALL opcode, when you should be using CALL instead. LCALL only applies for processors that have more than one page of program memory (such as the PIC16C57). LCALL uses 4 execution cycles, while CALL only uses 2. This is only a WARNING, and correct code is generated.

... May not be handled as preprocessor directive

The #DEFINE and #UNDEFINE directives generate this WARNING to inform you that these directives do not function exactly as you would expect them to operate in the C language. Please refer to the User's Guide descriptions for these directives to determine how they will behave.

... Not a single byte quantity

You have specified a literal value that is larger than 8-bits. This WARNING is produced when you use an opcode that requires a single-byte value (such as MOVLW). MPASM truncates to the lower 8 bits of the value. You may be more specific by preceding the value with a HIGH or LOW operator.

This number is being treated as a binary representation

The specified number is ambiguous, and could be interpreted as either a binary or hexadecimal number. MPASM is assuming that it is binary. Example: b0101.

MPASM expects hexadecimal numbers to be represented as 0xb0101 or as H'b0101'.
Quick Reference Guide

This quick reference guide is supplied to give you all of the instructions for the Microchip family of microcontrollers, including their description, function and status bits modified.

If more information is required, please refer to the data sheets for the PIC16/17 in question.

Highlights

- Directive Summary
- PIC16C5X Instruction Set and Notes
- PIC16CXX Instruction Set and Notes
- PIC17CXX Instruction Set and Notes

Terms

PIC16C5X

Microchip's low-end 8-bit microcontroller with 12-bit wide instruction set currently including: PIC16C54, PIC16C55, PIC16C56, PIC16C57, and PIC16C58.

PIC16CXX

Microchip's mid-range 8-bit microcontroller with 14-bit wide instruction set currently including: PIC16C71, PIC16C64, and PIC16C84.

PIC17CXX

Microchip's high-end 8-bit microcontroller family with 16-bit wide instruction set currently including PIC17C42.
### Directive Summary

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBANK</td>
<td>Future Feature</td>
<td></td>
</tr>
<tr>
<td>CBLOCK</td>
<td>Define a Block of Constants</td>
<td>cblock [expr]</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>Declare Symbol Constant</td>
<td>constant [label]=expr, ... [label]=expr</td>
</tr>
<tr>
<td>DATA</td>
<td>Create Numeric and Text Data</td>
<td>data expr,...,expr DATA &quot;text_string&quot;,...</td>
</tr>
<tr>
<td>DB</td>
<td>Declare Data of One Byte</td>
<td>db expr,...,expr</td>
</tr>
<tr>
<td>#DEFINE</td>
<td>Define a Text Substitution Label</td>
<td>define name [value]</td>
</tr>
<tr>
<td>DW</td>
<td>Declare Data of One Word</td>
<td>dw expr,...,expr</td>
</tr>
<tr>
<td>ELSE</td>
<td>Begin Alternative Assembly Block to IF</td>
<td>else</td>
</tr>
<tr>
<td>END</td>
<td>End Program Block</td>
<td>end</td>
</tr>
<tr>
<td>ENDC</td>
<td>End an Automatic Constant Block</td>
<td>endc</td>
</tr>
<tr>
<td>ENDIF</td>
<td>End conditional Assembly Block</td>
<td>endif</td>
</tr>
<tr>
<td>ENDM</td>
<td>End a Macro Definition</td>
<td>endm</td>
</tr>
<tr>
<td>ENDW</td>
<td>End a While Loop</td>
<td>endw</td>
</tr>
<tr>
<td>EQU</td>
<td>Define an Assembly Constant</td>
<td>&lt;label&gt; equ expr</td>
</tr>
<tr>
<td>ERROR</td>
<td>Issue an Error Message</td>
<td>error &quot;text_string&quot;</td>
</tr>
<tr>
<td>EXITM</td>
<td>Exit from a Macro</td>
<td>exitm</td>
</tr>
<tr>
<td>EXPAND</td>
<td>Expand Macro Listing</td>
<td>expand</td>
</tr>
<tr>
<td>FILL</td>
<td>Specify Memory Fill Value</td>
<td>fill expr</td>
</tr>
<tr>
<td>IF</td>
<td>Begin Conditionally Assembled Code Block</td>
<td>if expr</td>
</tr>
<tr>
<td>IFDEF</td>
<td>Execute If Symbol has Been Defined</td>
<td>ifdef label</td>
</tr>
<tr>
<td>IFNDEF</td>
<td>Execute If Symbol has not been Defined</td>
<td>ifdef label</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Include Additional Source File</td>
<td>include &quot;include_file&quot;</td>
</tr>
<tr>
<td>LIST</td>
<td>Listing Options</td>
<td>list [list_option],...[list_option]</td>
</tr>
<tr>
<td>LOCAL</td>
<td>Declare Local Macro Variable</td>
<td>local [local]</td>
</tr>
<tr>
<td>MACRO</td>
<td>Declare Macro Definition</td>
<td>macro [arg],...[arg]</td>
</tr>
<tr>
<td>MESSG</td>
<td>Create User Defined Message</td>
<td>messg &quot;message_text&quot;</td>
</tr>
<tr>
<td>NOLIST</td>
<td>Turn off Listing Output</td>
<td>nolist</td>
</tr>
<tr>
<td>ORG</td>
<td>Set Program Origin</td>
<td>org expr</td>
</tr>
<tr>
<td>PAGE</td>
<td>Insert Listing Page Eject</td>
<td>page</td>
</tr>
<tr>
<td>PROCESSOR</td>
<td>Set Processor Type</td>
<td>processor &lt;processor_type&gt;</td>
</tr>
<tr>
<td>RADIX</td>
<td>Specify Default Radix</td>
<td>radix default_radix</td>
</tr>
<tr>
<td>RES</td>
<td>Reserve Memory</td>
<td>res [mem_units]</td>
</tr>
<tr>
<td>SET</td>
<td>Define an Assembler Variable</td>
<td>set expr</td>
</tr>
<tr>
<td>SPACE</td>
<td>Insert Blank Listing Lines</td>
<td>space expr</td>
</tr>
<tr>
<td>SUBTITLE</td>
<td>Specify Program Subtitle</td>
<td>subtit &quot;sub_text&quot;</td>
</tr>
<tr>
<td>TITLE</td>
<td>Specify Program Title</td>
<td>title &quot;title_text&quot;</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>Declare a Substitution Label</td>
<td>variable [label]=[expr],...[label]=[expr]</td>
</tr>
<tr>
<td>WHILE</td>
<td>Perform Loop While Condition is True</td>
<td>while expr; : endw</td>
</tr>
</tbody>
</table>

**TABLE 9: DIRECTIVE SUMMARY**
PIC16C5X 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 highly orthogonal and is grouped into three basic categories:

- Byte Oriented operations
- Bit Oriented Operations
- Literal and Control Operations

The following tables list the instructions recognized by the MPASM assembler, where:

### TABLE 10: PIC16C5X OPERAND CODES

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>Register file address (0x00 to 0xFF)</td>
</tr>
<tr>
<td>$w$</td>
<td>Working register (accumulator)</td>
</tr>
<tr>
<td>$b$</td>
<td>Bit address within an 8 bit file register</td>
</tr>
<tr>
<td>$k$</td>
<td>Literal field, constant data or label.</td>
</tr>
<tr>
<td>$x$</td>
<td>Don't care location.</td>
</tr>
<tr>
<td>$d$</td>
<td>Destination select; $d = 0$: store result in $w$ ($f0A$), $d = 1$: store result in file register $f$. Default is $d = 1$.</td>
</tr>
</tbody>
</table>
### TABLE 11: PIC16C5X BYTE ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>11df</td>
<td>ffff</td>
<td>ADDWF f,d</td>
<td>W+f→d</td>
<td>c dc z</td>
<td>1,2,4</td>
</tr>
<tr>
<td>0001</td>
<td>01df</td>
<td>ffff</td>
<td>ANDWF f,d</td>
<td>W.AND.f →d</td>
<td>z</td>
<td>2,4</td>
</tr>
<tr>
<td>0000</td>
<td>01ff</td>
<td>ffff</td>
<td>CLRWF f</td>
<td>0 → f</td>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>0000</td>
<td>0110</td>
<td>0000</td>
<td>CLRWF 0</td>
<td>0 → W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>01df</td>
<td>ffff</td>
<td>COMP f,d</td>
<td>f → d</td>
<td>z</td>
<td>2,-1</td>
</tr>
<tr>
<td>0000</td>
<td>11df</td>
<td>ffff</td>
<td>DECF f,d</td>
<td>f - 1 → d</td>
<td>z</td>
<td>2,4</td>
</tr>
<tr>
<td>0010</td>
<td>11df</td>
<td>ffff</td>
<td>DCF f,d, skip if zero</td>
<td>f - 1 → d, skip if zero</td>
<td>None</td>
<td>2,4</td>
</tr>
<tr>
<td>0010</td>
<td>10df</td>
<td>ffff</td>
<td>INCF f,d</td>
<td>f + 1 → d</td>
<td>z</td>
<td>2,4</td>
</tr>
<tr>
<td>0011</td>
<td>11df</td>
<td>ffff</td>
<td>INCPS f,d, skip if zero</td>
<td>f + 1 → d, skip if zero</td>
<td>None</td>
<td>2,4</td>
</tr>
<tr>
<td>0001</td>
<td>00df</td>
<td>ffff</td>
<td>IORWF f,d</td>
<td>Wf → d</td>
<td>z</td>
<td>2,4</td>
</tr>
<tr>
<td>0010</td>
<td>00df</td>
<td>ffff</td>
<td>MOVF f,d</td>
<td>f → d</td>
<td>z</td>
<td>2,4</td>
</tr>
<tr>
<td>0000</td>
<td>01ff</td>
<td>ffff</td>
<td>MOVWF f</td>
<td>W → f</td>
<td>None</td>
<td>1,4</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td>No operation</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>01df</td>
<td>ffff</td>
<td>RLF f,d</td>
<td>f&lt;n&gt; → d&lt;n+1&gt;, C → d&lt;n&gt;</td>
<td>c</td>
<td>2,4</td>
</tr>
<tr>
<td>0011</td>
<td>00df</td>
<td>ffff</td>
<td>RRf f,d</td>
<td>f&lt;n&gt; → d&lt;n-1&gt;, C → d&lt;n&gt;, c</td>
<td>2,4</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>10df</td>
<td>ffff</td>
<td>SUBWF f,d</td>
<td>f - W → d[f+ W +1→d]</td>
<td>c dc z</td>
<td>1,2,4</td>
</tr>
<tr>
<td>0011</td>
<td>10df</td>
<td>ffff</td>
<td>SWAPF f,d</td>
<td>f&lt;0:3&gt; ← f&lt;4:7&gt; → d</td>
<td>None</td>
<td>2,4</td>
</tr>
<tr>
<td>0001</td>
<td>10df</td>
<td>ffff</td>
<td>XORWF f,d</td>
<td>W ⊕ f → d</td>
<td>z</td>
<td>2,4</td>
</tr>
</tbody>
</table>

### TABLE 12: PIC16C5X BIT ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>bbbf</td>
<td>ffff</td>
<td>BCF f,b</td>
<td>0 → f(b)</td>
<td>None</td>
<td>2,4</td>
</tr>
<tr>
<td>0101</td>
<td>bbbf</td>
<td>ffff</td>
<td>BSF f,b</td>
<td>1 → f(b)</td>
<td>None</td>
<td>2,4</td>
</tr>
<tr>
<td>0110</td>
<td>bbbf</td>
<td>ffff</td>
<td>BTFS C f,b</td>
<td>skip if f(b) = 0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>bbbf</td>
<td>ffff</td>
<td>BTFFS f,b</td>
<td>skip if f(b) = 1</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
TABLE 13: PIC16C5X LITERAL AND CONTROL OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110 kkkk kkkk</td>
<td>Ekk</td>
<td>ANDLW</td>
<td>k &amp; literal and W</td>
<td>k &amp; W → W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>1001 kkkk kkkk</td>
<td>9kk</td>
<td>CALL</td>
<td>Call subroutine</td>
<td>PC + 1 → TOS, k → PC</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>0000 0000 0100</td>
<td>004</td>
<td>CLRWD</td>
<td>Clear watchdog timer</td>
<td>0 → WD (and Prescaler if assigned)</td>
<td>to pd</td>
<td></td>
</tr>
<tr>
<td>101k kkkk kkkk</td>
<td>A</td>
<td></td>
<td>Goto address (k is nine bits)</td>
<td>k → PC(9 bits)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>1101 kkkk kkkk</td>
<td>Dkk</td>
<td>IORLW</td>
<td>k &amp; literal and W</td>
<td>k→W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>1100 kkkk kkkk</td>
<td>Ckk</td>
<td>MOVIL</td>
<td>Move Literal to W</td>
<td>k → W</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0000 0000 0010</td>
<td>002</td>
<td>OPTION</td>
<td>Load OPTION register</td>
<td>W → OPTION Register</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>1000 kkkk kkkk</td>
<td>8kk</td>
<td>RETLW</td>
<td>Return with literal in W</td>
<td>k → W, TOS → PC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0000 0000 0011</td>
<td>003</td>
<td>SLEEP</td>
<td>Go into stand by mode</td>
<td>0 → WD, stop oscillator</td>
<td>to pd</td>
<td></td>
</tr>
<tr>
<td>0000 0000 00ff</td>
<td>00f</td>
<td>TRIS</td>
<td>Tristate port f</td>
<td>W → I/O control reg f</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>1111 kkkk kkkk</td>
<td>Fkk</td>
<td>XORLW</td>
<td>k &amp; literal and W</td>
<td>k Φ W → W</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

PIC16C5X Notes

1. If the destination of any instruction is the program counter (register file 2), the 8-bit destination value will be loaded into the lower 8-bits of the program counter (PC) and the 9th bit of the PC will be cleared. For the PIC16C56 and PIC16C57, the upper 3 bits of the status register (register file 3), PA2:PA0, are loaded into the most significant 3 bits of the PC(11:9). In case of the GOTO instruction, the lower 9 bits of the PC are loaded with the destination address, and the 3 most significant bits of the PC(11:9) are loaded with PA2:PA0 from the status register.

2. When an I/O register is modified as a function of itself (i.e. MOVF 6, 1) the value used will be the value present on the pins themselves. For example, a tristated pin with data latch "1" but is driven low by an external device will be relatched in the low state.

3. The instruction "TRIS f", where f = 5, 6, or 7 causes the contents of the w register to be written to the tristate latches of the specified file (port). A one forces the pin to a high impedance state and disables the output buffers.

4. If this instruction is executed on file register f1 (and, where applicable d=1), the prescaler will be cleared if assigned to the RTCC.
Pic16Cxx 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).

All instructions execute in a single instruction cycle unless otherwise noted. Any unused opcode is executed as a NOP. The instruction set is highly orthogonal and is grouped into three basic categories:

- Byte Oriented operations
- Bit Oriented Operations
- Literal and Control Operations

The following tables list the instructions recognized by the MPASM assembler.
### TABLE 14: PIC16CXX BYTE ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0111 dfff ffff</td>
<td>07ff</td>
<td>ADDWF f,d</td>
<td>Add W and f</td>
<td>W+f→d</td>
<td>c dc z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0101 dfff ffff</td>
<td>05ff</td>
<td>ANDWF f,d</td>
<td>AND W and f</td>
<td>W.AND.f→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0001 lfff ffff</td>
<td>01ff</td>
<td>CLRf</td>
<td>Clear f</td>
<td>0→f</td>
<td>z</td>
<td>3</td>
</tr>
<tr>
<td>00 0001 0xxxx xxxxx</td>
<td>0100</td>
<td>CLRW</td>
<td>Clear W</td>
<td>0→W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>00 1001 dfff ffff</td>
<td>09ff</td>
<td>COMF f,d</td>
<td>Complement f</td>
<td>¬f→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0011 dfff ffff</td>
<td>03ff</td>
<td>DECF f,d</td>
<td>Decrement f</td>
<td>f→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1011 dfff ffff</td>
<td>08ff</td>
<td>DECFZ f,d</td>
<td>Decrement f, skip if zero</td>
<td>f→d, skip if zero</td>
<td>None</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1010 dfff ffff</td>
<td>0aff</td>
<td>INCF f,d</td>
<td>Increment f</td>
<td>f+1→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1111 dfff ffff</td>
<td>0bff</td>
<td>INCFZ f,d</td>
<td>Increment f, skip if zero</td>
<td>f+1→d, skip if zero</td>
<td>None</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0100 dfff ffff</td>
<td>04ff</td>
<td>IORWF f,d</td>
<td>Inclusive OR W and f</td>
<td>Wf→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1000 dfff ffff</td>
<td>08ff</td>
<td>MOVf</td>
<td>Move f</td>
<td>f→d</td>
<td>z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0000 lfff ffff</td>
<td>00ff</td>
<td>MOVW f</td>
<td>Move W to f</td>
<td>W→f</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>00 0000 0x0 0000</td>
<td>0000</td>
<td>NOP</td>
<td>No operation</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 1101 dfff ffff</td>
<td>0Dff</td>
<td>RLF f,d</td>
<td>Rotate left f</td>
<td>f&lt;n−d&lt;n+1&gt;, C→d&lt;f&gt;→C</td>
<td>c</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1100 dfff ffff</td>
<td>0Cff</td>
<td>RRF f,d</td>
<td>Rotate right f</td>
<td>f&lt;n−d&lt;n−1&gt;, C→d&lt;7&gt;, f&lt;0&gt;→C</td>
<td>c</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0110 dfff ffff</td>
<td>02ff</td>
<td>SUBWF f,d</td>
<td>Subtract W from f</td>
<td>f-W→d[f+W+1→d]</td>
<td>c dc z</td>
<td>2,3</td>
</tr>
<tr>
<td>00 1110 dfff ffff</td>
<td>08ff</td>
<td>SWAPF f,d</td>
<td>Swap haves f</td>
<td>f&lt;0:3&gt;→f&lt;4:7→d</td>
<td>None</td>
<td>2,3</td>
</tr>
<tr>
<td>00 0110 dfff ffff</td>
<td>06ff</td>
<td>XORWF f,d</td>
<td>Exclusive OR W and f</td>
<td>W⊕f→d</td>
<td>z</td>
<td>2,3</td>
</tr>
</tbody>
</table>

### TABLE 15: PIC16CXX BIT ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 00bb bfff ffff</td>
<td>1bff</td>
<td>BCF f,b</td>
<td>Bit clear f</td>
<td>0→f(b)</td>
<td>None</td>
<td>2,3</td>
</tr>
<tr>
<td>01 01bb bfff ffff</td>
<td>1bff</td>
<td>BSF f,b</td>
<td>Bit set f</td>
<td>1→f(b)</td>
<td>None</td>
<td>2,3</td>
</tr>
<tr>
<td>01 10bb bfff ffff</td>
<td>1bff</td>
<td>BTFSC f,b</td>
<td>Bit test, skip if clear</td>
<td>skip if f(b) = 0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>01 11bb bfff ffff</td>
<td>1bff</td>
<td>BTFSS f,b</td>
<td>Bit test, skip if set</td>
<td>skip if f(b) = 1</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 16: PIC16CXX LITERAL AND CONTROL OPERATIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 111x kkkk kkkk</td>
<td>3Ekk</td>
<td>ADDLW k</td>
<td>Add literal to W</td>
<td>k +W → W</td>
<td>c dc z</td>
<td></td>
</tr>
<tr>
<td>11 1001 kkkk kkkk</td>
<td>39kk</td>
<td>ANDLW k</td>
<td>And literal and W</td>
<td>k &amp; W → W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>10 0kkk kkkk kkkk</td>
<td>2kkk</td>
<td>CALL k</td>
<td>Call subroutine</td>
<td>PC + 1→TOS, k→PC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>00 0000 0110 0100</td>
<td>0064</td>
<td>CLRW T</td>
<td>Clear watch dog timer</td>
<td>0 → WDT (and Prescaler if assigned)</td>
<td>to pd</td>
<td></td>
</tr>
<tr>
<td>10 1kkk kkkk kkkk</td>
<td>2kkk</td>
<td>GOTO k</td>
<td>Goto address (k is nine bits)</td>
<td>k→PC(9 bits)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>11 1000 kkkk kkkk</td>
<td>38kk</td>
<td>IORLW k</td>
<td>Incl. OR literal and W</td>
<td>k∨W → W</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>11 00xx kkkk kkkk</td>
<td>30kk</td>
<td>MOVLW k</td>
<td>Move Literal to W</td>
<td>k → W</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>00 0000 0110 0010</td>
<td>0062</td>
<td>OPTION</td>
<td>Load OPTION register</td>
<td>W → OPTION Register</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>00 0000 0000 1001</td>
<td>0009</td>
<td>RETFIIB</td>
<td>Return from Interrupt</td>
<td>TOS → PC, 1 → GIE</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>11 01xx kkkk kkkk</td>
<td>34kk</td>
<td>RETLM k</td>
<td>Return with literal in W</td>
<td>k → W, TOS → PC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>00 0000 0000 1000</td>
<td>0008</td>
<td>RETURN</td>
<td>Return from subroutine</td>
<td>TOS → PC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>00 0000 0110 0011</td>
<td>0063</td>
<td>SLEEP</td>
<td>Go into stand by mode</td>
<td>0 → WDT, stop oscillator</td>
<td>to pd</td>
<td></td>
</tr>
<tr>
<td>11 110x kkkk kkkk</td>
<td>3Ckk</td>
<td>SUBLW k</td>
<td>Subtract W from literal</td>
<td>k - W → W</td>
<td>c dc z</td>
<td></td>
</tr>
<tr>
<td>00 0000 0110 0fff</td>
<td>006f</td>
<td>TRIS f</td>
<td>Tristate port f</td>
<td>W → I/O control reg f</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>11 1010 kkkk kkkk</td>
<td>3Akk</td>
<td>XORLW k</td>
<td>Exclusive OR literal and W</td>
<td>k ⊕ W → W</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 17: PIC16CXX SPECIAL INSTRUCTION MNEMONICS

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
<th>Equivalent Operation(s)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Carry</td>
<td>CLRC</td>
<td>BCF 3.0</td>
<td>-</td>
</tr>
<tr>
<td>Set Carry</td>
<td>SETC</td>
<td>BSF 3.0</td>
<td>-</td>
</tr>
<tr>
<td>Clear Digit Carry</td>
<td>CLRDC</td>
<td>BCF 3.1</td>
<td>-</td>
</tr>
<tr>
<td>Set Digit Carry</td>
<td>SETDC</td>
<td>BSF 3.1</td>
<td>-</td>
</tr>
<tr>
<td>Clear Zero</td>
<td>CLRZ</td>
<td>BCF 3.2</td>
<td>-</td>
</tr>
<tr>
<td>Set Zero</td>
<td>SETZ</td>
<td>BSF 3.2</td>
<td>-</td>
</tr>
<tr>
<td>Skip on Carry</td>
<td>SKPC</td>
<td>BTFSS 3.0</td>
<td>-</td>
</tr>
<tr>
<td>Skip on No Carry</td>
<td>SKPNC</td>
<td>BTFSC 3.0</td>
<td>-</td>
</tr>
<tr>
<td>Skip on Digit Carry</td>
<td>SKPNDc</td>
<td>BTFSC 3.1</td>
<td>-</td>
</tr>
<tr>
<td>Skip on Zero</td>
<td>SKPZ</td>
<td>BTFSS 3.2</td>
<td>-</td>
</tr>
<tr>
<td>Skip on Non Zero</td>
<td>SKPNZ</td>
<td>BTFSC 3.2</td>
<td>-</td>
</tr>
<tr>
<td>Test File</td>
<td>TSTF f</td>
<td>MOVF 1.0, z</td>
<td>Z</td>
</tr>
<tr>
<td>Move File to W</td>
<td>MOVFW f</td>
<td>MOVF 1.0, z</td>
<td>Z</td>
</tr>
<tr>
<td>Negate File</td>
<td>NEGF f,d</td>
<td>COMF 1.1, INCF f,d, Z</td>
<td></td>
</tr>
<tr>
<td>Add Carry to File</td>
<td>ADDCF f,d</td>
<td>BTFSC 3.0, INCF f,d, Z</td>
<td></td>
</tr>
<tr>
<td>Subtract Carry from File</td>
<td>SUBCF f,d</td>
<td>BTFSC 3.0, DECF f,d</td>
<td></td>
</tr>
<tr>
<td>Add Digit Carry to File</td>
<td>ADDDCF f,d</td>
<td>BTFSC 3.1, INCF f,d, Z</td>
<td></td>
</tr>
<tr>
<td>Subtract Digit Carry from File</td>
<td>SUBDCF f,d</td>
<td>BTFSC 3.1, DECF f,d, Z</td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>B k</td>
<td>GOTO k</td>
<td>-</td>
</tr>
<tr>
<td>Branch on Carry</td>
<td>BC k</td>
<td>BTFSC 3.0, GOTO k</td>
<td>-</td>
</tr>
<tr>
<td>Branch on No Carry</td>
<td>BNC k</td>
<td>BTFSS 3.0, GOTO k</td>
<td>-</td>
</tr>
<tr>
<td>Branch on Digit Carry</td>
<td>BDC k</td>
<td>BTFSC 3.1, GOTO k</td>
<td>-</td>
</tr>
<tr>
<td>Branch on No Digit Carry</td>
<td>BNDC k</td>
<td>BTFSS 3.1, GOTO k</td>
<td>-</td>
</tr>
<tr>
<td>Branch on Zero</td>
<td>BZ k</td>
<td>BTFSC 3.2, GOTO k</td>
<td>-</td>
</tr>
</tbody>
</table>
TABLE 17: PIC16CXX SPECIAL INSTRUCTION MNEMONICS (CONT)

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
<th>Equivalent Operation(s)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch on Non Zero</td>
<td>BNZ k</td>
<td>BTFSS 3,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GOTO k</td>
<td></td>
</tr>
<tr>
<td>Call across page boundary</td>
<td>LCALL k</td>
<td>BCF 3,5 or BSF 3,5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BCF 3,6 or BSF 3,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL k</td>
<td></td>
</tr>
</tbody>
</table>

PIC16CXX Notes

1. TRIS and OPTION instructions are included in the instruction set for upward compatibility with the PIC16C5X products. Microchip strongly recommends not using these instructions for new code development. Instead of using these instructions, directly address the TRIS and OPTION registers to obtain equivalent control. These instructions may not be supported in future PIC16CXX products.

2. When an I/O register is modified as a function of itself (i.e. MOVF 6, 1) the value used will be the value present on the pins themselves. For example, a tristated pin with data latch “1” but is driven low by an external device will be relatched in the low state.

3. If this instruction is executed on file register f1 (and, where applicable d=1), the prescaler will be cleared if assigned to the RTCC.
PIC17C42 Instruction Set

The PIC17C42 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. A few instructions operate solely on a file register (BSF for example).

All instructions execute in a single instruction cycle unless otherwise noted. Any unused opcode is executed as a NOP. The instruction set is highly orthogonal and is grouped into four basic categories:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The following tables list the instructions recognized by the MPASM assembler, where:

**TABLE 18: PIC17C42 OPERAND CODES**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>Register file address (0x00 to 0xFF)</td>
</tr>
<tr>
<td>( p )</td>
<td>Peripheral register file address (0x00 to 0x1f)</td>
</tr>
<tr>
<td>( b )</td>
<td>Bit address within an 8 bit file register</td>
</tr>
<tr>
<td>( i )</td>
<td>Table pointer control; ( i = 0 ): do not change, ( i = 1 ): increment after instruction execution.</td>
</tr>
<tr>
<td>( t )</td>
<td>Table byte select; ( t = 0 ): perform operation on lower byte, ( t = 1 ): perform operation on upper byte.</td>
</tr>
<tr>
<td>( k )</td>
<td>Literal field, constant data or label.</td>
</tr>
<tr>
<td>( x )</td>
<td>Don't care location.</td>
</tr>
<tr>
<td>( d )</td>
<td>Destination select; ( d = 0 ): store result in ( W ) (f0A), ( d = 1 ): store result in file register ( f ). Default is ( d = 1 ).</td>
</tr>
</tbody>
</table>
## TABLE 19: PIC17C42 DATA MOVE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>011p pppp ffff ffff</td>
<td>6pff</td>
<td>MOVFP f,p</td>
<td>Move f to p</td>
<td>f → p</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>1011 1000 kkkk kkkk</td>
<td>B8kk</td>
<td>MOVLB k</td>
<td>Move literal to BSR</td>
<td>k → BSR</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>010p pppp ffff ffff</td>
<td>4pff</td>
<td>MOVPF p,f</td>
<td>Move p to f</td>
<td>p → w</td>
<td>Z</td>
<td>4</td>
</tr>
<tr>
<td>0000 0001 ffff ffff</td>
<td>01ff</td>
<td>MOVWF f</td>
<td>Move W to F</td>
<td>w → f</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>1010 101i ffff ffff</td>
<td>a8ff</td>
<td>TABLDRD t,i,f</td>
<td>Read data from table latch into file f, then update table latch with 16-bit contents of memory location addressed by table pointer</td>
<td>TBLATH → f if t = 1, TBLATL → f if t = 0; ProgMem(TBLPTR)→TBLAT; TBLPTR+1→TBLPTR if i=1</td>
<td>None</td>
<td>8,10</td>
</tr>
<tr>
<td>111i ffff ffff</td>
<td>acff</td>
<td>TABLWT t,i,f</td>
<td>Write data from file f to table latch and then write 16-bit table latch to program memory location addressed by table pointer</td>
<td>f→TBLATH if t = 1, f→TBLATL if t = 0; TBLAT→ProgMem(TBLPTR); TBLPTR+1→TBLPTR if i=1</td>
<td>None</td>
<td>6</td>
</tr>
<tr>
<td>1010 001x ffff ffff</td>
<td>a0ff</td>
<td>TLRD t,f</td>
<td>Read data from table latch into file f (table latch unchanged)</td>
<td>TBLATH → f if t = 1, TBLATL → f if t = 0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>1010 011x ffff ffff</td>
<td>a4ff</td>
<td>TLWT t,f</td>
<td>Write data from file f into table latch</td>
<td>f → TBLATH if t = 1, f → TBLATL if t = 0</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
## TABLE 20: PIC17C42 ARITHMETIC AND LOGICAL INSTRUCTIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011 0001</td>
<td>kkkk kkkk</td>
<td>b1kk</td>
<td>ADDLW k</td>
<td>Add literal to W</td>
<td>(W+k_ → W)</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>0000 111d</td>
<td>ffff ffff</td>
<td>0eff</td>
<td>ADDWF f,d</td>
<td>Add W to F</td>
<td>(W+f) → d</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>0001 001d</td>
<td>ffff ffff</td>
<td>10ff</td>
<td>ADDWF f,d</td>
<td>Add W and Carry to f</td>
<td>(W+f+C) → d</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>1011 0111</td>
<td>kkkk kkkk</td>
<td>b5kk</td>
<td>ANDLW k</td>
<td>AND Literal and W</td>
<td>(W.AND.k) → W</td>
<td>z</td>
</tr>
<tr>
<td>0000 101d</td>
<td>ffff ffff</td>
<td>0aff</td>
<td>ANDWF f,d</td>
<td>AND W with f</td>
<td>(W.AND.f) → d</td>
<td>z</td>
</tr>
<tr>
<td>0010 100d</td>
<td>ffff ffff</td>
<td>28ff</td>
<td>CLRF f,d</td>
<td>Clear f and Clear d</td>
<td>0x00 → f, 0x00 → d</td>
<td>None 3</td>
</tr>
<tr>
<td>0001 011d</td>
<td>ffff ffff</td>
<td>12ff</td>
<td>COMF f,d</td>
<td>Complement f</td>
<td>→ d</td>
<td>z</td>
</tr>
<tr>
<td>1010 111d</td>
<td>ffff ffff</td>
<td>2eff</td>
<td>DAW f,d</td>
<td>Dec. adjust W, store in f,d</td>
<td>W adjusted → f and d</td>
<td>c</td>
</tr>
<tr>
<td>0000 011d</td>
<td>ffff ffff</td>
<td>06ff</td>
<td>DECF f,d</td>
<td>Decrement f</td>
<td>(f - 1) → f and d</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>0001 010d</td>
<td>ffff ffff</td>
<td>14ff</td>
<td>INCF f,d</td>
<td>Increment f</td>
<td>(f + 1) → f and d</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>1011 0111</td>
<td>kkkk kkkk</td>
<td>b3kk</td>
<td>IORLW k</td>
<td>Inclusive OR literal with W</td>
<td>(W.OR.k) → W</td>
<td>z</td>
</tr>
<tr>
<td>0000 100d</td>
<td>ffff ffff</td>
<td>08ff</td>
<td>IORWF f,d</td>
<td>Inclusive or W with f</td>
<td>(w.OR.f) → d</td>
<td>z</td>
</tr>
<tr>
<td>1011 0000</td>
<td>kkkk kkkk</td>
<td>b0kk</td>
<td>MOVWF k</td>
<td>Move literal to W</td>
<td>k → W</td>
<td>None</td>
</tr>
<tr>
<td>0010 111d</td>
<td>ffff ffff</td>
<td>2aff</td>
<td>NEG f,d</td>
<td>Negate W, store in f and d</td>
<td>(w + 1) → f, (w + 1) → d</td>
<td>ov c dc z 1,3</td>
</tr>
<tr>
<td>0001 101d</td>
<td>ffff ffff</td>
<td>1aff</td>
<td>RLCF f,d</td>
<td>Rotate left through carry</td>
<td>f&lt;n&gt;→d&lt;n+1&gt;, f&lt;7&gt;→c, C→d&lt;0&gt;</td>
<td>c</td>
</tr>
<tr>
<td>1010 001d</td>
<td>ffff ffff</td>
<td>22ff</td>
<td>RLNCF f,d</td>
<td>Rotate left (no carry)</td>
<td>f&lt;n&gt;→d&lt;n+1&gt;, f&lt;7&gt;→d&lt;0&gt;</td>
<td>None</td>
</tr>
<tr>
<td>0001 100d</td>
<td>ffff ffff</td>
<td>18ff</td>
<td>RRCF f,d</td>
<td>Rotate right through carry</td>
<td>f&lt;n&gt;→d&lt;n-1&gt;, f&lt;0&gt;→c, c→d&lt;7&gt;</td>
<td>None</td>
</tr>
<tr>
<td>0010 000d</td>
<td>ffff ffff</td>
<td>20ff</td>
<td>RRNCF f,d</td>
<td>Rotate right (no carry)</td>
<td>f&lt;n&gt;→d&lt;n-1&gt;, f&lt;0&gt;→d&lt;7&gt;</td>
<td>None</td>
</tr>
<tr>
<td>0010 101d</td>
<td>ffff ffff</td>
<td>2aff</td>
<td>SETF f,d</td>
<td>Set f and Set d</td>
<td>0xff→f, 0xff→d</td>
<td>None 3</td>
</tr>
<tr>
<td>1011 0010</td>
<td>kkkk kkkk</td>
<td>b2kk</td>
<td>SUBLW k</td>
<td>Subtract W from literal</td>
<td>(k-w) → w</td>
<td>ov c dc z</td>
</tr>
<tr>
<td>0000 010d</td>
<td>ffff ffff</td>
<td>04ff</td>
<td>SUBWF f,d</td>
<td>Subtract W from f</td>
<td>(f-w) → d</td>
<td>ov c dc z 1</td>
</tr>
<tr>
<td>0000 001d</td>
<td>ffff ffff</td>
<td>02ff</td>
<td>SUBWF f,d</td>
<td>Subtract from f with borrow</td>
<td>(f-w-c) → d</td>
<td>ov c dc z 1</td>
</tr>
<tr>
<td>0001 110d</td>
<td>ffff ffff</td>
<td>1cff</td>
<td>SWAPF f,d</td>
<td>Swap f</td>
<td>(f&lt;0:3&gt;→d&lt;4:7&gt;, f&lt;4:7&gt;→d&lt;0:3&gt;)</td>
<td>None</td>
</tr>
<tr>
<td>1011 0100</td>
<td>kkkk kkkk</td>
<td>b4kk</td>
<td>XORLW k</td>
<td>Exclusive OR literal with W</td>
<td>(W.XOR.k) → w</td>
<td>z</td>
</tr>
<tr>
<td>0000 110d</td>
<td>ffff ffff</td>
<td>0cff</td>
<td>XORWF f,d</td>
<td>Exclusive OR W with f</td>
<td>(W.XOR.f) → d</td>
<td>z</td>
</tr>
</tbody>
</table>

© 1994 Microchip Technology Inc.
### TABLE 21: PIC17C42 PROGRAM CONTROL INSTRUCTIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>111k</td>
<td>kkkk</td>
<td>CALL k</td>
<td>Subroutine call (within 8k page)</td>
<td>PC+1 (\rightarrow) TOS,</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>0011</td>
<td>0001</td>
<td>CPFSEQ f</td>
<td>Compare f/w, skip if (f = w)</td>
<td>(f-W, \text{skip if } f = W)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>0011</td>
<td>0010</td>
<td>CPFSGT f</td>
<td>Compare f/w, skip if (f &gt; w)</td>
<td>(f-W, \text{skip if } f &gt; W)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>0011</td>
<td>0000</td>
<td>CPFSLT f</td>
<td>Compare f/w, skip if (f &lt; w)</td>
<td>(f-W, \text{skip if } f &lt; W)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>0001</td>
<td>011d</td>
<td>DECFSZ f,d</td>
<td>Decrement f, skip if 0</td>
<td>((f-1) \rightarrow d, \text{skip if } 0)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>0010</td>
<td>011d</td>
<td>DCFSNZ f,d</td>
<td>Decrement f, skip if not 0</td>
<td>((f-1) \rightarrow d, \text{skip if } 0)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>110k</td>
<td>kkkk</td>
<td>GOTO k</td>
<td>Unconditional branch (within 8k)</td>
<td>(k \rightarrow \text{PC}&lt;12:0&gt;)</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>0001</td>
<td>111d</td>
<td>INCFSZ f,d</td>
<td>Increment f, skip if zero</td>
<td>((f+1) \rightarrow d, \text{skip if } 0)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>0010</td>
<td>010d</td>
<td>INFSNZ f,d</td>
<td>Increment f, skip if not zero</td>
<td>((f+1) \rightarrow d, \text{skip if } 0)</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>1011</td>
<td>0111</td>
<td>LCALL k</td>
<td>Long Call (within 64k)</td>
<td>((\text{PC}+1) \rightarrow \text{TOS})</td>
<td>None</td>
<td>5,8</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
<td>RETFIE</td>
<td>Return from interrupt, enable interrupt</td>
<td>(f3 \rightarrow \text{PC}+k \rightarrow \text{PCL} \cdot ^* \rightarrow \text{GLINTD})</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>1011</td>
<td>0110</td>
<td>RETLW k</td>
<td>Return with literal in W</td>
<td>(k \rightarrow W, \text{TOS} \rightarrow \text{PC}, (f3 \text{ unchanged}))</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
<td>RETURN</td>
<td>Return from subroutine</td>
<td>(\text{TOS} \rightarrow \text{PC}(f3 \text{ unchanged}))</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
<td>TSTFSZ f</td>
<td>Test f, skip if zero</td>
<td>(\text{skip if } f = 0)</td>
<td>None</td>
<td>7</td>
</tr>
</tbody>
</table>

### TABLE 22: PIC17C42 BIT HANDLING INSTRUCTIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1bbb</td>
<td>BCF f,b</td>
<td>Bit clear f</td>
<td>(0 \rightarrow f(b))</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>1000</td>
<td>0bbb</td>
<td>BSF f,b</td>
<td>Bit set f</td>
<td>(1 \rightarrow f(b))</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>1001</td>
<td>1bbb</td>
<td>BTFSF f,b</td>
<td>Bit test, skip if clear</td>
<td>(\text{skip if } f(b) = 0)</td>
<td>None</td>
<td>4,7</td>
</tr>
<tr>
<td>1001</td>
<td>0bbb</td>
<td>BTFSG f,b</td>
<td>Bit test, skip if set</td>
<td>(\text{skip if } f(b) = 1)</td>
<td>None</td>
<td>4,7</td>
</tr>
<tr>
<td>0011</td>
<td>1bbb</td>
<td>BTG f,b</td>
<td>Bit toggle f</td>
<td>(f(\bar{b}) \rightarrow f(b))</td>
<td>None</td>
<td>4</td>
</tr>
</tbody>
</table>
### TABLE 23: PIC17C42 SPECIAL CONTROL INSTRUCTIONS

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0100</td>
<td>0004</td>
<td>CLRWT</td>
<td>Clear watch dog timer</td>
<td>0→WDT, 0→WDT prescaler, 1→PD, 1→TO</td>
<td>PD, TO</td>
<td></td>
</tr>
<tr>
<td>0000 0000 0000 0100</td>
<td>0000</td>
<td>NOP</td>
<td>No operation</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0000 0000 0000 0011</td>
<td>0003</td>
<td>SLEEP</td>
<td>Enter sleep mode</td>
<td>Stop oscillator, power down, 0→WDT, 0→WDTPrescaler, 1→PD, 1→TO</td>
<td>PD, TO</td>
<td></td>
</tr>
</tbody>
</table>

### PIC17C42 Notes

1. 2's complement arithmetic
2. Unsigned arithmetic
3. If \( d = 1 \), only the file is affected; if \( d = 0 \), both \( w \) and the file are affected; if only \( w \) is required to be affected, then \( f = 0ah \) (File 0ah) must be defined.
4. The hex representation is not accurate. The value of the bit to be modified has to be incorporated into the third digit.
5. During an LCALL, the contents of File 03h are loaded into the MSB of the PC and kkkk kkkk is loaded into File 02h, the LSB of the PC.
6. Multiple cycle instructions for EPROM programming when table pointer selects internal EPROM. The instruction is terminated by an interrupt event. When writing to external program memory, it is a two cycle instruction.
7. Two cycle instructions when condition is true, else single cycle instruction.
8. Two cycle instruction, except for TABLRD to File 02h (Program Counter low byte) in which case it takes 3 cycles.
9. A skip means that instructions fetched during execution of current instruction are not executed. Instead, a NOP is executed.
10. Any instruction that writes to PCL (File 02h) is a two cycle instruction, except for TABLRD to File 02h, which is a three cycle instruction.
**MPASM USER'S GUIDE**

### Index

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Bit Split Format</td>
</tr>
<tr>
<td>32-Bit Hex Format</td>
</tr>
<tr>
<td><strong>A</strong></td>
</tr>
<tr>
<td>Alpha Character</td>
</tr>
<tr>
<td>Alpha Numeric</td>
</tr>
<tr>
<td>Alpha Release</td>
</tr>
<tr>
<td>ANSI 'C' Escape Sequences</td>
</tr>
<tr>
<td>Arithmetic Operators and Precedence</td>
</tr>
<tr>
<td>Assemble</td>
</tr>
<tr>
<td>Assemble to Object File</td>
</tr>
<tr>
<td>Assembler Command Line Options</td>
</tr>
<tr>
<td><strong>B</strong></td>
</tr>
<tr>
<td>Beta Release</td>
</tr>
<tr>
<td>Bulletin Board Access</td>
</tr>
<tr>
<td>Bulletin Board Usage</td>
</tr>
<tr>
<td><strong>C</strong></td>
</tr>
<tr>
<td>CBANK</td>
</tr>
<tr>
<td>CBLOCK</td>
</tr>
<tr>
<td>Command Line Interface</td>
</tr>
<tr>
<td>Comments</td>
</tr>
<tr>
<td>Compatibility Issues</td>
</tr>
<tr>
<td>CONSTANT</td>
</tr>
<tr>
<td>Control Directives</td>
</tr>
<tr>
<td>Cross Reference File</td>
</tr>
<tr>
<td>Customer Support</td>
</tr>
<tr>
<td><strong>D</strong></td>
</tr>
<tr>
<td>DATA</td>
</tr>
<tr>
<td>Data Directives</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<td>#DEFINE</td>
</tr>
<tr>
<td>Directives</td>
</tr>
<tr>
<td>Directive Summary</td>
</tr>
<tr>
<td>DOS</td>
</tr>
<tr>
<td>DW</td>
</tr>
<tr>
<td><strong>E</strong></td>
</tr>
<tr>
<td>ELSE</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>ENDC</td>
</tr>
<tr>
<td>ENDF</td>
</tr>
<tr>
<td>ENDM</td>
</tr>
<tr>
<td>ENDW</td>
</tr>
<tr>
<td>EQU</td>
</tr>
<tr>
<td>ERROR</td>
</tr>
<tr>
<td>EXITM</td>
</tr>
<tr>
<td>EXPAND</td>
</tr>
<tr>
<td>Error File</td>
</tr>
<tr>
<td>Error File Format (.ERR)</td>
</tr>
<tr>
<td>Error Messages</td>
</tr>
<tr>
<td>Expressions</td>
</tr>
<tr>
<td><strong>F</strong></td>
</tr>
<tr>
<td>FILL</td>
</tr>
<tr>
<td>Files</td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td>Hex Dump Type</td>
</tr>
<tr>
<td>High/Low</td>
</tr>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td>IF</td>
</tr>
<tr>
<td>IFDEF</td>
</tr>
<tr>
<td>IFNDEF</td>
</tr>
<tr>
<td>INCLUDE</td>
</tr>
<tr>
<td>Installation</td>
</tr>
<tr>
<td>Intel Hex Format (.HEX)</td>
</tr>
<tr>
<td>Intermediate Release</td>
</tr>
<tr>
<td><strong>L</strong></td>
</tr>
<tr>
<td>Labels</td>
</tr>
<tr>
<td>Linking</td>
</tr>
<tr>
<td>LIST</td>
</tr>
<tr>
<td>Listing</td>
</tr>
<tr>
<td>Listing Directives</td>
</tr>
<tr>
<td>Listing File</td>
</tr>
<tr>
<td>Listing File Format</td>
</tr>
<tr>
<td>LOCAL</td>
</tr>
<tr>
<td>Local Label</td>
</tr>
<tr>
<td><strong>M</strong></td>
</tr>
<tr>
<td>Macro</td>
</tr>
<tr>
<td>MACRO</td>
</tr>
<tr>
<td>Macro Directives</td>
</tr>
<tr>
<td>Macro Examples</td>
</tr>
<tr>
<td>Macro Syntax</td>
</tr>
<tr>
<td>Macro Usage</td>
</tr>
<tr>
<td>Mail</td>
</tr>
<tr>
<td>MESSG</td>
</tr>
<tr>
<td>Microchip BBS</td>
</tr>
<tr>
<td>Migration Path</td>
</tr>
<tr>
<td>Mnemonics</td>
</tr>
<tr>
<td>MPASM Default File Extensions</td>
</tr>
<tr>
<td><strong>N</strong></td>
</tr>
<tr>
<td>NOEXPAND</td>
</tr>
<tr>
<td>NOLIST</td>
</tr>
<tr>
<td>Numeric Constants and Radix</td>
</tr>
<tr>
<td><strong>O</strong></td>
</tr>
<tr>
<td>Object Code Formats</td>
</tr>
<tr>
<td>Operands</td>
</tr>
<tr>
<td>Operators</td>
</tr>
<tr>
<td>ORG</td>
</tr>
<tr>
<td><strong>P</strong></td>
</tr>
<tr>
<td>PAGE</td>
</tr>
<tr>
<td>PC</td>
</tr>
<tr>
<td>PIC16/17</td>
</tr>
<tr>
<td>PIC16C5X</td>
</tr>
<tr>
<td>PIC16C5X Instruction Set</td>
</tr>
<tr>
<td>PIC16C5X Notes</td>
</tr>
<tr>
<td>PIC16C5X Operand Codes</td>
</tr>
<tr>
<td>PIC16CXX</td>
</tr>
<tr>
<td>PIC16CXX Instruction Set</td>
</tr>
</tbody>
</table>
Index

PIC16CXX Notes 94
PIC17C42 Instruction Set 95
PIC17C42 Notes 99
PIC17C42 Operand Codes 95
PIC17CXX 85
Precedence 53, 57
PROCESSOR 41
Production Release 71

Q
Quick Reference Guide 85

R
Reader Response 103
RADIX 41
Radix Specifications 56
Recursion 48
Recursive Macros 50
Relocatable Object 6
RES 42

S
SET 42
Shell 9
Source Code 5
Source Code Formats 13
Source File 12
SPACE 43
Special Interest Groups 68
SUBTITLE 43
System Requirements 7

T
Text Strings 54
Text Substitution 49
TITLE 44

U
#UNDEFINE 44

V
VARIABLE 45

W
Warranty Registration 7
WHILE 46
READER RESPONSE

MPASM USER'S GUIDE

It is our intention to provide you with the best documentation possible to ensure successful use of your Microchip product. If you wish to provide your comments on organization, clarity, subject matter, and ways in which our documentation can better serve you, please FAX your comments to the Technical Publications Manager at (602) 786-7578.

Please list the following information, and use this outline to provide us with your comments about this Data Sheet.

To: Technical Publications Manager Total pages sent ________
RE: Reader Response
From: Name ________________________________
Company_______________________________
Address_______________________________
City / State / ZIP / Country______________
Telephone: (_____) _______ - __________
FAX: (_____) _______ - __________
Application (optional): __________________
Would you like a reply? _Y _N

Product: MPASM User's Guide Literature Number: DS33014C

Questions:
1. What are the best features of this document? ____________________________________________
2. How does this document meet your hardware and software development needs? ___________
3. Do you find the organization of this document easy to follow? If not, why? _______________
4. What additions to the document do you think would enhance the structure and subject matter? _____________
5. What deletions from the document could be made without affecting the overall usefulness? ___________
6. Is there any incorrect or misleading information (what and where)? _______________________
7. How would you improve this document? _____________________________________________
8. How would you improve our software, systems, and silicon products? ___________________