## **MPASM Assembler Overview**

An overview of MPASM assembler and its capabilities is presented.

- What is MPASM Assembler
- Assembler Migration Path
- <u>Assembler Compatibility Issues</u>
- How MPASM Assembler Helps You



#### What is MPASM Assembler

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

MPASM assembler supports all PICmicro MCU, memory, and secure data (KeeLoq)) products from Microchip Technology Inc.



Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

#### **Assembler Migration Path**

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



#### **Assembler Compatibility Issues**

MPASM assembler is compatible with all Microchip PICmicro MCU development systems currently in production. This includes MPLAB SIM (PICmicro MCU discrete-event simulator), MPLAB ICE 2000 (PICmicro MCU in-circuit emulator), MPLAB ICD 2 (incircuit debugger - PIC18 parts), PRO MATE II (device programmer) and PICSTART Plus (low-cost development programmer).

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



#### **How MPASM Assembler Helps You**

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

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



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## Getting Started with MPASM Assembler

An overview of assembler (MPASM assembler) operation and instructions for the installation of MPASM assembler on your system is shown.

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



#### **Overview of Assembler**

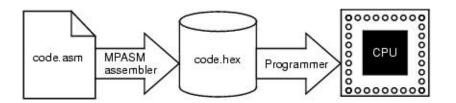
MPASM assembler can be used in two ways:

To generate absolute code that can be executed directly by a microcontroller.

To generate object code that can be linked with other separately assembled or compiled modules.

#### **Generating Absolute Code**

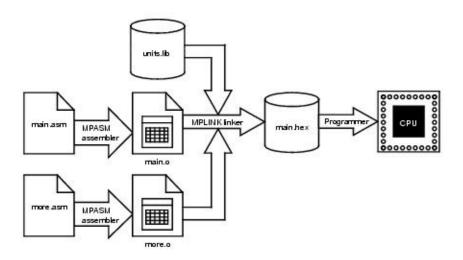
Absolute code is the default output from MPASM assembler. This process is shown below.



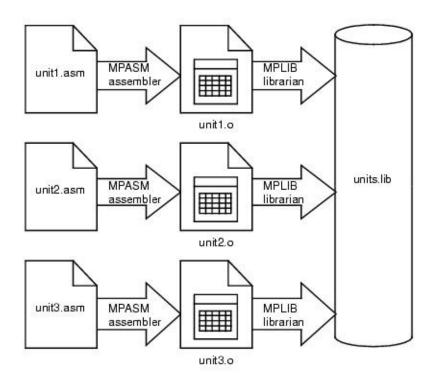
When a source file is assembled in this manner, all values used in the source file must be defined within that source file, or in files that have been explicitly included. If assembly proceeds without errors, a HEX file will be generated, containing the executable machine code for the target device.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

#### **Generating Object Code**

MPASM assembler also has the ability to generate an object module that can be linked with other modules using Microchip's MPLINK linker to form the final executable code. This method is very useful for creating reusable modules that do not have to be retested each time they are used.



Related modules can also be grouped and stored together in a library using Microchip's MPLIB librarian. Required libraries can be specified at link time, and only the routines that are needed will be included in the final executable.



Refer to Relocatable Objects for more information on the

differences between absolute and object assembly.



## **Assembler Input/Output Files**

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

#### **TABLE: INPUT FILES**

Source Code (.asm)Default source file extension input to assembler.Include File (.inc)Include (header) file

#### **TABLE: OUTPUT FILES**

Listing File (.lst)	Default output extension for listing files generated by assembler.
<u>Error File (.err)</u>	Output extension from assembler for error files.
<u>Hex File Formats (.hex, .hxl, .hxh)</u>	Output extension from assembler for hex files.
	Output extension for the symbol and debug file. This file may be output from assembler or MPLINK linker.
<u>Object File (.o)</u>	Output extension from assembler for object files.

#### Source Code (.asm)

Assembly source code is one programming language you may use to develop your application. The source code file may be created using any ASCII text file editor. It should conform to the following basic guidelines.

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

- Labels
- <u>Mnemonics</u>
- Operands
- <u>Comments</u>

The order and position of these are important. Labels must start in column one. Mnemonics may start in column two or beyond.

Operands follow the mnemonic. Comments may follow the operands, mnemonics or labels, and can start in any column. The maximum column width is 255 characters.

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

# Sample MPASM Assembler Source Code (Shows multiple operands)

; ; Sample MPASM Source Code. For illustration only. ; list p=16c54 Dest equ H'0B' org H'01FF' goto Start org H'0000' Start movlw H'0A' movwf Dest bcf Dest, 3 goto Start end

#### Labels

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

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

**Note:** Do not use labels with a leading underscore and number, e.g., \_2NDLOOP. Also, do not use the assembler reserved word Halt as a label.

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

#### **Mnemonics**

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

#### **Operands**

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

#### Comments

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

#### Include File (.inc)

Assembler include, or header, file. Usually contains device-specific register and bit assignments.

As an example, to add the standard header file for the PIC18F452 device to your assembly code, use:

```
#include <p18f452.inc>
```

## Listing File (.lst)

A listing file provides a mapping of source code to machine instructions. MPASM assembler and MPLINK linker can generate listing files.

#### Sample MPASM Assembler Listing File

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

The first column of numbers contains the base address in memory where the code will be placed. The second column displays the 32bit value of any symbols created with the SET, EQU, VARIABLE, CONSTANT, or CBLOCK directives. The third column is reserved for the machine instruction. This is the code that will be executed by the PICmicro MCU. The fourth column lists the associated source file line number for this line. The remainder of the line is reserved for the source code line that generated the machine code.

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

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

MPASM 01.99.21 Intermediate MANUAL.ASM 5-30-1997 15:31:05 PAGE 1 LOC OBJECT CODE LINE SOURCE TEXT VALUE 00001 ; 00002 ; Sample MPASM Source Code. For illustration only. 00003 ;

00004 list p=16c54 0000000B 00005 Dest equ H'0B' 00006 01FF 00007 org H'01FF' 01FF 0A00 00008 goto Start 00009 0000 00010 org H'0000' 00011 0000 0C0A 00012 Start movlw H'0A' 0001 002B 00013 movwf Dest 0002 0A00 00014 goto Start 00015 00016 end MPASM 01.99.21 Intermediate MANUAL.ASM 5-30-1997 15:31:05 PAGE 2 SYMBOL TABLE LABEL VALUE Dest 000000B Start 00000000 16C54 0000001 MEMORY USAGE MAP ('X' = Used, '-' = Unused) 0000 : XXX------01C0 : -----X All other memory blocks unused. Program Memory Words Used: 4 Program Memory Words Free: 508 Errors: 0 Warnings : 0 reported, 0 suppressed Messages : 0 reported, 0 suppressed

## **Error File (.err)**

MPASM assembler, by default, generates an error file. This file can be useful when debugging your code. The MPLAB IDE Source Level Debugger will automatically open this file in the case of an error. The format of the messages in the error file is: <type>[<number>] <file> <line> <description>

For example:

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

The error file will contain MPASM assembler errors, warnings and messages.

## Hex File Formats (.hex, .hxl, .hxh)

MPASM Assembler is capable of producing different hex file formats.

Format Name	Format Type	File Extension	Use
Intel Hex Format	INHX8M	.HEX	for standard programmers
Intel Split Hex Format	INHX8S	.HXL, .HXH	for odd/even ROM programmers
Intel Hex 32 Format	INHX32	.HEX	for 16-bit core programmers

#### **Intel Hex Format**

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

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

:ВВААААТТНННН....НННСС

where:

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

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

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

HH - is a two digit hexadecimal data byte, presented in lowbyte/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.

Example

#### Intel Split Hex Format

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

Example

<file\_name>.HXL

#### **Intel Hex 32 Format**

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

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

:BBAAAATTHHHH....HHHCC

where:

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

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

- TT is a two digit record type:
- 00 Data record

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

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

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

#### Symbol and Debug File (.cod)

A COD file is used by MPLAB IDE to debug code. The COD file name, including the path, has a 63 character limit. MPASM assembler and MPLINK linker can generate a COD file.

## **Object File (.0)**

Object files are the relocatable code produced from source files.

• MPASM assembler assembles source files into object files.

• MPLINK linker combines object files and library files, according to a linker script, into a single output file.

• MPLIB librarian combines several object files into a single library file.

< >

#### **Assembler Installation**

When MPLAB IDE is installed, the windowed version of MPASM assembler is also installed. You may obtain the MPLAB IDE software either from the latest MPLAB IDE CD-ROM or from our web site.

Actually, there are two versions of MPASM assembler:

- a Windows version, MPASMWIN.EXE (Recommended)
- a DOS version, MPASM.EXE, for DOS 5.0 or greater

#### **Available free with MPLAB IDE**

MPASMWIN.EXE has a Windows shell interface. MPASMWIN.EXE may be used with Windows 95/98/ME, Windows NT/2000 or Windows XP. You can use this version with MPLAB IDE (recommended) or stand-alone.

#### Available free with MPLAB C1X compilers

MPASM.EXE has a command line interface. MPASM.EXE may be used with DOS or a DOS window in Windows 3.x, Windows 95/98/ME, Windows NT/2000 or Windows XP. You can use it with MPLAB IDE, though MPASMWIN.EXE is recommended.



## **Assembler Usage with MPLAB IDE**

How to use MPASM assembler with MPLAB IDE v6.xx and later is discussed here. The windows version (mpasmwin.exe) of the assembler should be used with MPLAB IDE.

- MPLAB IDE Interface
- MPLAB IDE Projects
- Project and Assembler Setup



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

#### **MPLAB IDE Interface**

MPASM assembler may be used with the MPLAB IDE integrated development environment to provide GUI development of your application. In order to use MPASM assembler with MPLAB IDE, you must first install MPLAB IDE. The latest version of this free software is available at our website (http://www.microchip.com) or from any sales office (back cover). When you install MPLAB IDE, you will be installing MPASM assembler as well.

Once MPLAB IDE is installed on your PC, check the settings below to ensure that the assember is installed properly as a language tool.

1. From the MPLAB IDE menu bar, select <u>Project>Set Language</u> <u>Tool Locations</u> to open a dialog to set/check language tool executable location.

# FIGURE: MPASM ASSEMBLER EXECUTABLE LOCATION

E IAR Systems PIC18 Toolsuite Microchip ASM30 Toolsuite Microchip C17 Toolsuite Microchip C18 Toolsuite Microchip C30 Toolsuite Microchip C30 Toolsuite	icrochip ASM30 Toolsuite icrochip C17 Toolsuite icrochip C18 Toolsuite icrochip C30 Toolsuite icrochip MPASM Toolsuite Executables MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe)	IAR Systems Midran		<b>^</b>
e Microchip C17 Toolsuite Microchip C18 Toolsuite e Microchip C30 Toolsuite	icrochip C17 Toolsuite icrochip C38 Toolsuite icrochip C30 Toolsuite icrochip MPASM Toolsuite Executables MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe)			
e Microchip C18 Toolsuite e Microchip C30 Toolsuite	icrochip C18 Toolsuite icrochip C30 Toolsuite icrochip MPASM Toolsuite Executables MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe)			
∋ Microchip C30 Toolsuite	icrochip C30 Toolsuite icrochip MPASM Toolsuite Executables MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe)			
	icrochip MPASM Toolsuite = Executables MPASM Assembler (mpasmwin.exe) 			
- MICTOCHID MEASM LOOISUILE	- Executables MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe)			
	-	<mark>MPASM Ass</mark> <b>MPLINK Ob</b>	ject Linker (mplink.exe)	-
peation			BIDE\MCHIP Tools\mpasmwin exe	Browse
cation			B IDE\MCHIP_Tools\mpasmwin.exe	Browse.

- 1. In the dialog, under Registered Tools, select "Microchip MPASM Toolsuite". Click the "+" to expand.
- Select Executables. Click the "+" to expand.

• Select MPASM Assembler (mpasmwin.exe). Under Location, a path to the mpasmwin.exe file should be displayed. If no path is displayed, enter one or browse to the location of this file. By default, it is located at:

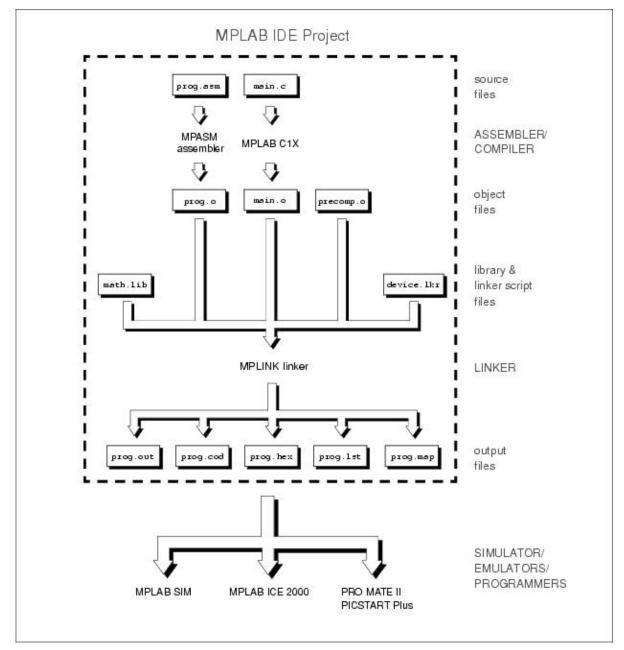
C:\Program Files\MPLAB IDE\MCHIP\_Tools\mpasmwin.exe

• Click OK.



#### **MPLAB IDE Projects**

A project in MPLAB IDE is a group of files needed to build an application, along with their associations to various build tools. Below a generic MPLAB IDE Project using the MPASM assembler tool is shown.



#### FIGURE: PROJECT RELATIONSHIPS

In this MPLAB IDE Project, an assembly source file (prog.asm) is shown with its associated assembler (MPASM assembler). MPLAB IDE will use this information to generate the object file prog.o for input into MPLINK linker.

The C source file main. c is also shown with its associated MPLAB C1X compiler. MPLAB IDE will use this information to generate an object file (main.0) for input into the linker (MPLINK linker). See either the MPLAB C17 Compiler User's Guide (DS51290) for PIC17CXXX devices or the MPLAB C18 Compiler User's Guide (DS51288) for PIC18XXXXX devices for more information on using these compilers.

In addition, precompiled object files (precomp.o) may be included in a project, with no associated tool required. Types of precompiled object files that are generally required in a project are:

- Start up code
- Initialization code
- Interrupt service routines
- Register definitions

Precompiled object files are often device and/or memory model dependent. For more information on available Microchip precompiled object files, see either the *MPLAB C17 Compiler Libraries* (DS51296) for PIC17CXXX devices or the *MPLAB C18 Compiler Libraries* (DS51297) for PIC18XXXXX devices.

Some library files (math.lib) are available with the compiler. Others may be built outside the project using the librarian tool (MPLIB librarian). See the MPLIB Object Librarian section later in this manual for more information on using the librarian. For more information on available Microchip libraries, see the MPLAB C1X library documents previously mentioned. The object files, along with library files and a linker script file (device.lkr) are used to generate the project output files via the linker (MPLINK linker). See the MPLINK Object Linker section later in this manual for more information on linker script files and using the linker.

The main output file generated by MPLINK linker is the **Hex file** (prog.hex), used by simulators (MPLAB SIM), emulators (MPLAB ICE 2000) and programmers (PRO MATE II and PICSTART Plus). The other output files are:

- **COFF file (.out).** Intermediate file used by MPLINK linker to generate Code file, Hex file, and Listing file.
- Code file (.cod). Debug file used by MPLAB IDE.
- Listing file (.lst). Original source code, side-by-side with final binary code.
- **Map file (.map).** Shows the memory layout after linking. Indicates used and unused memory regions.

< >

#### **Project and Assembler Setup**

To set up an MPLAB IDE project for the first time, it is advisable to use the built-in Project Wizard (*Project>Project Wizard*.) In this wizard, you will be able to select a language toolsuite that uses MPASM assembler, e.g., the Microchip MPASM Toolsuite. For more on the wizard, and MPLAB IDE projects, see MPLAB IDE documentation.

Once you have a project set up, you may then set up properties of MPASM assembler in MPLAB IDE.

 From the MPLAB IDE menu bar, select <u>Project>Build</u> <u>Options>Project</u> to open a dialog to set/check project build options.

**Note:** MPASM assembler does not recognize include path information specified in MPLAB IDE.

• Click on the MPASM Assembler tab and enter/change assembler settings.



# Assembler Usage without MPLAB IDE

How to use MPASM assembler without MPLAB IDE is discussed here.

The command-line version (**mpasm.exe**) may be run from the command line or a command shell. The windows version (**mpasmwin.exe**) may be from the command line or a Windows shell.

- Command Line Interface
- Command Shell Interface
- Windows Shell Interface
- Troubleshooting



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **Command Line Interface**

MPASM assembler can be invoked through the command line interface (command prompt) as follows:

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

or

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

where

/<0ption> - refers to one of the command line options

<filename> - is the file being assembled

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

mpasmwin /e /l test

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

Option	Result
/ <option></option>	Enables the option
/ <option>+</option>	Enables the option
/ <option>-</option>	Disables the option
	If appropriate, enables the option and directs the output to the specified file

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

• mpasmwin - a Windows interface is displayed, which includes a Help button

mpasm - the assembler help panel is displayed (same as mpasm /?)

Option	Default	Description
?	N/A	Displays the assembler help panel (mpasm.exe).
а	INHX8M	Generate absolute .cod and .hex output directly from assembler; /a <hex-format>, where <hex-format> is one of [INHX8M   INHX8S   INHX32].</hex-format></hex-format>
С	On	Enables/Disables case sensitivity.
d	N/A	Define a text string substitution; /d <label>[=<value>].</value></label>
e	On	Enable/Disable/Set Path for error file. /e Enable /e- Disable /e <path>error.file Enables/sets path</path>
h	N/A	Displays the assembler help panel.
l	On	Enable/Disable/Set Path for list file /l Enable /l+ Enable /l - Disable /l <path>list.file Enables/sets path</path>
m	On	Enable/Disable macro expansion.
0	Off	Enable/Disable/Set Path for object file. /o Enable /o+ Enable /o- Disable /o <path>object.file Enables/sets path</path>
р	None	Set the processor type; /p <processor_type>, where <processor_type> is a PICmicro MCU device, e.g., PIC16C54.</processor_type></processor_type>
q	Off	Enable/Disable quiet mode (suppress screen output).
r	Hex	Defines default radix; /r <radix>, where <radix> is one of [ HEX   DEC   OCT ].</radix></radix>
t	8	List file tab size; /t <size>.</size>
w	0	Set message level; /w <value>, where <value> is one of [0 1 2]. 0 all messages 1 errors and warnings 2 errors only</value></value>
x	Off	Enable/Disable/Set Path for cross reference file. /x Enable /x+ Enable /x- Disable /x <path>xref.file Enables/sets path</path>
У	Disabled	Enable/Disable extended instruction set. /y Enable /y+ Enable /y- Disable Can only be enabled for processors which support the extended instruction set and





Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **Command Shell Interface**

The MPASM assembler command shell interface displays a screen in Text Graphics mode. It is invoked by executing mpasm.exe in Windows Explorer.

On this screen, you can fill in the name of the source file you want to assemble and other information.

#### FIGURE: TEXT GRAPHICS MODE DISPLAY

Source		(c)1993-2003 Microchip Technology Inc./Byte Craft Li			
Jource	File		*.ASM		
Processor	Т уре		None		
Error	File		Yes		
oss Reference	File		No		
Listing	File		Yes		
Hex Dump	Т уре		I NHX8M	.HEX	
ble to Object	File		No		
	Error oss Reference Listing Hex Dump	Error File oss Reference File Listing File Hex Dump Type	Error File : oss Reference File : Listing File : Hex Dump Type :	Processor Type : None Error File : Yes oss Reference File : No Listing File : Yes Hex Dump Type : INHX8M ble to Object File : No	

#### **Source File**

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

#### **Processor Type**

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

#### **Error File**

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

#### **Cross Reference File**

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

#### **Listing File**

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

### **HEX Dump Type**

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

#### **Assemble to Object File**

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



### **Windows Shell Interface**

MPASM assembler for Windows provides a graphical interface for setting assembler options. It is invoked by executing mpasmwin.exe in Windows Explorer.

#### FIGURE: MPASM ASSEMBLER WINDOWS SHELL INTERFACE

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

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

Option	Usage			
Radix	Override any source file radix settings.			
Warning Level	Override any source file message level settings.			
Hex Output	Override any source file hex file format settings.			
Generated Files	Enable/disable various output files.			
Case Sensitivity	Enable/disable case sensitivity.			
Macro Expansion	Override any source file macro expansion settings.			
Processor	Override any source file processor settings.			
Tab Size	Set the list file tab size.			
Extra Options	Any additional command line options. See <u>Command Line Interface</u> for more details.			
Save Settings on Exit	Save these settings in mplab.ini. They will be used the next time you run mpasmwin.exe.			
Extended Mode	Enable extended mode.			



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## Troubleshooting

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

#### FIGURE: PROPERTIES DIALOG - MPASM.EXE

General Program Font Memory Conventional memory	Summary	Compatibility	Misc	Screen	
Conventional memory	Memory	Font	Program	General	
Convertional memory   Total:   Auto   Protected   256   512   Expanded (EMS) memory   Total:   None   1280   1536   1792   Extended (XMS) memory   Total:   0   2560   2816   3072   3284   3824   3840   Total:   Auto		environment: Auto 256 512 768 1024 1280 1536 1792 2048 2304 2560 2816 3072 3328 3328 3840	IS) memory S) memory	Total: Auto Pro Expanded (EM Total: None Extended (XM Total: 0 V Use MS-DOS prote	

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

<	>
and the second se	and the second second second

# **Directives**

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 here.

**Note:** Although MPASM assembler is often used with MPLINK object linker, MPASM assembler directives are not supported by MPLINK linker. See MPLINK object linker documentation for more information on linker options to control listing and hex file output.

Directives discussed are:

**Note:** Directives are not case-sensitive, e.g., cblock may be executed as CBLOCK, cblock, Cblock, etc

- \_\_BADRAM Identify Unimplemented RAM
- <u>BADROM Identify Unimplemented ROM</u>
- \_\_\_CONFIG Set Processor Configuration Bits
- \_\_IDLOCS Set Processor ID Locations
- \_\_\_MAXRAM Define Maximum RAM Location
- \_\_\_MAXROM Define Maximum ROM Location
- #DEFINE Define a Text Substitution Label
- #INCLUDE Include Additional Source File

- <u>#UNDEFINE Delete a Substitution Label</u>
- BANKISEL Generate Indirect Bank Selecting Code
- BANKSEL Generate Bank Selecting Code
- <u>CBLOCK Define a Block of Constants</u>
- <u>CODE Begin an Object File Code Section</u>
- <u>CODE\_PACK Begin an Object File Packed Code Section</u>
- <u>CONSTANT Declare Symbol Constant</u>
- DA Store Strings in Program Memory
- Data Create Numeric and Text Data
- DB Declare Data of One Byte
- DE Declare EEPROM Data Byte
- <u>DT Define Table</u>
- DW Declare Data of One Word
- ELSE Begin Alternative Assembly Block to IF
- END End Program Block
- ENDC End an Automatic Constant Block
- ENDIF End Conditional Assembly Block
- ENDM End a Macro Definition
- ENDW End a While Loop
- EQU Define an Assembler Constant
- ERROR Issue an Error Message
- ERRORLEVEL Set Message Level
- EXITM Exit from a Macro
- EXPAND Expand Macro Listing
- EXTERN Declare an Externally Defined Label
- FILL Specify Memory Fill Value
- GLOBAL Export a Label
- IDATA Begin an Object File Initialized Data Section

- IF Begin Conditionally Assembled Code Block
- IFDEF Execute If Symbol has Been Defined
- IFNDEF Execute If Symbol has not Been Defined
- LIST Listing Options
- LOCAL Declare Local Macro Variable
- MACRO Declare Macro Definition
- MESSG Create User Defined Message
- NOEXPAND Turn off Macro Expansion
- NOLIST Turn off Listing Output
- ORG Set Program Origin
- PAGE Insert Listing Page Eject
- PAGESEL Generate Page Selecting Code
- PROCESSOR Set Processor Type
- RADIX Specify Default Radix
- <u>RES Reserve Memory</u>
- SET Define an Assembler Variable
- SPACE Insert Blank Listing Lines
- SUBTITLE Specify Program Subtitle
- TITLE Specify Program Title
- UDATA Begin an Object File Uninitialized Data Section
- UDATA\_ACS Begin an Object File Access Uninitialized Data
   Section
- <u>UDATA\_OVR Begin an Object File Overlayed Uninitialized Data</u>
   <u>Section</u>
- <u>UDATA\_SHR Begin an Object File Shared Uninitialized Data</u>
   <u>Section</u>
- VARIABLE Declare Symbol Variable
- WHILE Perform Loop While Condition is True



# **\_BADRAM - Identify Unimplemented RAM**

**Note:** badram is preceded by two underline characters, with no space in between these characters. A space is added here for readibility only.

#### **Syntax**

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

#### Description

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

#### **Example**

See the example for \_ \_maxram.

#### See Also

\_\_<u>MAXRAM</u>



# **\_BADROM - Identify Unimplemented ROM**

**Note:** badrom is preceded by two underline characters, with no space in between these characters. A space is added here for readibility only.

#### **Syntax**

\_\_badrom <expr>[-<expr>][, <expr>[-<expr>]]

#### Description

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

Specifically, a warning will be raised in the following circumstances:

- the target of a G0T0 or CALL instruction is evaluated by the assembler to a constant, and falls in a bad ROM region
- the target of an LGOTO or LCALL psuedo-op is evaluated by the assembler to a constant, and falls in a bad ROM region

• a . hex file is being generated, and part of an instruction falls in a bad ROM region

# Example

See the example for \_ \_maxrom.

#### See Also





Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### \_\_CONFIG - Set Processor Configuration Bits

**Note: config** is preceded by two underline characters, with no space in between these characters. A space is added here for readability only.

### **Syntax**

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

# Description

Sets the processor's configuration bits to the value described by <expr>. For PIC18CXXX devices, the address of a valid configuration byte must also be specified by <addr>. Refer to individual PICmicro microcontroller data sheets for a description of the configuration bits.

Before this directive is used, the processor must be declared through the command line, the list directive, or the processor directive. If this directive is used with the PIC17CXXX family, the Hex file output format must be set to INHX32 through the command line or the list directive.

### Examples

#### example 1:

list p=17c42,f=INHX32 ;define processor and select output format \_\_\_\_config H'FFFF' ;default configuration bits

#### example 2:

list p=16f877a ;list directive to define processor

#include <p16f877a.inc> ;include file with config bit definitions
\_\_\_config \_HS\_OSC & \_WDT\_OFF & \_LVP\_OFF ;Set oscillator to HS,
;watchdog time off,
;low-voltage prog. off

#### example 3:

list p=18f8720 ;list directive to define processor #include <p18f8720.inc> ;include file with config bit definitions \_\_CONFIG \_CONFIG1H, \_OSCS\_OFF\_1H & \_HS\_OSC\_1H ;Set osc info \_\_CONFIG \_CONFIG2L, \_BOR\_OFF\_2L & \_PWRT\_OFF\_2L ;Set power info

#### See Also

\_\_\_\_IDLOCS LIST PROCESSOR



### **\_IDLOCS - Set Processor ID Locations**

**Note:** idlocs is preceded by two underline characters, with no space in between these characters. A space is added here for readibility only.

#### **Syntax**

\_\_idlocs <expr> or \_\_idlocs <expr1>, <expr2>

#### Description

For PIC12CXXX and PIC16CXXX devices, \_\_\_idlocs sets the four ID locations to the hexadecimal value of <expr>. For PIC18CXXX devices, \_\_\_idlocs sets the two-byte device ID <expr1> to the hexadecimal value of <expr2>. This directive is not valid for the PIC17CXXX family.

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

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

#### Example

\_\_idlocs H'1234'

#### See Also

<u>CONFIG LIST PROCESSOR</u>



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### \_\_MAXRAM - Define Maximum RAM Location

**Note:** maxram is preceded by two underline characters, with no space in between these characters. A space is added here for readibility only.

#### **Syntax**

\_\_maxram <expr>

#### Description

The \_\_\_\_maxram and \_\_\_\_badram directives together flag accesses to unimplemented registers. \_\_\_\_maxram defines the absolute maximum valid RAM address and initializes the map of valid RAM addresses to all addresses valid at and below <expr>. <expr> must be greater than or equal to the maximum page 0 RAM address and less than 1000H. This directive is designed for use with the \_\_\_\_badram directive. Once the \_\_\_\_maxram directive is used, strict RAM address checking is enabled, using the RAM map specified by \_\_\_\_badram.

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

#### Example

list p=16c622 \_\_maxram H'0BF' \_\_badram H'07'-H'09', H'0D'-H'1E' \_\_badram H'87'-H'89', H'8D', H'8F'-H'9E' movwf H'07' ; Generates invalid RAM warning movwf H'87' ; Generates invalid RAM warning ; and truncation message

#### See Also

\_\_BADRAM



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

#### \_\_MAXROM - Define Maximum ROM Location

**Note:** maxrom is preceded by two underline characters, with no space in between these characters. A space is added here for readibility only.

## **Syntax**

\_\_maxrom <expr>

# Description

The \_ \_maxrom and \_ \_badrom directives together flag accesses to unimplemented registers. \_ \_maxrom defines the absolute maximum valid ROM address and initializes the map of valid ROM addresses to all addresses valid at and below <expr>. <expr> must be greater than or equal to the maximum ROM address of the target device. This directive is designed for use with the \_ \_badrom directive. Once the

\_\_maxrom directive is used, strict ROM address checking is enabled, using the ROM map specified by \_\_badrom.

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

### Example

list p=12c508 \_\_maxrom 0x1FF \_\_badrom 0x2 - 0x4, 0xA org 0x5 goto 0x2 ; generates a warning call 0x3 ; generates a warning org 0xA movlw 5; generates a warning

#### **See Also**

BADROM



#### **#DEFINE - Define a Text Substitution Label**

#### **Syntax**

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

# Description

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

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

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

#### 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

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



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **#INCLUDE - Include Additional Source File**

#### **Syntax**

#include <<include\_file>>
#include "<include\_file>"

## Description

The specified file is read in as source code. The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement. Upon end-of-file, source code assembly will resume from the original source file. Up to 5 levels of nesting are permitted. Up to 255 include files are allowed.

<include\_file> may be enclosed in quotes or angle brackets. If a fully qualified path is specified, only that path will be searched. Otherwise, the search order is: current working directory, source file directory, MPASM assembler executable directory.

#### Example

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

#### See Also

#### <u>#DEFINE</u> <u>#UNDEFINE</u>



#### **#UNDEFINE - Delete a Substitution Label**

#### **Syntax**

#undefine <label>

# Description

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

#### Example

#define length 20
:
#undefine length

#### See Also

**#DEFINE #INCLUDE IFDEF IFNDEF** 



#### BANKISEL - Generate Indirect Bank Selecting Code

#### **Syntax**

bankisel <label>

# Description

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

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

For more information, refer to <u>Relocatable Objects</u>.

# Example

movlw Var1 movwf FSR bankisel Var1 : movwf INDF

#### **See Also**

#### **BANKSEL PAGESEL**



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

\_

#### **BANKSEL - Generate Bank Selecting Code**

#### **Syntax**

banksel <label>

#### Description

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

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

For more information, refer to <u>Relocatable Objects</u>.

#### Example

banksel Var1 movwf Var1

#### See Also

BANKISEL PAGESEL



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

### **CBLOCK - Define a Block of Constants**

#### **Syntax**

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

## Description

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

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

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

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

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

### Example

cblock 0x20 ; name\_1 will be assigned 20 name\_1, name\_2 ; name\_2, 21 and so on

name\_3, name\_4 ; name\_4 is assigned 23. endc cblock 0x30 TwoByteVar: 0, TwoByteHigh, TwoByteLow Queue: QUEUE\_SIZE QueueHead, QueueTail Double1:2, Double2:2 endc

#### See Also

**ENDC** 



Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

### **CODE - Begin an Object File Code Section**

#### **Syntax**

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

# Description

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

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

For more information, refer to <u>Relocatable Objects</u>.

### Example

RESET code H'01FF' goto START

#### See Also

EXTERN CODE\_PACK GLOBAL IDATA UDATA\_ACS UDATA\_OVR UDATA\_SHR



#### CODE\_PACK - Begin an Object File Packed Code Section

#### **Syntax**

[<label>] code\_pack [<ROM address>]

# Description

For use when generating an object file. Declares the beginning of a section of program code or ROM data where a padding byte of zero is not appended to an odd number of bytes. If <label> is not specified, the section is named . Code. The starting address is initialized to <ROM address> or will be assigned at link time if no address is specified. If <ROM address> is specified, it must be word-aligned.

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

For more information, refer to <u>Relocatable Objects</u>.

Note: This directive is only available for the PIC18 family of devices.

#### Example

00001 LIST P=18Cxx 00002 00003 packed code\_pack H'1F0' 0001F0 01 02 03 00004 DB 1, 2, 3 0001F3 04 05 00005 DB 4, 5 00006 00007 padded code 000000 0201 0003 00008 DB 1, 2, 3 000004 0504 00009 DB 4, 5 00010 00011 END

#### See Also

EXTERN CODE GLOBAL IDATA UDATA UDATA\_ACS UDATA\_OVR UDATA\_SHR



# **CONSTANT - Declare Symbol Constant**

# **Syntax**

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

# Description

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

### Example

```
variable RecLength=64 ; Set Default
; RecLength
constant BufLength=512 ; Init BufLength
. ; RecLength may
```

.; be reset later

```
.; in RecLength=128
```

.;

constant MaxMem=RecLength+BufLength ;CalcMaxMem

### See Also

SET VARIABLE

Microchip Technology Inc.



<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

### **DA - Store Strings in Program Memory**

#### **Syntax**

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

# Description

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

#### Examples

• da "abcdef"

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

• da "12345678" ,0

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

• da 0xFFFF

will put 0x3FFF into program memory



#### **Data - Create Numeric and Text Data**

#### **Syntax**

[<label>] data <expr>,[,<expr>,...,<expr>]
[<label>] data "<text\_string>"[,"<text\_string>",...]

### Description

Initialize one or more words of program memory with data. The data may be in the form of constants, relocatable or external labels, or expressions of any of the above. The data may also consist of ASCII character strings, <text\_string>, enclosed in single quotes for one character or double quotes for strings. Single character items are placed into the low byte of the word, while strings are packed two to a word. If an odd number of characters are given in a string, the final byte is zero. On all families except the PIC18CXXX, the first character is in the most significant byte of the word. On the PIC18CXXX, the first character is in the least significant byte of the word.

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

#### 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**

#### <u>DB DE DT DW IDATA</u>



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

\_

#### **DB - Declare Data of One Byte**

#### **Syntax**

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

# Description

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

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

#### Example

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

#### See Also

DATA DE DT DW IDATA



#### **DE - Declare EEPROM Data Byte**

#### **Syntax**

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

#### Description

Although designed for initializing EEPROM data on the PIC16F8X, the directive can be used at any location for any processor.

#### PIC18XXXX

Reserve memory word bytes are packed.

When using **de**, make sure to specify the start of data memory at 0xF00000 for use with programmers.

#### **Other PICmicro's**

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

When using **de**, make sure to specify the start of data memory at 0x2100 for use with programmers.

#### Example

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



#### DATA DB DT DW



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

-

#### **DT - Define Table**

### **Syntax**

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

# Description

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

#### Example

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

#### See Also

DATA DB DE DW



#### **DW - Declare Data of One Word**

#### **Syntax**

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

# Description

Reserve program memory words for data, initializing that space to specific values. For PIC18CXXX devices, dw functions like db. Values are stored into successive memory locations and the location counter is incremented by one. Expressions may be literal strings and are stored as described in the data directive.

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

#### Example

dw 39, "diagnostic 39", (d\_list\*2+d\_offset) dw diagbase-1

#### See Also

DATA DB IDATA



# ELSE - Begin Alternative Assembly Block to IF

#### **Syntax**

else

#### 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

ENDIF IF

< >

#### **END - End Program Block**

### **Syntax**

end

#### Description

Indicates the end of the program.

#### Example

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

#### See Also

#### <u>ORG</u>



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

#### **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

ELSE IF



#### **ENDM - End a Macro Definition**

#### **Syntax**

endm

### **Description**

Terminates a macro definition begun with macro.

#### Example

make\_table macro arg1, arg2 dw arg1, 0; null terminate table name res arg2; reserve storage endm

#### See Also

MACRO EXITM



#### Microchip Technology Inc. Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210

Microchip's E-mail Address

#### **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

<u>WHILE</u>



#### **EQU - Define an Assembler Constant**

#### **Syntax**

<label> equ <expr>

#### Description

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

#### Example

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

#### See Also

<u>SET</u>



#### **ERROR - Issue an Error Message**

#### **Syntax**

error "<text\_string>"

# Description

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

#### Example

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

### See Also

MESSG



#### **ERRORLEVEL - Set Message Level**

### Syntax

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

# Description

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

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

Error messages cannot be disabled. The setting of 0, 1, or 2 overrides individual message disabling or enabling.

### Example

errorlevel 1, -202

#### See Also

LIST ERROR



#### **EXITM - Exit from a Macro**

#### **Syntax**

exitm

#### Description

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

#### Example

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

#### See Also

#### ENDM MACRO



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **EXPAND - Expand Macro Listing**

#### **Syntax**

expand

#### Description

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

#### See Also

MACRO NOEXPAND



#### EXTERN - Declare an Externally Defined Label

#### **Syntax**

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

# Description

For use when generating an object file. Declares symbol names that may be used in the current module but are defined as global in a different module.

The extern statement must be included before the <label> is used. At least one label must be specified on the line. If <label> is defined in the current module, MPASM assembler will generate a duplicate label error.

For more information, refer to <u>Relocatable Objects</u>.

#### Example

extern Function : call Function

### See Also

GLOBAL IDATA UDATA UDATA\_ACS UDATA\_OVR UDATA\_SHR



Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **FILL - Specify Memory Fill Value**

#### **Syntax**

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

#### Description

Generates <count> occurrences of the program word or byte (PIC18CXXX devices), <expr>. If bounded by parentheses, <expr> can be an assembler instruction.

#### **Examples**

#### example 1

fill 0x1009, 5 ; fill with a constant fill (GOTO RESET\_VECTOR), NEXT\_BLOCK-\$

#### example 2

list p=18f252 org 0x12 foo goto \$ org 0x100 fill(goto foo), (h'8000'-\$)/2 ;Divide by 2 for 2-word ;instructions end

#### See Also

DATA DW ORG



### **GLOBAL - Export a Label**

#### **Syntax**

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

# Description

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

For more information, refer to Relocatable Objects.

#### Example

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

#### See Also

EXTERN IDATA UDATA UDATA ACS UDATA\_OVR UDATA\_SHR



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Fax: (480) 899-9210 Microchip's E-mail Address

# IDATA - Begin an Object File Initialized Data Section

#### **Syntax**

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

# Description

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

The linker will generate a look-up table entry for each byte specified in an idata section. You must then link or include the appropriate initialization code. Examples of initialization code that may be used and modified as needed may be found with MPLINK linker sample application examples.

```
Note: This directive is not available for 12-bit core devices.
```

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

For more information, refer to <u>Relocatable Objects</u>.

#### Example

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

#### See Also

EXTERN GLOBAL UDATA UDATA\_ACS UDATA\_OVR UDATA\_SHR



#### IF - Begin Conditionally Assembled Code Block

#### **Syntax**

if <expr>

#### Description

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

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

if's may be nested up to 16 deep.

#### Example

if version == 100; check current version movlw 0x0a movwf io\_1 else movlw 0x01a movwf io\_2 endif



#### ELSE ENDIF



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

-

#### **IFDEF - Execute If Symbol has Been Defined**

#### **Syntax**

ifdef <label>

### Description

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

#### Example

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

#### See Also

<u>#DEFINE</u> <u>#UNDEFINE</u> <u>ELSE</u> <u>ENDIF</u> <u>IFNDEF</u>



# IFNDEF - Execute If Symbol has not Been Defined

#### **Syntax**

ifndef <label>

# Description

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

### Example

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

#### See Also

<u>#DEFINE</u> <u>#UNDEFINE</u> <u>ELSE</u> <u>ENDIF</u> <u>IFDEF</u>



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Fax: (480) 899-9210 Microchip's E-mail Address

### **LIST - Listing Options**

#### **Syntax**

list [<list\_option>, ..., <list\_option>]

#### Description

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

# **List Directive Options**

Option	Default	Description
b=nnn	8	Set tab spaces.
c=nnn	132	Set column width.
f= <format></format>	INHX8M	Set the hex file output. <format> can be INHX32, INHX8M, or INHX8S.</format>
free	FIXED	Use free-format parser. Provided for backward compatibility.
fixed	FIXED	Use fixed-format parser.
mm= {ON OFF}	On	Print memory map in list file.
n=nnn	60	Set lines per page.
p= <type></type>	None	Set processor type; for example, PIC16C54.
pe= <type></type>	None	Set processor type and enable extended instruction set, for example; LIST pe=PIC18F4620 Only valid with processors which support the extended instruction set and the generic processor PIC18CXX. Is overridden by command-line option /y-(disable extended instruction set).
r= <radix></radix>	hex	Set default radix: hex, dec, oct.
st= {ON OFF}	On	Print symbol table in list file.
t= {ON OFF}	Off	Truncate lines of listing (otherwise wrap).

w= {0 1 2}	0	Set the message level. See ERRORLEVEL.
x= {0N 0FF}	On	Turn macro expansion on or off.

Note: All list options are evaluated as decimal numbers.

### Example

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

#### See Also

ERRORLEVEL EXPAND NOEXPAND NOLIST PROCESSOR RADIX



#### **LOCAL - Declare Local Macro Variable**

#### **Syntax**

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

# Description

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

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

#### Example

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

#### See Also

#### ENDM MACRO



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **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.

Arguments are read in from the source line, stored in a linked list and then counted. The maximum number of arguments would be the number of arguments that would fit on the source line, after the label and macro terms. The maximum source line length is 200.

A macro can call another macro, or may call itself recursively. The maximum number of nested macro calls is 16.

Please refer to 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 call sys\_21 ; read file call endm



#### ENDM EXITM LOCAL



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

\_

#### **MESSG - Create User Defined Message**

#### **Syntax**

messg "<message\_text>"

#### Description

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

#### Example

mssg\_macro macro messg "mssg\_macro-001 invoked without argument" endm

#### See Also

**ERROR** 



#### **NOEXPAND - Turn off Macro Expansion**

#### **Syntax**

noexpand

# Description

Turns off macro expansion in the listing file.

#### See Also

#### **EXPAND**



#### **NOLIST - Turn off Listing Output**

#### **Syntax**

nolist

#### Description

Turn off listing file output.

#### See Also

#### <u>LIST</u>



#### Microchip Technology Inc. Microchip's Web Site

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

## **ORG - Set Program Origin**

#### **Syntax**

[<label>] org <expr>

#### Description

Set the program origin for subsequent code at the address defined in  $\langle expr \rangle$ . If  $\langle label \rangle$  is specified, it will be given the value of the  $\langle expr \rangle$ . If no org is specified, code generation will begin at address zero.

For PIC18CXXX devices, only even <expr> values are allowed.

When generating an object file, the org directive is interpreted as introducing an absolute CODE section with an internally generated name. For example:

L1: org 0x200

is interpreted as:

.scnname CODE 0x200 L1:

where **. SCNNAME** is generated by the assembler, and will be distinct from every name previously generated in this context.

#### Example

int\_1 org 0x20
; Vector 20 code goes here
int\_2 org int\_1+0x10

; Vector 30 code goes here

#### See Also

FILL RES END



## **PAGE - Insert Listing Page Eject**

#### **Syntax**

page

#### Description

Inserts a page eject into the listing file.

#### See Also

LIST SUBTITLE TITLE



#### **PAGESEL - Generate Page Selecting Code**

#### **Syntax**

pagesel <label>

#### Description

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

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

For PIC18CXXX devices, this command will do nothing.

For more information, refer to <u>Relocatable Objects</u>.

#### Example

pagesel GotoDest goto GotoDest : pagesel CallDest call CallDest

#### **See Also**

#### **BANKISEL BANKSEL**



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

\_

#### **PROCESSOR - Set Processor Type**

#### **Syntax**

processor <processor\_type>

#### Description

Sets the processor type to <processor\_type>.

#### Example

processor 16C54

#### See Also

<u>LIST</u>



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210

Fax: (480) 899-9210 Microchip's E-mail Address

## **RADIX - Specify Default Radix**

#### **Syntax**

radix <default\_radix>

## Description

Sets the default radix for data expressions. The default radix is hex. Valid radix values are:

- hex hexadecimal (base 16)
- dec decimal (base 10)
- oct octal (base 8)

You may also specify a radix using the list directive. For specifying the radix of constants, see <u>Numeric Constants and</u> <u>Radix</u>.

#### Example

radix dec

#### See Also

<u>LIST</u>



#### **RES - Reserve Memory**

#### **Syntax**

[<label>] res <mem\_units>

## Description

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

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

#### Example

buffer res 64 ; reserve 64 address locations of storage

#### See Also

FILL ORG



#### **SET - Define an Assembler Variable**

#### **Syntax**

<label> set <expr>

## Description

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

#### Example

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

#### See Also

EQU VARIABLE



#### **SPACE - Insert Blank Listing Lines**

#### **Syntax**

space <expr>

#### Description

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

#### Example

space 3 ;Inserts three blank lines

#### See Also

<u>LIST</u>



#### **SUBTITLE - Specify Program Subtitle**

#### **Syntax**

subtitle "<sub\_text>"

## Description

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

#### Example

subtitle "diagnostic section"

#### See Also

#### LIST TITLE



## **TITLE - Specify Program Title**

#### **Syntax**

title "<title\_text>"

## Description

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

#### Example

title "operational code, rev 5.0"

#### See Also

LIST SUBTITLE



#### UDATA - Begin an Object File Uninitialized Data Section

#### **Syntax**

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

# Description

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

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

For more information, refer to <u>Relocatable Objects</u>.

#### Example

udata Var1 res 1 Double res 2

## See Also

EXTERN GLOBAL IDATA UDATA\_ACS UDATA\_OVR UDATA\_SHR



<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### UDATA\_ACS - Begin an Object File Access Uninitialized Data Section

#### **Syntax**

[<label>] udata\_acs [<RAM address>]

## Description

For use when generating an object file. Declares the beginning of a section of access uninitialized data. If <label> is not specified, the section is named .udata\_acs. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. This directive is used to declare variables that are allocated in access RAM of PIC18CXXX devices. No code can be generated in this segment. The res directive should be used to reserve space for data.

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

For more information, refer to <u>Relocatable Objects</u>.

#### Example

udata\_acs Var1 res 1 Double res 2

#### See Also

EXTERN GLOBAL IDATA UDATA UDATA OVR UDATA SHR



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### UDATA\_OVR - Begin an Object File Overlayed Uninitialized Data Section

#### **Syntax**

[<label>] udata\_ovr [<RAM address>]

# Description

For use when generating an object file. Declares the beginning of a section of overlayed uninitialized data. If <label> is not specified, the section is named .udata\_ovr. The starting address is initialized to the specified address or will be assigned at link time if no address is specified. The space declared by this section is overlayed by all other udata\_ovr sections of the same name. It is an ideal way of declaring temporary variables since it allows multiple variables to be declared at the same memory location. No code can be generated in this segment. The res directive should be used to reserve space for data.

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

For more information, refer to <u>Relocatable Objects</u>.

# Example

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

#### See Also

EXTERN GLOBAL IDATA UDATA UDATA\_ACS UDATA\_SHR



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

#### **Syntax**

[<label>] udata\_shr [<RAM address>]

## Description

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

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

For more information, refer to <u>Relocatable Objects</u>.

#### Example

Temps udata\_shr Temp1 res 1 Temp2 res 1 Temp3 res 1

#### See Also

EXTERN GLOBAL IDATA UDATA UDATA ACS UDATA OVR



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

#### **VARIABLE - Declare Symbol Variable**

#### **Syntax**

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

# Description

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

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

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

#### Example

Please refer to the example given for the constant directive.

#### See Also

CONSTANT SET



#### WHILE - Perform Loop While Condition is True

#### **Syntax**

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

#### Description

The lines between the while and the endw are assembled as long as  $\langle expr \rangle$  evaluates to TRUE. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. A relational TRUE expression is guaranteed to return a non-zero value; FALSE a value of zero.

A while loop can contain at most 100 lines and be repeated a maximum of 256 times. while loops can be nested up to 8 deep.

#### 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

> <

# **Directive Usage**

How to use the MPASM assembler directive language is shown using examples.

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.

For a reference listing of all directives discussed in examples here, please see <u>Directives</u>.

**Note:** Although MPASM assembler is often used with MPLINK object linker, MPASM assembler directives are not supported by MPLINK linker. See MPLINK object linker documentation for more information on linker options to control listing and hex file output.

There are six basic types of directives provided by the assembler:

**Note:** Directives are not case-sensitive, e.g., cblock may be executed as CBLOCK, cblock, Cblock, etc.

- <u>Control Directives</u>
- <u>Conditional Assembly Directives</u>
- Data Directives
- Listing Directives

- <u>Macro Directives</u>
- <u>Object File Directives</u>

For all directive types:

Additional Directive Examples

< >

Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Fax: (480) 899-9210 Microchip's E-mail Address

#### **Control Directives**

Control directives control how code is assembled.

List of Control Directives

Control directive examples available:

- Multiple Directive Example 1
- Multiple Directive Example 2
- ORG PIC16CXXX Example
- ORG PIC18CXXX Example
- <u>RADIX Example</u>
- <u>SET/EQU Example</u>
- <u>UNDEFINE/DEFINE Example</u>
- VARIABLE/CONSTANT Example



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

#### **Conditional Assembly Directives**

Conditional assembly directives permit sections of conditionally assembled code.

List of Conditional Assembly Directives

Conditional assembly directive examples available:

- IF/ELSE/ENDIF Example
- IFDEF Example
- <u>WHILE/ENDW Example</u>



#### **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.

• List of Data Directives

Data directive examples available:

- <u>CBLOCK/ENDC Example</u>
- <u>CONFIG PIC16CXXX Example</u>
- <u>CONFIG PIC18CXXX Example</u>
- DA Example
- DATA PIC16CXXX Example
- DATA PIC18CXXX Example
- DB PIC16CXXX Example
- DB PIC18CXXX Example
- DE PIC16CXXX Example
- DE PIC18CXXX Example
- FILL PIC16CXXX Example
- FILL PIC18CXXX Example
- IDLOC PIC16CXXX Example
- IDLOC PIC18CXXX Example
- <u>RES Example</u>



#### Microchip Technology Inc. Microchip's Web Site

Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

## **Listing Directives**

Listing Directives are those directives that control the MPASM assembler listing file format. They allow the specification of titles, pagination, and other listing control. Some listing directives also control how code is assembled.

• List of Listing Directives

Listing directive examples available:

- ERROR Example
- ERRORLEVEL Example
- <u>MESSG Example</u>



#### **Macro Directives**

These directives control the execution and data allocation within macro body definitions.

List of Macro Directives

Macro directive examples available:

- EXITM Example
- LOCAL Example
- MACRO/ENDM Example



# **Object File Directives**

These directives are used only when creating an object file.

List of Object File Directives

Object file directive examples available:

- BANKISEL Example
- BANKSEL Example
- <u>CODE Example</u>
- GLOBAL/EXTERN Example
- IDATA Example
- PAGESEL Example
- UDATA Example
- <u>UDATA\_ACS Example</u>
- <u>UDATA\_OVR Example</u>
- <u>UDATA\_SHR Example</u>



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Additional Directive Examples**

Short examples of use for each directive are listed under each directive topic. See <u>Directives</u>.

Examples of use for multiple directives are available from the following sources:

- readme.asm Serial EEPROM Support
- Application Notes, Technical Briefs
  - Embedded Control Handbook, Volume 1 (DS00092)
- Embedded Control Handbook, Volume 2 Math Library (DS00167)
- Embedded Control Handbook, Update 2000 (DS00711)
- Website http://www.microchip.com
- Code Examples and Templates
  - MPLAB IDE installation directory
- Website http://www.microchip.com



# **Relocatable Objects**

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

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



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Header Files**

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



### **Program Memory**

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

### **Absolute Code**

Start CLRW OPTION

#### **Relocatable Code**

```
CODE
Start CLRW
OPTION
.
```

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

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

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

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

# **Example Relocatable Code**

Reset CODE H'01FF' GOTO Start Main CODE CLRW OPTION



# **Instruction Operands**

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

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

where:

• <relocatable symbol> is any label that defines a program or data memory address

• <constant offset> is an expression that is resolvable at assembly time to a value between -32768 and 32767

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

Operands of the form:

<relocatable symbol> - <relocatable symbol>

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

If HIGH is used, only bits 8 through15 of the expression will be used. If LOW is used, only bits 0 through 7 of the expression will be used. If UPPER is used, only bits 16 through 21 of the expression will be used.



# **RAM Allocation**

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

• UDATA - Uninitialized data. This is the most common type of data section. Locations reserved in this section are not initialized and can be accessed only by the labels defined in this section or by indirect accesses.

• UDATA\_ACS - Uninitialized access data. This data section is used for variables that will be placed in access RAM of PIC18CXXX devices. Access RAM is used as quick data access for specified instructions.

• UDATA\_OVR - Uninitialized overlaid data. This data section is used for variables that can be declared at the same address as other variables in the same module or in other linked modules. A typical use of this sec tion is for temporary variables.

• UDATA\_SHR - Uninitialized shared data. This data section is used for variables that will be placed in RAM that is unbanked or shared across all banks.

• IDATA - Initialized data. The linker will generate a lookup table that can be used to initialize the variables in this section to the specified values. The locations reserved by this section can be accessed only by the labels defined in this section or by indirect accesses.

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

## **Absolute Code**

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

#### **Relocatable Code**

IDATA HistoryVector DB 0 UDATA InputGain RES 1 OutputGain RES 1 UDATA\_OVR Templ RES 1 Temp2 RES 1 Temp3 RES 1

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

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

## **Relocatable Code**

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



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Configuration Bits and ID Locations**

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



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **Accessing Labels From Other Modules**

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

#### **Relocatable Code, Defining Module**

UDATA InputGain RES 1 OutputGain RES 1 GLOBAL InputGain, OutputGain CODE Filter GLOBAL Filter : ; Filter code

#### **Relocatable Code, Referencing Module**

EXTERN InputGain, OutputGain, Filter UDATA Reading RES 1 CODE ... MOVLW GAIN1

MOVWF InputGain MOVLW GAIN2 MOVWF OutputGain MOVF Reading,W CALL Filter



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Paging and Banking Issues**

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

### **Absolute Code**

```
LIST P=12C509
#include "P12C509.INC"
Varl EQU H'10'
Var2 EQU H'30'
...
MOVLW InitialValue
BCF FSR, 5
MOVWF Varl
BSF FSR, 5
MOVWF Var2
BSF STATUS, PA0
CALL Subroutine
...
Subroutine CLRW ;In Page 1
...
RETLW 0
```

#### **Relocatable Code**

LIST P=12C509 #include "P12C509.INC" UDATA Varl RES 1 Var2 RES 1 ... CODE MOVLW InitialValue BANKSEL Varl MOVWF Varl BANKSEL Var2 MOVWF Var2 PAGESEL Subroutine CALL Subroutine ...

Subroutine CLRW

**RETLW 0** 

• • •



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

## **Unavailable Directives**

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

### **Absolute Code**

Reset ORG H'01FF' GOTO Start

### **Relocatable Code**

Reset CODE H'01FF' GOTO Start



#### Microchip Technology Inc. Microchip's Web Site

Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **Generating the Object Module**

Once the code conversion is complete, the object module is generated by requesting an object file on the command line or in the shell interface. When using MPASM assembler for Windows, check the checkbox labeled "Object File." When using the DOS command line interface, specify the /0 option and toggle "Assemble to Object File" to "Yes." The output file will have a .0 extension.



### **Code Examples**

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

<u>Absolute Code</u> becomes <u>Relocatable Code</u>, <u>Calling File</u> and <u>Relocatable Code</u>, <u>Library Routine</u>.

### **Absolute Code**

LIST P=16C54 #INCLUDE "P16C5X.INC" cblock H '020' mulcnd; 8 bit multiplicand mulplr; 8 bit multiplier H\_byte ; High byte of the 16 bit result L\_byte ; Low byte of the 16 bit result count; loop counter endc mpy clrf H\_byte clrf L\_byte movlw 8 movwf count movf mulcnd,w bcf STATUS,C ;Clear carry bit Loop rrf mulplr,F btfsc STATUS,C addwf H\_byte,F rrf H\_byte,F rrf L\_byte,F decfsz count.F goto loop retlw 0  Since an eight-by-eight bit multiply is a useful, generic routine, it would be handy to break this off into a separate object file that can be linked in when required. The above file can be broken into two files: a calling file representing an application and a generic routine that could be incorporated in a library.

# **Relocatable Code, Calling File**

LIST P=16C54 #INCLUDE "P16C5x.INC" EXTERN mulcnd, mulplr, H\_byte, L\_byte EXTERN mpy CODE start clrw option main movf PORTB, W movwf mulplr movf PORTB, W movwf mulcnd call\_m call mpy ; The result is in H\_byte & L\_byte goto main Reset CODE H'0lFF' goto start END

#### **Relocatable Code, Library Routine**

LIST P=16C54 #INCLUDE "P16C5x.INC" UDATA mulcnd RES l; 8 bit multiplicand mulplr RES 1; 8 bit multiplier H\_byte RES 1 ; High byte of the 16 bit result L\_byte RES 1; Low byte of the 16 bit result count RES 1; loop counter GLOBAL mulcnd, mulplr, H\_byte, L\_byte CODE mpy GLOBAL mpy clrf H\_byte clrf L byte movlw 8 movwf count movf muland, W bcf STATUS, C ; Clear carry bit loop rrf mulplr, F btfsc STATUS, C addwf H\_byte, F rrf H byte, F rrf L byte, F decfsz count, F goto loop retlw 0 **END** 



Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Macro Language**

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

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

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

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

- <u>Macro Syntax</u>
- <u>Macro Directives Defined</u>
- Macro Text Substitution
- <u>Macro Usage</u>
- <u>Macro Code Examples</u>



# **Macro Syntax**

MPASM assembler macros are defined according to the following syntax:

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

where <label> is a valid assembler label and <arg> is any number of optional arguments supplied to the macro (that will fit on the source line.) 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 assembler directives, PICmicro MCU assembly instructions, or MPASM assembler macro directives (LOCAL for example.) The assembler continues to process the body of the macro until an EXITM or ENDM directive is encountered.

Note: Forward references to macros are not permitted.



# **Macro Directives Defined**

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

Macro Directives

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

**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 assembler, but as mentioned before, you are encouraged to write new code based on the constructs defined within this help file to ensure upward compatibility.



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **Macro Text Substitution**

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

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

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

```
define_table macro
local a = 0
while a < 3
entry#v(a) dw 0
a += 1
endw
endm
when invoked, would generate:
entry0 dw 0
entry1 dw 0
entry2 dw 0
entry3 dw 0
```



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

## **Macro Usage**

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

<macro\_name> [<arg>, ..., <arg>]

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

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

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



## **Macro Code Examples**

The following are examples of macros:

- Eight-by-Eight Multiply
- <u>Constant Compare</u>

# **Eight-by-Eight Multiply**

```
subtitle "macro definitions"
page
; multiply - is an eight by eight multiply macro,
; executing in program memory, optimized for speed,
; straight line code.
; This macro has five parameters as defined here:
; arg1 - first eight bit literal to be multiplied
; arg2 - second eight bit literal to be multiplied
; dest_hi - memory location for high byte of result
; dest_lo - memory location for low byte of result
; temp - memory location for temporary storage
; During the execution of this macro, the w register is
; destroyed.
; The result of multiply is a 16 bit value stored in the
; two eight bit registers (dest_hi, dest_lo)
; This macro is written for the PIC17C42.
multiply macro arg1, arg2, dest_hi, dest_lo, temp
```

```
local i = 0 ; Establish a local index
; variable and initialize it.
movlw arg1; Setup the eight bit
movwf temp; literal multiplier in the
; memory location temp.
movlw arg2; Setup the eight bit
; literal multiplicand in the
; w register.
clrf dest_hi, F ; Clear both the high and
clrf dest_lo, F ; the low destination
; registers.
bcf ALUSTA, C ; Clear the carry bit.
while i < 8; Use the loop to check all
; eight bits of the
; multiplier (temp).
btfsc temp, i ; Test the current
addwf dest_hi, F; multiplier bit, if temp,I
; then add the multiplicand
; to the high register.
rrcf dest_hi, F ; For each pass in the
rrcf dest_lo, F ; loop, right shift each
; destination register using
; the carry bit.
i += 1 ; Place this increment in
; column 1 to avoid
; Warning [207].
endw;
endm;
```

The macro declares all of the required arguments. In this case, there are five. 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 adds for each bit set in the eight bits of the multiplier and uses right shifts. The WHILE directive is used for this function, continuing the loop until "I" is greater than or equal to eight.

## **Constant Compare**

As another example, if the following macro were written:

include "16cxx.reg" ; compare file to constant and jump if file : >= constant.cfl\_jge macro file, con, jump\_to movlw con & 0xff subwf file, w btfsc status, carry goto jump\_to endm and invoked by: cfl\_jge switch\_val, max\_switch, switch\_on it would produce: movlw max switch & 0xff subwf switch val, w btfsc status, carry goto switch\_on



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Expression Syntax and Operation**

Various expression formats, syntax, and operations used by MPASM assembler are described here.

- <u>Text Strings</u>
- <u>Numeric Constants and Radix</u>
- <u>Arithmetic Operators and Precedence</u>



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **Text Strings**

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

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

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

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

### **Code Examples**

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

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

### **Escape Characters**

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

Escape Character	Description	Hex Value
\a	Bell (alert) character	
\b	Backspace character	
∖f	Form feed character	
\n	New line character	
\r	Carriage return character	0D
\t	Horizontal tab character	
\v	Vertical tab character	
۱۱	Backslash	
\?	Question mark character	
\'	Single quote (apostrophe)	
\"	Double quote character	
\000	Octal number (zero, Octal digit, Octal digit)	
∖xHH	Hexadecimal number	

#### TABLE: ANSI `C' ESCAPE SEQUENCES



#### **Microchip Technology Inc.**

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

### **Numeric Constants and Radix**

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

**Note:** The radix for numeric constants can be different from the default radix specifed with the directives radix or list r=. Also, allowable default radices are limited to hexadecimal, decimal, and octal.

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:

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

#### TABLE: RADIX SPECIFICATIONS



## **Arithmetic Operators and Precedence**

Arithmatic operators and their precedence are listed in <u>Table: Arithmetic Operators and Precedence</u>.

Selected operators are discussed in greater detail in subsections following the table.

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

#### TABLE: ARITHMETIC OPERATORS AND PRECEDENCE

-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
=	Inclusive OR, set equal	flags  = ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment	i ++
	Decrement	i

# High/Low/Upper

#### **Syntax**

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

#### Description

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

#### Example

movlw low size ; handle the lsb's
movpf wreg, low size\_lo
movlw high size ; handle the msb's
movpf wreg, high size\_hi

### Increment/Decrement (++/--)

#### **Syntax**

<variable>++ <variable>--

#### Description

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

#### Example

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

< >

#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

### Troubleshooting

Error messages, warning messages and general messages are produced by the MPASM assembler. These messages always appear in the listing file directly above each line in which the error occurred.

The messages are stored in the error file (.err) if no MPASM assembler 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.

Limitations of the assembler tool are also listed here.

- <u>Assembler Errors</u>
- <u>Assembler Warnings</u>
- <u>Assembler Messages</u>
- <u>Assembler Limitations</u>



#### **Assembler Errors**

MPASM assembler errors are listed numerically below:

#### **101 ERROR:**

User error, invoked with the ERROR directive.

#### **102 Out of memory.**

Not enough memory for macros, #defines or internal processing. Eliminate any TSR's, close any open applications, and try assembling the file again. If this error was obtained using the Real Mode DOS executable, try using either the Windows version (MPASMWIN) or DPMI version (MPASM\_DP)

#### **103 Symbol table full.**

No more memory available for the symbol table. Eliminate any TSR's, close any open applications, and try assembling the file again. If this error was obtained using the Real Mode DOS executable, try using either the Windows version (MPASMWIN) or DPMI version (MPASM\_DP)

#### **104 Temp file creation error.**

Could not create a temporary file. Check the available disk space.

#### 105 Cannot open file.

Could not open a file. If it is a source file, the file may not exist. If it is an output file, the old version may be write protected.

### **106 String substitution too complex.**

Too much nesting of #defines.

### 107 Illegal digit.

An illegal digit in a number. Valid digits are 0-1 for binary, 0-7 for octal, 0-9 for decimal, and 0-9, a-f, and A-F for hexadecimal.

#### **108 Illegal character.**

An illegal character in a label. Valid characters for labels are alphabetic (a..f, A..F), numeric (0-9), the underscore (\_), and the question mark (?). Labels may not begin with a numeric.

#### 109 Unmatched (

An open parenthesis did not have a matching close parenthesis. For example, "DATA (1+2".

#### 110 Unmatched )

An close parenthesis did not have a matching open parenthesis. For example, DATA 1+2).

#### **111 Missing symbol.**

An EQU or SET statement did not have a symbol to assign the value to.

#### **112 Missing operator.**

An arithmetic operator was missing from an expression. For example, DATA 1 2.

#### **113 Symbol not previously defined.**

A symbol was referenced that has not yet been defined. Only addresses may be used as forward references. Constants and variables must be declared before they are used.

#### 114 Divide by zero.

Division by zero encountered during an expression evaluation.

#### **115 Duplicate label.**

A label was declared as a constant (e.g., with the EQU or CBLOCK directive) in more than one location.

# **116 Address label duplicated or different in second pass.**

The same label was used in two locations. Alternately, the label was used only once but evaluated to a different location on the second pass. This often happens when users try to write page-bit setting macros that generate different numbers of instructions based on the destination.

#### **117 Address wrapped around 0.**

The location counter can only advance to FFFF. After that, it wraps back to 0.

#### **118 Overwriting previous address contents.**

Code was previously generated for this address.

#### **119 Code too fragmented.**

The code is broken into too many pieces. This error is very rare, and will only occur in source code that references addresses above 32K (including configuration bits).

#### **120 Call or jump not allowed at this address.**

A call or jump cannot be made to this address. For example, CALL destinations on the PIC16C5x family must be in the lower half of the page.

#### **121 Illegal label.**

Labels are not allowed on certain directive lines. Simply put the label on its own line, above the directive. Also, HIGH, LOW, PAGE, and BANK are not allowed as labels.

#### **122 Illegal opcode.**

Token is not a valid opcode.

#### **123 Illegal directive.**

Directive is not allowed for the selected processor; for example, the

\_ IDLOCS directive on the PIC17C42.

### **124 Illegal argument.**

An illegal directive argument; for example, LIST STUPID.

### **125 Illegal condition.**

A bad conditional assembly. For example, an unmatched ENDIF.

### **126 Argument out of range.**

Opcode or directive argument out of the valid range; for example, TRIS 10.

#### **127 Too many arguments.**

Too many arguments specified for a macro call.

### **128** Missing argument(s).

Not enough arguments for a macro call or an opcode.

### 129 Expected.

Expected a certain type of argument. The expected list will be provided.

### **130 Processor type previously defined.**

A different family of processor is being selected.

### **131 Processor type is undefined.**

Code is being generated before the processor has been defined. Note that until the processor is defined, the opcode set is not known.

#### 132 Unknown processor.

The selected processor is not a valid processor.

#### **133 Hex file format INHX32 required.**

An address above 32K was specified. For example, specifying the configuration bits on the PIC17CXXX family.

#### 134 Illegal hex file format.

An illegal hex file format was specified in the LIST directive.

#### 135 Macro name missing.

A macro was defined without a name.

#### **136 Duplicate macro name.**

A macro name was duplicated.

### **137 Macros nested too deep.**

The maximum macro nesting level was exceeded.

#### **138 Include files nested too deep.**

The maximum include file nesting level was exceeded.

#### 139 Maximum of 100 lines inside WHILE-ENDW.

A WHILE-ENDW can contain at most 100 lines.

# **140 WHILE must terminate within 256** iterations.

A WHILE-ENDW loop must terminate within 256 iterations. This is to prevent infinite assembly.

#### 141 WHILEs nested too deep.

The maximum WHILE-ENDW nesting level was exceeded.

### 142 IFs nested too deep.

The maximum IF nesting level was exceeded.

#### 143 Illegal nesting.

Macros, IF's and WHILE's must be completely nested; they cannot overlap. If you have an IF within a WHILE loop, the ENDIF must come before the ENDW.

### 144 Unmatched ENDC.

ENDC found without a CBLOCK.

### 145 Unmatched ENDM.

ENDM found without a MACRO definition.

#### **146 Unmatched EXITM.**

EXITM found without a MACRO definition.

## 147 Directive not allowed when generating an object file.

The ORG directive is not allowed when generating an object file. Instead, declare a data or code section, specifying the address if necessary.

# **148 Expanded source line exceeded 200 characters.**

The maximum length of a source line, after #DEFINE and macro parameter substitution, is 200 characters. Note that #DEFINE substitution does not include comments, but macro parameter substitution does.

## **149 Directive only allowed when generating an object file section.**

Certain directives, such as GLOBAL and EXTERN, only have meaning when an object file is generated. They cannot be used when generating absolute code.

# 150 Labels must be defined in a code or data section when making an object file.

When generating an object file, all data and code address labels

must be defined inside a data or code section. Symbols defined by the EQU and SET directives can be defined outside of a section.

# **151 Operand contains unresolvable labels or is too complex.**

When generating an object file, operands must be of the form [HIGH|LOW]([<relocatable address label>]+[<offset>]).

# **152 Executable code and data must be defined in an appropriate section.**

When generating an object file, all executable code and data declarations must be placed within appropriate sections.

## **153 Page or Bank bits cannot be evaluated for the operand.**

The operand of a PAGESEL, BANKSEL or BANKISEL directive must be of the form <relocatable address label> or <constant>.

# **154 Each object file section must be contiguous.**

Object file sections, except UDATA\_OVR sections, cannot be stopped and restarted within a single source file. To resolve this problem, either name each section with its own name or move the code and data declarations such that each section is contiguous. This error will also be generated if two sections of different types are given the same name.

#### **155 All overlaid sections of the same name**

#### must have the same starting address.

If multiple UDATA\_OVR sections with the same name are declared, they must all have the same starting address.

#### **156 Operand must be an address label.**

When generating object files, only address labels in code or data sections may be declared global. Variables declared by the SET or EQU directives may not be exported.

#### **157 UNKNOWN ERROR.**

An error has occurred which the assembler cannot understand. It is not any of the errors described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this error.



#### **Assembler Warnings**

MPASM assembler warnings are listed numerically below:

#### **201 Symbol not previously defined.**

Symbol being #undefined was not previously defined.

# **202 Argument out of range. Least significant bits used.**

Argument did not fit in the allocated space. For example, literals must be 8 bits.

#### 203 Found opcode in column 1.

An opcode was found in column one, which is reserved for labels.

#### 204 Found pseudo-op in column 1.

A pseudo-op was found in column one, which is reserved for labels.

#### **205 Found directive in column 1.**

A directive was found in column one, which is reserved for labels.

### **206 Found call to macro in column 1.**

A macro call was found in column one, which is reserved for labels.

#### 207 Found label after column 1.

A label was found after column one, which is often due to a misspelled opcode.

#### **208 Label truncated at 32 characters.**

Maximum label length is 32 characters.

#### 209 Missing quote.

A text string or character was missing a quote. For example, DATA 'a.

### 210 Extra ),

An extra comma was found at the end of the line.

#### **211 Extraneous arguments on the line.**

Extra arguments were found on the line. These warnings should be investigated, since they are often indications of the free-format parser interpreting something in a manner other than was intended (try assembling OPTION EQU 0x81 with LIST FREE).

### **212 Expected**

Expected a certain type of argument. A description should be provided. For the warning, an assumption is made about the argument.

#### **213 The EXTERN directive should only be**

#### used when making a .o file.

The EXTERN directive only has meaning if an object file is being created. This warning has been superseded by Error 149.

### 214 Unmatched (

An unmatched parenthesis was found. The warning is used if the parenthesis is not used for indicating order of evaluation.

#### **215 Processor superseded by command line. Verify processor symbol.**

The processor was specified on the command line as well as in the source file. The command line has precedence.

#### **216 Radix superseded by command line.**

The radix was specified on the command line as well as in the source file. The command line has precedence.

# **217 Hex file format specified on command line.**

The hex file format was specified on the command line as well as in the source file. The command line has precedence.

#### 218 Expected DEC, OCT, HEX. Will use HEX.

Bad radix specification.

#### **219 Invalid RAM location specified.**

If the \_\_MAXRAM and \_\_BADRAM directives are used, this warning flags use of any RAM locations declared as invalid by these directives. Note that the provided header files include \_\_MAXRAM and \_\_BADRAM for each processor.

# 220 Address exceeds maximum range for this processor.

A ROM location was specified that exceeds the processor's memory size.

#### 221 Invalid message number.

The message number specified for displaying or hiding is not a valid message number.

#### **222 Error messages cannot be disabled.**

Error messages cannot be disabled with the ERRORLEVEL command.

#### **223 Redefining processor**

The selected processor is being reselected by the LIST or PROCESSOR directive.

## **224 Use of this instruction is not recommended.**

Use of the TRIS and OPTION instructions is not recommended for

a PIC16CXXX device.

#### **225 Invalid label in operand.**

Operand was not a valid address. For example, if the user tried to issue a CALL to a MACRO name.

#### **226 UNKNOWN WARNING**

A warning has occurred which the assembler cannot understand. It is not any of the warnings described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this warning.



#### **Assembler Messages**

MPASM assembler messages are listed numerically below:

#### **301 MESSAGE:**

User message, invoked with the MESSG directive.

#### 302 Register in operand not in bank 0. Ensure that bank bits are correct.

Register address was specified by a value that included the bank bits. For example, RAM locations in the PIC16CXXX are specified with 7 bits in the instruction and one or two bank bits.

## 303 Program word too large. Truncated to core size.

Program words for the PIC16C5X may only be 12-bits; program words for the PIC16CXXX may only be 14-bits.

# **304 ID Locations value too large. Last four hex digits used.**

Only four hex digits are allowed for the ID locations.

#### **305 Using default destination of 1 (file).**

If no destination bit is specified, the default is used.

# **306 Crossing page boundary - ensure page bits are set.**

Generated code is crossing a page boundary.

#### **307 Setting page bits.**

Page bits are being set with the LCALL or LGOTO pseudo-op.

# 308 Warning level superseded by command line value.

The warning level was specified on the command line as well as in the source file. The command line has precedence.

## 309 Macro expansion superseded by command line.

Macro expansion was specified on the command line as well as in the source file. The command line has precedence.

## **310 Superseding current maximum RAM and RAM map.**

The \_\_\_MAXRAM directive has been used previously.

## **312 Page or Bank selection not needed for this device. No code generated.**

If a device contains only one ROM page or RAM bank, no page or bank selection is required, and any PAGESEL, BANKSEL, or

BANKISEL directives will not generate any code.

### 313 CBLOCK constants will start with a value of 0.

If the first CBLOCK in the source file has no starting value specified, this message will be generated.

#### **314 UNKNOWN MESSAGE**

A message has occurred which the assembler cannot understand. It is not any of the messages described in this appendix. Contact your Microchip Field Application Engineer (FAE) if you cannot debug this message.



### **Assembler Limitations**

#### **General Limitations**

- MPASM assembler only looks in the path of the executable or the file being assembled for included files. Therefore, Include Path information entered in the MPLAB IDE Edit Project dialog will NOT be used by the assembler.
- Source line limit (expanded) = 200 characters
- File name limit = 8.3 format

#### **Directive Limitations**

- while nest limit = 8 deep while loop limit = 100 lines while iteration limit = 256
- if nest limit = 16 deep
- include nest limit = 5 levels include max. number of files = 255
- macro nest limit = 16 deep macro source line limit (expanded) = 200 characters
- Do not use #includes in macros.



### **Instruction Sets**

PICmicro MCU instruction sets are used in developing applications with MPASM assembler, MPLINK object linker and MPLIB object librarian.

Instructions are listed here based on device core size. As of the time of publication of this document, the following core sizes map to the following devices:

- 12-Bit Core Devices: PIC12C5XX, PIC12CE5XX, PIC16X5X, PIC16C505
- 14-Bit Core Devices: PIC12C67X, PIC12CE67X, PIC12F629/675, PIC16XXXX
- 16-Bit Core Devices: PIC17CXXX
- Extended 16-Bit Core Devices: PIC18XXXXX

Topics covered are:

- <u>Key to PICmicro Family Instruction Sets</u>
- 12-Bit Core Instruction Set
- <u>14-Bit Core Instruction Set</u>
- 16-Bit Core Instruction Set
- Key to Extended 16-Bit Core Instruction Set
- Extended 16-Bit Core Instruction Set



### Key to PICmicro Family Instruction Sets

Field	Description					
<b>Register F</b> i	iles					
dest	Destination either the WREG register or the specified register file location. See d.					
f	Register file address (5-bit, 7-bit or 8-bit).					
р	Peripheral register file address (5-bit).					
r	Port for TRIS.					
x	Don't care ( $^0$ or $^1$ ). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.					
Literals						
k	Literal field, constant data or label. k 4-bit. kk 8-bit. kkk 12-bit.					
Bits						
b	Bit address within an 8-bit file register (0 to 7).					
d	Destination select bit. d = 0: store result in WREG d = 1: store result in file register f <b>(default)</b>					
i	Table pointer control. i = 0: do not change. i = 1: increment after instruciton execution.					
s	Destination select bit. s = 0: store result in file register f and WREG s = 1: store result in file register f <b>(default)</b>					
t	Table byte select. t = 0: perform operation on lower byte. t = 1: perform operation on upper byte.					
11	Bit values, as opposed to Hex value.					
Named Re	gisters					
BSR	Bank Select Register. Used to select the current RAM bank.					
OPTION	OPTION Register.					
PCL	Program Counter Low Byte.					
РСН	Program Counter High Byte.					
PCLATH	Program Counter High Byte Latch.					
PCLATU	Program Counter Upper Byte Latch.					
PRODH	Product of Multiply High Byte.					
PRODL	Product of Multiply Low Byte.					
TBLATH	Table Latch (TBLAT) High Byte.					
TBLATL	Table Latch (TBLAT) Low Byte.					

TBLPTR	16-bit Table Pointer (TBLPTRH:TBLPTRL). Points to a Program Memory location.							
WREG	Working register (accumulator).							
Named Bi	Named Bits							
C, DC, Z, OV, N	ALU Status bits: Carry, Digit Carry, Zero, Overflow, Negative.							
TO	Time-out bit.							
PD	Power-down bit.							
GIE	Global Interrupt Enable bit(s).							
Named De	vice Features							
PC	Program Counter.							
TOS	Top-of-Stack.							
WDT	Watchdog Timer.							
Misc. Desc	riptors							
()	Contents.							
$\rightarrow, \leftrightarrow$	Assigned to.							
< >	Register bit field.							



#### **12-Bit Core Instruction Set**

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

The instruction set is grouped into the following catagories: Byteoriented file register operations, bit-oriented file register operations, and core literal and control operations. Instructions are listed by catagory in the tables below. Instruction opcode is show in Hex by certain making assumptions, either listed in the key or as a footnote. For more information on the opcode bit values for each instruction, as well as the number of cycles per instruction, status bits affected and complete instruction details, see the relevent device data sheet.

Hex	Mnemo	onic	Description	Function		
1Ef*	ADDWF	f,d	Add W and f	WREG + f $\rightarrow$ dest		
16f*	ANDWF	f,d	AND W and f	WREG .AND. $f \rightarrow dest$		
06f	CLRF	f	Clear f	$0 \rightarrow f$		
040	CLRW		Clear W	$0 \rightarrow WREG$		
26f*	COMF	f,d	Complement f	.NOT. $f \rightarrow dest$		
0Ef*	DECF	f,d	Decrement f	$f - 1 \rightarrow dest$		
2Ef*	DECFSZ	f,d	Decrement f, skip if zero	f - 1 $\rightarrow$ dest, skip if zero		
2Af*	INCF	f,d	Increment f	$f + 1 \rightarrow dest$		
3Ef*	INCFSZ	f,d	Increment f, skip if zero	$f + 1 \rightarrow \text{dest, skip if zero}$		
12f*	IORWF	f,d	Inclusive OR W and f	WREG .OR. $f \rightarrow \text{dest}$		
22f*	/		Move f	f → dest		
02f	MOVWF	f	Move W to f	WREG $\rightarrow$ f		
000	NOP		No operation			
36f*	RLF	f,d	Rotate left f	register f		

TABLE: 12-BIT CORE BYTE-ORIENTED FILE REGISTER OPERATIONS

32f*	RRF	f,d	Rotate right f	register f	
0Af*	SUBWF	f,d	Subtract W from f	$f - WREG \rightarrow dest$	
3Af*	SWAPF	f,d	Swap halves f	$f(0:3) \leftrightarrow f(4:7) \rightarrow dest$	
1Af*	1Af* XORWF f,d Exclusive OR W and t		Exclusive OR W and f	WREG .XOR. $f \rightarrow dest$	
* Assu	* Assuming default bit value for d.				

#### **TABLE: 12-BIT CORE BIT-ORIENTED FILE REGISTER OPERATIONS**

Hex	Mnem	onic	Description	Function
4bf	BCF	f,b	Bit clear f	$0 \rightarrow f(b)$
5bf			Bit set f	$1 \rightarrow f(b)$
6bf			Bit test, skip if clear	skip if f(b) = 0
7bf	BTFSS	f,b	Bit test, skip if set	skip if f(b) = 1

#### **TABLE: 12-BIT CORE LITERAL AND CONTROL OPERATIONS**

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



#### **14-Bit Core Instruction Set**

Microchip's mid-range 8-bit microcontroller family uses a 14-bit wide instruction set. This instruction set consists of 36 instructions, each a single 14-bit wide word. Most instructions operate on a file register, f, and the working register, WREG (accumulator). The result can be directed either to the file register or the WREG register or to both in the case of some instructions. A few instructions operate solely on a file register (BSF, for example).

The instruction set is grouped into the following catagories: Byteoriented file register operations, bit-oriented file register operations, and core literal and control operations. Instructions are listed by catagory in the tables below. Instruction opcode is show in Hex by certain making assumptions, either listed in the key or as a footnote. For more information on the opcode bit values for each instruction, as well as the number of cycles per instruction, status bits affected and complete instruction details, see the relevent device data sheet.

I				/- · · · · · · · · · · · · · · · · · · ·
Hex	Mnemo	onic	Description	Function
07df	ADDWF f,d		Add W and f	$W + f \rightarrow d$
05df	ANDWF	f,d	AND W and f	W .AND. $f \rightarrow d$
01'1'f	CLRF	f	Clear f	$0 \rightarrow f$
01xx	CLRW		Clear W	$0 \rightarrow W$
09df	COMF	f,d	Complement f	.NOT. $f \rightarrow d$
03df	DECF	f,d	Decrement f	$f - 1 \rightarrow d$
0Bdf	DECFSZ	f,d	Decrement f, skip if zero	$f - 1 \rightarrow d$ , skip if 0
0Adf	INCF	f,d	Increment f	$f + 1 \rightarrow d$
0Fdf	INCFSZ	f,d	Increment f, skip if zero	$f + 1 \rightarrow d$ , skip if 0
04df	IORWF	f,d	Inclusive OR W and f	W .OR. $f \rightarrow d$
08df	MOVF	f,d	Move f	$f \rightarrow d$
00'1'f	MOVWF	f	Move W to f	$W \rightarrow f$
0000	NOP		No operation	

TABLE: 14-BIT CORE BYTE-ORIENTED FILE REGISTER OPERATIONS

0Ddf	RLF	f,d	Rotate left f	register f
			-	register f
0Cdf	RRF	f,d	Rotate right f	
02df	SUBWF	f,d	Subtract W from f	$f - W \rightarrow d$
0Edf	SWAPF	f,d	Swap halves f	$f(0:3) \leftrightarrow f(4:7) \rightarrow d$
06df	XORWF	f,d	Exclusive OR W and f	W.XOR. $f \rightarrow d$

#### TABLE: 14-BIT CORE BIT-ORIENTED FILE REGISTER OPERATIONS

Hex	Mnemo	onic	Description	Function
4bf		f,b	Bit clear f	$0 \rightarrow f(b)$
5bf	BSF	f,b	Bit set f	$1 \rightarrow f(b)$
6bf	BTFSC	f,b	Bit test, skip if clear	skip if f(b) = 0
7bf			Bit test, skip if set	skip if f(b) = 1

#### TABLE: 14-BIT CORE LITERAL AND CONTROL OPERATIONS

Hex	Mnemo	onic	Description	Function	
3Ekk	ADDLW	kk	Add literal to W	$kk + WREG \rightarrow WREG$	
39kk	ANDLW	kk	AND literal and W	kk .AND. WREG $\rightarrow$ WREG	
2'0'kkk	CALL	kkk	Call subroutine	$PC + 1 \rightarrow TOS, kk \rightarrow PC$	
0064	CLRWDT		Clear watchdog timer	$0 \rightarrow WDT$ (and Prescaler if assigned)	
2'1'kkk	GOTO	kkk	Goto address (k is nine bits)	$kk \rightarrow PC(9 bits)$	
38kk	IORLW	kk	Incl. OR literal and W	kk .OR. WREG $\rightarrow$ WREG	
30kk	MOVLW	kk	Move Literal to W	$kk \rightarrow WREG$	
0062	OPTION		Load OPTION register	WREG $\rightarrow$ OPTION Register	
0009	RETFIE		Return from Interrupt	TOS $\rightarrow$ PC, 1 $\rightarrow$ GIE	
34kk	RETLW	kk	Return with literal in W	$kk \rightarrow WREG, TOS \rightarrow PC$	
0008	RETURN		Return from subroutine	$TOS \rightarrow PC$	
0063	SLEEP		Go into Standby Mode	$0 \rightarrow$ WDT, stop oscillator	
3Ckk	SUBLW	kk	Subtract W from literal	$kk - WREG \rightarrow WREG$	
006r	TRIS	r	Tristate port r	WREG → I/O control reg r	
3Akk	XORLW	kk	Exclusive OR literal and W	kk .XOR. WREG $\rightarrow$ WREG	

#### TABLE: 12-BIT/14-BIT CORE SPECIAL INSTRUCTION MNEMONICS

Mnemonic	Description	Equivalent Operation(s)	Status					
		BTFSC 3,0						

ADDCF	f,d	Add Carry to File	INCF	f,d	Z
ADDDCF	f,d	Add Digit Carry to File	BTFSC INCF	3,1 f,d	Z
В	k	Branch	GOTO	k	-
вс	k	Branch on Carry	BTFSC GOTO	3,0 k	-
BDC	k	Branch on Digit Carry	BTFSC GOTO	3,1 k	-
BNC	k	Branch on No Carry	BTFSS GOTO	3,0 k	-
BNDC	k	Branch on No Digit Carry	BTFSS GOTO	3,1 k	-
BNZ	k	Branch on No Zero	BTFSS GOTO	3,2 k	-
BZ	k	Branch on Zero	BTFSC GOTO	3,2 k	-
CLRC		Clear Carry	BCF	3,0	-
CLRDC		Clear Digit Carry	BCF	3,1	-
CLRZ		Clear Zero	BCF	3,2	-
LCALL	k	Long Call	BCF/BSF BCF/BSF CALL	<b>0x0A,3</b> 0x0A,4 k	
LGOT0	k	Long GOTO	BCF/BSF BCF/BSF GOTO	<b>0x0A,3</b> 0x0A,4 k	
MOVFW	f	Move File to W	MOVF	f,0	Z
NEGF	f,d	Negate File	COMF INCF	f,1 f,d	Z
SETC		Set Carry	BSF	3,0	-
SETDC		Set Digit Carry	BSF	3,1	-
SETZ		Set Zero	BSF	3,2	-
SKPC		Skip on Carry	BTFSS	3,0	-
SKPDC		Skip on Digit Carry	BTFSS	3,1	-
SKPNC		Skip on No Carry	BTFSC	3,0	-
SKPNDC		Skip on No Digit Carry	BTFSC	3,1	-
SKPNZ		Skip on Non Zero	BTFSC	3,2	-
SKPZ		Skip on Zero	BTFSS	3,2	-
SUBCF	f,d	Subtract Carry from File	BTFSC DECF	3,0 f,d	Z
SUBDCF	f,d	Subtract Digit Carry from File	BTFSC	3,1	z

			DECF	f,d	
TSTF	f	Test File	MOVF	f,1	Z



#### **16-Bit Core Instruction Set**

Microchip's high-performance 8-bit microcontroller family uses a 16-bit wide instruction set. This instruction set consists of 55 instructions, each a single 16-bit wide word. Most instructions operate on a file register, f, and the working register, WREG (accumulator). The result can be directed either to the file register or the WREG register or to both in the case of some instructions. Some devices in this family also include hardware multiply instructions. A few instructions operate solely on a file register (BSF for example).

The instruction set is grouped into the following catagories: Byteoriented file register operations, bit-oriented file register operations, and core literal and control operations. Instructions are listed by catagory in the tables below. Instruction opcode is show in Hex by certain making assumptions, either listed in the key or as a footnote. For more information on the opcode bit values for each instruction, as well as the number of cycles per instruction, status bits affected and complete instruction details, see the relevent device data sheet.

Hex	Mnem	nonic	Description	Function		
0Ff*	ADDWF	f,d	Add WREG to F	$(WREG + f) \rightarrow dest$		
11f*	ADDWFC	f,d	Add WREG and Carry to f	$(WREG + f + C) \rightarrow dest$		
0Bf*	ANDWF	f,d	AND WREG with f	(WREG .AND. f) $\rightarrow$ dest		
29f*	CLRF	f,s	Clear dest	$0 \times 00 \rightarrow \text{dest}$		
13f*	COMF	f,d	Complement f	.NOT. $f \rightarrow dest$		
31f	CPFSEQ	f	Compare f, WREG, skip if f = WREG	f-WREG, skip if f = WREG		
32f	CPFSGT	f	Compare f, WREG, skip if f > WREG	f-WREG, skip if f > WREG		
30f	CPFSLT	f	Compare f, WREG, skip if f< WREG f-WREG, skip if f < WREG			
2Ff*	DAW	f,s	Dec. adjust WREG, store in dest WREG adjusted $\rightarrow$ dest			
07f*	DECF	f,d	Decrement f $(f - 1) \rightarrow \text{dest}$			
17f*	DECFSZ	f,d	Decrement f, skip if 0 $(f - 1) \rightarrow \text{dest}$ , skip if 0			
27f*	DCFSNZ	f,d	Decrement f, skip if not 0	$(f - 1) \rightarrow \text{dest, skip if not } 0$		

TABLE: 16-BIT BYTE-ORIENTED FILE REGISTER OPERATIONS

1 +	тисг		La concerca de C		
			Increment f	$\frac{(f+1) \rightarrow \text{dest}}{(f+1) \rightarrow \text{dest}}$	
	INCFSZ		Increment f, skip if zero	$(f + 1) \rightarrow \text{dest, skip if } 0$	
	INFSNZ		Increment f, skip if not zero	$(f + 1) \rightarrow \text{dest, skip if not } 0$	
	IORWF		Inclusive or WREG with f	$(WREG .OR. f) \rightarrow dest$	
	MOVFP	f,p	Move f to p	f → p	
	MOVPF	p,f	Move p to f	$p \rightarrow f$	
	MOVWF	f	Move WREG to F	WREG → f	
		f	Multiply WREG and f	$(WREG x f) \rightarrow PRODH:PRODL$	
2Df*	NEGW	f,s	Negate WREG, store in dest	-WREG → dest	
0000	NOP		No operation	No operation	
1Bf*	RLCF	f,d	Rotate left through carry	register f	
23f*	RLNCF	f,d	Rotate left (no carry)	register f 70 ◀	
19f*	RRCF	f,d	Rotate right through carry	register f ►C ► 70	
21f*	RRNCF	f,d	Rotate right (no carry)	register f ▶ 70	
2Af*	SETF	f,s	Set dest	0xFF → dest	
05f*	SUBWF	f,d	Subtract WREG from f	$(f - WREG) \rightarrow d$	
03f*	SUBWFB	f,d	Subtract from f with borrow	$(f - WREG - c) \rightarrow d$	
1Df*	SWAPF	f,d	Swap f	$f(0:3) \rightarrow d(4:7),$ $f(4:7) \rightarrow d(0:3)$	
A8f	TABLRD	t,i,f		TBLATH $\rightarrow$ f if t=1, TBLATL $\rightarrow$ f if t=0; ProgMem(TBLPTR) $\rightarrow$ TBLAT; TBLPTR + 1 $\rightarrow$ TBLPTR if i=1	
ACf	TABLWT	t,i,f	Write data from file f to table latch and then write 16-bit table latch to program memory location addressed by table pointer	f → TBLATH if t = 1, f → TBLATL if t = 0; TBLAT → ProgMem(TBLPTR); TBLPTR + 1 → TBLPTR if i=1	
A0f	TLRD	t,f	Read data from table latch into file f (table latch unchanged)	$TBLATH \rightarrow f \text{ if } t = 1$ TBLATL $\rightarrow f \text{ if } t = 0$	
A4f	TLWT	t,f	Write data from file f into table latch	$f \rightarrow TBLATH \text{ if } t = 1$ $f \rightarrow TBLATL \text{ if } t = 0$	
33f	TSTFSZ	f	Test f, skip if zero	skip if f = 0	
		f,d	Exclusive OR WREG with f	(WREG .XOR. f) $\rightarrow$ dest	
	P	P	<del>,</del>		

#### TABLE: 16-BIT CORE BIT-ORIENTED FILE REGISTER OPERATIONS

Hex	Mnem	onic	Description	Function
8'1'bf	BCF	f,b	Bit clear f	$0 \rightarrow f(b)$
8'0'bf	BSF	f,b	Bit set f	$1 \rightarrow f(b)$
9'1'bf	BTFSC	f,b	Bit test, skip if clear	skip if f(b) = 0
9'0'bf	BTFSS	f,b	Bit test, skip if set	skip if f(b) = 1
3'1'bf	BTG	f,b	Bit toggle f	.NOT. $f(b) \rightarrow f(b)$

#### TABLE: 16-BIT CORE LITERAL AND CONTROL OPERATIONS

Hex	Mnemonic		Description	Function
B1kk	ADDLW	kk	Add literal to WREG	$(WREG + kk) \rightarrow WREG$
B5kk	ANDLW	kk	AND Literal and WREG	(WREG .AND. kk) $\rightarrow$ WREG
Ekkk	CALL	kkk	Subroutine call (within 8k page)	PC+1 → TOS,k → PC(12:0), k(12:8) → PCLATH(4:0), PC(15:13) → PCLATH(7:5)
0004	CLRWT		Clear watchdog timer $\begin{array}{c} 0 \rightarrow WDT, 0 \rightarrow WDT \text{ prescaler}, \\ 1 \rightarrow \overline{PD}, 1 \rightarrow \overline{TO} \end{array}$	
Ckkk	GOTO	kkk	Unconditional branch (within 8k)	$k \rightarrow PC(12:0)$ k(12:8) → PCLATH(4:0), PC(15:13) → PCLATH(7:5)
B3kk	IORLW	kk	Inclusive OR literal with W	(WREG .OR. kk) $\rightarrow$ WREG
B7kk	LCALL	kk	Long Call (within 64k)	$(PC+1) \rightarrow TOS; kk \rightarrow PCL,$ $(PCLATH) \rightarrow PCH$
B8xk	MOVLB	k	Move literal to low nibble in BSR	$k \rightarrow BSR (3:0)$
BAkx	MOVLR	k	Move literal to high nibble in BSR	$k \rightarrow BSR(7:4)$
B0kk	MOVLW	kk	Move literal to WREG	$kk \rightarrow WREG$
BCkk	MULLW	kk	Multiply literal and WREG	$(kk \times WREG) \rightarrow PRODH:PRODL$
0005	RETFIE		Return from interrupt, enable interrupt	$\begin{array}{l} (\text{PCLATH}) \rightarrow \text{PCH:k} \rightarrow \text{PCL} \\ 0 \rightarrow \text{GLINTD} \end{array}$
B6kk	RETLW	kk	Return with literal in WREG	kk → W, TOS → PC, (PCLATH unchanged)
0002	RETURN		Return from subroutine	TOS → PC (PCLATH unchanged)
0003	SLEEP		Enter Sleep Mode	Stop oscillator, power down, $0 \rightarrow WDT$ , $0 \rightarrow WDT$ Prescaler $1 \rightarrow \overline{PD}$ , $1 \rightarrow \overline{TO}$
B2kk	SUBLW	kk	Subtract WREG from literal	$(kk - WREG) \rightarrow WREG$
B4kk	XORLW	kk	Exclusive OR literal with WREG	$(WREG .XOR. kk) \rightarrow WREG$



### **Key to Extended 16-Bit Core Instruction Set**

Field	Description					
Register Fi	Register Files					
dest	Destination either the WREG register or the specified register file location. See <b>d</b> .					
f	Register file address. f 8-bit (0x00 to 0xFF). f ' 12-bit (0x000 to 0xFFF). This is the source address. f" 12-bit (0x000 to 0xFFF). This is the destination address.					
r	0, 1 or 2 for FSR number.					
x	Don't care (`0' or `1'). The assembler will generate code with $x = 0$ . It is the recommended form of use for compatibility with all Microchip software tools.					
Literals						
k	Literal field, constant data or label. k 4-bit. kk 8-bit. kkk 12-bit.					
Offsets, In	crements/Decrements					
n	The relative address (2's complement number) for relative branch instructions, or the direct address for Call/Branch and Return instructions.					
* *+ *- +*	The mode of the TBLPTR register for the table read and table write instructions. Only used with table read (TBLRD) and table write (TBLWT) instructions: No Change to register Post-Increment register Post-Decrement register Pre-Increment register					
Bits						
a	RAM access bit a = 0: RAM location in Access RAM (BSR register is ignored) a = 1: RAM bank is specified by BSR register <b>(default)</b>					
b	Bit address within an 8-bit file register (0 to 7).					
d	Destination select bit d = 0: store result in WREG d = 1: store result in file register f <b>(default)</b>					
s	Fast Call/Return mode select bit s = 0: do not update into/from shadow registers <b>(default)</b> s = 1: certain registers loaded into/from shadow registers (Fast mode)					
11	Bit values, as opposed to Hex value.					
Named Re	gisters					
BSR	Bank Select Register. Used to select the current RAM bank.					
FSR	File Select Register.					
PCL	Program Counter Low Byte.					
РСН	Program Counter High Byte.					

PCLATH	Program Counter High Byte Latch.
PCLATU	Program Counter Upper Byte Latch.
PRODH	Product of Multiply High Byte.
PRODL	Product of Multiply Low Byte.
STATUS	Status Register
TABLAT	8-bit Table Latch.
TBLPTR	21-bit Table Pointer (points to a Program Memory location).
WREG	Working register (accumulator).
Named Bi	ts
C, DC, Z, OV, N	ALU Status bits: Carry, Digit Carry, Zero, Overflow, Negative.
TO	Time-out bit.
PD	Power-down bit.
PEIE	Peripheral Interrupt Enable bit.
GIE, GIEL/H	Global Interrupt Enable bit(s).
Named De	vice Features
MCLR	Master clear device reset.
PC	Program Counter.
TOS	Top-of-Stack.
WDT	Watchdog Timer.
Misc. Desc	riptors
()	Contents.
_→	Assigned to.
< >	Register bit field.



### **Extended 16-Bit Core Instruction Set**

Microchip's new high-performance 8-bit microcontroller family uses a 16-bit wide instruction set. This instruction set consists of 76 instructions, each a single 16-bit wide word (2 bytes). Most instructions operate on a file register, f, and the working register, WREG (accumulator). The result can be directed either to the file register or the WREG register or to both in the case of some instructions. A few instructions operate solely on a file register (BSF for example).

Hex	Mnemonic		Description	Function
27f*	ADDWF	f,d,a	ADD WREG to f	WREG+f $\rightarrow$ dest
23f*	ADDWFC	f,d,a	ADD WREG and Carry bit to f	WREG+f+C $\rightarrow$ dest
17f*	ANDWF	f,d,a	AND WREG with f	WREG .AND. $f \rightarrow dest$
6Bf*	CLRF	f,a	Clear f	$0 \rightarrow f$
1Ff*	COMF	f,d,a	Complement f	.NOT. $f \rightarrow dest$
63f*	CPFSEQ	f,a	Compare f with WREG, skip if f=WREG	f-WREG, if f=WREG, PC+4 $\rightarrow$ PC else PC+2 $\rightarrow$ PC
65f*	CPFSGT	тэ	Compare f with WREG, skip if f > WREG	f-WREG, if f > WREG, PC+4 $\rightarrow$ PC else PC+2 $\rightarrow$ PC
61f*	CPFSLT	f,a	Compare f with WREG, skip if f < WREG	f-WREG, if $f < WREG$ , $PC+4 \rightarrow PC$ else $PC+2 \rightarrow PC$
07f*	DECF	f,d,a	Decrement f	$f-1 \rightarrow dest$
2Ff*	DECFSZ	f,d,a	Decrement f, skip if 0	f-1 $\rightarrow$ dest, if dest=0, PC+4 $\rightarrow$ PC else PC+2 $\rightarrow$ PC
4Ff*	DCFSNZ	f,d,a	Decrement f, skip if not 0	f-1 → dest, if dest ¼ 0, PC+4 → PC else PC+2 → PC
2Bf*	INCF	f,d,a	Increment f	$f+1 \rightarrow dest$
3Ff*	INCFSZ	f,d,a	Increment f, skip if 0	f+1 → dest, if dest=0, PC+4 → PC else PC+2 → PC
4Bf*	INFSNZ	f,d,a	Increment f, skip if not 0	f+1 $\rightarrow$ dest, if dest Â <sup>1</sup> /4 0, PC+4 $\rightarrow$ PC else PC+2 $\rightarrow$ PC
13f*	IORWF	f,d,a	Inclusive OR WREG with f	WREG .OR. $f \rightarrow dest$
53f*	MOVF	f,d,a	Move f	$f \rightarrow dest$
Cf' Ff"	MOVFF	f',f"	Move f' to fd" (second word)	f' → f"
6Ff*	MOVWF	f,a	Move WREG to f	WREG $\rightarrow$ f
03f*	MULWF	f,a	Multiply WREG with f	WREG * f → PRODH:PRODL

TABLE: EXTENDED 16-BIT CORE BYTE-ORIENTED REGISTER OPERATIONS

6Df*	NEGF	f,a	Negate f	$-f \rightarrow f$			
37f*	RLCF	f,d,a	Rotate left f through Carry	register f			
47f*	RLNCF	f,d,a	Rotate left f (no carry)	register f 70 ◀			
33f*	RRCF	f,d,a	Rotate right f through Carry	register f			
43f*	RRNCF	f,d,a	Rotate right f (no carry)	register f			
69f*	SETF	f,a	Set f	$0xFF \rightarrow f$			
57f*	SUBFWB	f,d,a	Subtract f from WREG with Borrow	WREG-f-C → dest			
5Ff*	SUBWF	f,d,a	Subtract WREG from f	f-WREG → dest			
5Bf*	SUBWFB	f,d,a	Subtract WREG from f with Borrow	f-WREG-C → dest			
3Bf*	SWAPF	f,d,a	Swap nibbles of f	f<3:0> → dest<7:4>, f<7:4> → dest<3:0>			
67f*	TSTFSZ	f,a	Test f, skip if 0	PC+4 $\rightarrow$ PC, if f=0, else PC+2 $\rightarrow$ PC			
1Bf*	LBf* XORWF f, d, a Exclusive OR WREG with f WREG.XOR. $f \rightarrow dest$						
* Assu	ming defa	ult bit va	lues for d and a.				

#### TABLE: EXTENDED 16-BIT CORE BIT-ORIENTED REGISTER OPERATIONS

Hex	Mnemonic		Description	Function				
91f*	BCF	f,b,a	Bit Clear f	$0 \rightarrow f \le b >$				
81f*	BSF	f,b,a	Bit Set f	$1 \rightarrow f \le b >$				
B1f*	BTFSC	f,b,a	Bit test f, skip if clear	if f <b>=0, PC+4 <math>\rightarrow</math> PC, else PC+2 <math>\rightarrow</math> PC</b>				
Alf*	BTFSS	f,b,a	Bit test f, skip if set	if f <b>=1, PC+4 <math>\rightarrow</math> PC, else PC+2 <math>\rightarrow</math> PC</b>				
71f* BTG f, b, a Bit Toggle f			Bit Toggle f	$f < b > \rightarrow f < b >$				
* Assu	* Assuming b = 0 and default bit value for a.							

#### TABLE: EXTENDED 16-BIT CORE CONTROL OPERATIONS

Hex	Mnemonic		Description	Function			
E2n	BC n		Branch if Carry	if C=1, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC			
E6n	BN n Branch if Negative		Branch if Negative	if N=1, PC+2+2*n $\rightarrow$ PC,else PC+2 $\rightarrow$ PC			
E3n	BNC	BNC n Branch if Not Carry		if C=0, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC			
E7n	BNN n Branch if Not Negative		Branch if Not Negative	if N=0, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC			
E5n	BNOV	n	Branch if Not Overflow	if OV=0, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC			

E1n	BNZ	n	Branch if Not Zero	if Z=0, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC
E4n	BOV	n	Branch if Overflow	if OV=1, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC
D'0'n	BRA	n	Branch Unconditionally	$PC+2+2*n \rightarrow PC$
E0n	BZ	n	Branch if Zero	if Z=1, PC+2+2*n $\rightarrow$ PC, else PC+2 $\rightarrow$ PC
ECkk* Fkkk	CALL	n,s	Call Subroutine 1st word 2nd word	PC+4 → TOS, n → PC<20:1>, if s=1, WREG → WREGs, STATUS → STATUSs, BSR → BSRs
0004	CLRWDT		Clear Watchdog Timer	$\begin{array}{l} 0 \rightarrow \text{WDT, } 0 \rightarrow \text{WDT postscaler,} \\ 1 \rightarrow \overline{\text{TO}}, 1 \rightarrow \overline{\text{PD}} \end{array}$
0007	DAW		Decimal Adjust WREG	if WREG<3:0> >9 or DC=1, WREG<3:0>+6 $\rightarrow$ WREG<3:0>, else WREG<3:0> $\rightarrow$ WREG<3:0>; if WREG<7:4> >9 or C=1, WREG<7:4>+6 $\rightarrow$ WREG<7:4>, else WREG<7:4> $\rightarrow$ WREG<7:4>;
EFkk Fkkk	бото	n	Go to address 1st word 2nd word	n → PC<20:1>
0000	NOP		No Operation	No Operation
Fxxx	NOP		No Operation	No Operation (2-word instructions)
0006	POP		Pop top of return stack (TOS)	TOS-1 → TOS
0005	PUSH		Push top of return stack (TOS)	$PC + 2 \rightarrow TOS$
D'1'n	RCALL	n	Relative Call	PC+2 → TOS, PC+2+2*n → PC
00FF	RESET		Software device reset	Same as MCLR reset
0010*	RETFIE	s	Return from interrupt (and enable interrupts)	TOS $\rightarrow$ PC, 1 $\rightarrow$ GIE/GIEH or PEIE/GIEL, if s=1, WREGs $\rightarrow$ WREG, STATUSs $\rightarrow$ STATUS, BSRs $\rightarrow$ BSR, PCLATU/PCLATH unchngd.
0012*	RETURN	s	Return from subroutine	TOS $\rightarrow$ PC, if s=1, WREGs $\rightarrow$ WREG, STATUSs $\rightarrow$ STATUS, BSRs $\rightarrow$ BSR, PCLATU/PCLATH are unchanged
0003	SLEEP		Enter SLEEP Mode	$0 \rightarrow \text{WDT}, 0 \rightarrow \text{WDT}$ postscaler, $1 \rightarrow \overline{\text{TO}}, 0 \rightarrow \overline{\text{PD}}$
* Assun	ning defau	lt bit v	value for s.	~

#### TABLE: EXTENDED 16-BIT CORE LITERAL OPERATIONS

Hex	Mnemonic		Description	Function					
0Fkk	ADDLW	kk	Add literal to WREG	WREG+kk → WREG					
0Bkk	ANDLW	kk	AND literal with WREG	WREG .AND. $kk \rightarrow WREG$					
	IORLW	kk	Inclusive OR literal with WREG	WREG .OR. $kk \rightarrow WREG$					
EErk F0kk	LFSR	r,kk	Move literal (12 bit) 2nd word to FSRr 1st word	kk → FSRr					
010k	MOVLB	k	Move literal to BSR<3:0>	kk → BSR					
0Ekk	MOVLW kk		Move literal to WREG	$kk \rightarrow WREG$					

0Dkk	MULLW	kk	Multiply literal with WREG	WREG * kk $\rightarrow$ PRODH:PRODL
0Ckk	RETLW	kk	Return with literal in WREG	kk → WREG
08kk	SUBLW	kk	Subtract WREG from literal	kk-WREG → WREG
0Akk	XORLW	kk	Exclusive OR literal with WREG	WREG .XOR. $kk \rightarrow WREG$

Hex	Mnemonic	Description	Function
0008	TBLRD*	Table Read	Prog Mem (TBLPTR) $\rightarrow$ TABLAT
0009	TBLRD*+	Table Read with post-increment	Prog Mem (TBLPTR) → TABLAT TBLPTR +1 → TBLPTR
000A	TBLRD*-	$12 \text{ DIA} R A277 WITH DOST_70Cromont$	Prog Mem (TBLPTR) → TABLAT TBLPTR -1 → TBLPTR
000B	TBLRD+*	Lable Read with pro-increment	TBLPTR +1 → TBLPTR Prog Mem (TBLPTR) → TABLAT
000C	TBLWT*	Table Write	TABLAT $\rightarrow$ Prog Mem(TBLPTR)
000D	TBLWT*+	Table Write with post-increment	TABLAT $\rightarrow$ Prog Mem(TBLPTR) TBLPTR +1 $\rightarrow$ TBLPTR
000E	TBLWT*-	Table Write with post-decrement	TABLAT $\rightarrow$ Prog Mem(TBLPTR) TBLPTR -1 $\rightarrow$ TBLPTR
000F	TBLWT+*	I able Write with pre-increment	TBLPTR $+1 \rightarrow$ TBLPTR TABLAT $\rightarrow$ Prog Mem(TBLPTR)



# **Useful Tables**

Some useful tables are included for reference here.

- <u>ASCII Character Set</u>
- Hexadecimal to Decimal Conversion



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## **ASCII Character Set**

.

		Мо	st Siç	gnificar	nt I	Nib	ble		
	HEX	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	Ρ	Ì	р
	1	SOH	DC1	!	1	А	Q	a	q
	2	STX	DC2	"	2	В	R	b	r
	3	ETX	DC3	#	3	С	S	с	S
	4	EOT	DC4	\$	4	D	Т	d	t
	5	ENQ	NAK	%	5	Е	U	е	u
Least Significant Nibble	6	ACK	SYN	&	6	F	V	f	v
Least Significant Nibble	7	Bell	ЕТВ	1	7	G	W	g	w
	8	BS	CAN	(	8	Н	Х	h	х
	9	HT	EM	)	9	Ι	Y	i	У
	Α	LF	SUB	*	Ŀ	J	Ζ	j	z
	В	VT	ESC	+	•	К	]	k	{
	С	FF	FS	,	<	L	١	1	
	D	CR	GS	-	=	Μ	]	m	}
	E	SO	RS		>	Ν	^	n	~
	F	SI	US	/	?	0		0	DEL

-



### **Hexadecimal to Decimal Conversion**

This appendix describes how to convert hexadecimal to decimal. For each HEX digit, find the associated decimal value. Add the numbers together.

	L	_ow I	Byte				
HEX 1000	Dec	HEX 100	Dec	HEX 10	Dec	HEX 1	Dec
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
А	40960	А	2560	A	160	A	10
В	45056	В	2816	В	176	В	11
С	49152	С	3072	С	192	С	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

#### For example, HEX A38F converts to 41871 as follows:

HEX 1000's Digit	HEX 100's Digit	HEX 10's Digit	HEX 1's Digit	Result
40960	768	128	15	41871 Decimal



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210

Microchip's E-mail Address

# Support

The general information contained here will be useful to know when working with Microchip Technology's MPASM assembler (the assembler), MPLINK object linker (the linker), and MPLIB object librarian (the librarian).

- <u>Recommended Reading</u>
- The Microchip Web Site
- Development Systems Customer Notification Service
- <u>Customer Support</u>



## **Recommended Reading**

Other useful documents are listed below.

#### Readme File - readme.asm

For the latest information on using MPASM assembler, read the readme.asm file (an ASCII text file) in the MPLAB IDE directory. The README file contains update information and known issues that may not be included in the user's guide or the on-line help file.

#### Readme File - readme.lkr

For the latest information on using MPLINK linker and MPLIB librarian, read the readme.lkr file (an ASCII text file) in the MPLAB IDE directory. The README file contains update information and known issues that may not be included in the user's guide or the on-line help file.

#### MPASM Assembler, MPLINK Object Linker, and MPLIB Object Librarian User's Guide (DS00000)

This user's guide describes how to use the Microchip PICmicro MCU MPASM assembler, MPLINK object linker and MPLIB object librarian.

# MPASM and MPLINK PICmicro Quick Reference Card (DS30400)

A quick reference card (QRC) containing MPASM assembler directive language summary, MPASM assembler radix types supported, MPLINK object linker command line options, MPLIB object librarian usage format and examples, PIC18CXXX core special function register files, ASCII character set, and PICmicro MCU instruction set summaries.

#### Microchip Technical Library CD-ROM (DS00161)

This CD-ROM contains comprehensive application notes, data sheets, and technical briefs for all of Microchip products. To obtain this CD-ROM, contact the nearest Microchip Sales and Service location (see back page).

#### Embedded Control Handbook Vol.1 & 2 and the Embedded Control Handbook Update 2000 (DS00092, DS00167, and DS00711)

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

The application notes described in these manuals are also obtainable from Microchip sales and service locations or from the Microchip website (http://www.microchip.com).

#### Microsoft<sup>®</sup> Windows<sup>®</sup> Manuals

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



## **The Microchip Web Site**

Microchip provides online support on the Microchip World Wide Web (WWW) site. The website is used by Microchip as a means to make files and information easily available to customers. To view the site, you must have access to the Internet and a web browser such as Netscape Navigator or Microsoft Internet Explorer.

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

http://www.microchip.com

The web site provides a variety of services. Users may download files for the latest development tools, data sheets, application notes, user's guides, articles, and sample programs. A variety information specific to the business of Microchip is also available, including listings of Microchip sales offices, distributors and factory representatives.

#### **Technical Support**

- Frequently Asked Questions (FAQ)
- Online Discussion Groups Conferences for products, Development Systems, technical information and more
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip products

#### **Developer's Toolbox**

- Design Tips
- Device Errata

#### Other available information

- Latest Microchip Press Releases
- Listing of seminars and events
- Job Postings



### **Development Systems Customer Notification Service**

Microchip started the customer notification service to help our customers keep current on Microchip products with the least amount of effort. Once you subscribe, you will receive email notification whenever we change, update, revise or have errata related to your specified product family or development tool of interest.

Go to the Microchip WWW web page (http://www.microchip.com) and click on Customer Change Notification under Items of Interest. Follow the instructions to register.

The Development Systems product group categories are:

- Compilers
- Emulators
- In-Circuit Debuggers
- MPLAB
- Programmers

Here is a description of these categories:

COMPILERS - The latest information on Microchip C compilers and other language tools. These include the MPLAB C17, MPLAB C18 and MPLAB C30 C compilers; MPASM and MPLAB ASM30 assemblers; MPLINK and MPLAB LINK30 object linkers; and MPLIB and MPLAB LIB30 object librarians.

EMULATORS - The latest information on Microchip in-circuit emulators.This includes the MPLAB ICE 2000 and MPLAB ICE 4000. IN-CIRCUIT DEBUGGERS - The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD and MPLAB ICD 2.

MPLAB - The latest information on Microchip MPLAB IDE, the Windows Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM and MPLAB SIM30 simulators, MPLAB IDE Project Manager and general editing and debugging features.

PROGRAMMERS - The latest information on Microchip device programmers. These include the PRO MATE II device programmer and PICSTART Plus development programmer.



### **Customer Support**

Users of Microchip products can receive assistance through several channels:

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

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

Corporate Applications Engineers (CAEs) may be contacted at (480) 792-7627.

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

The Hotline Numbers are:

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

1-480-792-7302 for the rest of the world.



# Glossary

#### **Absolute Section**

A section with a fixed (absolute) address that cannot be changed by the linker.

#### Access Memory (PIC18 Only)

Special registers on PIC18XXXXX devices that allow access regardless of the setting of the bank select register (BSR).

#### Address

Value that identifies a location in memory.

#### **Alphabetic Character**

Alphabetic characters are those characters that are letters of the arabic alphabet (a, b, ..., z, A, B, ..., Z).

#### Alphanumeric

Alphanumeric characters are comprised of alphabetic characters and decimal digits (0,1, ..., 9).

#### **Anonymous Structure**

An unnamed structure that is a member of a C union. The members of an anonymous structure may be accessed as if they were members of the enclosing union. For example, in the following code, hi and lo are members of an anonymous structure inside the union caster.

```
union castaway
int intval;
```

```
struct {
  char lo; //accessible as caster.lo
  char hi; //accessible as caster.hi
  };
  } caster;
ANSI
```

American National Standards Institute is an organization responsible for formulating and approving standards in the United States.

#### Application

A set of software and hardware that may be controlled by a PICmicro microcontroller.

#### Archive

A collection of relocatable object modules. It is created by assembling multiple source files to object files, and then using the archiver to combine the object files into one library file. A library can be linked with object modules and other libraries to create executable code.

#### Archiver

A tool that creates and manipulates libraries.

### ASCII

American Standard Code for Information Interchange is a character set encoding that uses 7 binary digits to represent each character. It includes upper and lower case letters, digits, symbols and control characters.

### Assembler

A language tool that translates assembly language source code

into machine code.

#### Assembly Language

A programming language that describes binary machine code in a symbolic form.

#### **Assigned Section**

A section which has been assigned to a target memory block in the linker command file.

#### **Asynchronous Events**

Multiple events that do not occur at the same time. This is generally used to refer to interrupts that may occur at any time during processor execution.

#### **Asynchronous Stimulus**

Data generated to simulate external inputs to a simulator device.

#### Binary

The base two numbering system that uses the digits 0-1. The rightmost digit counts ones, the next counts multiples of 2, then  $2^2 = 4$ , etc.

#### Breakpoint, Hardware

An event whose execution will cause a halt.

#### Breakpoint, Software

An address where execution of the firmware will halt. Usually achieved by a special break instruction.

#### Build

Compile and link all the source files for an application.

#### С

A general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators.

#### **Calibration Memory**

A special function register or registers used to hold values for calibration of a PICmicro microcontroller on-board RC oscillator or other device peripherals.

#### **Central Processing Unit**

The part of a device that is responsible for fetching the correct instruction for execution, decoding that instruction, and then executing that instruction. When necessary, it works in conjunction with the arithmetic logic unit (ALU) to complete the execution of the instruction. It controls the program memory address bus, the data memory address bus, and accesses to the stack.

#### COFF

Common Object File Format. An object file of this format contains machine code, debugging and other information.

#### **Command Line Interface**

A means of communication between a program and its user based solely on textual input and output.

#### Compiler

A program that translates a source file written in a high-level language into machine code.

#### **Conditional Compilation**

The act of compiling a program fragment only if a certain constant expression, specified by a preprocessor directive, is true.

#### **Configuration Bits**

Special-purpose bits programmed to set PICmicro microcontroller modes of operation. A configuration bit may or may not be preprogrammed.

#### **Control Directives**

Directives in assembly language code that cause code to be included or omitted based on the assembly-time value of a specified expression.

### CPU

See Central Processing Unit.

#### **Cross Reference File**

A file that references a table of symbols and a list of files that references the symbol. If the symbol is defined, the first file listed is the location of the definition. The remaining files contain references to the symbol.

#### **Data Directives**

Data directives are those that control the assembler's allocation of program or data memory and provide a way to refer to data items symbolically; that is, by meaningful names.

#### **Data Memory**

On Microchip MCU and DSC devices, data memory (RAM) is comprised of general purpose registers (GPRs) and special function registers (SFRs). Some devices also have EEPROM data memory.

#### **Device Programmer**

A tool used to program electrically programmable semiconductor devices such as microcontrollers.

#### Directives

Statements in source code that provide control of the language tool's operation.

#### Download

Download is the process of sending data from a host to another device, such as an emulator, programmer or target board.

#### EEPROM

Electrically Erasable Programmable Read Only Memory. A special type of PROM that can be erased electrically. Data is written or erased one byte at a time. EEPROM retains its contents even when power is turned off.

#### Emulation

The process of executing software loaded into emulation memory as if it were firmware residing on a microcontroller device.

#### **Emulation Memory**

Program memory contained within the emulator.

#### Emulator

Hardware that performs emulation.

#### **Emulator System**

The MPLAB ICE 2000 and 4000 emulator systems include the pod, processor module, device adapter, cables, and MPLAB IDE software.

#### Endianess

Describes order of bytes in a multi-byte object.

#### EPROM

Erasable Programmable Read Only Memory. A programmable read-only memory that can be erased usually by exposure to ultraviolet radiation.

### **Error File**

A file containing error messages and diagnostics generated by a language tool.

#### Event

A description of a bus cycle which may include address, data, pass count, external input, cycle type (fetch, R/W), and time stamp. Events are used to describe triggers, breakpoints and interrupts.

### Export

Send data out of the MPLAB IDE in a standardized format.

#### **Extended Microcontroller Mode**

In extended microcontroller mode, on-chip program memory as well as external memory is available. Execution automatically switches to external if the program memory address is greater than the internal memory space of the PIC17CXXX or PIC18CXXX device.

#### **External Label**

A label that has external linkage.

#### **External Linkage**

A function or variable has external linkage if it can be referenced from outside the module in which it is defined.

#### **External Symbol**

A symbol for an identifier which has external linkage. This may be a reference or a definition.

#### **External Symbol Resolution**

A process performed by the linker in which external symbol definitions from all input modules are collected in an attempt to resolve all external symbol references. Any external symbol references which do not have a corresponding definition cause a linker error to be reported.

#### **External Input Line**

An external input signal logic probe line (TRIGIN) for setting an event based upon external signals.

#### **External RAM**

Off-chip Read/Write memory.

#### **File Registers**

On-chip data memory, including general purpose registers (GPRs) and special function registers (SFRs).

#### Flash

A type of EEPROM where data is written or erased in blocks instead of bytes.

#### FNOP

Forced No Operation. A forced NOP cycle is the second cycle of a two-cycle instruction. Since the PICmicro microcontroller architecture is pipelined, it prefetches the next instruction in the physical address space while it is executing the current instruction. However, if the current instruction changes the program counter, this prefetched instruction is explicitly ignored, causing a forced NOP cycle.

#### **Frame Pointer**

A pointer that references the location on the stack that separates the stack-based arguments from the stack-based local variables. Provides a convenient base from which to access local variables and other values for the current function.

#### **Free-Standing**

A C compiler implementation that accepts any strictly conforming program that does not use complex types and in which the use of the features specified in the ISO library clause is confined to the contents of the standard headers <float.h>, <iso646.h>, <limits.h>, <stddef.h>, and <stdint.h>.

#### GPR

General Purpose Register. The portion of device data memory (RAM) avaliable for general use.

#### Halt

A stop of program execution. Executing Halt is the same as stopping at a breakpoint.

### HEX Code

Executable instructions stored in a hexadecimal format code. HEX

code is contained in a HEX file.

#### **HEX File**

An ASCII file containing hexadecimal addresses and values (HEX code) suitable for programming a device.

#### Hexadecimal

The base 16 numbering system that uses the digits 0-9 plus the letters A-F (or a-f). The digits A-F represent hexadecimal digits with values of (decimal) 10 to 15. The right-most digit counts ones, the next counts multiples of 16, then  $16^2 = 256$ , etc.

#### **High Level Language**

A language for writing programs that is further removed from the processor than assembly.

#### ICD

In-Circuit Debugger. MPLAB ICD and MPLAB ICD 2 are Microchip's in-circuit debuggers for PIC16F87X and PIC18FXXX devices, respectively. These ICDs work with MPLAB IDE.

#### ICE

In-Circuit Emulator. MPLAB ICE 2000 and 4000 are Microchip's incircuit emulators that work with MPLAB IDE.

#### IDE

Integrated Development Environment. MPLAB IDE is Microchip's integrated development environment.

#### IEEE

Institute of Electrical and Electronics Engineers.

#### Import

Bring data into the MPLAB IDE from an outside source, such as from a HEX file.

#### **Instruction Set**

The collection of machine language instructions that a particular processor understands.

#### Instructions

A sequence of bits that tells a central processing unit to perform a particular operation and can contain data to be used in the operation.

#### Internal Linkage

A function or variable has internal linkage if it can not be accessed from outside the module in which it is defined.

#### International Organization for Standardization

An organization that sets standards in many businesses and technologies, including computing and communications.

#### Interrupt

A signal to the CPU that suspends the execution of a running application and transfers control to an Interrupt Service Routine (ISR) so that the event may be processed.

#### **Interrupt Handler**

A routine that processes special code when an interrupt occurs.

#### **Interrupt Request**

An event which causes the processor to temporarily suspend

normal instruction execution and to start executing an interrupt handler routine. Some processors have several interrupt request events allowing different priority interrupts.

#### **Interrupt Service Routine**

A function that is invoked when an interrupt occurs.

#### **Interrupt Service Routine**

User-generated code that is entered when an interrupt occurs. The location of the code in program memory will usually depend on the type of interrupt that has occurred.

### IRQ

See Interrupt Request.

#### ISO

See International Organization for Standardization.

#### ISR

See Interrupt Service Routine.

#### Latency

The time between an event and its response.

#### Librarian

See Archiver.

#### Library

See Archive.

#### Linker

A language tool that combines object files and libraries to create executable code, resolving references from one module to another.

#### **Linker Script Files**

Linker script files are the command files of a linker. They define linker options and describe available memory on the target platform.

#### **Listing Directives**

Listing directives are those directives that control the assembler listing file format. They allow the specification of titles, pagination and other listing control.

#### **Listing File**

A listing file is an ASCII text file that shows the machine code generated for each C source statement, assembly instruction, assembler directive, or macro encountered in a source file.

#### **Little Endianess**

A data ordering scheme for multibyte data whereby the least significant byte is stored at the lower addresses.

#### Local Label

A local label is one that is defined inside a macro with the LOCAL directive. These labels are particular to a given instance of a macro's instantiation. In other words, the symbols and labels that are declared as local are no longer accessible after the ENDM macro is encountered.

#### **Logic Probes**

Up to 14 logic probes can be connected to some Microchip emulators. The logic probes provide external trace inputs, trigger

output signal, +5V, and a common ground.

### **Machine Code**

The representation of a computer program that is actually read and interpreted by the processor. A program in binary machine code consists of a sequence of machine instructions (possibly interspersed with data). The collection of all possible instructions for a particular processor is known as its "instruction set".

#### Machine Language

A set of instructions for a specific central processing unit, designed to be usable by a processor without being translated.

#### Macro

Macroinstruction. An instruction that represents a sequence of instructions in abbreviated form.

#### **Macro Directives**

Directives that control the execution and data allocation within macro body definitions.

#### **Make Project**

A command that rebuilds an application, re-compiling only those source files that have changed since the last complete compilation.

#### MCU

Microcontroller Unit. An abbreviation for microcontroller. Also uC.

#### **Memory Models**

Versions of libraries and/or precompiled object files based on a device's memory (RAM/ROM) size and structure.

#### **Memory Models**

A description that specifies the size of pointers that point to program memory.

#### Message

Text displayed to alert you to potential problems in language tool operation. A message will not stop operation.

#### Microcontroller

A highly integrated chip that contains a CPU, RAM, program memory, I/O ports, and timers.

#### **Microcontroller Mode**

One of the possible program memory configurations of the PIC17CXXX and PIC18CXXX families of microcontrollers. In microcontroller mode, only internal execution is allowed. Thus, only the on-chip program memory is available in microcontroller mode.

#### **Microprocessor Mode**

One of the possible program memory configurations of the PIC17CXXX and PIC18CXXX families of microcontrollers. In microprocessor mode, the on-chip program memory is not used. The entire program memory is mapped externally.

#### **Mnemonics**

Text instructions that can be translated directly into machine code. Also referred to as Opcodes.

#### **MPASM Assembler**

Microchip Technology's relocatable macro assembler for PICmicro microcontroller devices, KeeLoq devices and Microchip memory

devices.

#### MPLAB ASM30

Microchip's relocatable macro assembler for dsPIC30F digitial signal controller devices.

#### MPLAB C1X

Refers to both the MPLAB C17 and MPLAB C18 C compilers from Microchip. MPLAB C17 is the C compiler for PIC17CXXX devices and MPLAB C18 is the C compiler for PIC18CXXX and PIC18FXXXX devices.

#### MPLAB C30

Microchip's C compiler for dsPIC30F digitial signal controller devices.

#### MPLAB ICD 2

Microchip's in-circuit debugger for PIC16F87X, PIC18FXXX and dsPIC30FXXXX devices. The ICD works with MPLAB IDE. The main component of each ICD is the module. A complete system consists of a module, header, demo board, cables, and MPLAB IDE Software.

#### MPLAB ICE 2000

Microchip's in-circuit emulator for PICmicro MCU's that works with MPLAB IDE.

#### MPLAB ICE 4000

Microchip's in-circuit emulator for dsPIC DSC's that works with MPLAB IDE.

#### MPLAB IDE

Microchip's Integrated Development Environment.

### MPLAB LIB30

MPLAB LIB30 archiver/librarian is an object librarian for use with COFF object modules created using either MPLAB ASM30 or MPLAB C30 C compiler.

#### MPLAB LINK30

MPLAB LINK30 is an object linker for the Microchip MPLAB ASM30 assembler and the Microchip MPLAB C30 C compiler.

#### MPLAB SIM

Microchip's simulator that works with MPLAB IDE in support of PICmicro MCU devices.

#### MPLAB SIM30

Microchip's simulator that works with MPLAB IDE in support of dsPIC DSC devices.

#### **MPLIB Object Librarian**

MPLIB librarian is an object librarian for use with COFF object modules created using either MPASM assembler (mpasm or mpasmwin v2.0) or MPLAB C1X C compilers.

#### **MPLINK Object Linker**

MPLINK linker is an object linker for the Microchip MPASM assembler and the Microchip MPLAB C17 or C18 C compilers. MPLINK linker also may be used with the Microchip MPLIB librarian. MPLINK linker is designed to be used with MPLAB IDE, though it does not have to be.

#### MRU

Most Recently Used. Refers to files and windows available to be selected from MPLAB IDE main pull down menus.

#### **Nesting Depth**

The maximum level to which macros can include other macros.

#### Node

MPLAB IDE project component.

#### Non Real-Time

Refers to the processor at a breakpoint or executing single step instructions or MPLAB IDE being run in simulator mode.

#### Non-Volatile Storage

A storage device whose contents are preserved when its power is off.

#### NOP

No Operation. An instruction that has no effect when executed except to advance the program counter.

#### **Object Code**

The machine code generated by an assembler or compiler.

#### **Object File**

A file containing machine code and possibly debug information. It may be immediately executable or it may be relocatable, requiring linking with other object files, e.g. libraries, to produce a complete executable program.

#### **Object File Directives**

Directives that are used only when creating an object file.

## Octal

The base 8 number system that only uses the digits 0-7. The rightmost digit counts ones, the next digit counts multiples of 8, then  $8^2 = 64$ , etc.

## **Off-Chip Memory**

Off-chip memory refers to the memory selection option for the PIC17CXXX or PIC18CXXX device where memory may reside on the target board, or where all program memory may be supplied by the Emulator. The Memory tab accessed from Options > Development Mode provides the Off-Chip Memory selection dialog box.

## Opcodes

Operational Codes. See Mnemonics.

### Operators

Symbols, like the plus sign `+' and the minus sign `-', that are used when forming well-defined expressions. Each operator has an assigned precedence that is used to determine order of evaluation.

## ΟΤΡ

One Time Programmable. EPROM devices that are not in windowed packages. Since EPROM needs ultraviolet light to erase its memory, only windowed devices are erasable.

## Pass Counter

A counter that decrements each time an event (such as the execution of an instruction at a particular address) occurs. When the pass count value reaches zero, the event is satisfied. You can

assign the Pass Counter to break and trace logic, and to any sequential event in the complex trigger dialog.

## PC

Personal Computer or Program Counter.

#### PC Host

Any IBM<sup>™</sup> or compatible personal computer running a supported Windows operating system.

#### **PICmicro MCUs**

PICmicro microcontrollers (MCUs) refers to all Microchip microcontroller families.

#### **PICSTART Plus**

A developmental device programmer from Microchip. Programs 8-, 14-, 28-, and 40-pin PICmicro microcontrollers. Must be used with MPLAB IDE Software.

#### Pod, Emulator

The external emulator box that contains emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic.

#### **Power-on-Reset Emulation**

A software randomization process that writes random values in data RAM areas to simulate uninitialized values in RAM upon initial power application.

#### Pragma

A directive that has meaning to a specific compiler. Often a pragma is used to convey implementation-defined information to the compiler. MPLAB C30 uses attributes to convey this information.

## PRO MATE II

A device programmer from Microchip. Programs all PICmicro microcontrollers and most memory and Keeloq devices. Can be used with MPLAB IDE or stand-alone.

#### **Program Counter**

The location that contains the address of the instruction that is currently executing.

#### **Program Memory**

The memory area in a device where instructions are stored. Also, the memory in the emulator or simulator containing the downloaded target application firmware.

## Project

A set of source files and instructions to build the object and executable code for an application.

#### **Prototype System**

A term referring to a user's target application, or target board.

### **PWM Signals**

Pulse Width Modulation Signals. Certain PICmicro MCU devices have a PWM peripheral.

## Qualifier

An address or an address range used by the Pass Counter or as an event before another operation in a complex trigger.

### Radix

The number base, HEX, or decimal, used in specifying an address.

### RAM

Random Access Memory (Data Memory). Memory in which information can be accessed in any order.

#### Raw Data

The binary representation of code or data associated with a section.

### **Real-Time**

When released from the halt state in the emulator or MPLAB ICD mode, the processor runs in real-time mode and behaves exactly as the normal chip would behave. In real-time mode, the real-time trace buffer of MPLAB ICE is enabled and constantly captures all selected cycles, and all break logic is enabled. In the emulator or MPLAB ICD, the processor executes in real-time until a valid breakpoint causes a halt, or until the user halts the emulator. In the simulator real-time simply means execution of the microcontroller instructions as fast as they can be simulated by the host CPU.

### **Recursive Calls**

A function that calls itself, either directly or indirectly.

## Recursion

The concept that a function or 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.

### Reentrant

A function that may have multiple, simultaneously active instances.

This may happen due to either direct or indirect recursion or through execution during interrupt processing.

#### Relocatable

An object file whose sections have not been assigned to a fixed location in memory.

#### ROM

Read Only Memory (Program Memory). Memory that cannot be modified.

#### Run

The command that releases the emulator from halt, allowing it to run the application code and change or respond to I/O in real time.

#### **Runtime Model**

Describes the use of target architecture resources.

#### Section

A named sequence of code or data.

#### **Section Attribute**

A characteristic ascribed to a section (e.g., an access section).

#### SFR

See Special Function Registers.

### Shell

The MPASM assembler shell is a prompted input interface to the macro assembler. There are two MPASM assembler shells: one for the DOS version and one for the Windows version.

## Simulator

A software program that models the operation of devices.

## Single Step

This command steps though code, one instruction at a time. After each instruction, MPLAB IDE updates register windows, watch variables, and status displays so you can analyze and debug instruction execution. You can also single step C compiler source code, but instead of executing single instructions, MPLAB IDE will execute all assembly level instructions generated by the line of the high level C statement.

## Skew

The information associated with the execution of an instruction appears on the processor bus at different times. For example, the executed Opcodes appears on the bus as a fetch during the execution of the previous instruction, the source data address and value and the destination data address appear when the Opcodes is actually executed, and the destination data value appears when the next instruction is executed. The trace buffer captures the information that is on the bus at one instance. Therefore, one trace buffer entry will contain execution information for three instructions. The number of captured cycles from one piece of information to another for a single instruction execution is referred to as the skew.

## Skid

When a hardware breakpoint is used to halt the processor, one or more additional instructions may be executed before the processor halts. The number of extra instructions executed after the intended breakpoint is referred to as the skid.

## Source Code

The form in which a computer program is written by the

programmer. Source code is written in some formal programming language which can be translated into or machine code or executed by an interpreter.

#### Source File

An ASCII text file containing source code.

#### **Special Function Registers**

The portion of data memory (RAM) dedicated to registers that control I/O processor functions, I/O status, timers, or other modes or peripherals.

#### Stack, Hardware

Locations in PICmicro microcontroller where the return address is stored when a function call is made.

#### Stack, Software

Memory used by an application for storing return addresses, function parameters, and local variables. This memory is typically managed by the compiler when developing code in a high-level language.

#### Static RAM or SRAM

Static Random Access Memory. Program memory you can Read/Write on the target board that does not need refreshing frequently.

#### **Status Bar**

The Status Bar is located on the bottom of the MPLAB IDE window and indicates such current information as cursor position, development mode and device, and active tool bar.

#### Step Into

This command is the same as Single Step. Step Into (as opposed to Step Over) follows a CALL instruction into a subroutine.

## Step Over

Step Over allows you to debug code without stepping into subroutines. When stepping over a CALL instruction, the next breakpoint will be set at the instruction after the CALL. If for some reason the subroutine gets into an endless loop or does not return properly, the next breakpoint will never be reached. The Step Over command is the same as Single Step except for its handling of CALL instructions.

## Stimulus

Input to the simulator, i.e., data generated to exercise the response of simulation to external signals. Often the data is put into the form of a list of actions in a text file. Stimulus may be asynchronous, synchronous (pin), clocked and register.

### Stopwatch

A counter for measuring execution cycles.

### **Storage Class**

Determines the lifetime of an object.

## **Storage Qualifier**

Indicates special properties of an object (e.g., volitile).

## Symbol

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include

function names, variable names, section names, file names, struct/enum/union tag names, etc. Symbols in MPLAB IDE refer mainly to variable names, function names and assembly labels. The value of a symbol after linking is its value in memory.

## **System Window Control**

The system window control is located in the upper left corner of windows and some dialogs. Clicking on this control usually pops up a menu that has the items "Minimize," "Maximize," and "Close."

## Target

Refers to user hardware.

## **Target Application**

Software residing on the target board.

## **Target Board**

The circuitry and programmable device that makes up the target application.

### **Target Processor**

The microcontroller device on the target application board.

### Template

Lines of text that you build for inserting into your files at a later time. The MPLAB Editor stores templates in template files.

## Tool Bar

A row or column of icons that you can click on to execute MPLAB IDE functions.

#### Trace

An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to MPLAB IDE's trace window.

### **Trace Memory**

Trace memory contained within the emulator. Trace memory is sometimes called the trace buffer.

## Trigger Output

Trigger output refers to an emulator output signal that can be generated at any address or address range, and is independent of the trace and breakpoint settings. Any number of trigger output points can be set.

## **Uninitialized Data**

Data which is defined without an initial value. In C,

#### int myVar;

defines a variable which will reside in an uninitialized data section.

### Upload

The Upload function transfers data from a tool, such as an emulator or programmer, to the host PC or from the target board to the emulator.

#### Vector

The memory locations from which an application starts execution when a specific event occurs, such as a reset or interrupt.

### Warning

An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

#### Watch Variable

A variable that you may monitor during a debugging session in a watch window.

#### Watch Window

Watch windows contain a list of watch variables that are updated at each breakpoint.

#### Watchdog Timer

A timer on a PICmicro microcontroller that resets the processor after a selectable length of time. The WDT is enabled or disabled and set up using configuration bits.

#### WDT

See Watchdog Timer.



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **List of Control Directives**

Directive	Description	Syntax	
CONSTANT	Declare Symbol Constant	<pre>constant <label> [= <expr>,,<label> [= <expr>] ]</expr></label></expr></label></pre>	
#DEFINE	Define a Text Substitution Label	#define <name> [[(<arg>,,<arg>)] <value>]</value></arg></arg></name>	
END	End Program Block	end	
EQU	Define an Assembly Constant	<label> equ <expr></expr></label>	
#INCLUDE	Include Additional Source File	include < <include_file>&gt; include " <include_file>"</include_file></include_file>	
ORG	Set Program Origin	<label> org <expr></expr></label>	
PROCESSOR	Set Processor Type	processor <processsor_type></processsor_type>	
RADIX	Specify Default Radix	radix <default_radix></default_radix>	
SET	Define an Assembler Variable	<label> set <expr></expr></label>	
#UNDEFINE	Delete a Substitution Label	#undefine <label></label>	
VARIABLE	Declare Symbol Variable	<pre>variable <label> [= <expr>,, <label> [= <expr>] ]</expr></label></expr></label></pre>	

< >

Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **Multiple Directive Example 1**

Directives highlighted in this example are:

- processor
- radix
- #include
- equ
- org
- end

## **Program Functional Description**

This program continually alternates the output on the Port B pins from 1's to 0's. Two delay routines using interrupts provide the timing for the alternating output. If LEDs were attached to Port B, they would flash (1=on, 0=off).

The type of PICmicro MCU is set using processor, and the radix is set to hexadecimal using radix. The standard header file for the processor selected is included using #include. Registers are assigned using the equ directive. Sections of code are blocked out using the org statement. Finally, the program is finished with an end.

### **Commented Code Listing**

- •\*\*\*\*\*\*
- ;\* MPASM Assembler Control Directives \*
- ;\* Example Program 1 \*
- ;\* Alternate output on Port B between \*

processor 16f877 ;Set the processor **radix** hex ;Set the radix #include <p16f877.inc> ;Include header file DTEMP equ 0x20 ;Set temp register DFLAG equ 0x21 ;Set flag register DFL0 equ 0x00 ;Set flag bit org 0x00 ;Reset Vector goto Start org 0x04 ;Interrupt Vector goto ServInt org 0x06 ;Start Program Start clrf PORTB ;Clear PortB bsf STATUS, RP0 ;Select Bank 1 clrf TRISB ;Set PortB as output bcf STATUS, RP0 ;Select Bank 0 bsf INTCON, GIE ;Enable Global Int's bsf INTCON, TOIE ;Enable Timer0 Int Loop movlw 0xFF movwf PORTB ;Set PortB call Delay1;Wait clrf PORTB ;Clear PortB bsf PCLATH,3 ;Select Page 3 bsf PCLATH,4 call Delay2; Wait bcf PCLATH,3 ;Select Page 0 bcf PCLATH,4 goto Loop ;Repeat ServInt ;Interrupt Serice Routine bsf STATUS, RP0 ;Select Bank 1 bsf OPTION REG, T0CS ;Stop Timer0 bcf STATUS, RP0 ;Select Bank 0 bcf INTCON, T0IF ;Clear overflow flag bcf DFLAG, DFL0 ;Clear flag bit

retfie

\* Delay 1 Routine - Timer0 delay loop \* \*\*\*\*\*\* Delay1 movlw 0xF0 ;Set Timer0 value movwf TMR0 ;0x00-longest delay ;0xFF-shortest delay clrf DFLAG bsf DFLAG, DFL0 ;Set flag bit bsf STATUS, RP0 ;Select Bank 1 bcf OPTION REG, T0CS ;Start Timer0 bcf STATUS, RP0 ;Select Bank 0 TLoop btfsc DFLAG, DFL0 ;Wait for overflow goto TLoop ;Timer0 0xFF->0x00 return ;\* Delay 2 Routine - Decrement delay loop \* org 0x1900 ;Page 3 Delay2 movlw 0xFF ;Set DTEMP value movwf DTEMP ;0x00-shortest delay ;0xFF-longest delay DLoop decfsz DTEMP, F goto DLoop ;End loop when DTEMP=0 return end

#### **Additional Comments**

Header Files

A header file is included in the program flow with the #include

directive.

#include <p16f877.inc> ;Include header file

Angle brackets are used to enclose the name of the file to be included, although quotes may also be used. You may specify the complete path to the included file, or let the assembler search for it. For more on search order, see the discussion of the **#include** directive ().

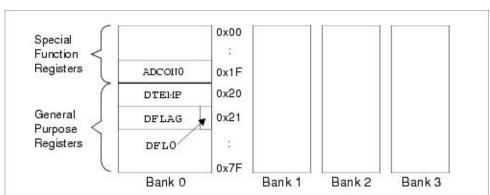
A header file is extremely useful for specifying often-used constants, such as register and pin names. This information can be typed in once, and then the file can be included in any code using the processor with those registers and pins.

**Register and Bit Assignments** 

You can specify your own registers and bits by using the equ directive, as is done in the following lines.

DTEMP equ 0x20 ;Set temp register DFLAG equ 0x21 ;Set flag register DFL0 equ 0x00 ;Set flag bit

DTEMP and DFLAG are assigned to the values  $0 \times 20$  and  $0 \times 21$  respectively. They will be used in delay loops in the program to stand for the general purpose registers (GPRs)  $0 \times 20$  and  $0 \times 21$ . DFL0 is assigned the value  $0 \times 00$  and will be used as the name for pin 0 in the DFLAG register.

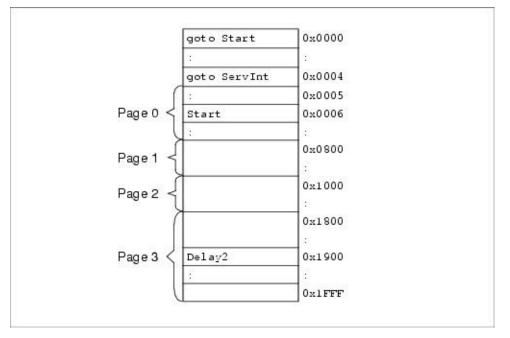


#### FIGURE: PIC16F877 REGISTER FILE MAP

Using ORG

The org directive is used to specify the program origin for specific sections of code. If no org is used, code generation begins at address zero. For Example 1, org is used to specify code at 0x00 (reset address), 0x04 (interrupt address), 0x06 (program start address) and 0x1900 (Delay2 address).

#### FIGURE: PIC16F877 PROGRAM MEMORY MAP



Most of the program is contained on page 0. However, the code for delay routine Delay2 has been placed on page 3. When calling this routine, you must remember to use the paging bits in the PCLATH to select page 3, and then use them to switch back to page 0 on the return.

bsf PCLATH,3 ;Select Page 3 bsf PCLATH,4 call Delay2 ;Wait bcf PCLATH,3 ;Select Page 0 bcf PCLATH,4



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **Multiple Directive Example 2**

Directives highlighted in this example are:

- #define
- #undefine
- equ
- constant
- variable
- set

#### **Program Functional Description**

This program performs several calculations using definded constants and variables. As in control directives - example 1, processor is used to specify the processor type, radix is used to specify the radix used, and #include is used to include a header file. See example 1 for more on these directives.

### **Commented Code Listing**

**#define** Tdistance2 25 ;Define the symbol :Tdistance2 **#undefine** Tdistance2 ;Remove Tdistance2 from ;the symbol table distance\_reg equ 0x20 ;Set up distance\_reg ;at GPR 0x20 org 0x00 ;Reset Vector goto Start org 0x06 ;Start Program Start movlw Tdistance1 ;Move value of Tdistance1 movwf distance reg; into distance reg **constant** distance1=10 ;Declare distance1 ;a constant symbol variable distance2 ;Declare distance2 ;a variable symbol distance3 set 10 ;Define a value for ;the symbol distance3

Set symbol distance3 to 10.

```
distance2=15 ;Give distance2 an
;initial value
distance2=distance1+distance2 ;Add distance1
;to distance2
distance3 set 15 ;Change value of distance3
distance2=distance2+distance3 ;Add distance3
;to distance2
movlw distance2 ;Move value of distance2
movwf distance_reg ;into distance_reg
end
```

#### **Additional Comments**

Using Watch Windows

Once the program begins, the value of Tdistance1 is placed

into distance\_reg. This can be observed in a watch window, where the value of distance\_reg will become 50. The symbol Tdistance1 will not be found in the watch window symbol list, as symbols defined using the #define directive are not available for viewing in MPLAB IDE.

The final lines of the example program write the final value of distance2 to distance\_reg. If you had a watch window open to see distance\_reg loaded with the value of 50, you will see it change to 3A. Remember that the radix is hexadecimal, so hex addition was used to determine the distance2 value.

Looking in the watch window symbol list, you will find the symbols distance1, distance2 and distance3. However, they will have no values. These symbol values are not actually stored on the PICmicro device, but implemented only in the assembler.



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **ORG PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• org

#### **Program Functional Description**

This example shows the usage of the **Org** directive. Code generation begins at an address spcified by **Org** <address>. The origin of a data table also can be specified by this directive. A data table may be placed either in a program memory region or in an EE data memory region, as in case of PICmicro device with EE data FLASH.

#### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. org 0000 ;The following code will be ;placed in reset address 0. goto Main ;Jump to an address whose label ;is 'Main'. org 0004 ;The following code will be ;placed in interrupt address 4. goto int\_routine ;Jump to an address whose label ;is 'int\_routine'. org 0010 ;The following code section will ;placed starting from address 10H. Main ; ;Write your main program here. ; ;

goto Main ;Loop back to 'Main'. org 0100; The following code section will ;be placed starting from address :100H. int routine ; ;Write your interrupt service ; ;routine here. retfie ;Return from interrupt. org 1000 ;You can create a data or ;character table starting from ;any address in program memory. ;In this case the address is ;1000h. ch\_tbl1 da "PICwithFLASH" ;6 program memory locations ;(starting from 1000h) will ;be filled with six 14-bit ;packed numbers, each ;representing two 7-bit ASCII ;characters. org 2100 ;The absolue address 2100h is ;mapped to the 0000 location of ;EE data memory in PIC16Fxxx. ;You can create a data or ;character table starting from ;any address in EE data memory. ch\_tbl2 de "PICwithFLASH" ;12 EE data memory locations ;(starting from 0) will be ;filled with 12 ASCII ;characters. end



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **ORG PIC18CXXX Example**

Directives highlighted in this example for PIC18CXXX devices are:

• org

#### **Program Functional Description**

This example shows the usage of the **Org** directive. Code generation begins at an address spcified by **Org** <address>. The origin of a data table also can be specified by this directive. A data table may be placed either in a program memory region or in an EE data memory region, as in case of PICmicro device with EE data FLASH.

#### **Commented Code Listing**

list p=18c452 ;Select the device. #include <p18c452.inc> ;Include standard header file ;for the selected device. org 0000 ;The following code will be ;programmed in reset address 0. goto Main ;Jump to an address whose label is ;'Main'. org 0008 ;The following code will be programmed in high priority ; interrupt address 8. goto int\_hi ;Jump to an address whose label is ;'int hi'. org 0018 ;The following code will be ;programmed in low priority ;interrupt address 18h. goto int\_lo ;Jump to an address whose label is ;'int lo'.

```
org 0010 ;The following code section will
;be programmed starting from
:address 10H.
Main
; ;Write your main program here.
goto Main ;Loop back to 'Main'
org 0100 ;The following code section will
;be programmed starting from
;address 100H.
int hi
; ;Write your high priority
; ;interrupt service routine here.
retfie ;Return from interrupt.
org 0200 ;The following code section will
;be programmed starting from
;address 200H.
int lo
; ;Write your low priority
; ;interrupt service routine here.
retfie ;Return from interrupt.
org 1000 ;You can create a data or
;character table starting from any
;address in program memory. In
;this case the address is 1000h.
ch_tbl1 db "PICwithFLASH"
end
```



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **RADIX Example**

Directives highlighted in this example are:

- list r=
- radix

## **Program Functional Description**

This example shows the usage of the radix directive for data presentation. If not declared, then the default radix is in hex(adecimal).

## **Commented Code Listing**

list p=16f877,r=dec ;Select the device and set :radix as decimal. #include <p16f877.inc> ;Include standard header file ;for the selected device. movlw 50H ;50 is in hex movlw 0x50 ;Another way of declaring 50 hex movlw 50O;50 is in octal movlw 50;50 is not declared as hex or ;octal or decimal. So by default ; it is in decimal as default radix ; is declared as decimal. radix oct ;Use `radix' to declare default :radix as octal. movlw 50H ;50 is in hex. movlw 0x50 ;Another way of declaring 50 hex. movlw .50 ;50 is in decimal. movlw 50;50 is not declared as hex or ;octal or decimal. So by default

;it is in octal as default radix ;is declared as octal. radix hex ;Now default radix is in hex. movlw .50 ;50 is declared in decimal. movlw 500 ;50 is declared in octal movlw 50 ;50 is not declared as hex or ;octal or decimal. So by default ;it is in hex as default radix ;is declared as hex. end



#### Microchip Technology Inc. Microchip's Web Site

Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## **SET/EQU Example**

Directives highlighted in this example are:

- set
- equ

#### **Program Functional Description**

This example shows the the usage of the set directive, used for creating symbols which may be used in MPASM assembler expressions only. The symbols created with this directive do not occupy any physical memory location of microcontroller.

### **Commented Code Listing**

```
list p=16f877 ;Select the device.
#include <p16f877.inc> ;Include standard header file
;for the selected device.
perimeter set 0 ;The label 'perimeter' is
;assigned value 0.
area set 0 ;The label 'area' is assigned
;value 0.
lngth equ 50H ;The label 'lngth' is assigned
;the value 50H.
wdth equ 25H ;The label 'wdth' is assigned
;the value 25H.
perimeter set 2*(lngth+wdth) ;Both 'perimeter' and
area set lngth*wdth ;'area' values are
;reassigned.
end
```



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## **UNDEFINE/DEFINE Example**

Directives highlighted in this example are:

- #undefine
- #define

### **Program Functional Description**

This example shows the the usage of **#UNDEFINE** directive. A symbol name previously defined with the **#DEFINE** directive, is removed from the symbol table if **#UNDEFINE** directive is used. The same symbol may be redefined again.

#### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. area set 0; The label 'area' is assigned ;the value 0. #define lngth 50H ;Label 'lngth' is assigned ;the value 50H. #define wdth 25H ;Label 'wdth' is assigned ;the value 25H area set lngth\*wdth ;Reassignment of label 'area'. ;So 'area' will be reassigned a ;value equal to 50H\*25H. #undefine lngth ;Undefine label 'lngth'. #undefine wdth ;Undefine label 'wdth' **#define** lngth 0 ;Define label 'lngth' to '0'. end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

## **VARIABLE/CONSTANT Example**

Directives highlighted in this example are:

- variable
- constant

#### **Program Functional Description**

This example shows the the usage of the variable directive, used for creating symbols which may be used in MPASM assembler expressions only. The symbols created with this directive do not occupy any physical memory location of microcontroller.

#### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. variable perimeter=0 ;The symbol 'perimeter' is ; initialized to 0 variable area ;If a symbol is declared as ;variable, then initialization ; is optional, i.e. it may or may ;not be initialized. constant lngth=50H ;The symbol 'lngth' is ; initialized to 50H. constant wdth=25H ;The symbol 'wdth' is ; initialized to 25H. ;A constant symbol always needs ;to be initialized. perimeter=2\*(lngth+wdth);The value of a CONSTANT cannot ;be reassigned after having been ;initialized once. So 'lngth' and ;'wdth' cannot be reassigned. But ;'perimeter' has been declared ;as variable, and so can be ;reassigned. area=lngth\*wdth end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **List of Conditional Assembly Directives**

Directive	Description	Syntax
ELSE	Begin Alternative Assembly Block to IF	else
ENDIF	End Conditional Assembly Block	endif
ENDW	End a While Loop	endw
IF	Begin Conditionally Assembled Code Block	if <expr></expr>
IFDEF	Execute If Symbol is Defined	ifdef <label></label>
IFNDEF	Execute If Symbol is Not Defined	ifndef <label></label>
WHILE	Perform Loop While Condition is True	while <expr></expr>



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **IF/ELSE/ENDIF** Example

Directives highlighted in this example are:

- if
- else
- endif

## **Program Functional Description**

This program demonstrates the utility of IF, ELSE and ENDIF assembly directives.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. variable config ;variable used to define ;required configuration of ;PORTA & PORTB config set D'1' RST CODE H'0' ;The code section named RST ;is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled

;'service\_int'. PGM CODE ;This is the begining of the code ;section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start if config==H'0' ;If config==H'0' is true, clrw ;assemble the mnemonics up to movwf TRISA ;the directive 'else'. movlw H'ff' movwf TRISB else clrw ;If config==H'0' is false, movwf TRISB ;assemble the mnemonics up to movlw H'ff' ;the directive 'endif'. movwf TRISA endif goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **IFDEF Example**

Directives highlighted in this example are:

- #define
- ifdef
- else
- endif

## **Program Functional Description**

This program uses the control directive #define, along with the ifdef, else and endif directives to seletively assemble code for use with either an emulator or an acutal part. The list directives title and list p= are used to set the title and processor and display this information in the list file. The control directive #include is used to include the standard header file for the selected device.

#### **Commented Code Listing**

```
; Define EMULATOR at the top of this file
; (#define EMULATOR)
; This will set the I2C PORT, SDA and SCL lines
; to communicate over Port A, pins 0 and 1. It
; also assembles in the necessary TRIS
; instructions to allow reading from the SDA line.
; To convert the code for the actual part, simply comment
; out the #define EMULATOR line and reassemble.
#ifdef EMULATED
I2C_PORT EQU 5 ; Port A control register,
; used for I2C
SCL EQU 01H; EEPROM Clock, SCL (I/O bit 7)
SDA EQU 00H; EEPROM Data, SDA (I/O bit 6)
#else
I2C_PORT EQU GPIO ; Port B control register,
; used for I2C
SCL EQU 07H; EEPROM Clock, SCL (I/O bit 7)
SDA EQU 06H; EEPROM Data, SDA (I/O bit 6)
#endif
START BIT
BCF I2C_PORT,SDA ; Start bit, SDA and SCL
; preset to "1"
```



# **WHILE/ENDW** Example

Directives highlighted in this example are:

- while
- endw

# **Program Functional Description**

This example shows the usefulness of directive while to perform a loop while a certain condition is true. This directive is used with the endw directive.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. variable i ;Define the symbol 'i' as a ;variable. reg\_hi equ 20 ;Assign value 20H to label ;reg hi. reg\_lo equ 21 ;Assign value 21H to label ;reg\_lo. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address whose label ;is 'start'. shift\_right macro by\_n ;Beginning of a macro, which ;shifts register data n times. ;Code length generated after ;assembly, varies depending upon ;the value of parameter 'by\_n'.

```
i=0 ;Initialize variable i.
while i< by_n ;Following 3 lines of assembly
;code are repeated as long as
;i< by_n.
bcf STATUS,C ;Clear carry bit.
rrf reg_hi ;reg_hi and reg_lo contains
rrf reg_lo ;16-bit data which is rotated
;right through carry.
i+=1 ;Increment loop counter i.
endw; End while loop. The loop will
;break here after i=by_n.
endm ;End of 'shift_right' macro.
org 0010 ;My main program starts at 10H.
start ;The label 'start' is equal to
;10H.
shift_right 3 ;Shift right 3 times the 16-bit
;data in reg_hi and reg_lo. This
; is an example. A value 8 will
;shift data 8 times.
goto $
end
```



# **List of Data Directives**

Directive	Description	Syntax
BADRAM	Specify invalid RAM locations	badram <expr></expr>
CBLOCK	Define a Block of Constants	cblock [ <expr>]</expr>
CODE_PACK	No Padding at End of Odd Byte	code_pack
CONFIG	Set configuration fuses	config <expr> OR config <addr>, <expr></expr></addr></expr>
DA	Store Strings in Program Memory	[ <label>] da <expr> [, <expr2>,, <exprn>]</exprn></expr2></expr></label>
DATA	Create Numeric and Text Data	data <expr>,[,<expr>,,<expr>] data " <text_string>" [,"<text_string>",]</text_string></text_string></expr></expr></expr>
DB	Declare Data of One Byte	db <expr>[,<expr>,,<expr>]</expr></expr></expr>
DE	Declare EEPROM Data	de <expr>[,<expr>,,<expr>]</expr></expr></expr>
DT	Define Table	dt <expr>[,<expr>,,<expr>]</expr></expr></expr>
DW	Declare Data of One Word	dw <expr>[,<expr>,,<expr>]</expr></expr></expr>
ENDC	End an Automatic Constant Block	endc
FILL	Specify Memory Fill Value	fill <expr>, <count></count></expr>
IDLOCS	Set ID locations	idlocs <expr></expr>
MAXRAM	Specify maximum RAM address	maxram <expr></expr>
RES	Reserve Memory	res <mem_units></mem_units>



# **CBLOCK/ENDC Example**

Directives highlighted in this example are:

- cblock
- endc

# **Program Functional Description**

This example shows the usage of CBLOCK and ENDC directives for defining constants or variablers in data memory space. The same directives can be used for program memory space also.

The program calculates the perimeter of a rectangle. Length and width of the rectangle will be stored in buffers addressed by length (22H) and width (23H). The calculated perimeter will be stored in the double-precision buffer addressed by perimeter (i.e.20H and 21H).

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. CBLOCK 0x20 ;Starting address of program or ;data memory space. Here the value ;is 20H, which is in data memory ;space. perimeter:2 ;The label perimeter is 2-byte ;wide. Address 20H and 21H is ;assigned to the label perimeter. length ;Address 22H is assigned to the ;label length.

```
width ;Address 23H is assigned to the
;label width.
ENDC ;This directive must be supplied
;at the end of CBLOCK list to
:terminate the list.
clrf perimeter ;Clear the buffer addressed by
;'perimeter' i.e. address 20H.
clrf perimeter+1 ;Clear address 21H.
movf length, w; Move the data present in the
;register addressed by 'length'
:to 'w'
addwf width,w ;Add data in 'w' with data in the
;register addressed by 'width'.
movwf perimeter ;Move 'w' to the register
;addressed by 20H.
incfsz perimeter+1 ;Increment register 21H if carry
; is generated.
bcf STATUS,C ;Clear carry bit in STATUS
;register.
rlf perimeter+1
rlf perimeter
incfsz perimeter+1 ;High byte of perimeter is in
;21H and low byte is in 20H.
goto $
end
```



# **CONFIG PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• \_ \_ config

## **Program Functional Description**

This program demonstrates the utility of the <u>config</u> directive. This directive is used to program configuration bits in the configuration register during device programming.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. **\_config** \_LP\_OSC ;Configuration register is ;programmed to select low ;power oscillator. Refer to ;data sheet for details of ;configuration register. RST CODE H'0' ;The code section named RST ;is placed at H'0.' ;The instruction 'goto start' ; is placed in code section RST. goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'4' ;The code section named INTRT ; is placed at H'4'. ;The instruction 'goto ;service\_int' is placed in code ;section INTRT. goto service\_int ;Jumps to the location

;labelled 'service\_int'. PGM CODE ;This is the begining of the ;code section named PGM. It is ;a relocatable code section ;since no absolute address is ;given along with directive start ;'CODE'. goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **CONFIG PIC18CXXX Example**

Directives highlighted in this example for PIC18CXXX devices are:

• \_ \_ config

### **Program Functional Description**

This program demonstrates the utility of the <u>config</u> directive. This directive is used to program configuration bits in the configuration register during device programming.

### **Commented Code Listing**

list p=18c452 ;Select the device. #include <p18c452.inc> ;Include standard header file ;for the selected device. ;code protect disabled. \_\_CONFIG \_CONFIG0, \_CP\_OFF\_0 ;Oscillator switch disabled, RC oscillator with OSC2 ;as I/O pin. \_\_CONFIG \_CONFIG1, \_OSCS\_OFF\_1 & \_RCIO\_OSC\_1 ;Brown-OutReset enabled, BOR Voltage is 2.5v CONFIG CONFIG2, BOR ON 2 & BORV 25 2 ;Watch Dog Timer enable, Watch Dog Timer PostScaler :count - 1:128 \_CONFIG \_CONFIG3, \_WDT\_ON\_3 & \_WDTPS\_128\_3 ;CCP2 pin Mux enabled CONFIG CONFIG5, CCP2MX ON 5 ;Stack over/underflow Reset enabled CONFIG CONFIG6, STVR ON 6 RST CODE H'0' ;The code section named RST ; is placed at H'0'. ;The instruction 'goto start'

; is placed in code section RST. goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'8' ;The code section named INTRT ;is placed at H'4'. ;The instruction 'goto service\_int' ; is placed in code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ;This is the begining of the code ;section named PGM. It is a ;relocatable code section ;since no absolute address is ;given along with directive start ;'CODE'. goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **DA Example**

Directives highlighted in this example are:

• da

# **Program Functional Description**

This example shows the usefulnes of directive da in storing a character string in the program memory of 14-bit architecture devices. This directive generates a packed 14-bit number representing two 7-bit ASCII characters.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. start ;Write your main program here to ; display the string given in ;'Ch\_stng'. goto \$ ORG 1000 ;Store the string starting from ;1000H. Ch stng **da** "PICmicro" Sngl\_ch da "A" ;7-bit ASCII equivalents of 'A' ;and a NULL charater will be packed ;in a 14-bit number. da 0xff55 ;Places 3f55 in program memory. ;No packing.

end

## **Additional Comments**

#### **Determining 14-Bit Numbers**

For the following statement:

Ch\_stng da "PICmicro"

directive da produces four 14-bit numbers: 2849, 21ED, 34E3 and 396F representing the ASCII equivalent of PI, Cm, ic and ro.

To see how the 14-bit numbers are determined, let's look at the ASCII values of P and I, which are 50h(01010000) and 49h(01001001) respectively. Each is presented in 7-bit as (0)1010000 and (0)1001001 respectively. The packed 14-bit number is 101000 01001001, which is stored as (00)101000 01001001 or 2849.



# **DATA PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• data

## **Program Functional Description**

This example shows the usefulnes of directive data in storing one or more words in program memory.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. start ;Write your main program here to ; display the string given in ;'Ch stng'. goto \$ ORG 1000 ;Store the string starting from ;1000H. Ch\_stng **data** 'M', 'C', 'U' ;3 program memory locations ;will be filled with ASCII ;equivalent of 'M', 'C' and ;'U'. tb1\_dta data 0xffff,0xaa55 ;Places 3fffh and 2a55h in ;two consecutive program ;memory locations. As program ;memory is 14-bit wide,

;the last nibble can store ;a maximum value 3. end



# DATA PIC18CXXX Example

Directives highlighted in this example for PIC18CXXX devices are:

• data

## **Program Functional Description**

This example shows the usefulnes of directive data in storing one or more words in program memory.

# **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. start ;Write your main program here to ; display the string given in ;'Ch stng'. goto \$ ORG 1000 ;Store the string starting from ;1000H. In PIC18Cxxx devices, the ;first character is in least ;significant byte. Ch stng **data** 'M', 'C', 'U' ;3 program memory locations ;will be filled with ASCII ;equivalent of 'M', 'C' and :'U'. Ch\_stg1 data "MCU" ;2 program memory locations ;will be filled with two

;words (16-bit numbers), ;each representing ASCI ;equivalent of two ;characters. The last ;character will be taken as ;NULL in case odd number of ;characters are specified. tb1\_dta data 0xffff,0xaa55 ;Places ffff and aa55 in ;two consecutive ;program memory location. end



# **DB PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• db

# **Program Functional Description**

This example shows the usefulness of directive **db** in storing one or more byte or character in program memory.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. start ;Write your main program here. goto \$ ORG 1000 ;Store the string starting from ;1000H. Ch\_stng **db** 0,'M',0,'C',0,'U' tb1\_dta db 0,0xff ;Places 00ff in program memory ;location. end



# **DB PIC18CXXX Example**

Directives highlighted in this example for PIC18CXXX devices are:

• db

# **Program Functional Description**

This example shows the usefulnes of directive **db** in storing one or more byte or character in program memory.

# **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. ORG 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. start ;Write your main program here to ; display the string given in ;'Ch stng'. goto \$ ORG 1000 ;Store the string starting from ;1000H. In PIC18Cxxx devices, the ;first character is in least ;significant byte. Ch stng db 'M', 'C', 'U' tb1 dta db 0,0xff ;Places ff00 in program memory ;location. end



# **DE PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• de

# **Program Functional Description**

This example shows the usage of the **de** directive. This directive is designed mainly for initializing data in the EE data memory region of PICmicro devices with EE data FLASH.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. org 2100 ;The absolue address 2100h is ;mapped to the 0000 location of ;EE data memory. ch\_tbl2 de "PICmicro" ;6 EE data memory locations ;(starting from 0) will be filled ;with 6 ASCII characters. end



# **DE PIC18CXXX Example**

Directives highlighted in this example for PIC18CXXX devices are:

• de

# **Program Functional Description**

This example shows the usage of the **de** directive. This directive is designed mainly for initializing data in the EE data memory region of PICmicro devices with EE data FLASH.

# **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. org F0 ;The absolue address F0h is ;mapped to the 0000 location of ;EE data memory. ch\_tbl2 de "PICmicro" ;6 EE data memory locations ;(starting from 0) will be filled ;with 6 ASCII characters. end



# FILL PIC16CXXX Example

Directives highlighted in this example for PIC16CXXX devices are:

• fill

# **Program Functional Description**

The fill directive is used to program successive program memory locations with a constant or an assembly instruction.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. org 0000 ;The following code will be ;programmed in reset address 0. goto start ;Jump to an address labelled ;'start'. FILL 0, INTRPT-\$; Fill with 0 up to address 3. **INTRPT** org 0004 goto ISR FILL (goto start), start-\$ ;Fill upto address 0Fh with ;instruction <goto start>. **ORG 0010** start ;Write your main program here. FILL (nop), 5 ;Fill 5 locations with NOPs. goto \$ ISR; RETFIE **END** 



# FILL PIC18CXXX Example

Directives highlighted in this example for PIC18CXXX devices are:

• fill

# **Program Functional Description**

The fill directive is used to program successive program memory locations with a constant or an assembly instruction. For PIC18CXXX devices, only an even number is allowed to be specified as a count of locations to be filled.

# **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. org 0000 ;Following code will be programmed ; in reset address 0. goto start ;Jump to an address labelled ;'start'. FILL 0, HI INT-\$; Fills 0 in 2 program memory ;locations: 0004 and 0006. HI\_INT org 0008 goto INTR\_H FILL (goto start),6 ;Fills 6 locations (each location ; is 2 bytes wide) with 3 numbers :of 2 word wide instructions ;<goto start> LO\_INT org 0018 goto INTR L FILL 10a9, start-\$; Fills address 1Ch and 1Eh with ;10a9h

```
ORG 0020
start ;Write your main program here
;
FILL (nop), 4 ;Fills 2 locations (4 bytes) with
;NOP
goto $
INTR_H ;
RETFIE
INTR_L ;
RETFIE
END
```



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **IDLOC PIC16CXXX Example**

Directives highlighted in this example for PIC16CXXX devices are:

• \_\_idloc

## **Program Functional Description**

This program demonstrates the utility of the \_\_idlocs directive. This directive is used to program device ID bits in the IDLOC register during device programming.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file :for the selected device. idlocs H'1234' ;Sets device ID to 1234. RST CODE H'0' ;The code section named RST ; is placed at H'0'. ;The instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'.The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service int ;Jumps to the location labelled ;'service int'. PGM CODE ;This is the begining of the code ;section named PGM. It is a ;relocatable code section since ;no absolute address is given along

start ;with directive 'CODE'. goto \$ service\_int retfie end



# **IDLOC PIC18CXXX Example**

Directives highlighted in this example for PIC18CXXX devices are:

• \_\_idloc

# **Program Functional Description**

This program demonstrates the utility of the \_\_idlocs directive. This directive is used to program device ID bits in the IDLOC register during device programming.

# **Commented Code Listing**

list p=18c452 ;Select device.
#include <p18c452.inc> ;Include standard header file
;for the selected device.

\_\_IDLOCS \_IDLOC0, H'1' ;IDLOC register 0 will be ;programmed to 1.

\_\_IDLOCS \_IDLOC1, H'2' ;IDLOC register 1 will be ;programmed to 2.

\_\_IDLOCS \_IDLOC2, H'3' ;IDLOC register 2 will be ;programmed to 3.

\_\_IDLOCS \_IDLOC3, H'4' ;IDLOC register 3 will be ;programmed to 4.

\_\_IDLOCS \_IDLOC4, H'5' ;IDLOC register 4 will be ;programmed to 5.

\_\_IDLOCS \_IDLOC5, H'6' ;IDLOC register 5 will be ;programmed to 6.

\_\_IDLOCS \_IDLOC6, H'7' ;IDLOC register 6 will be ;programmed to 7.

\_\_IDLOCS \_IDLOC7, H'8' ;IDLOC register 7 will be ;programmed to 8.

RST CODE H'0' ;The code section named RST

; is placed at H'0'. The instruction ;'goto start' is placed in code ;section RST. goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'8' ;The code section named INTRT is ;placed at H'4'.The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the begining of the code ;section named PGM. It is a ;relocatable code section since ;no absolute address is given along start ;with directive 'CODE'. goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **RES Example**

Directives highlighted in this example are:

• res

### **Program Functional Description**

This example shows the advantage of res directive in developing relocatable code. The program calculates the perimeter of a rectangle. Length and width of the rectangle will be stored in buffers addressed by length and width. The calculated perimeter will be stored in the double-precision buffer addressed by perimeter.

# **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. UDATA ;This directive allows the ;following data to be placed only ; in the data area. perimeter **res** 2 ;Two locations of memory are ;reserved for the label ;'perimeter'. Addresses of the ;memory locations will be ;allocated by MPLINK. length res 1 ;One location of memory is ;reserved for the label 'length'. ;Address of the memory location ;will be allocated by MPLINK. width res 1; One location of memory is :reserved for the label 'width'.

;Address of the memory location ;will be allocated by MPLINK. Start CODE 0000 ;Following code will be placed in ;address 0. goto PER\_CAL ;Jump to label PER\_CAL CODE ;CODE directive here dictates that ;the following lines of code will ;be placed in program memory, but ;the starting address will be ;decided by MPLINK. PER CAL clrf perimeter ;Clear the buffers addressed by clrf perimeter+1 ;'perimeter'. movf length,w;Move the data present in the ;register addressed by 'length' ;to 'w'. addwf width,w ;Add data in 'w' with data in the ;register addressed by 'width' movwf perimeter ;Move 'w' to the register ;addressed by 'perimeter'. incfsz perimeter+1 ;Increment 'perimeter+1' if carry ; is generated. bcf STATUS,C ;Clear carry bit in STATUS ;register. rlf perimeter+1 rlf perimeter incfsz perimeter+1 goto \$ end



# **List of Listing Directives**

Directive	Description	Syntax
ERROR	Issue an Error Message	error " <text_string>"</text_string>
ERRORLEVEL	Set Messge Level	errorlevel 0 1 2 <+-> <msg></msg>
LIST	Listing Options	<pre>list [<option>[,,<option>]]</option></option></pre>
MESSG	Create User Defined Message	<pre>messg "<message_text>"</message_text></pre>
NOLIST	Turn off Listing Output	nolist
PAGE	Insert Listing Page Eject	page
SPACE	Insert Blank Listing Lines	space [ <expr>]</expr>
SUBTITLE	Specify Program Subtitle	subtitl " <sub_text>"</sub_text>
TITLE	Specify Program Title	title " <title_text>"</title_text>

> <

# **ERROR Example**

Directives highlighted in this example are:

• error

# **Program Functional Description**

This program demonstrates the utility of the error assembler directive, which sets an error message to be printed in the listing file and error file.

# **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. variable baudrate ;variable used to define ;required baud rate baudrate set D'5600' ;Enter the required value of ;baud rate here. if (baudrate!=D'1200')&&(baudrate!=D'2400')&& (baudrate!=D'4800')&&(baudrate!=D'9600')&& (baudrate!=D'19200') error "Selected baud rate is not supported" endif RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'.The instruction

;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the begining of the ;code section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **ERRORLEVEL Example**

Directives highlighted in this example are:

errorlevel

## **Program Functional Description**

This program demonstrates the utility of the errorlevel assembler directive, which sets the type of messages that are printed in the listing file and error file.

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. errorlevel 0 ;Display/print messages, ;warnings and errors. messg "CAUTION: This program has errors" errorlevel 1 ;Display/print warnings ;and errors. messg "CAUTION: This program has errors" group1 udata 0x20 group1\_var1 res 1 ;Label of this directive is not ;at column 1. This will generate ;a warning no. 207. errorlevel -207 ;This disables warning whose ;msgnum is 207. group1 var2 res 1 ;label of this directive is also ;not at column 1, but no warning ; is displayed/printed. errorlevel +207 ;This enables warning whose ;msgnum is 207

group2 udata

errorlevel 2 ;Display/print errors

group2\_var1 res 1 ;label of this directive is not

;at column 1. This will generate

;a warning no. 207.

errorlevel 1 ;Display/print warnings

;and errors.

group2\_var2 res 1 ;label of this directive is not

;at column 1. This will generate

;a warning no. 207.

RST CODE H'0 ' ;The code section named RST is

;placed at H'0'. The instruction

;'goto start' is placed in code

;section RST.

goto start ;Jumps to the location labelled

;'start'.

INTRT CODE H'4 ' ;The code section named INTRT is

;placed at H'4'. The instruction

;'goto service\_int' is placed in

;code section INTRT

goto service\_int ;Label 'service\_int' is not

;defined. Hence this generates

;error[113].

PGM CODE ;This is the begining of the code

;section named 'PGM'. It is a

;relocatable code section since

;no absolute address is given along

;with directive 'CODE'

start

movwf group1\_var1

goto \$

end



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **MESSG Example**

Directives highlighted in this example are:

• messg

### **Program Functional Description**

This program demonstrates the utility of the **messg** assembler directive, which sets a message to be printed in the listing file and error file.

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. variable baudrate ;variable used to define ;required baud rate baudrate set D'5600' ;Enter the required value of ;baud rate here. if (baudrate!=D'1200')&&(baudrate!=D'2400')&& (baudrate!=D'4800')&&(baudrate!=D'9600')&& (baudrate!=D'19200') error "Selected baud rate is not supported" messg "only baud rates 1200,2400,4800,9600 & 19200 Hz "&& "are supported" endif RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled ;'start'.

INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'.The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the begining of the ;code section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start goto \$ service\_int retfie end



#### Microchip Technology Inc.

Microchip's Web Site Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **List of Macro Directives**

Directive	Description	Syntax
ENDM	End a Macro Definition	endm
EXITM	Exit from a Macro	exitm
EXPAND	Expand Macro Listing	expand
LOCAL	Declare Local Macro Variable	local <label> [,<label>]</label></label>
MACR0	Declare Macro Definition	<label> macro [<arg>,,<arg>]</arg></arg></label>
NOEXPAND	Turn off Macro Expansion	noexpand



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **EXITM Example**

Directives highlighted in this example are:

• exitm

## **Program Functional Description**

This program demonstrates the utility of the exitm assembler directive, which causes an immediate exit from a macro. It is used in the example to exit from the macro when certain conditions are met.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. result equ 20 ;Assign value 20H to label ;result. ORG 0000 ;The following code will be placed ; in reset address 0. goto start ;Jump to an address whose label is :'start'. add MACRO num1, num2; 'add' is a macro. The values of ;'num1' and 'num2' must be passed ;to this macro. if num1>0xff ;If num1>255 decimal, extim :force immediate return from ;macro during assembly. else if num2>0xff ;If num2>255 decimal, extim ;force immediate return from ;macro during assembly.

else

movlw num1 ;Load W register with a literal ;value assigned to the label ;'num1'. movwf result ;Load W register to an address ;location assigned to the label ;'result'. movlw num2 ;Load W register with a literal ;value assigned to the label ;'num2'. addwf result ;Add W register with the memory ;location addressed by 'result' ;and load the result back to ;'result'. endif endif endm ;End of 'add' MACRO org 0010 ;My main program starts at 10H. start ;The label 'start' is assigned an ;address 10H. add .100,.256 ;Call 'add' MACRO with decimal ;numbers 100 and 256 assigned to ;'num1' and 'num2' labels, ;respactively. EXTIM directive in ;macro will force return. end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# LOCAL Example

Directives highlighted in this example are:

local

### **Program Functional Description**

This code demonstrates the utility of local directive, which declares that the specified data elements are to be considered in local context to the macro.

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. incr equ 2 ;Assembler variable incr is set ;equal to 2. add\_incr macro ;Declaration of macro 'add\_incr'. local incr ;Local assembler variable 'incr'. incr set 3 ;Local 'incr' is set to 3, in :contrast to 'incr' value ;of 2 in main code. clrw ;w register is set to zero addlw incr ;w register is added to incr and ;result placed back endm; in w register. RST CODE H'0' ;The code section named RST is ;placed at H'0'. The instruction ;'goto start' is placed in code ;section RST. goto start ;Jumps to the location labelled ;'start'.

INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4' The instruction ;'goto service\_int' is placed in :code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the begining of the code ;section named 'PGM'. It is a ;relocatable code section since ;no absolute address is given along :with directive 'CODE' start clrw ;W register set to zero. addlw incr ;W register is added with the ;value of incr which is now equal ;to 2. add\_incr ;W register is added with the ;value of incr which is now equal ;to 3 (value set locally in the ;macro add\_incr). clrw ;W register is set to zero again. addlw incr ; incr is added to W register and ;result placed in W register. ; incr value is again 2, not ;affected by the value set in the ;macro. goto \$ service int retfie end



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **MACRO/ENDM** Example

Directives highlighted in this example are:

- macro
- endm

## **Program Functional Description**

This code demonstrates the utility of macro directive, which is used to define a macro.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file :for the selected device. result equ 20 ;Assign value 20H to label ;result. ORG 0000 ;The following code will be placed ; in reset address 0. goto start ;Jump to an address whose label is ;'start'. add MACRO num1, num2; 'add' is a macro. The values of ;'num1' and 'num2' must be passed :to this macro. movlw num1 ;Load W register with a literal ;value assigned to the label ;'num1'. movwf result ;Load W register to an address ;location assigned to the label ;'result'. movlw num2 ;Load W register with a literal

;value assigned to the label ;'num2'. addwf result ;Add W register with the memory ;location addressed by 'result' ;and load the result back to ;'result'. endm ;end of 'add' MACRO org 0010 ;Main program starts at 10H. start ;The label 'start' is assigned an ;address 10H. add .100,.90 ;Call 'add' MACRO with decimal ;numbers 100 and 90 assigned to ;'num1' and 'num2' labels, ;respactively. 100 and 90 will be ;added and the result will be in ;'result'. end



#### Microchip Technology Inc. Microchip's Web Site Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **List of Object File Directives**

Directive	Description	Syntax
BANKISEL	Generate RAM bank selecting code for indirect addressing	bankisel <label></label>
BANKSEL	Generate RAM bank selecting code	banksel <label></label>
CODE	Begins executable code section	[ <name>] code [<address>]</address></name>
EXTERN	Declares an external label	extern <label> [ , <label>]</label></label>
GLOBAL	Exports a defined label	extern <label> [ . <label>]</label></label>
IDATA	Begins initialized data section	[ <name>] idata [<address>]</address></name>
PAGESEL	Generate ROM page selecting code	pagesel <label></label>
UDATA	Begins uninitialized data section	[ <name>] udata [<address>]</address></name>
UDATA_ACS	Begins access uninitialized data section	[ <name>] udata_acs [<address>]</address></name>
UDATA_OVR	Begins overlayed uninitialized data section	[ <name>] udata_ovr [<address>]</address></name>
UDATA_SHR	Begins shared uninitialized data section	[ <name>] udata_shr [<address>]</address></name>



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **BANKISEL Example**

Directives highlighted in this example are:

bankisel

## **Program Functional Description**

This program demonstrates the utility of the bankisel directive. This directive generates the appropriate code to set/clear the IRP bit of the STATUS register for an indirect access.

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. group1 udata H'20' ;group1 data stored at locations ;starting at H'20'(IRP bit 0). group1\_var1 res 1 ;group1\_var1 located at H'20'. group1\_var2 res 1 ;group1\_var2 located at H'21'. group2 udata H'120' ;group2 data stored at locations ;starting at H'120'(IRP bit 1). group2\_var1 res 1 ;group2\_var1 located at H'120'. group2\_var2 res 1 ;group2\_var2 located at H'121'. RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'.The instruction ;'goto service\_int' is placed in

;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ;This is the beginning of the ;code section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start movlw H'20' ;This part of the code addresses movwf FSR ;variables group1\_var1 & bankisel group1\_var1 ;group1\_var2 indirectly. clrf INDF incf FSR,F clrf INDF movwf FSR bankisel group2\_var1 clrf INDF incf FSR,F clrf INDF goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **BANKSEL Example**

Directives highlighted in this example are:

banksel

### **Program Functional Description**

This program demonstrates the utility of the banksel directive. This directive generates the appropriate code to set/clear the RP0 and RP1 bits of the STATUS register.

#### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. group1 udata H'20' ;group1 data stored at locations ;starting at H'20'(bank 0). group1\_var1 res 1 ;group1\_var1 located at H'20'. group1\_var2 res 1 ;group1\_var2 located at H'21'. group2 udata H'A0' ;group2 data stored at locations ;starting at H'A0'(bank 1) group2\_var1 res 1 group2\_var2 res 1 RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed in

;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ;This is the beginning of the ;code section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start banksel group1\_var1 ;This directive generates code ;to set/clear bank select bits ;RP0 & RP1 of STATUS register ;depending upon the address of ;group1\_var1. clrf group1\_var1 clrf group1\_var2 banksel group2\_var1 ;This directive generates code ;to set/clear bank select bits ;RP0 & RP1 of STATUS register ;depending upon the address of ;group2\_var1. clrf group2\_var1 clrf group2\_var2 goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **CODE Example**

Directives highlighted in this example are:

code

## **Program Functional Description**

This program demonstrates the utility of the code directive, which declares the beginning of a section of program code.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ; This is the begining of the ;code section named PGM. It is a ;relocatable code section since ;no absolute address is given ;along with directive 'CODE'. start

clrw goto \$ **CODE** ;This is a relocatable code nop ;section since no address is ; ;specified with the CODE ; ;directive. ; ; service\_int retfie end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **GLOBAL/EXTERN Example**

Directives highlighted in this example are:

- global
- extern

### **Program Functional Description**

The program main.asm, along with sub.asm, demonstrate the utility of the GLOBAL and EXTERN directives, which make it possible to use symbols in modules other than where they are defined.

## **Commented Code Listing**

```
;main.asm
list p=16f877 ;Select the device.
#include <p16f877.inc> ;Include standard header file
:for the selected device.
UDATA
delay_value res 1
GLOBAL delay_value ;The variable 'delay_value',
;declared GLOBAL in this
;module, is included in an
:EXTERN directive in the module
;sub.asm.
EXTERN delay ;The variable 'delay', declared
;EXTERN in this module, is
;declared GLOBAL in the module
;sub.asm.
```

RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service int' is placed in :code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ; This is the begining of the ;code section named PGM. It is a relocatable code section since; ;no absolute address is given ;along with directive 'CODE'. start movlw D'10' movwf delay value xorlw H'80' call delay goto start service\_int retfie end :sub.asm list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. GLOBAL delay ;The variable 'delay' declared ;GLOBAL in this module is ;included in an EXTERN directive ; in the module main.asm. EXTERN delay\_value ;The variable 'delay\_value' ;declared EXTERN in this module

; is declared GLOBAL in the ;module main.asm. PGM CODE delay decfsz delay\_value,1 goto delay return end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 Microchip's E-mail Address

# **IDATA Example**

Directives highlighted in this example are:

• idata

## **Program Functional Description**

The directive idata is used when generating an object file. It reserves RAM locations for variables and directs the linker to generate a lookup table that may be used to initialize the variables specified in this section. The Starting Address of the lookup table can be obtained from the Map (.map) file.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. group1 IDATA 0x20 ;Initialized data at location :20h. group1\_var1 res 1 ;group1\_var1 located at 0x20, ; initialized with 0. group1\_var2 res 1 ;group1\_var2 located at 0x21, ;initialized with 0. group2 IDATA ;Declaration of group2 data. The addresses for variables under ;this data section are allocated ;automatically by the linker. group2\_var1 db 1,2,3,4 ;4 bytes in RAM are reserved. group2\_var2 dw H'1234 ';1 word in RAM is reserved. RST CODE 0x0 ;The code section named RST is ;placed at H'0'. ;Following instruction 'goto

;start' is placed in code
;section RST.
goto start ;Jumps to the location labelled
;'start'.
PGM CODE ;Code section named PGM is
;declared. It is a relocatable
;code section since no absolute
;address is specified.
start
• •
• •
end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **PAGESEL Example**

Directives highlighted in this example are:

pagesel

### **Program Functional Description**

This program demonstrates the utility of the pagesel directive, which generates the appropriate code to set/clear PCLATH bits

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. RST CODE H'0' ;The code section named RST ; is placed at H'0'. The ;instruction 'goto start' is ;placed in code section RST. goto start ;Jumps to the location labelled ;'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM0 CODE H'500' ;The code section named PGM0 is ;placed at H'500'. start PAGESEL page1\_pgm ;address bits 12 & 11 of ;page1\_pgm are copied to PCLATH ;4 & 3 respectively.

goto page1\_pgm PGM1 CODE H'900' ;The code section named PGM1 is ;placed at H'900'. Label ;page1\_pgm is located in this page1\_pgm ;code section. goto \$ service\_int retfie end



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **UDATA Example**

Directives highlighted in this example are:

• udata

## **Program Functional Description**

This program demonstrates the utility of the udata directive, which declares the beginning of a section of uninitialized data. This directive is used when generating an object file.

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. group1 udata 0x20 ;group1 data stored at locations ;starting at 0x20. group1\_var1 res 1 ;group1\_var1 located at 0x20. group1\_var2 res 1 ;group1\_var2 located at 0x21. group2 udata ;Declaration of group2 data. The :addresses for variables under group2 var1 res 1 ;this data section are allocated group2\_var2 res 1 ;automatically by the linker. RST CODE H'0'; The code section named RST is ;placed at H'0'. The instruction ;'goto start' is placed in code :section RST. goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'.The instruction ;'goto service\_int' is placed in

;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the beginning of the code ;section named PGM. It is a ;relocatable code section since ;no absolute address is given along start ;with directive 'CODE'. banksel group1\_var1 clrf group1\_var1 clrf group1\_var2 banksel group2\_var1 clrf group2\_var1 clrf group2\_var2 goto \$ service\_int retfie end



#### Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200

Fax: (480) 899-9210 Microchip's E-mail Address

# **UDATA\_ACS Example**

Directives highlighted in this example for PIC18CXXX devices are:

• udata\_acs

### **Program Functional Description**

This program demonstrates the utility of the udata\_acs directive, which is used when generating an object file. This directive declares the beginning of a section of uninitialized data.

### **Commented Code Listing**

list p=18c452 ;Select device. #include <p18c452.inc> ;Include standard header file ;for the selected device. group1 udata\_acs 0x20 ;group1 data stored at access ;RAM locations starting at 0x20. group1\_var1 res 1 ;group1\_var1 located at 0x20. group1\_var2 res 1 ;group1\_var2 located at 0x21. group2 udata acs ;Declaration of group2 data. The :addresses for data under this :secton are allocated ;automatically by the linker. group2\_var1 res 1 ;All addresses be will allocated group2\_var2 res 1 ;in access RAM space only. RST CODE H'0' ;The code section named RST is ;placed at H'0'. The instruction ;'goto start' is placed in code :section RST. goto start ;Jumps to the location labelled :'start' INTRT CODE H'8' ;The code section named INTRT is

```
;placed at H'4'. The instruction
;'goto service_int' is placed in
;code section INTRT.
goto service_int ;Jumps to the location labelled
;'service_int'.
PGM CODE ;This is the begining of the code
;section named PGM. It is a
;relocatable code section since
;no absolute address is given along
:with directive 'CODE'.
start
clrf group1_var1,A ;group1_var1 initialized to zero
clrf group1_var2,A ;group1_var2 initialized to zero
clrf group2_var1,A ;group2_var1 initialized to zero
clrf group2_var2,A ;group2_var2 initialized to zero
goto $
service_int
retfie
end
```



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **UDATA\_OVR Example**

Directives highlighted in this example are:

• udata\_ovr

## **Program Functional Description**

This program demonstrates the utility of the udata\_ovr directive, which is used when generating an object file. This directive declares the beginning of a section of overlayed uninitialized data.

## **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. same\_var udata\_ovr H'20' ;Declares an overlayed ;uninitialized data section ;named'same\_var' starting at var1 res 1 ;location H'20'. same var udata ovr H'20' ;Declares an overlayed :uninitialized data section var2 res 1 ; with the same name as the one ;declared above. Thus variables ;var1 and var2 are allocated ;at the same address. RST CODE H'0' ;The code section named RST is ;placed at H'0'.The instruction ;'goto start' is placed in code ;section RST. goto start ;Jumps to the location labelled ;'start'.

INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed in ;code section INTRT. goto service\_int ;Jumps to the location labelled ;'service\_int'. PGM CODE ;This is the begining of the code ;section named PGM. It is a ;relocatable code section ;since no absolute address is given ;along with directive 'CODE' start banksel var1 ;Any operation on var1 affects movlw H'FF' ;var2 also since both variables movwf var1 ;are overlaid. comf var2 goto \$ service\_int retfie end



#### Microchip Technology Inc.

<u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>

# **UDATA\_SHR Example**

Directives highlighted in this example for PIC16FXXX devices are:

• udata\_shr

### **Program Functional Description**

This program demonstrates the utility of the udata\_shr directive, which is used when generating an object file. This directive declares the beginning of a section of shared uninitialized data. This directive is used to declare variables that are allocated in RAM that is shared across all RAM banks (i.e. unbanked RAM.)

### **Commented Code Listing**

list p=16f877 ;Select the device. #include <p16f877.inc> ;Include standard header file ;for the selected device. shared\_data udata\_shr ;Declares the beginning of a data ;section named 'shared data', var res 1; which is shared by all banks. 'var' is the location which can ;be accessed irrespective of :banksel bits. bank0\_var udata 0X20 ;Declares beginning of a data var0 res 1 ;section named 'bank0 var', :which is in bank0. var0 is ;allocated the address 0x20. bank1 var udata 0xa0 ;Declares beginning of a data var1 res 1 ;section named 'bank1\_var', ;which is in bank1. var1 is ;allocated the addess 0xa0 bank2\_var udata 0x120 ;Declares beginning of a data

var2 res 1 ;section named 'bank2\_var', ;which is in bank2. var2 is ;allocated the addess 0x120 bank3 var udata 0x1a0 ;Declares beginning of a data var3 res 1 ;section named 'bank3 var', ;which is in bank3. var3 is ;allocated the addess 0x1a0 RST CODE H'0'; The code section named RST is ;placed at H'0'. The instruction ;'goto start' is placed in :code section RST. goto start ;Jumps to the location labelled :'start'. INTRT CODE H'4' ;The code section named INTRT is ;placed at H'4'. The instruction ;'goto service\_int' is placed ;in code sectionINTRT. goto service\_int ;Jumps to the location labelled ;'service int'. PGM CODE ; This is the beginning of the code ;section named PGM. It is a ;relocatable code section since ;no absolute address is given along start ;with directive 'CODE'. banksel var0 ;Select bank0. movlw H'00' movwf var ;var is accessible from bank0. banksel var1 ;Select bank1. movlw H'01' movwf var ;var is accessible from bank1 ;also. banksel var2 ;Select bank2. movlw H'02' movwf var ;var is accessible from bank2 ;also. banksel var3 ;Select bank3. movlw H'03'

movwf var ;var is accessible from bank3 ;also. goto \$ service\_int retfie end



Microchip Technology Inc. <u>Microchip's Web Site</u> Voice: (480) 792-7200 Fax: (480) 899-9210 <u>Microchip's E-mail Address</u>