

# **16-bit MCU and DSC Programmer's Reference Manual**

High-Performance Microcontrollers (MCU) and Digital Signal Controllers (DSC)

© 2005-2011 Microchip Technology Inc.

#### Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION. QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

# QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV ISO/TS 16949:2009

#### Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC<sup>32</sup> logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

 $\ensuremath{\mathsf{SQTP}}$  is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2005-2011, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

ISBN: 978-1-61341-357-9

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



# **Table of Contents**

# <u>PAGE</u>

SECTION 1. INTRODUCTION	5
Introduction	
Manual Objective	
Development Support	
Style and Symbol Conventions	
Instruction Set Symbols	
SECTION 2. PROGRAMMER'S MODEL	
16-bit MCU and DSC Core Architecture Overview	
Programmer's Model	
Working Register Array	
Default Working Register (WREG)	
Software Stack Frame Pointer	
Software Stack Pointer	
Stack Pointer Limit Register (SPLIM)	
Accumulator A and Accumulator B (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
Program Counter	
TBLPAG Register	19
PSVPAG Register (PIC24F, PIC24H, dsPIC30F and dsPIC33F)	
RCOUNT Register	20
DCOUNT Register (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
DOSTART Register (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
DOEND Register (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
STATUS Register	
Core Control Register	
Shadow Registers	
DO Stack (dsPIC33E Devices)	25
SECTION 3. INSTRUCTION SET OVERVIEW	37
Introduction	38
Instruction Set Overview	
Instruction Set Summary Tables	
SECTION 4. INSTRUCTION SET DETAILS	51
Data Addressing Modes	
Program Addressing Modes	
Instruction Stalls	
Byte Operations	64
Word Move Operations	66
Using 10-bit Literal Operands	69
Software Stack Pointer and Frame Pointer	70
Conditional Branch Instructions	
Z Status Bit	77
Assigned Working Register Usage	
DSP Data Formats (dsPIC30F, dsPIC33F and dsPIC33E Devices)	81
Accumulator Usage (dsPIC30F, dsPIC33F and dsPIC33E Devices)	83
Accumulator Access (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
DSP MAC Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
DSP Accumulator Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
Scaling Data with the FBCL Instruction (dsPIC30F, dsPIC33F and dsPIC33E Devices)	
Normalizing the Accumulator with the FBCL Instruction (dsPIC30F, dsPIC33F and dsPIC33E Devices)	

Extended-precison Arithmetic using mixed-sign	multiplications (dsPIC33E only) 91
SECTION 5. INSTRUCTION DESCRIPTIONS	93
Instruction Symbols	
Instruction Encoding Field Descriptors Introduction	
Instruction Description Example	
Instruction Descriptions	
SECTION 6. BUILT-IN FUNCTIONS	445
Introduction Built-in Function List	
SECTION 7. REFERENCE	483
Instruction Bit Map	
Instruction Set Summary Table	
Revision History	
SECTION 8. INDEX	497
SECTION 9. WORLDWIDE SALES AND SERVICE	502





# Section 1. Introduction

# HIGHLIGHTS

This section of the manual contains the following major topics:

1.1	Introduction	6
1.2	Manual Objective	6
1.3	Development Support	6
1.4	Style and Symbol Conventions	7
1.5	Instruction Set Symbols	8

## 1.1 INTRODUCTION

Microchip Technology focuses on products for the embedded control market. Microchip is a leading supplier of the following devices and products:

- 8-bit General Purpose Microcontrollers (PIC® MCUs)
- 16-bit Digital Signal Controllers (dsPIC® DSCs)
- 16-bit and 32-bit Microcontrollers (MCUs)
- Speciality and Standard Nonvolatile Memory Devices
- Security Devices (KEELOQ<sup>®</sup> Security ICs)
- Application-specific Standard Products

Information about these devices and products, with corresponding technical documentation, is available on the Microchip web site (www.microchip.com).

## 1.2 MANUAL OBJECTIVE

This manual is a software developer's reference for the 16-bit MCU and DSC device families. It describes the Instruction Set in detail and also provides general information to assist the development of software for the 16-bit MCU and DSC device families.

This manual does not include detailed information about the core, peripherals, system integration or device-specific information. The user should refer to the specific device family reference manual for information about the core, peripherals and system integration. For device-specific information, the user should refer to the specific device data sheets. The information that can be found in the data sheets includes:

- Device memory map
- Device pinout and packaging details
- · Device electrical specifications
- · List of peripherals included on the device

Code examples are given throughout this manual. These examples are valid for any device in the 16-bit MCU and DSC families.

## 1.3 DEVELOPMENT SUPPORT

Microchip offers a wide range of development tools that allow users to efficiently develop and debug application code. Microchip's development tools can be broken down into four categories:

- Code generation
- Hardware/Software debug
- Device programmer
- Product evaluation boards

Information about the latest tools, product briefs and user guides can be obtained from the Microchip web site (www.microchip.com) or from your local Microchip Sales Office.

Microchip offers other reference tools to speed the development cycle. These include:

- Application Notes
- Reference Designs
- Microchip web site
- Local Sales Offices with Field Application Support
- Corporate Support Line

The Microchip web site also lists other sites that may be useful references.

# 1.4 STYLE AND SYMBOL CONVENTIONS

Throughout this document, certain style and font format conventions are used. Table 1-1 provides a description of the conventions used in this document.

Table 1-1:Document Conventions

Symbol or Term	Description
set	To force a bit/register to a value of logic '1'.
clear	To force a bit/register to a value of logic '0'.
Reset	1. To force a register/bit to its default state.
	<ol> <li>A condition in which the device places itself after a device Reset occurs. Some bits will be forced to '0' (such as interrupt enable bits), while others will be forced to '1' (such as the I/O data direction bits).</li> </ol>
0xnnnn	Designates the number 'nnnn' in the hexadecimal number system. These conventions are used in the code examples. For example, 0x013F or 0xA800.
: (colon)	Used to specify a range or the concatenation of registers/bits/pins. One example is ACCAU:ACCAH:ACCAL, which is the concatenation of three registers to form the 40-bit Accumulator. Concatenation order (left-right) usually specifies a positional relationship (MSb to LSb, higher to lower).
<>	Specifies bit locations in a particular register. One example is SR<7:5> (or IPL<2:0>), which specifies the register and associated bits or bit locations.
LSb, MSb	Indicates the Least Significant or Most Significant bit in a field.
LSB, MSB	Indicates the Least/Most Significant Byte in a field of bits.
lsw, msw	Indicates the least/most significant word in a field of bits
Courier New Font	Used for code examples, binary numbers and for Instruction mnemonics in the text.
Times New Roman Font, Italic	Used for equations and variables.
Times New Roman Font, Bold Italic	Used in explanatory text for items called out from a figure, equation, or example.
Note:	A Note presents information that we want to re-emphasize, either to help you avoid a common pitfall, or make you aware of operating differences between some device family members. A Note can be in a box, or when used in a table or figure, it is located at the bottom of the table or figure.

1

# 1.5 INSTRUCTION SET SYMBOLS

The summary tables in Section 3.2 "Instruction Set Overview" and Section 7.2 "Instruction Set Summary Table", and the instruction descriptions in Section 5.4 "Instruction Descriptions" utilize the symbols shown in Table 1-2.

Table 1-2:	Symbols Used in Instruc	tion Summary Tables and Descriptions
------------	-------------------------	--------------------------------------

Symbol <sup>(1)</sup>	Description
{ }	Optional field or operation
[text]	The location addressed by text
(text)	The contents of text
#text	The literal defined by text
a ∈ [b, c, d]	"a" must be in the set of [b, c, d]
<n:m></n:m>	Register bit field
{label:}	Optional label name
Acc	Accumulator A or Accumulator B
AWB	Accumulator Write Back
bit4	4-bit wide bit position (0:7 in Byte mode, 0:15 in Word mode)
Expr	Absolute address, label or expression (resolved by the linker)
f	File register address
lit1	1-bit literal (0:1)
lit4	4-bit literal (0:15)
lit5	5-bit literal (0:31)
lit8	8-bit literal (0:255)
lit10	10-bit literal (0:255 in Byte mode, 0:1023 in Word mode)
lit14	14-bit literal (0:16383)
lit16	16-bit literal (0:65535)
lit23	23-bit literal (0:8388607)
Slit4	Signed 4-bit literal (-8:7)
Slit6	Signed 6-bit literal (-32:31) (range is limited to -16:16)
Slit10	Signed 10-bit literal (-512:511)
Slit16	Signed 16-bit literal (-32768:32767)
TOS	Top-of-Stack
Wb	Base working register
Wd	Destination working register (direct and indirect addressing)
Wdo	Destination working register (direct and indirect addressing, including indirect addressing with offset)
Wm, Wn	Working register divide pair (dividend, divisor)
Wm * Wm	Working register multiplier pair (same source register)
Wm * Wn	Working register multiplier pair (different source registers)
Wn	Both source and destination working register (direct addressing)
Wnd	Destination working register (direct addressing)
Wns	Source working register (direct addressing)
WREG	Default working register (assigned to W0)
Ws	Source working register (direct and indirect addressing)
Wso	Source working register (direct and indirect addressing, including indirect addressing with offset)
Wx	Source Addressing mode and working register for X data bus prefetch
Wxd	Destination working register for X data bus prefetch
Wy	Source Addressing mode and working register for Y data bus prefetch
Wyd	Destination working register for Y data bus prefetch

**Note 1:** The range of each symbol is instruction dependent. Refer to **Section 5. "Instruction Descriptions"** for the specific instruction range.



# Section 2. Programmer's Model

# HIGHLIGHTS

This section of the manual contains the following major topics:

16-bit MCU and DSC Core Architecture Overview	. 10
Programmer's Model	. 14
Working Register Array	. 18
Default Working Register (WREG)	. 18
Software Stack Frame Pointer	. 18
	Programmer's Model Working Register Array Default Working Register (WREG)

2

# Programmer's Model

# 2.1 16-BIT MCU AND DSC CORE ARCHITECTURE OVERVIEW

This section provides an overview of the 16-bit architecture features and capabilities for the following families of devices:

- 16-bit Microcontrollers (MCU):
  - PIC24F
  - PIC24H
  - PIC24E
- 16-bit Digital Signal Controllers (DSC):
  - dsPIC30F
  - dsPIC33F
  - dsPIC33E

#### 2.1.1 Features Specific to 16-bit MCU and DSC Core

The core of the 16-bit MCU and DSC devices is a 16-bit (data) modified Harvard architecture with an enhanced instruction set. The core has a 24-bit instruction word, with an 8-bit Op code field. The Program Counter (PC) is 23 bits wide and addresses up to 4M x 24 bits of user program memory space. An instruction prefetch mechanism is used to help maintain throughput and provides predictable execution. The majority of instructions execute in a single cycle.

#### 2.1.1.1 REGISTERS

The 16-bit MCU and DSC devices have sixteen 16-bit working registers. Each of the working registers can act as a data, address or offset register. The 16th working register (W15) operates as a software Stack Pointer for interrupts and calls.

#### 2.1.1.2 INSTRUCTION SET

The instruction set is almost identical for the 16-bit MCU and DSC architectures. The instruction set includes many Addressing modes and was designed for optimum C compiler efficiency.

#### 2.1.1.3 DATA SPACE ADDRESSING

The data space can be addressed as 32K words or 64 Kbytes. The upper 32 Kbytes of the data space memory map can optionally be mapped into program space at any 16K program word boundary, which is a feature known as Program Space Visibility (PSV). The program to data space mapping feature lets any instruction access program space as if it were the data space, which is useful for storing data coefficients.

**Note:** Some devices families support Extended Data Space (EDS) addressing. See the specific device data sheet and family reference manual for more details on this feature.

#### 2.1.1.4 ADDRESSING MODES

The core supports Inherent (no operand), Relative, Literal, Memory Direct, Register Direct, Register Indirect, and Register Offset Addressing modes. Each instruction is associated with a predefined Addressing mode group, depending upon its functional requirements. As many as seven Addressing modes are supported for each instruction.

For most instructions, the CPU is capable of executing a data (or program data) memory read, a working register (data) read, a data memory write and a program (instruction) memory read per instruction cycle. As a result, 3-operand instructions can be supported, allowing A + B = C operations to be executed in a single cycle.

#### 2.1.1.5 ARITHMETIC AND LOGIC UNIT

A high-speed, 17-bit by 17-bit multiplier is included to significantly enhance the core's arithmetic capability and throughput. The multiplier supports Signed, Unsigned, and Mixed modes, as well as 16-bit by 16-bit, or 8-bit by 8-bit integer multiplication. All multiply instructions execute in a single cycle.

The 16-bit Arithmetic Logic Unit (ALU) is enhanced with integer divide assist hardware that supports an iterative non-restoring divide algorithm. It operates in conjunction with the REPEAT instruction looping mechanism, and a selection of iterative divide instructions, to support 32-bit (or 16-bit) divided by 16-bit integer signed and unsigned division. All divide operations require 19 cycles to complete, but are interruptible at any cycle boundary.

#### 2.1.1.6 EXCEPTION PROCESSING

The 16-bit MCU and DSC devices have a vectored exception scheme with support for up to 8 sources of non-maskable traps and up to 246 interrupt sources. In both families, each interrupt source can be assigned to one of seven priority levels.

#### 2.1.2 PIC24E and dsPIC33E Features

In addition to the information provided in Section 2.1.1 "Features Specific to 16-bit MCU and DSC Core", this section describes the enhancements that are available in the PIC24E and dsPIC33E families of devices.

#### 2.1.2.1 DATA SPACE ADDRESSING

The Base Data Space address is used in conjunction with a read or write page register (DSRPAG or DSWPAG) to form an Extended Data Space (EDS) address, which can also be used for PSV access. The EDS can be addressed as 8 M words or 16 Mbytes. Refer to **Section 3.** "Data **Memory**" (DS70595) in the "dsPIC33E/PIC24E Family Reference Manual" for more details on EDS, PSV, and table accesses.

Note:	Some PIC24F devices also support Extended Data Space. Refer to Section 44.
	"CPU with EDS" (DS39732) and Section 45. "Data Memory with EDS"
	(DS39733) of the PIC24F Family Reference Manual for details.

#### 2.1.2.2 AUTOMATIC MIXED-SIGN MULTIPLICATION MODE (dsPIC33E ONLY)

In addition to signed and unsigned DSP multiplications, dsPIC33E devices support mixed-sign (unsigned-signed and signed-unsigned) multiplications without the need to dynamically reconfigure the multiplication mode and shift data to account for the difference in operand formats. This mode is particularly beneficial for executing extended-precision (32-bit and 64-bit) algorithms. Besides DSP instructions, MCU multiplication (MUL) instructions can also utilize either accumulator as a result destination, thereby enabling faster extended-precision arithmetic. Refer to 4.10.1 "Implied DSP Operands (dsPIC30F, dsPIC33F and dsPIC33E Devices)" and 4.18 "Extended-precison Arithmetic using mixed-sign multiplications (dsPIC33E only)" for more details on mixed-sign DSP multiplications.

#### 2.1.2.3 MCU MULTIPLICATIONS WITH 16-BIT RESULT

16x16-bit MUL instructions include an option to store the product in a single 16-bit working register rather than a pair of registers. This feature helps free up a register for other purposes, in cases where the numbers being multiplied are small in magnitude and therefore expected to provide a 16-bit result. See the individual MUL instruction descriptions in **5.4** "Instruction Descriptions" for more details.

#### 2.1.2.4 HARDWARE STACK FOR DO LOOPS (dsPIC33E ONLY)

The single-level DO loop shadow register-set has been replaced by 4-level deep DO loop hardware stack. This provides automatic DO loop register save/restore for up to 3 levels of DO loop nesting, resulting in more efficient implementation of nested loops. Refer to 2.19 "DO Stack (dsPIC33E Devices)" for more details on DO loop nesting in dsPIC33E devices.

#### 2.1.2.5 DSP CONTEXT SWITCH SUPPORT (dsPIC33E ONLY)

In dsPIC33E devices, the DSP overflow and saturation status bits are writable. This allows the state of the DSP Engine to be efficiently saved and restored while switching between DSP tasks. See **2.16.4** "DSP ALU Status Bits (dsPIC30F, dsPIC33F and dsPIC33E Devices)" for more details on DSP status bits.

#### 2.1.2.6 EXTENDED CALL AND GOTO INSTRUCTIONS

The new CALL.L Wn and GOTO.L Wn instructions extend the capabilities of the CALL Wn and GOTO Wn by enabling 32-bit addresses for computed branch/call destinations. In these enhanced instructions, the destination address is provided by a pair of working registers rather than a single 16-bit register. See the CALL.L and GOTO.L instruction descriptions in **5.4 "Instruction Descriptions"** for more details.

#### 2.1.2.7 COMPARE-BRANCH INSTRUCTIONS

dsPIC33E/PIC24E devices feature conditional Compare-Branch (CPBxx) instructions. These instructions extend the capabilities of the Compare-Skip (CPSxx) instructions by allowing branches rather than only skipping over a single instruction. See the CPBEQ, CPBNE, CPBGT and CPBLT instruction descriptions in **5.4** "Instruction Descriptions" for more details on compare-branch instructions.

#### 2.1.3 dsPIC30F, dsPIC33F, and dsPIC33E Features

In addition to the information provided in Section 2.1.1 "Features Specific to 16-bit MCU and DSC Core", this section describes the DSP enhancements that are available in the dsPIC30F, dsPIC33F, and dsPIC33E families of devices.

#### 2.1.3.1 PROGRAMMING LOOP CONSTRUCTS

Overhead free program loop constructs are supported using the DO instruction, which is interruptible.

#### 2.1.3.2 DSP INSTRUCTION CLASS

The DSP class of instructions.are seamlessly integrated into the architecture and execute from a single execution unit.

#### 2.1.3.3 DATA SPACE ADDRESSING

The data space is split into two blocks, referred to as X and Y data memory. Each memory block has its own independent Address Generation Unit (AGU). The MCU class of instructions operate solely through the X memory AGU, which accesses the entire memory map as one linear data space. The DSP dual source class of instructions operates through the X and Y AGUs, which splits the data address space into two parts. The X and Y data space boundary is arbitrary and device-specific.

#### 2.1.3.4 MODULO AND BIT-REVERSED ADDRESSING

Overhead-free circular buffers (modulo addressing) are supported in both X and Y address spaces. The modulo addressing removes the software boundary checking overhead for DSP algorithms. Furthermore, the X AGU circular addressing can be used with any of the MCU class of instructions. The X AGU also supports bit-reverse addressing, to greatly simplify input or output data reordering for radix-2 FFT algorithms.

#### 2.1.3.5 DSP ENGINE

The DSP engine features a high-speed, 17-bit by 17-bit multiplier, a 40-bit ALU, two 40-bit saturating accumulators and a 40-bit bidirectional barrel shifter. The barrel shifter is capable of shifting a 40-bit value, up to 16 bits right, or up to 16 bits left, in a single cycle. The DSP instructions operate seamlessly with all other instructions and have been designed for optimal real-time performance. The MAC instruction and other associated instructions can concurrently fetch two data operands from memory while multiplying two working registers. This requires that

the data space be split for these instructions and linear for all others. This is achieved in a transparent and flexible manner through dedicating certain working registers to each address space.

#### 2.1.3.6 EXCEPTION PROCESSING

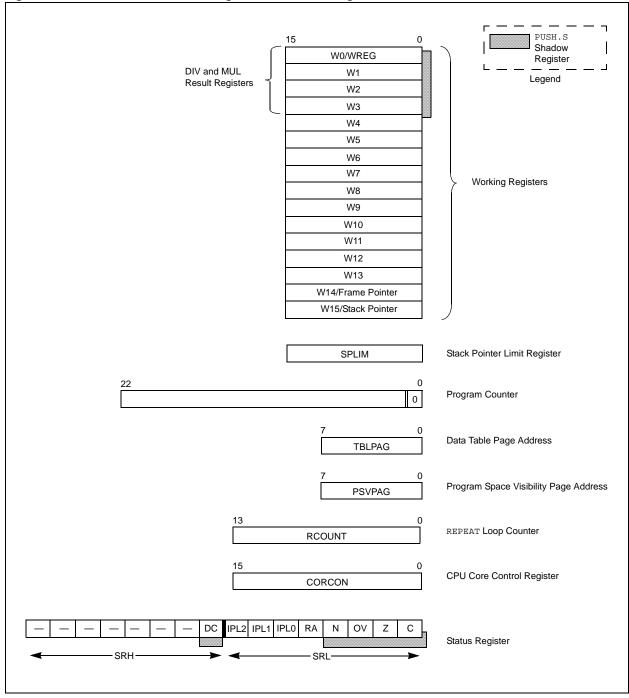
The dsPIC30F devices have a vectored exception scheme with support for up to 8 sources of non-maskable traps and up to 54 interrupt sources. The dsPIC33F and dsPIC33E have a similar exception scheme, but support up to 118, and up to 246 interrupt sources, respectively. In all three families, each interrupt source can be assigned to one of seven priority levels.

Refer to **Section 6** and **28** " **Interrupts**" of the dsPIC30F Family Reference Manual, **Sections 6**, **29**, **32**, **41**, **47** and **53** of the dsPIC33F/PIC24H Family Reference Manual and **Section 6** of the dsPIC33E/PIC24E Family Reference Manual, for more details on Exception Processing.

2

# 2.2 PROGRAMMER'S MODEL

Figure 2-1 through Figure 2-4 show the programmer's model diagrams for the 16-bit MCU and DSC families of devices.



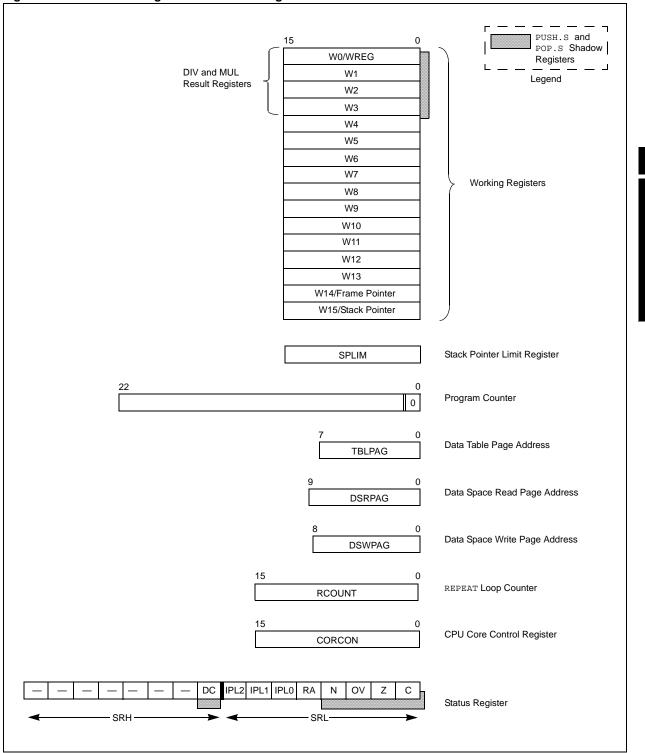
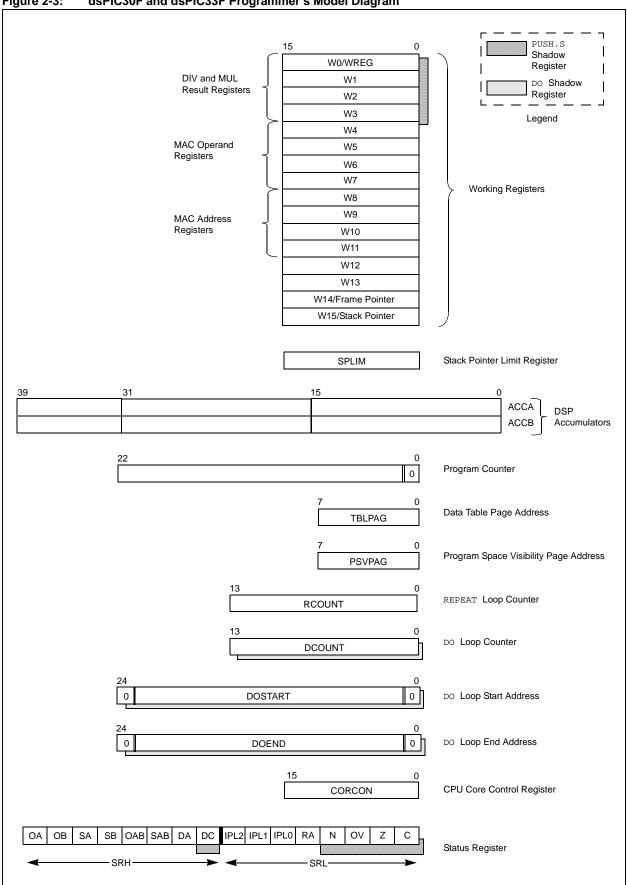


Figure 2-2: PIC24E Programmer's Model Diagram

Programmer's Model





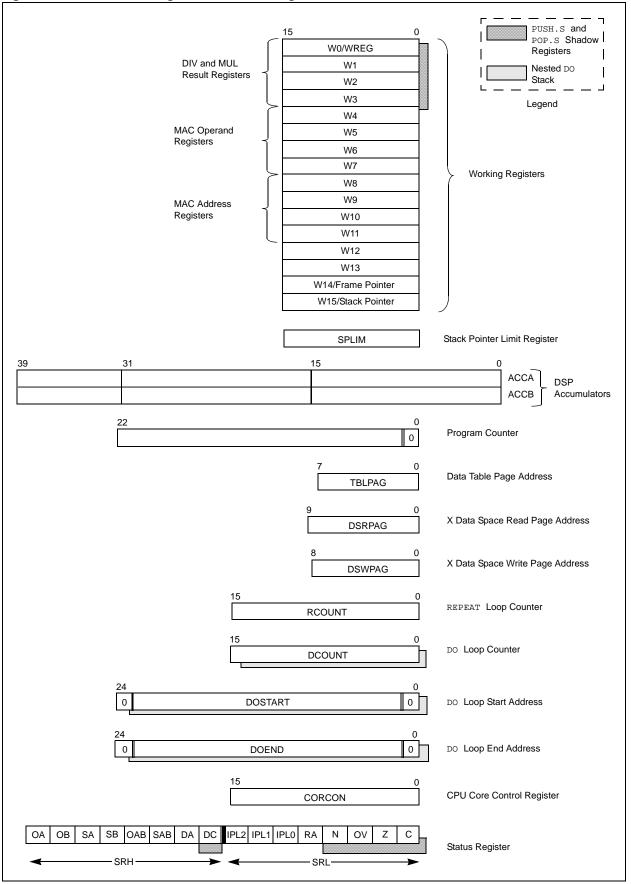


Figure 2-4: dsPIC33E Programmer's Model Diagram

Programmer's Model All registers in the programmer's model are memory mapped and can be manipulated directly by the instruction set. A description of each register is provided in Table 2-1.

Note: Unless otherwise specified, the Programmer's Model Register Descriptions in Table 2-1 apply to all MCU and DSC device families.

Table 2-1. Programmer's Model Register Descriptions				
Description				
CPU Core Configuration register				
23-bit Program Counter				
Program Space Visibility Page Address register				
Extended Data Space (EDS) Read Page register				
Extended Data Space (EDS) Write Page register				
REPEAT Loop Count register				
Stack Pointer Limit Value register				
ALU and DSP Engine STATUS register				
Table Memory Page Address register				
Working register array				
40-bit DSP Accumulators				
DO Loop Count register				
DO Loop Start Address register				
DO Loop End Address register				

Table 2-1: Programmer's Model Register Descriptions

**Note 1:** This register is only available on PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

- 2: This register is only available on PIC24E and dsPIC33E devices.
- **3:** This register is only available on dsPIC30F, dsPIC33F, and dsPIC33E devices.

# 2.3 WORKING REGISTER ARRAY

The 16 working (W) registers can function as data, address or offset registers. The function of a W register is determined by the instruction that accesses it.

Byte instructions, which target the working register array, only affect the Least Significant Byte (LSB) of the target register. Since the working registers are memory mapped, the Least *and* Most Significant Bytes can be manipulated through byte-wide data memory space accesses.

# 2.4 DEFAULT WORKING REGISTER (WREG)

The instruction set can be divided into two instruction types: working register instructions and file register instructions. The working register instructions use the working register array as data values or as addresses that point to a memory location. In contrast, file register instructions operate on a specific memory address contained in the instruction opcode.

File register instructions that also utilize a working register do not specify the working register that is to be used for the instruction. Instead, a default working register (WREG) is used for these file register instructions. Working register, W0, is assigned to be the WREG. The WREG assignment is not programmable.

# 2.5 SOFTWARE STACK FRAME POINTER

A frame is a user-defined section of memory in the stack, used by a function to allocate memory for local variables. W14 has been assigned for use as a Stack Frame Pointer with the link (LNK) and unlink (ULNK) instructions. However, if a Stack Frame Pointer and the LNK and ULNK instructions are not used, W14 can be used by any instruction in the same manner as all other W registers. On dsPIC33E and PIC24E devices, a Stack Frame Active (SFA) Status bit is used to support nested stack frames. See Section 4.7.2 "Software Stack Frame Pointer" for detailed information about the Frame Pointer.

# 2.6 SOFTWARE STACK POINTER

W15 serves as a dedicated Software Stack Pointer, and will be automatically modified by function calls, exception processing and returns. However, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies reading, writing and manipulating the Stack Pointer. Refer to **Section 4.7.1** "Software Stack Pointer" for detailed information about the Stack Pointer.

## 2.7 STACK POINTER LIMIT REGISTER (SPLIM)

The SPLIM is a 16-bit register associated with the Stack Pointer. It is used to prevent the Stack Pointer from overflowing and accessing memory beyond the user allocated region of stack memory. Refer to **Section 4.7.3** "**Stack Pointer Overflow**" for detailed information about the SPLIM.

# 2.8 ACCUMULATOR A AND ACCUMULATOR B (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

Accumulator A (ACCA) and Accumulator B (ACCB) are 40-bit wide registers, utilized by DSP instructions to perform mathematical and shifting operations. Each accumulator is composed of 3 memory mapped registers:

- AccxU (bits 39-32)
- AccxH (bits 31-16)
- AccxL (bits 15-0)

In dsPIC33E devices, Accumulator A and Accumulator B can also be used as destination registers in MCU MUL.xx instructions. This helps reduce the execution time of extended-precision arithmetic operations.

Refer to Section 4.12 "Accumulator Usage (dsPIC30F, dsPIC33F and dsPIC33E Devices)" for details on using ACCA and ACCB.

# 2.9 PROGRAM COUNTER

The Program Counter (PC) is 23 bits wide. Instructions are addressed in the 4M x 24-bit user program memory space by PC<22:1>, where PC<0> is always set to '0' to maintain instruction word alignment and provide compatibility with data space addressing. This means that during normal instruction execution, the PC increments by 2.

Program memory located at 0x800000 and above is utilized for device configuration data, Unit ID and Device ID. This region is not available for user code execution and the PC can not access this area. However, one may access this region of memory using table instructions. For details on accessing the configuration data, Unit ID, and Device ID, refer to the specific device family reference manual.

# 2.10 TBLPAG REGISTER

The TBLPAG register is used to hold the upper 8 bits of a program memory address during table read and write operations. Table instructions are used to transfer data between program memory space and data memory space. For details on accessing program memory with the table instructions, refer to the family reference manual of the specific device.

# 2.11 PSVPAG REGISTER (PIC24F, PIC24H, dsPIC30F AND dsPIC33F)

Program space visibility allows the user to map a 32-Kbyte section of the program memory space into the upper 32 Kbytes of data address space. This feature allows transparent access of constant data through instructions that operate on data memory. The PSVPAG register selects the 32-Kbyte region of program memory space that is mapped to the data address space. For details on program space visibility, refer to the specific device family reference manual. Programmer's

# 2.12 RCOUNT REGISTER

The 14-bit RCOUNT register (16-bit for PIC24E and dsPIC33E devices) register contains the loop counter for the REPEAT instruction. When a REPEAT instruction is executed, RCOUNT is loaded with the repeat count of the instruction, either "lit14" for the "REPEAT #lit14" instruction ("lit15" for the "REPEAT #lit15" instruction for PIC24E and dsPIC33E devices), or the 14 LSb of the Wn register for the "REPEAT Wn" instruction (entire Wn for PIC24E and dsPIC33E devices). The REPEAT loop will be executed RCOUNT + 1 time.

- **Note 1:** If a REPEAT loop is executing and gets interrupted, RCOUNT may be cleared by the Interrupt Service Routine to break out of the REPEAT loop when the foreground code is re-entered.
  - 2: Refer to the specific device family reference manual for complete details about REPEAT loops.

# 2.13 DCOUNT REGISTER (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The 14-bit DCOUNT register (16-bit for dsPIC33E devices) contains the loop counter for hardware DO loops. When a DO instruction is executed, DCOUNT is loaded with the loop count of the instruction, either "lit14" for the "DO #lit14, Expr" instruction ("lit15" for the "DO #lit15, Expr" instruction for dsPIC33E devices) or the 14 LSb of the Ws register for the "DO Ws, Expr" instruction (entire Wn for dsPIC33E devices). The DO loop will be executed DCOUNT + 1 times.

- Note 1: In dsPIC30F and dsPIC33F devices, the DCOUNT register contains a shadow register. See Section 2.18 "Shadow Registers" for information on shadow registers.
  - 2: The dsPIC33E devices have a 4-level-deep, nested DO stack instead of a shadow register.
  - **3:** Refer to the specific device family reference manual for complete details about DO loops.

# 2.14 DOSTART REGISTER (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The DOSTART register contains the starting address for a hardware DO loop. When a DO instruction is executed, DOSTART is loaded with the address of the instruction that follows the DO instruction. This location in memory is the start of the DO loop. When looping is activated, program execution continues with the instruction stored at the DOSTART address after the last instruction in the DO loop is executed. This mechanism allows for zero overhead looping.

- Note 1: For dsPIC30F and dsPIC33F devices, DOSTART has a shadow register. See Section 2.18 "Shadow Registers" for information on shadowing.
  - 2: The dsPIC33E devices have a 4-level-deep, nested DO stack instead of a shadow register. The DOSTART register is read-only in dsPIC33E devices.
  - **3:** Refer to the specific device family reference manual for complete details about DO loops.

# 2.15 DOEND REGISTER (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The DOEND register contains the ending address for a hardware DO loop. When a DO instruction is executed, DOEND is loaded with the address specified by the expression in the DO instruction. This location in memory specifies the last instruction in the DO loop. When looping is activated and the instruction stored at the DOEND address is executed, program execution will continue from the DO loop start address (stored in the DOSTART register).

- Note 1: For dsPIC30F and dsPIC33F devices, DOEND has a shadow register. See Section 2.18 "Shadow Registers" for information on shadow registers.
  - The dsPIC33E devices have a 4-level-deep, nested DO stack instead of a shadow register.
  - **3:** Refer to the specific device family reference manual for complete details about DO loops.

## 2.16 STATUS REGISTER

The 16-bit STATUS register maintains status information for the instructions which have been executed most recently. Operation Status bits exist for MCU operations, loop operations and DSP operations. Additionally, the STATUS register contains the CPU Interrupt Priority Level bits, IPL<2:0>, which are used for interrupt processing.

Depending on the MCU and DSC family, one of the following STATUS registers is used:

- Register 2-1 for PIC24F, PIC24H, and PIC24E devices
- Register 2-2 for dsPIC30F and dsPIC33F devices
- Register 2-3 for dsPIC33E devices

#### 2.16.1 MCU ALU Status Bits

The MCU operation Status bits are either affected or used by the majority of instructions in the instruction set. Most of the logic, math, rotate/shift and bit instructions modify the MCU Status bits after execution, and the conditional Branch instructions use the state of individual Status bits to determine the flow of program execution. All conditional branch instructions are listed in **Section 4.8 "Conditional Branch Instructions"**.

The Carry (C), Zero (Z), Overflow (OV), Negative (N), and Digit Carry (DC) bits show the immediate status of the MCU ALU by indicating whether an operation has resulted in a Carry, Zero, Overflow, <u>Negative</u> result, or Digit Carry. When a subtract operation is performed, the C flag is used as a Borrow flag.

The Z Status bit is useful for extended precision arithmetic. The Z Status bit functions like a normal Z flag for all instructions except those that use a carry or borrow input (ADDC, CPB, SUBB and SUBBR). See Section 4.9 "Z Status Bit" for more detailed information.

- **Note 1:** All MCU bits are shadowed during execution of the PUSH.S instruction and they are restored on execution of the POP.S instruction.
  - 2: All MCU bits, except the DC flag (which is not in the SRL), are stacked during exception processing (see Section 4.7.1 "Software Stack Pointer").

#### 2.16.2 REPEAT Loop Status Bit

The REPEAT Active bit (RA) is used to indicate when looping is active. The RA flag indicates that a REPEAT instruction is being executed, and it is only affected by the REPEAT instructions. The RA flag is set to '1' when the instruction being repeated begins execution, and it is cleared when the instruction being repeated completes execution for the last time.

Since the RA flag is also read-only, it may not be directly cleared. However, if a REPEAT or its target instruction is interrupted, the Interrupt Service Routine may clear the RA flag of the SRL, which resides on the stack. This action will disable looping once program execution returns from the Interrupt Service Routine, because the restored RA will be '0'.

2

#### 2.16.3 DO Active bit (DA) (dsPIC30F, dsPIC33F and dsPIC33E Devices)

The DO Active bit (DA) is used to indicate when looping is active. The DO instructions affect the DA flag, which indicates that a DO loop is active. The DA flag is set to '1' when the first instruction of the DO loop is executed, and it is cleared when the last instruction of the loop completes final execution.

The DA flag is read-only. This means that looping is not initiated by writing a '1' to DA, nor is it terminated by writing a '0' to DA. If a DO loop must be terminated prematurely, the EDT bit, CORCON<11>, should be used.

# 2.16.4 DSP ALU Status Bits (dsPIC30F, dsPIC33F and dsPIC33E Devices)

The high byte of the STATUS Register (SRH) is used by the DSP class of instructions, and it is modified when data passes through one of the adders. The SRH provides status information about overflow and saturation for both accumulators. The Saturate A, Saturate B, Overflow A and Overflow B (SA, SB, OA, OB) bits provide individual accumulator status, while the Saturate AB and Overflow AB (SAB, OAB) bits provide combined accumulator status. The SAB and OAB bits provide an efficient method for the software developer to check the register for saturation or overflow.

The OA and OB bits are used to indicate when an operation has generated an overflow into the guard bits (bits 32 through 39) of the respective accumulator. This condition can only occur when the processor is in Super Saturation mode, or if saturation is disabled. It indicates that the operation has generated a number which cannot be represented with the lower 31 bits of the accumulator. The OA and OB bits are writable in dsPIC33E devices.

The SA and SB bits are used to indicate when an operation has generated an overflow out of the MSb of the respective accumulator. The SA and SB bits are active, regardless of the Saturation mode (Disabled, Normal or Super) and may be considered "sticky". Namely, once the SA or SB bit is set to '1', it can only be cleared manually by software, regardless of subsequent DSP operations. When it is required, the BCLR instruction can be used to clear the SA or SB bit.

In addition, the SA and SB bits can be set by software in dsPIC33E devices, enabling efficient context state switching.

For convenience, the OA and OB bits are logically ORed together to form the OAB flag, and the SA and SB bits are logically ORed to form the SAB flag. These cumulative Status bits provide efficient overflow and saturation checking when an algorithm is implemented. Instead of interrogating the OA and the OB bits independently for arithmetic overflows, a single check of OAB can be performed. Likewise, when checking for saturation, SAB may be examined instead of checking both the SA and SB bits. Note that clearing the SAB flag will clear both the SA and SB bits.

#### 2.16.5 Interrupt Priority Level Status Bits

The three Interrupt Priority Level (IPL) bits of the SRL, SR<7:5>, and the IPL3 bit, CORCON<3>, set the CPU's IPL which is used for exception processing. Exceptions consist of interrupts and hardware traps. Interrupts have a user-defined priority level between 0 and 7, while traps have a fixed priority level between 8 and 15. The fourth Interrupt Priority Level bit, IPL3, is a special IPL bit that may only be read or cleared by the user. This bit is only set when a hardware trap is activated and it is cleared after the trap is serviced.

The CPU's IPL identifies the lowest level exception which may interrupt the processor. The interrupt level of a pending exception must always be greater than the CPU's IPL for the CPU to process the exception. This means that if the IPL is 0, all exceptions at priority Level 1 and above may interrupt the processor. If the IPL is 7, only hardware traps may interrupt the processor.

When an exception is serviced, the IPL is automatically set to the priority level of the exception being serviced, which will disable all exceptions of equal and lower priority. However, since the IPL field is read/write, one may modify the lower three bits of the IPL in an Interrupt Service Routine to control which exceptions may preempt the exception processing. Since the SRL is stacked during exception processing, the original IPL is always restored after the exception is serviced. If required, one may also prevent exceptions from nesting by setting the NSTDIS bit (INTCON1<15>).

**Note:** For more detailed information on exception processing, refer to the family reference manual of the specific device.

## 2.17 CORE CONTROL REGISTER

For all MCU and DSC devices, the 16-bit CPU Core Control register (CORCON), is used to set the configuration of the CPU. This register provides the ability to map program space into data space.

In addition to setting CPU modes, the CORCON register contains status information about the IPL<3> Status bit, which indicates if a trap exception is being processed.

Depending on the MCU and DSC family, one of the following CORCON registers is used:

- Register 2-4 for PIC24F and PIC24H devices
- Register 2-5 for PIC24E devices
- Register 2-6 for dsPIC30F and dsPIC33F devices
- Register 2-7 for dsPIC33E devices

#### 2.17.1 dsPIC30F, dsPIC33F, and dsPIC33E Specific bits

In addition to setting CPU modes, the following features are available through the CORCON register:

- Set the ACCA and ACCB saturation enable
- Set the Data Space Write Saturation mode
- Set the Accumulator Saturation and Rounding modes
- Set the Multiplier mode for DSP operations
- Terminate DO loops prematurely
- Provide status information about the DO loop nesting level (DL<2:0>)
- Select fixed or variable interrupt latency (dsPIC33E only)

#### 2.17.1.1 PIC24E and dsPIC33E SPECIFIC BITS

A Status bit (SFA) is available that indicates whether the Stack Frame is active.

Note: PIC24E and dsPIC33E devices do not have a PSV control bit, it has been replaced by the SFA bit.

### 2.18 SHADOW REGISTERS

A shadow register is used as a temporary holding register and can transfer its contents to or from the associated host register when instructed. Some of the registers in the programmer's model have a shadow register, which is utilized during the execution of a DO, POP.S, or PUSH.S instruction. Shadow register usage is shown in Table 2-2.

Note: The DO instruction is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

Table 2-2: Automatic Shadow Register Usage

Location	<sub>DO</sub> (1)	POP.S/PUSH.S
DCOUNT <sup>(1)</sup>	Yes	_
DOSTART <sup>(1)</sup>	Yes	—
DOEND <sup>(1)</sup>	Yes	—
STATUS Register – DC, N, OV, Z and C bits	—	Yes
W0-W3		Yes

**Note 1:** The DO shadow registers are only available in dsPIC30F and dsPIC33F devices.

For dsPIC30F and dsPIC33F devices, since the DCOUNT, DOSTART and DOEND registers are shadowed, the ability to nest DO loops without additional overhead is provided. Since all shadow registers are one register deep, up to one level of DO loop nesting is possible. Further nesting of DO loops is possible in software, with support provided by the DO Loop Nesting Level Status bits (DL<2:0>) in the CORCON register (CORCON<10:8>).

**Note:** All shadow registers are one register deep and not directly accessible. Additional shadowing may be performed in software using the software stack.

#### 2.19 DO STACK (dsPIC33E DEVICES)

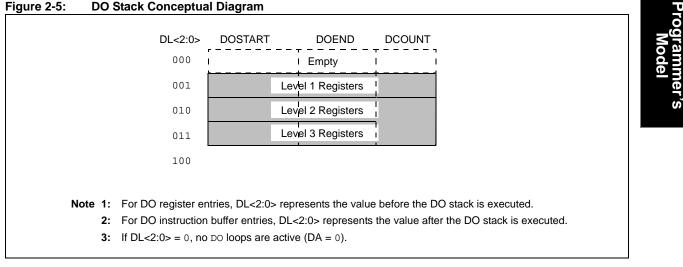
The DO stack is used to preserve the following elements associated with a DO loop underway when another DO loop is encountered (i.e., a nested DO loop).

- DOSTART register value
- DOEND register value
- · DCOUNT register value

Note that the DO level status field (DL<2:0>) also acts as a pointer to address the DO stack. After the DO instruction is executed, the DO level status field (DL<2:0>) points to the next free entry.

The DOSTART, DOEND, and DCOUNT registers each have an associated hardware stack that allows the DO loop hardware to support up to three levels of nesting. A conceptual representation of the DO stack is shown in Figure 2-5.

Figure 2-5: **DO Stack Conceptual Diagram** 



U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0			
_	_	_	_	—		_	DC			
bit 15							bit			
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0			
IPL2 <sup>(1,2)</sup>	IPL1 <sup>(1,2)</sup>	IPL0 <sup>(1,2)</sup>	RA	N	OV	Z	С			
bit 7							bit			
Legend:				U = Unimplen	nented bit, rea	d as '0'				
R = Readab	le bit	W = Writable bit		C = Clearable	e bit					
-n = Value a	t POR	'1' = Bit is set		'0' = Bit is cle	ared	x = Bit is unkr	nown			
hit 15 0	Unimploment	tad. Dood oo f	0'							
bit 15-9 bit 8	-	i <b>ted:</b> Read as ' U Half Carry/Bo								
				(for byte-sized c	data) or 8th low	order bit (for wo	ord-sized dat			
	of the res	ult occurred								
	•		low order bit	(for byte-sized	data) or 8th lov	v order bit (for wo	ord-sized dat			
		ult occurred		· · · · (1 2)						
bit 7-5		PU Interrupt Pri		atus bits()-/ 5). User interrup	ots disabled					
		110 = CPU Interrupt Priority Level is 6 (14) 101 = CPU Interrupt Priority Level is 5 (13)								
		100 = CPU Interrupt Priority Level is 4 (12)								
		011 = CPU Interrupt Priority Level is 3 (11)								
		010 = CPU Interrupt Priority Level is 2 (10) 001 = CPU Interrupt Priority Level is 1 (9)								
		nterrupt Priority								
bit 4		Loop Active bit								
		oop in progress								
1 1 0		oop not in prog	ress							
bit 3	N: MCU ALU 1 = Result wa	0								
		as negative as non-negative	e (zero or pos	itive)						
bit 2		U Overflow bit	- ( F							
		This bit is used for signed arithmetic (2's complement). It indicates an overflow of the magnitude that								
	causes the sign bit to change state.									
	<ul> <li>1 = Overflow occurred for signed arithmetic (in this arithmetic operation)</li> <li>0 = No overflow occurred</li> </ul>									
bit 1		: MCU ALU Zero bit = An operation that affects the Z bit has set it at some time in the past								
	0 = The most recent operation that affects the Z bit has cleared it (i.e., a non-zero result)									
bit 0	<b>C</b> : MCU ALU Carry/Borrow bit									
	1 = A carry-out from the MSb occurred									
	0 = No carry-	out from the M	Sb occurred							
Note 1: T	he IPL<2:0> bits	are concatenat	ted with the If	PL3 bit (CORCO	DN<3>) to form	n the CPU Interr	upt Priority			
L	evel. The value in									
	PL<3> = 1.	uo hito are ratio	المتعطيين بالصماد			) 1 Defender	the forsily			
<b>2:</b> T	110 1PL<2:0> Stat	e IPL<2:0> Status bits are read only when the NSTDIS bit (INTCON1<15>) = 1. Refer to the family								

2: The IPL<2:0> Status bits are read only when the NSTDIS bit (INTCON1<15>) = 1. Refer to the family reference manual of the specific device family to see the associated interrupt register.

R-0	R-0	R/C-0	R/C-0	R-0	R/C-0	R-0	R/W-0	
OA	OB	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	OAB	SAB <sup>(1,2,3)</sup>	DA <sup>(4)</sup>	DC	
bit 15	÷			•			bit	
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	
IPL2 <sup>(5)</sup>	IPL1 <sup>(5)</sup>	IPL0 <sup>(5)</sup>	RA	N N	OV	Z	C	
bit 7		11 20		14		L	bit	
Legend:				<b>.</b>				
R = Readab		W = Writable		C = Clearabl				
-n = Value a	It POR	'1' = Bit is set		'0' = Bit is cle	eared	x = Bit is unkr	nown	
bit 15	1 = Accumula	lator A Overflo ator A overflow	ed					
		ator A has not						
bit 14	1 = Accumula	lator B Overflo ator B overflow ator B has not	ed					
bit 13	1 = Accumula		ted or has bee	en saturated si	ince this bit was	last cleared		
bit 12	<b>SB:</b> Accumul	<ul> <li>0 = Accumulator A is not saturated</li> <li>SB: Accumulator B Saturation bit<sup>(1, 2)</sup></li> <li>1 = Accumulator B is saturated or has been saturated at since this bit was last cleared</li> <li>0 = Accumulator B is not saturated</li> </ul>						
bit 11	1 = Accumula	DB Combined A ator A or B has Accumulator A	overflowed					
bit 10	1 = Accumula		aturated or ha	s been saturat	ted since this bit	was last cleare	d	
bit 9	DA: DO Loop 1 = DO loop in	<ul> <li>0 = Neither Accumulator A nor B is saturated</li> <li>DA: DO Loop Active bit<sup>(4)</sup></li> <li>1 = DO loop in progress</li> <li>0 = DO loop not in progress</li> </ul>						
bit 8	<b>DC:</b> MCU AL 1 = A carry-o	<ul> <li>0 = DO loop not in progress</li> <li>DC: MCU ALU Half Carry bit</li> <li>1 = A carry-out from the MSb of the lower nibble occurred</li> <li>0 = No carry-out from the MSb of the lower nibble occurred</li> </ul>						
bit 7-5	IPL<2:0>: In 111 = CPU II 110 = CPU II 101 = CPU II 100 = CPU II 011 = CPU II 010 = CPU II 010 = CPU II 001 = CPU II	terrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority	Level bits <sup>(5)</sup> / Level is 7 (19 / Level is 6 (14 / Level is 5 (13 / Level is 4 (12 / Level is 3 (11 / Level is 2 (10 / Level is 1 (9)	5). User interru 4) 3) 2) 1) 0)				
	This bit may be re Once this bit is set			v by software				
	Clearing this bit wi		-	, _, _,				
	his bit is read-on							

- **4:** This bit is read-only.
- **5:** The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1.

Programmer's Model

Register 2-2:	SR: CPU STATUS Register (dsPIC30F and dsPIC33F Devices) (Continued)
bit 4	RA: REPEAT Loop Active bit
	1 = REPEAT loop in progress 0 = REPEAT loop not in progress
bit 3	N: MCU ALU Negative bit
	1 = The result of the operation was negative
	0 = The result of the operation was not negative
bit 2	OV: MCU ALU Overflow bit
	1 = Overflow occurred
	0 = No overflow occurred
bit 1	Z: MCU ALU Zero bit
	1 = The result of the operation was zero
	0 = The result of the operation was not zero
bit 0	C: MCU ALU Carry/Borrow bit
	1 = A carry-out from the MSb occurred
	0 = No carry-out from the MSb occurred

- Note 1: This bit may be read or cleared, but not set.
  - 2: Once this bit is set, it must be cleared manually by software.
  - **3:** Clearing this bit will clear SA and SB.
  - 4: This bit is read-only.
  - **5:** The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1.

R/W-0	R/W-0	R/W-0	R/W-0	R/C-0	R/C-0	R -0	R/W-0
OA	OB	SA(3)	SB <sup>(3)</sup>	OAB	SAB	DA	DC
bit 15							bit
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
IPL2(1,2		IPL0 <sup>(1,2)</sup>	RA	N	OV	Z	C
bit 7		11 20	101		0.	_	bit
Lonondi						Lee (0)	
Legend:	hla hit		L:1		nented bit, read	as '0'	
R = Reada		W = Writable		C = Clearable		Dit is under	
-n = Value	at POR	'1' = Bit is set		'0' = Bit is cle	ared	x = Bit is unkr	nown
bit 15	1 = Accumula	lator A Overflov ator A has over ator A has not c	lowed				
bit 14	1 = Accumula	lator B Overflov ator B has over ator B has not c	lowed				
bit 13	1 = Accumula	lator A Saturation ator A is saturat ator A is not sat	ed or has bee	n saturated sir	nce this bit was	last cleared	
bit 12	1 = Accumula	<ul> <li>SB: Accumulator B Saturation Status bit</li> <li>1 = Accumulator B is saturated or has been saturated since this bit was last cleared</li> <li>0 = Accumulator B is not saturated</li> </ul>					
bit 11	1 = Accumula	<b>OAB:</b> OA    OB Combined Accumulator Overflow Status bit 1 = Accumulator A or B has overflowed 0 = Neither Accumulator A nor B has overflowed					
bit 10	1 = Accumula	<ul> <li>SAB: SA    SB Combined Accumulator Status bit</li> <li>1 = Accumulator A or B is saturated or has been saturated since this bit was last cleared</li> <li>0 = Neither Accumulator A nor B is saturated</li> </ul>					
bit 9	1 = DO <b>loop i</b>	<ul> <li>DA: DO Loop Active bit</li> <li>1 = DO loop in progress</li> <li>0 = DO loop not in progress</li> </ul>					
bit 8	DC: MCU AL 1 = A carry-o of the res 0 = No carry-	U Half Carry/Bo ut from the 4th ult occurred	low order bit (1	-	data) or 8th low data) or 8th low		
	The IPL<2:0> bits Level. The value i	n parentheses i	ndicates the II	PL, if IPL3 = 1.	User interrupts	are disabled w	/hen IPL3 = 1
	The IPL<2:0> Stat reference manual						tamily

**3:** A data write to SR can modify the SA or SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA/SB bit write race-condition, the SA and SB bits should not be modified using bit operations.

2

Programmer's Model

Register 2-3:	SR: CPU STATUS Register (dsPIC33E Devices) (Continued)
bit 7-5	IPL<2:0>: CPU Interrupt Priority Level Status bits <sup>(1,2)</sup> 111 = CPU Interrupt Priority Level is 7 (15). User interrupts disabled 110 = CPU Interrupt Priority Level is 6 (14) 101 = CPU Interrupt Priority Level is 5 (13) 100 = CPU Interrupt Priority Level is 4 (12) 011 = CPU Interrupt Priority Level is 3 (11) 010 = CPU Interrupt Priority Level is 2 (10) 001 = CPU Interrupt Priority Level is 1 (9) 000 = CPU Interrupt Priority Level is 0 (8)
bit 4	RA: REPEAT Loop Active bit 1 = REPEAT loop in progress 0 = REPEAT loop not in progress
bit 3	N: MCU ALU Negative bit 1 = Result was negative 0 = Result was non-negative (zero or positive)
bit 2	<ul> <li>OV: MCU ALU Overflow bit</li> <li>This bit is used for signed arithmetic (2's complement). It indicates an overflow of the magnitude that causes the sign bit to change state.</li> <li>1 = Overflow occurred for signed arithmetic (in this arithmetic operation)</li> <li>0 = No overflow occurred</li> </ul>
bit 1	<b>Z:</b> MCU ALU Zero bit 1 = The result of the operation was zero 0 = The result of the operation was not zero
bit 0	<b>C:</b> MCU ALU Carry/Borrow bit 1 = A carry-out from the MSb of the result occurred 0 = No carry-out from the MSb of the result occurred
Note 1: Th	ne IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority

- **Note 1:** The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1. User interrupts are disabled when IPL3 = 1.
  - 2: The IPL<2:0> Status bits are read only when NSTDIS bit (INTCON1<15>) = 1. Refer to the family reference manual of the specific device family to see the associated interrupt register.
  - **3:** A data write to SR can modify the SA or SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA/SB bit write race-condition, the SA and SB bits should not be modified using bit operations.

Register 2-4:	CORCON: Core Control Register (PIC24F and PIC24H Devices)								
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0		
	_	—	_		_	—	—		
bit 15							bit 8		
r									
U-0	U-0	U-0	U-0	R/C-0	R/W-0	U-0	U-0		
—	—	—	—	IPL3 <sup>(1,2)</sup>	PSV	—	—		
bit 7							bit C		
Legend:		C = Clearable bit		R = Readabl	R = Readable bit		W = Writable bit		
-n = Value at POR '		'1' = Bit is set		'0' = Bit is cleared		x = Bit is unknown			
U = Unimplem	ented bit, read	d as '0'							
bit 15-4	Unimpleme	Unimplemented: Read as '0'							
bit 3		IPL3: Interrupt Priority Level 3 Status bit <sup>(1,2)</sup>							
		•	•	reater (trap exc	•	,			
				ess (no trap exc		d)			
bit 2	-	-		pace Enable bit					
	•	n space visible i	•						
	0	n space not visil	•	ace					
1 1 4 0			(						

bit 1-0 Unimplemented: Read as '0'

Note 1: This bit may be read or cleared, but not set.

2: This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.

2

# 16-bit MCU and DSC Programmer's Reference Manual

Register 2-5:	CORCON:	Core Control R	egister (PIC	24E Devices)			
R/W-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
VAR	—	—	—	—	—	—	—
bit 15							bit 8
[							
U-0	U-0	U-0	U-0	R/C-0	R-0	U-0	U-0
		—		IPL3 <sup>(1,2)</sup>	SFA	—	_
bit 7							bit 0
Legend:							
R = Readable bit		W = Writable bit		U = Unimplemented bit, read as '0'			
-n = Value at PC	)R	'1' = Bit is set		0' = Bit is cleared $x = Bit is$			iown
bit 15	VAR: Variable	e Exception Pro	cessing Later	ncy Control bit			
		bounded deterr					
	<ul> <li>0 = Fixed (fully deterministic) exception processing latency</li> <li>Unimplemented: Read as '0'</li> </ul>						
	•			(1)			
	IPL3: CPU Interrupt Priority Level Status bit 3 <sup>(1)</sup> 1 = CPU interrupt priority level is greater than 7 0 = CPU interrupt priority level is 7 or less						
		rame Active Sta					
		me is active. V		address 0x00	000 to 0xFFFF,	regardless of	DSRPAG and
	0 = Stack frar	ne is not active	. W14 and W	15 address of I	EDS or Base Da	ata Space	
bit 1-0	Unimplemen	ted: Read as '0	)'				

- Note 1: This bit may be read or cleared, but not set.
  - 2: The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU interrupt priority level.

U-0	U-0	U-0	R/W-0	R(0)/W-0	R-0	R-0	R-0
_	_	_	US	EDT <sup>(1)</sup>		DL<2:0> <sup>(2,3)</sup>	
bit 15							bit
R/W-0	R/W-0	R/W-1	R/W-0	R/C-0	R/W-0	R/W-0	R/W-0
SATA	SATB	SATDW	ACCSAT	IPL3 <sup>(4,5)</sup>	PSV	RND	IF
bit 7	SAID	SAIDW	ACCOAT	IFLO	FOV	RIND	bit
Legend:		C = Clearable	e bit	R = Readable	e bit	W = Writable b	oit
-n = Value at	t POR	'1' = Bit is se	t	'0' = Bit is clea	ared	x = Bit is unkn	own
U = Unimple	mented bit, read	as '0'					
bit 15-13	Unimplemen	ted: Read as	ʻ∩'				
bit 12	-		ultiplier Mode S	Select bit			
	1 = Unsigned	mode enable		tiply operations			
bit 11	•	•	ation Control b loop at end of	bit <sup>(1)</sup> f current iteratio	n		
	110 = DO loo 110 = DO loo 011 = DO loo 010 = DO loo 001 = DO loo	ping is nested ping is nested ping is nested ping is nested ping is nested ping is active, ping is not active	at 5 levels at 4 levels at 3 levels at 2 levels but not nested	l (just 1 level)			
bit 7	SATA: ACCA Saturation Enable bit 1 = Accumulator A saturation enabled						
bit 6	SATB: ACCE 1 = Accumula	<ul> <li>0 = Accumulator A saturation disabled</li> <li>SATB: ACCB Saturation Enable bit</li> <li>1 = Accumulator B saturation enabled</li> <li>0 = Accumulator B saturation disabled</li> </ul>					
bit 5	1 = Data spac	a Space Write ce write satura ce write satura	tion enabled	gine Saturation	Enable bit		
bit 4	1 = 9.31 satu	cumulator Sat ration (Super S ration (Norma		Select bit			
bit 3	1 = CPU Inte	<ul> <li>IPL3: Interrupt Priority Level 3 Status bit<sup>(4, 5)</sup></li> <li>1 = CPU Interrupt Priority Level is 8 or greater (trap exception activated)</li> <li>0 = CPU Interrupt Priority Level is 7 or less (no trap exception activated)</li> </ul>					
bit 2	1 = Program	space visible i		ace Enable bit Ice			
Note 1: T	his bit will always	read '0'.					
	L<2:1> are read-						
	he first two levels		sting are hand	lled by hardwar	e.		
÷• •				,	-		

- 4: This bit may be read or cleared, but not set.
- 5: This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.

© 2005-2011 Microchip Technology Inc.

#### Register 2-6: CORCON: Core Control Register (dsPIC30F and dsPIC33F Devices) (Continued)

- bit 1
   RND: Rounding Mode Select bit

   1 = Biased (conventional) rounding enabled

   0 = Unbiased (convergent) rounding enabled

   bit 0
   IF: Integer or Fractional Multiplier Mode Select bit

   1 = Integer mode enabled for DSP multiply operations

   0 = Fractional mode enabled for DSP multiply operations
- **Note 1:** This bit will always read '0'.
  - **2:** DL<2:1> are read-only.
  - 3: The first two levels of DO loop nesting are handled by hardware.
  - 4: This bit may be read or cleared, but not set.
  - 5: This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0				
VAR	_	US<	:1:0>	EDT <sup>(1)</sup>		DL<2:0>					
bit 15							bi				
R/W-0	R/W-0	R/W-1	R/W-0	R/C-0	R-0	R/W-0	R/W-0				
SATA	SATB	SATDW	ACCSAT	IPL3 <sup>(2,3)</sup>	SFA	RND	IF				
bit 7							bi				
Legend:											
R = Readabl	e bit	W = Writable	bit	U = Unimplem	ented bit, rea	ad as '0'					
-n = Value at	POR	'1' = Bit is set	t	'0' = Bit is clea	red	x = Bit is unkr	nown				
bit 15	VAR: Variable	e Exception Pr	ocessing Later	ncy Control bit							
		•	•	ption processing	g latency						
	0 = Fixed (ful	ly deterministic	c) exception pr	ocessing latenc	y Y						
bit 14	Unimplemen	ted: Read as '	0'								
bit 13-12		P Multiply Uns	igned/Signed	Control bits							
		11 = Reserved 10 = DSP engine multiplies are mixed-sign									
	01 = DSP engine multiplies are unsigned										
		gine multiplies	•								
bit 11	EDT: Early DO Loop Termination Control bit <sup>(1)</sup>										
	<ul> <li>1 = Terminate executing DO loop at end of current loop iteration</li> <li>0 = No effect</li> </ul>										
bit 10-8		DL<2:0>: DO Loop Nesting Level Status bits									
	111 = 7 DO loops active										
	•										
	•										
	001 = 1  DO IO 000 = 0  DO IO	•									
bit 7		Saturation En	able bit								
		ator A saturatio									
	0 = Accumula	ator A saturatio	n disabled								
bit 6	SATB: ACCB Saturation Enable bit										
	1 = Accumulator B saturation enabled 0 = Accumulator B saturation disabled										
bit 5	SATDW: Data Space Write from DSP Engine Saturation Enable bit										
	1 = Data space write saturation enabled										
	0 = Data space write saturation disabled										
bit 4	ACCSAT: Accumulator Saturation Mode Select bit										
		ration (super s ration (normal									
	() = 1.51 Gam	a such a	sacaraciony								
bit 3		-	Level Status k	oit 3(2)							
bit 3	IPL3: CPU In	terrupt Priority									

Note 1: This bit always reads as '0'.

- **2:** This bit may be read or cleared, but not set.
- 3: The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU interrupt priority level.

2 Programmer's Model

Register 2-7:	CORCON: Core Control Register (dsPIC33E Devices) (Continued)
bit 2	<ul> <li>SFA: Stack Frame Active Status bit</li> <li>1 = Stack frame is active. W14 and W15 address 0x0000 to 0xFFFF, regardless of DSRPAG and DSWPAG values.</li> <li>0 = Stack frame is not active. W14 and W15 address of EDS or Base Data Space</li> </ul>
bit 1	RND: Rounding Mode Select bit
	<ul> <li>1 = Biased (conventional) rounding enabled</li> <li>0 = Unbiased (convergent) rounding enabled</li> </ul>
bit 0	<ul> <li>IF: Integer or Fractional Multiplier Mode Select bit</li> <li>1 = Integer mode enabled for DSP multiply</li> <li>0 = Fractional mode enabled for DSP multiply</li> </ul>

#### Note 1: This bit always reads as '0'.

- 2: This bit may be read or cleared, but not set.
- 3: The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU interrupt priority level.



# **Section 3. Instruction Set Overview**

# HIGHLIGHTS

This section of the manual contains the following major topics:

3.1	Introduction	. 38
3.2	Instruction Set Overview	. 38
3.3	Instruction Set Summary Tables	. 40

# 3.1 INTRODUCTION

The 16-bit MCU and DSC instruction set provides a broad suite of instructions that support traditional microcontroller applications, and a class of instructions that support math intensive applications. Since almost all of the functionality of the 8-bit PIC MCU instruction set has been maintained, this hybrid instruction set allows an easy 16-bit migration path for users already familiar with the PIC microcontroller.

## 3.2 INSTRUCTION SET OVERVIEW

Depending on the device family, the 16-bit MCU and DSC instruction set contains up to 84 instructions, which can be grouped into the functional categories shown in Table 3-1. Table 1-2 defines the symbols used in the instruction summary tables, Table 3-2 through Table 3-11. These tables define the syntax, description, storage and execution requirements for each instruction. Storage requirements are represented in 24-bit instruction words and execution requirements are represented in instruction cycles.

Functional Group	Summary Table	Page Number
Move Instructions	Table 3-2	40
Math Instructions	Table 3-3	41
Logic Instructions	Table 3-4	43
Rotate/Shift Instructions	Table 3-5	44
Bit Instructions	Table 3-6	45
Compare/Skip and Compare/Branch Instructions	Table 3-7	46
Program Flow Instructions	Table 3-8	47
Shadow/Stack Instructions	Table 3-9	49
Control Instructions	Table 3-10	49
DSP Instructions <sup>(1)</sup>	Table 3-11	50

Table 3-1: Instruction Groups

**Note 1:** DSP instructions are only available in the dsPIC30F, dsPIC33F, and dsPIC33E device families.

Most instructions have several different Addressing modes and execution flows, which require different instruction variants. For instance, depending on the device family, there are up to six unique ADD instructions and each instruction variant has its own instruction encoding. Instruction format descriptions and specific instruction operation are provided in **Section 5. "Instruction Descriptions**". Additionally, a composite alphabetized instruction set table is provided in **Section 7. "Reference**".

## 3.2.1 Multi-Cycle Instructions

As the instruction summary tables show, most instructions execute in a single cycle, with the following exceptions:

- Note: The DO and DIVF instructions are only available in the dsPIC30F, dsPIC33F, and dsPIC33E device families.
- Instructions DO, MOV.D, POP.D, PUSH.D, TBLRDH, TBLRDL, TBLWTH and TBLWTL require 2 cycles to execute
- Instructions DIV.S, DIV.U and DIVF are single-cycle instructions, which should be executed 18 consecutive times as the target of a REPEAT instruction
- Instructions that change the program counter also require 2 cycles to execute, with the
  extra cycle executed as a NOP. Compare-skip instructions, which skip over a 2-word
  instruction, require 3 instruction cycles to execute, with 2 cycles executed as a NOP.
  Compare-branch instructions (dsPIC33E/PIC24E devices only) require 5 instruction cycles
  to execute when the branch is taken.
- The RETFIE, RETLW and RETURN are a special case of an instruction that changes the program counter. These execute in 3 cycles, unless an exception is pending and then they execute in 2 cycles.
  - Note 1: Instructions which access program memory as data, using Program Space Visibility (PSV), will incur a one or two cycle delay for PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices, whereas using PSV in dsPIC33E and PIC24E devices incurs a 4-cycle delay based on Flash memory access time. However, regardless of which device is being used, when the target instruction of a REPEAT loop accesses program memory as data, only the first execution of the target instruction is subject to the delay. See the specific device family reference manual for details.
    - 2: All instructions may incur an additional delay on some device families, depending on Flash memory access time. For example, PIC24E and dsPIC33E devices have a 3-cycle Flash memory access time. However, instruction pipelining increases the effective instruction execution throughput. Refer to Section 2. "CPU" of the specific device family reference manual for details on instruction timing.
    - 3: All read and read-modify-write operations (including bit operations) on non-CPU Special Function Registers (e.g., I/O Port, peripheral control, or status registers; interrupt flags, etc.) in PIC24E and dsPIC33E devices require 2 instruction cycles to execute. However, all write operations on both CPU and non-CPU Special Function Registers, and all read and read-modify-write operations on CPU Special Function Registers require 1 instruction cycle.

## 3.2.2 Multi-Word Instructions

As defined by Table 3-2, almost all instructions consume one instruction word (24 bits), with the exception of the CALL, DO and GOTO instructions, which are Program Flow Instructions, listed in Table 3-8. These instructions require two words of memory because their opcodes embed large literal operands.

# 3.3 INSTRUCTION SET SUMMARY TABLES

### Table 3-2: Move Instructions

A	ssembly Syntax	Description	Words	Cycles	Page Number
EXCH	Wns,Wnd	Swap Wns and Wnd	1	1	243
MOV	f {,WREG} <sup>(1)</sup>	Move f to destination	1	1	279
MOV	WREG, f	Move WREG to f	1	1	280
MOV	f,Wnd	Move f to Wnd	1	1 <sup>(4)</sup>	281
MOV	Wns,f	Move Wns to f	1	1	282
MOV.B	#lit8,Wnd	Move 8-bit literal to Wnd	1	1	283
MOV	<pre>#lit16,Wnd</pre>	Move 16-bit literal to Wnd	1	1	284
MOV	[Ws+Slit10],Wnd	Move [Ws + signed 10-bit offset] to Wnd	1	1 <sup>(4)</sup>	285
MOV	Wns,[Wd+Slit10]	Move Wns to [Wd + signed 10-bit offset]	1	1	286
MOV	Wso,Wdo	Move Wso to Wdo	1	1 <sup>(4)</sup>	287
MOV.D	Ws,Wnd	Move double Ws to Wnd:Wnd + 1	1	2 <sup>(4)</sup>	289
MOV.D	Wns,Wd	Move double Wns:Wns + 1 to Wd	1	2	289
MOVPAG	#lit10,DSRPAG <sup>(2)</sup>	Move 10-bit literal to DSRPAG	1	1	291
MOVPAG	#lit9,DSWPAG <sup>(2)</sup>	Move 9-bit literal to DSWPAG	1	1	291
MOVPAG	#lit8,TBLPAG <sup>(2)</sup>	Move 8-bit literal to TBLPAG	1	1	291
MOVPAG W	In, DSRPAG <sup>(2)</sup>	Move Wn to DSRPAG	1	1	292
MOVPAG W	In, DSWPAG <sup>(2)</sup>	Move Wn to DSWPAG	1	1	292
MOVPAG W	In, TBLPAG <sup>(2)</sup>	Move Wn to TBLPAG	1	1	292
SWAP	Wn	Wn = byte or nibble swap Wn	1	1	426
TBLRDH	[Ws],Wd	Read high program word to Wd	1	2 <sup>(3)</sup>	427
TBLRDL	[Ws],Wd	Read low program word to Wd	1	2 <sup>(3)</sup>	429
TBLWTH	Ws,[Wd]	Write Ws to high program word	1	2 <sup>(4)</sup>	431
TBLWTL	Ws,[Wd]	Write Ws to low program word	1	2 <sup>(4)</sup>	433

**Note 1:** When the optional { , WREG} operand is specified, the destination of the instruction is WREG. When { , WREG} is not specified, the destination of the instruction is the file register f.

2: The MOVPAG instruction is only available in dsPIC33E and PIC24E devices.

**3:** In dsPIC33E and PIC24E devices, these instructions require 3 additional cycles – compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

4: In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

Table 3-3: Asso	Math Instruction	Description	Words	Cycles	Page
		-		_	Number
ADD	f {,WREG} <sup>(1)</sup>	Destination = f + WREG	1	1 <sup>(5)</sup>	99
ADD	#lit10,Wn	Wn = lit10 + Wn	1	1	100
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	1	1	101
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1 <sup>(5)</sup>	102
ADDC	f {,WREG} <b>(1)</b>	Destination = $f + WREG + (C)$	1	1 <sup>(5)</sup>	106
ADDC	#lit10,Wn	Wn = lit10 + Wn + (C)	1	1	107
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	1	1	108
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1 <sup>(5)</sup>	110
DAW.B	Wn	Wn = decimal adjust Wn	1	1	216
DEC	f {,WREG}(1)	Destination = $f - 1$	1	1 <sup>(5)</sup>	217
DEC	Ws,Wd	Wd = Ws - 1	1	1 <sup>(5)</sup>	218
DEC2	f {,WREG}(1)	Destination = $f - 2$	1	1 <sup>(5)</sup>	220
DEC2	Ws,Wd	Wd = Ws - 2	1	1 <sup>(5)</sup>	221
DIV.S	Wm, Wn	Signed 16/16-bit integer divide, Q →W0, R →W1	1	18 <b>(2)</b>	224
DIV.SD	Wm, Wn	Signed 32/16-bit integer divide, $Q \rightarrow W0$ , $R \rightarrow W1$	1	18 <sup>(2)</sup>	224
DIV.U	Wm, Wn	Unsigned 16/16-bit integer divide, Q - W0, R $\rightarrow$ W1	1	18 <sup>(2)</sup>	226
DIV.UD	Wm, Wn	Unsigned 32/16-bit integer divide, Q - W0, R $\rightarrow$ W1	1	18 <b>(2)</b>	226
DIVF	Wm, Wn	Signed 16/16-bit fractional divide, Q - W0, R $\rightarrow$ W1	1	18 <sup>(2)</sup>	228
INC	f {,WREG}(1)	Destination = f + 1	1	1 <sup>(5)</sup>	254
INC	Ws,Wd	Wd = Ws + 1	1	1 <sup>(5)</sup>	255
INC2	f {,WREG} <sup>(1)</sup>	Destination = f + 2	1	1 <sup>(5)</sup>	257
INC2	Ws,Wd	Wd = Ws + 2	1	1 <sup>(5)</sup>	258
MUL	f	W3:W2 = f * WREG	1	1 <sup>(5)</sup>	303
MUL.SS	Wb,Ws,Wnd	{Wnd + 1,Wnd} = signed(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	305
MUL.SS	Wb,Ws,Acc <sup>(4)</sup>	Accumulator = signed(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	307
MUL.SU	Wb,#lit5,Wnd	{Wnd + 1, Wnd} = signed(Wb) * unsigned(lit5)	1	1	308
MUL.SU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = signed(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	310
MUL.SU	Wb,Ws,Acc <b><sup>(4)</sup></b>	Accumulator = signed(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	312
MUL.SU	Wb,#lit5,Acc <sup>(4)</sup>	Accumulator = signed(Wb) * unsigned(lit5)	1	1	314
MUL.US	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsigned(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	315
MUL.US	Wb,Ws,Acc <b>(4)</b>	Accumulator = unsigned(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	317
MUL.UU	Wb,#lit5,Wnd	{Wnd + 1, Wnd} = unsigned(Wb) * unsigned(lit5)	1	1	319
MUL.UU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsigned(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	320
MUL.UU	Wb,Ws,Acc <b>(4)</b>	Accumulator = unsigned(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	322
MUL.UU	Wb,#lit5,Acc <sup>(4)</sup>	Accumulator = unsigned(Wb) * unsigned(lit5)	1	1	323
MULW.SS	Wb,Ws,Wnd <b>(3)</b>	Wnd = signed(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	324

Table 3-3:	Math Instructions
------------	-------------------

Note 1: When the optional { , WREG} operand is specified, the destination of the instruction is WREG. When { , WREG} is not specified, the destination of the instruction is the file register f.

2: The divide instructions must be preceded with a "REPEAT #17" instruction, such that they are executed 18 consecutive times.

3: These instructions are only available in dsPIC33E and PIC24E devices.

4: These instructions are only available in dsPIC33E devices.

 In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

Ass	embly Syntax	Description	Words	Cycles	Page Number
MULW.SU	Wb,Ws,Wnd <b>(3)</b>	Wnd = signed(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	326
MULW.SU	Wb,#lit5,Wnd <sup>(3)</sup>	Wnd = signed(Wb) * unsigned(lit5)	1	1	328
MULW.US	Wb,Ws,Wnd <b>(3)</b>	Wnd = unsigned(Wb) * signed(Ws)	1	1 <sup>(5)</sup>	329
MULW.UU	Wb,Ws,Wnd <b>(3)</b>	Wnd = unsigned(Wb) * unsigned(Ws)	1	1 <sup>(5)</sup>	331
MULW.UU	Wb,#lit5,Wnd <sup>(3)</sup>	Wnd = unsigned(Wb) * unsigned(lit5)	1	1	332
SE	Ws,Wnd	Wnd = signed-extended Ws	1	1 <sup>(5)</sup>	393
SUB	f {,WREG} <b>(1)</b>	Destination = f – WREG	1	1 <sup>(5)</sup>	405
SUB	#lit10,Wn	Wn = Wn - Iit10	1	1	406
SUB	Wb,#lit5,Wd	Wd = Wb - lit5	1	1	407
SUB	Wb,Ws,Wd	Wd = Wb - Ws	1	1 <sup>(5)</sup>	408
SUBB	f {,WREG} <b>(1)</b>	Destination = f – WREG – $(\overline{C})$	1	1 <sup>(5)</sup>	411
SUBB	#lit10,Wn	$Wn = Wn - lit10 - (\overline{C})$	1	1	412
SUBB	Wb,#lit5,Wd	$Wd = Wb - lit5 - (\overline{C})$	1	1	413
SUBB	Wb,Ws,Wd	$Wd = Wb - Ws - (\overline{C})$	1	1 <sup>(5)</sup>	415
SUBBR	f {,WREG} <b>(1)</b>	Destination = WREG – f – $(\overline{C})$	1	1 <sup>(5)</sup>	417
SUBBR	Wb,#lit5,Wd	$Wd = lit5 - Wb - (\overline{C})$	1	1	418
SUBBR	Wb,Ws,Wd	$Wd = Ws - Wb - (\overline{C})$	1	1 <sup>(5)</sup>	420
SUBR	f {,WREG} <b>(1)</b>	Destination = WREG – f	1	1 <sup>(5)</sup>	422
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	1	1	423
SUBR	Wb,Ws,Wd	Wd = Ws - Wb	1	1 <sup>(5)</sup>	424
ZE	Ws,Wnd	Wnd = zero-extended Ws	1	1 <sup>(5)</sup>	442

### Table 3-3: Math Instructions (Continued)

**Note 1:** When the optional { , WREG} operand is specified, the destination of the instruction is WREG. When { , WREG} is not specified, the destination of the instruction is the file register f.

2: The divide instructions must be preceded with a "REPEAT #17" instruction, such that they are executed 18 consecutive times.

- **3:** These instructions are only available in dsPIC33E and PIC24E devices.
- 4: These instructions are only available in dsPIC33E devices.
- In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

	Assembly Syntax	Description	Words	Cycles	Page Number
AND	f {,WREG} <sup>(1)</sup>	Destination = f .AND. WREG	1	1 <sup>(2)</sup>	112
AND	#lit10,Wn	Wn = lit10 .AND. Wn	1	1	113
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	1	1	114
AND	Wb,Ws,Wd	Wd = Wb .AND. Ws	1	1 <sup>(2)</sup>	115
CLR	f	f = 0x0000	1	1	184
CLR	WREG	WREG = 0x0000	1	1	184
CLR	Wd	Wd = 0x0000	1	1	185
COM	f {,WREG} <sup>(1)</sup>	Destination = $\overline{f}$	1	1 <sup>(2)</sup>	189
COM	Ws,Wd	$Wd = \overline{Ws}$	1	1 <sup>(2)</sup>	190
IOR	f {,WREG} <sup>(1)</sup>	Destination = f .IOR. WREG	1	1 <sup>(2)</sup>	260
IOR	#lit10,Wn	Wn = lit10 .IOR. Wn	1	1	261
IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	1	1	262
IOR	Wb,Ws,Wd	Wd = Wb .IOR. Ws	1	1 <sup>(2)</sup>	263
NEG	f {,WREG} <sup>(1)</sup>	Destination = $\overline{f} + 1$	1	1 <sup>(2)</sup>	333
NEG	Ws,Wd	$Wd = \overline{Ws} + 1$	1	1 <sup>(2)</sup>	333
SETM	f	f = 0xFFFF	1	1	395
SETM	WREG	WREG = 0xFFFF	1	1	395
SETM	Wd	Wd = 0xFFFF	1	1	396
XOR	f {,WREG} <sup>(1)</sup>	Destination = f .XOR. WREG	1	1 <sup>(2)</sup>	437
XOR	#lit10,Wn	Wn = lit10 .XOR. Wn	1	1	438
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1	439
XOR	Wb,Ws,Wd	Wd = Wb .XOR. Ws	1	1 <sup>(2)</sup>	440

Table 3-4:Logic Instructions

**Note 1:** When the optional { , WREG} operand is specified, the destination of the instruction is WREG. When { , WREG} is not specified, the destination of the instruction is the file register f.

2: In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

	Assembly Syntax	Description	Words	Cycles	Page Number
ASR	f {,WREG} <b>(1)</b>	Destination = arithmetic right shift f, LSb $\rightarrow$ C	1	1 <sup>(2)</sup>	117
ASR	Ws,Wd	Wd = arithmetic right shift Ws, LSb $\rightarrow$ C	1	1 <sup>(2)</sup>	119
ASR	Wb,#lit4,Wnd	Wnd = arithmetic right shift Wb by lit4, LSb $\rightarrow$ C	1	1	121
ASR	Wb,Wns,Wnd	Wnd = arithmetic right shift Wb by Wns, LSb $\rightarrow$ C	1	1	122
LSR	f {,WREG} <sup>(1)</sup>	Destination = logical right shift f, LSb $\rightarrow$ C	1	1 <sup>(2)</sup>	269
LSR	Ws,Wd	Wd = logical right shift Ws, LSb $\rightarrow$ C	1	1 <sup>(2)</sup>	271
LSR	Wb,#lit4,Wnd	Wnd = logical right shift Wb by lit4, LSb $\rightarrow$ C	1	1	273
LSR	Wb,Wns,Wnd	Wnd = logical right shift Wb by Wns, LSb $\rightarrow$ C	1	1	274
RLC	f {,WREG} <sup>(1)</sup>	Destination = rotate left through Carry f	1	1 <sup>(2)</sup>	373
RLC	Ws,Wd	Wd = rotate left through Carry Ws	1	1 <sup>(2)</sup>	375
RLNC	f {,WREG} <sup>(1)</sup>	Destination = rotate left (no Carry) f	1	1 <sup>(2)</sup>	377
RLNC	Ws,Wd	Wd = rotate left (no Carry) Ws	1	1 <sup>(2)</sup>	379
RRC	f {,WREG} <sup>(1)</sup>	Destination = rotate right through Carry f	1	1 <sup>(2)</sup>	381
RRC	Ws,Wd	Wd = rotate right through Carry Ws	1	1 <sup>(2)</sup>	383
RRNC	f {,WREG} <sup>(1)</sup>	Destination = rotate right (no Carry) f	1	1 <sup>(2)</sup>	385
RRNC	Ws,Wd	Wd = rotate right (no Carry) Ws	1	1 <sup>(2)</sup>	387
SL	f {,WREG} <sup>(1)</sup>	Destination = left shift f, MSb $\rightarrow$ C	1	1 <sup>(2)</sup>	399
SL	Ws,Wd	Wd = left shift Ws, MSb $\rightarrow$ C	1	1 <sup>(2)</sup>	401
SL	Wb,#lit4,Wnd	Wnd = left shift Wb by lit4, MSb $\rightarrow$ C	1	1	403
SL	Wb,Wns,Wnd	Wnd = left shift Wb by Wns, MSb $\rightarrow$ C	1	1	404

## Table 3-5: Rotate/Shift Instructions

**Note 1:** When the optional { , WREG} operand is specified, the destination of the instruction is WREG. When { , WREG} is not specified, the destination of the instruction is the file register f.

2: In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

Table 3-6:	Bit Instructions	Description	Words	Cycles <sup>(1)</sup>	Page
A330		Description		Oycles. /	Number
BCLR	f,#bit4	Bit clear f	1	1	123
BCLR	Ws,#bit4	Bit clear Ws	1	1	124
BSET	f,#bit4	Bit set f	1	1	152
BSET	Ws,#bit4	Bit set Ws	1	1	153
BSW.C	Ws,Wb	Write C bit to Ws <wb></wb>	1	1	155
BSW.Z	Ws,Wb	Write $\overline{Z}$ bit to Ws <wb></wb>	1	1	155
BTG	f,#bit4	Bit toggle f	1	1	157
BTG	Ws,#bit4	Bit toggle Ws	1	1	158
BTST	f,#bit4	Bit test f to Z	1	1	168
BTST.C	Ws,#bit4	Bit test Ws to C	1	1	169
BTST.Z	Ws,#bit4	Bit test Ws to Z	1	1	169
BTST.C	Ws,Wb	Bit test Ws <wb> to C</wb>	1	1	171
BTST.Z	Ws,Wb	Bit test Ws <wb> to Z</wb>	1	1	171
BTSTS	f,#bit4	Bit test f to Z, then set f	1	1	173
BTSTS.C	Ws,#bit4	Bit test Ws to C then set Ws	1	1	175
BTSTS.Z	Ws,#bit4	Bit test Ws to Z then set Ws	1	1	175
FBCL	Ws,Wnd	Find bit change from left (MSb) side	1	1	244
FF1L	Ws,Wnd	Find first one from left (MSb) side	1	1	246
FF1R	Ws,Wnd	Find first one from right (LSb) side	1	1	248

Table 3-6: Bit Instructions

**Note 1:** In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

Ass	sembly Syntax	Description	Words	Cycles <sup>(1)</sup>	Page Number
BTSC	f,#bit4	Bit test f, skip if clear	1	1 (2 or 3) <sup>(5)</sup>	160
BTSC	Ws,#bit4	Bit test Ws, skip if clear	1	1 (2 or 3) <sup>(5)</sup>	162
BTSS	f,#bit4	Bit test f, skip if set	1	1 (2 or 3) <sup>(5)</sup>	164
BTSS	Ws,#bit4	Bit test Ws, skip if set	1	1 (2 or 3) <sup>(5)</sup>	166
CP	f	Compare (f – WREG)	1	1 <sup>(5)</sup>	191
CP	Wb,#lit5 <sup>(2)</sup>	Compare (Wb – lit5)	1	1	192
CP	Wb,#lit8 <sup>(3)</sup>	Compare (Wb – lit8)	1	1	193
CP	Wb,Ws	Compare (Wb – Ws)	1	1 <sup>(5)</sup>	194
CP0	f	Compare (f – 0x0000)	1	1 <sup>(5)</sup>	196
CP0	Ws	Compare (Ws – 0x0000)	1	1 <sup>(5)</sup>	197
CPB	f	Compare with Borrow (f – WREG – $\overline{C}$ )	1	1 <sup>(5)</sup>	198
CPB	Wb,#lit5 <sup>(2)</sup>	Compare with Borrow (Wb – lit5 – $\overline{C}$ )	1	1	199
CPB	Wb,#lit8 <sup>(3)</sup>	Compare with Borrow (Wb – lit8 – $\overline{C}$ )	1	1	200
CPB	Wb,Ws	Compare with Borrow (Wb – Ws – $\overline{C}$ )	1	1 <sup>(5)</sup>	201
CPBEQ	Wb,Wn,Expr <sup>(3)</sup>	Compare Wb with Wn, branch if =	1	1 (5) <sup>(4)</sup>	203
CPBGT	Wb,Wn,Expr <sup>(3)</sup>	Signed compare Wb with Wn, branch if >	1	1 (5) <sup>(4)</sup>	204
CPBLT	Wb,Wn,Expr <sup>(3)</sup>	Signed compare Wb with Wn, branch if <	1	1 (5) <sup>(4)</sup>	205
CPBNE	Wb,Wn,Expr <sup>(3)</sup>	Compare Wb with Wn, branch if ≠	1	1 (5) <sup>(4)</sup>	204
CPSEQ	Wb, Wn	Compare (Wb – Wn), skip if =	1	1 (2 or 3)	207
CPSGT	Wb, Wn	Signed compare (Wb – Wn), skip if >	1	1 (2 or 3)	211
CPSLT	Wb, Wn	Signed compare (Wb – Wn), skip if <	1	1 (2 or 3)	212
CPSNE	Wb, Wn	Compare (Wb – Wn), skip if ≠	1	1 (2 or 3)	214

### Table 3-7: Compare/Skip and Compare/Branch Instructions

**Note 1:** Conditional skip instructions execute in 1 cycle if the skip is not taken, 2 cycles if the skip is taken over a one-word instruction and 3 cycles if the skip is taken over a two-word instruction.

2: This instruction is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

3: This instruction is only available in dsPIC33E and PIC24E devices.

4: Compare-branch instructions in dsPIC33E/PIC24E devices execute in 1 cycle if the branch is not taken and 5 cycles if the branch is taken.

5: In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

Ass	embly Syntax	Description	Words	Cycles	Page Number
BRA	Expr	Branch unconditionally	1	2 <sup>(8)</sup>	126
BRA	Wn	Computed branch	1	2 <sup>(8)</sup>	128
BRA	C,Expr	Branch if Carry (no Borrow)	1	1 (2) <sup>(1,8)</sup>	130
BRA	GE,Expr	Branch if greater than or equal	1	1 (2) <sup>(1,8)</sup>	132
BRA	GEU,Expr	Branch if unsigned greater than or equal	1	1 (2) <sup>(1,8)</sup>	134
BRA	GT,Expr	Branch if greater than	1	1 (2) <sup>(1,8)</sup>	135
BRA	GTU,Expr	Branch if unsigned greater than	1	1 (2) <sup>(1,8)</sup>	136
BRA	LE,Expr	Branch if less than or equal	1	1 (2) <sup>(1,8)</sup>	137
BRA	LEU,Expr	Branch if unsigned less than or equal	1	1 (2) <sup>(1,8)</sup>	138
BRA	LT,Expr	Branch if less than	1	1 (2) <sup>(1,8)</sup>	139
BRA	LTU,Expr	Branch if unsigned less than	1	1 (2) <sup>(1,8)</sup>	140
BRA	N,Expr	Branch if Negative	1	1 (2) <sup>(1,8)</sup>	141
BRA	NC,Expr	Branch if not Carry (Borrow)	1	1 (2) <sup>(1,8)</sup>	142
BRA	NN,Expr	Branch if not Negative	1	1 (2) <sup>(1,8)</sup>	143
BRA	NOV, Expr	Branch if not Overflow	1	1 (2) <sup>(1,8)</sup>	144
BRA	NZ,Expr	Branch if not Zero	1	1 (2) <sup>(1,8)</sup>	145
BRA	OA,Expr	Branch if Accumulator A Overflow	1	1 (2) <sup>(1,8)</sup>	146
BRA	OB,Expr	Branch if Accumulator B Overflow	1	1 (2) <sup>(1,8)</sup>	147
BRA	OV,Expr	Branch if Overflow	1	1 (2) <sup>(1,8)</sup>	148
BRA	SA,Expr	Branch if Accumulator A Saturate	1	1 (2) <sup>(1,8)</sup>	149
BRA	SB,Expr	Branch if Accumulator B Saturate	1	1 (2) <sup>(1,8)</sup>	150
BRA	Z,Expr	Branch if Zero	1	1 (2) <sup>(1,8)</sup>	151
CALL	Expr	Call subroutine	2	2 <sup>(8)</sup>	177
CALL	Wn	Call indirect subroutine	1	2 <sup>(8)</sup>	180
CALL.L	Wn <sup>(4)</sup>	Call indirect subroutine (long address)	1	4	183
DO	<pre>#lit14,Expr(6)</pre>	Do code through PC + Expr, (lit14 + 1) times	2	2	230
DO	<pre>#lit15,Expr<sup>(7)</sup></pre>	Do code through PC + Expr, (lit15 + 1) times	2	2	233
DO	Wn,Expr <sup>(3)</sup>	Do code through PC + Expr, (Wn + 1) times	2	2	235
GOTO	Expr	Go to address	2	2 <sup>(8)</sup>	250
GOTO	Wn	Go to address indirectly	1	2 <sup>(8)</sup>	251
GOTO.L	Wn <sup>(4)</sup>	Go to indirect (long address)	1	4	253
RCALL	Expr	Relative call	1	2 <sup>(8)</sup>	347
RCALL	Wn	Computed call	1	2 <sup>(8)</sup>	351
REPEAT	#lit14 <sup>(5)</sup>	Repeat next instruction (lit14 + 1) times	1	1	355

Table 3-8:Program Flow Instructions

**Note 1:** Conditional branch instructions execute in 1 cycle if the branch is not taken, or 2 cycles if the branch is taken.

- **2:** RETURN instructions execute in 3 cycles, but if an exception is pending, they execute in 2 cycles.
- 3: This instruction is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.
- 4: This instruction is only available in dsPIC33E and PIC24E devices.
- 5: This instruction is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.
- 6: This instruction is only available in dsPIC30F and dsPIC33F devices.
- 7: This instruction is only available in dsPIC33E devices.
- 8: In dsPIC33E and PIC24E devices, these instructions require 2 additional cycles (4 cycles overall) when the branch is taken.
- 9: In dsPIC33E and PIC24E devices, these instructions require 3 additional cycles.

Instruction Se Overview

Assembly Syntax	Description	Words	Cycles	Page Number
REPEAT #lit15 <sup>(4)</sup>	Repeat next instruction (lit15 + 1) times	1	1	357
REPEAT Wn	Repeat next instruction (Wn + 1) times	1	1	359
RETFIE	Return from interrupt enable	1	3 (2) <sup>(2,9)</sup>	365
RETLW #lit10,Wn	Return with lit10 in Wn	1	3 (2) <sup>(2,9)</sup>	367
RETURN	Return from subroutine	1	3 (2) <sup>(2,9)</sup>	371

### Table 3-8: Program Flow Instructions (Continued)

**Note 1:** Conditional branch instructions execute in 1 cycle if the branch is not taken, or 2 cycles if the branch is taken.

2: RETURN instructions execute in 3 cycles, but if an exception is pending, they execute in 2 cycles.

3: This instruction is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

4: This instruction is only available in dsPIC33E and PIC24E devices.

5: This instruction is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

6: This instruction is only available in dsPIC30F and dsPIC33F devices.

7: This instruction is only available in dsPIC33E devices.

8: In dsPIC33E and PIC24E devices, these instructions require 2 additional cycles (4 cycles overall) when the branch is taken.

9: In dsPIC33E and PIC24E devices, these instructions require 3 additional cycles.

Asse	mbly Syntax	Description	Words	Cycles	Page Number
LNK	#lit14	Link Frame Pointer	1	1	267
POP	f	POP TOS to f	1	1	337
POP	Wdo	POP TOS to Wdo	1	1	338
POP.D	Wnd	Double POP from TOS to Wnd:Wnd + 1	1	2	339
POP.S		POP shadow registers	1	1	340
PUSH	f	PUSH f to TOS	1	1 <sup>(1)</sup>	341
PUSH	Wso	PUSH Wso to TOS	1	1 <sup>(1)</sup>	342
PUSH.D	Wns	PUSH double Wns:Wns + 1 to TOS	1	2	343
PUSH.S		PUSH shadow registers	1	1	345
ULNK		Unlink Frame Pointer	1	1	435

#### Table 3-9: Shadow/Stack Instructions

**Note 1:** In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

### Table 3-10:Control Instructions

Assem	ibly Syntax	Description	Words	Cycles	Page Number
CLRWDT		Clear Watchdog Timer	1	1	188
DISI	#lit14	Disable interrupts for (lit14 + 1) instruction cycles	1	1	223
NOP		No operation	1	1	336
NOPR		No operation	1	1	336
PWRSAV	#lit1	Enter Power-saving mode lit1	1	1	346
RESET		Software device Reset	1	1	363

	Assembly Syntax	Description	Words	Cycles	Page Number
ADD	Acc	Add accumulators	1	1	103
ADD	Wso,#Slit4,Acc	16-bit signed add to Acc	1	1 <sup>(1)</sup>	104
CLR	Acc,[Wx],Wxd,[Wy],Wyd,AWB	Clear Acc	1	1	186
ED	Wm*Wm,Acc,[Wx],[Wy],Wxd	Euclidean distance (no accumulate)	1	1	239
EDAC	Wm*Wm,Acc,[Wx],[Wy],Wxd	Euclidean distance	1	1	241
LAC	Wso,#Slit4,Acc	Load Acc	1	1 <sup>(1)</sup>	265
MAC	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd,AWB	Multiply and accumulate	1	1	275
MAC	Wm*Wm,Acc,[Wx],Wxd,[Wy],Wyd	Square and accumulate	1	1	277
MOVSAC	Acc,[Wx],Wxd,[Wy],Wyd,AWB	Move Wx to Wxd and Wy to Wyd	1	1	293
MPY	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd	Multiply Wn by Wm to Acc	1	1	295
MPY	Wm*Wm,Acc,[Wx],Wxd,[Wy],Wyd	Square to Acc	1	1	297
MPY.N	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd	-(Multiply Wn by Wm) to Acc	1	1	299
MSC	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd,AWB	Multiply and subtract from Acc	1	1	301
NEG	Acc	Negate Acc	1	1	335
SAC	Acc,#Slit4,Wdo	Store Acc	1	1	389
SAC.R	Acc,#Slit4,Wdo	Store rounded Acc	1	1	391
SFTAC	Acc,#Slit6	Arithmetic shift Acc by Slit6	1	1	397
SFTAC	Acc,Wn	Arithmetic shift Acc by (Wn)	1	1	398
SUB	Acc	Subtract accumulators	1	1	410

### Table 3-11: DSP Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)

**Note 1:** In dsPIC33E and PIC24E devices, read and read-modify-write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.



# **Section 4. Instruction Set Details**

# HIGHLIGHTS

This section of the manual contains the following major topics:

4.1	Data Addressing Modes	52
4.2	Program Addressing Modes	61
4.3	Instruction Stalls	62
4.4	Byte Operations	64
4.5	Word Move Operations	66
4.6	Using 10-bit Literal Operands	69
4.7	Software Stack Pointer and Frame Pointer	70
4.8	Conditional Branch Instructions	76
4.9	Z Status Bit	77
4.10	Assigned Working Register Usage	78
4.11	DSP Data Formats (dsPIC30F, dsPIC33F and dsPIC33E Devices)	81
4.12	Accumulator Usage (dsPIC30F, dsPIC33F and dsPIC33E Devices)	83
4.13	Accumulator Access (dsPIC30F, dsPIC33F and dsPIC33E Devices)	84
4.14	DSP MAC Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)	84
4.15	DSP Accumulator Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)	88
4.16	Scaling Data with the FBCL Instruction (dsPIC30F, dsPIC33F and dsPIC33E Devices	s) 88
4.17	Normalizing the Accumulator with the FBCL Instruction (dsPIC30F, dsPIC33F and dsPIC33E Devices)	90

# 4.1 DATA ADDRESSING MODES

The 16-bit MCU and DSC devices support three native Addressing modes for accessing data memory, along with several forms of immediate addressing. Data accesses may be performed using file register addressing, register direct or indirect addressing, and immediate addressing, allow a fixed value to be used by the instruction.

File register addressing provides the ability to operate on data stored in the lower 8K of data memory (Near RAM), and also move data between the working registers and the entire 64K data space. Register direct addressing is used to access the 16 memory mapped working registers, W0:W15. Register indirect addressing is used to efficiently operate on data stored in the entire 64K data space (and also Extended Data Space, in the case of dsPIC33E/PIC24E), using the contents of the working registers as an effective address. Immediate addressing does not access data memory, but provides the ability to use a constant value as an instruction operand. The address range of each mode is summarized in Table 4-1.

Addressing Mode	Address Range	
File Register	0x0000-0x1FFF <sup>(1)</sup>	
Register Direct	0x0000-0x001F (working register array W0:W15)	
Register Indirect	0x0000-0xFFFF	
Immediate	N/A (constant value)	

Table 4-1: 16-bit MCU and DSC Addressing Modes

**Note 1:** The address range for the File Register MOV is 0x0000-0xFFFE.

## 4.1.1 File Register Addressing

File register addressing is used by instructions which use a predetermined data address as an operand for the instruction. The majority of instructions that support file register addressing provide access to the lower 8 Kbytes of data memory, which is called the Near RAM. However, the MOV instruction provides access to all 64 Kbytes of memory using file register addressing. This allows the loading of the data from any location in data memory to any working register, and storing the contents of any working register to any location in data memory. It should be noted that file register addressing supports both byte and word accesses of data memory, with the exception of the MOV instruction, which accesses all 64K of memory as words. Examples of file register addressing are shown in Example 4-1.

Most instructions, which support file register addressing, perform an operation on the specified file register and the default working register WREG (see **Section 2.4 "Default Working Register (WREG)**"). If only one operand is supplied in the instruction, WREG is an implied operand and the operation results are stored back to the file register. In these cases, the instruction is effectively a read-modify-write instruction. However, when both the file register and the WREG register are specified in the instruction, the operation results are stored in the WREG register are stored in the instruction, the operation results are stored in the WREG register and the contents of the file register are unchanged. Sample instructions that show the interaction between the file register and the WREG register are shown in Example 4-2.

Note: Instructions which support file register addressing use 'f' as an operand in the instruction summary tables of Section 3. "Instruction Set Overview".

Example 4-1: File Register Addressing

```
DEC
            0x1000
                             ; decrement data stored at 0x1000
Before Instruction:
    Data Memory 0x1000 = 0x5555
After Instruction:
    Data Memory 0x1000 = 0x5554
    MOV
            0x27FE, W0
                            ; move data stored at 0x27FE to W0
Before Instruction:
    W0 = 0x5555
    Data Memory 0x27FE = 0x1234
After Instruction:
    W0 = 0 \times 1234
    Data Memory 0x27FE = 0x1234
```



```
AND
            0x1000
                             ; AND 0x1000 with WREG, store to 0x1000
Before Instruction:
    W0 (WREG) = 0x332C
    Data Memory 0x1000 = 0x5555
After Instruction:
    W0 (WREG) = 0x332C
    Data Memory 0 \times 1000 = 0 \times 1104
    AND
            0x1000, WREG ; AND 0x1000 with WREG, store to WREG
Before Instruction:
    W0 (WREG) = 0x332C
    Data Memory 0x1000 = 0x5555
After Instruction:
    W0 (WREG) = 0 \times 1104
    Data Memory 0x1000 = 0x5555
```

## 4.1.2 Register Direct Addressing

Register direct addressing is used to access the contents of the 16 working registers (W0:W15). The Register Direct Addressing mode is fully orthogonal, which allows any working register to be specified for any instruction that uses register direct addressing, and it supports both byte and word accesses. Instructions which employ register direct addressing use the contents of the specified working register as data to execute the instruction, therefore this Addressing mode is useful only when data already resides in the working register core. Sample instructions which utilize register direct addressing are shown in Example 4-3.

Another feature of register direct addressing is that it provides the ability for dynamic flow control. Since variants of the DO and REPEAT instruction support register direct addressing, flexible looping constructs may be generated using these instructions.

Note: Instructions which must use register direct addressing, use the symbols Wb, Wn, Wns and Wnd in the summary tables of **Section 3. "Instruction Set Overview**". Commonly, register direct addressing may also be used when register indirect addressing may be used. Instructions which use register indirect addressing, use the symbols Wd and Ws in the summary tables of **Section 3. "Instruction Set Overview**". Instruction Set

Deta

S

	; Exchange W2 and W3
Before Instruction:	
W2 = 0x3499	
$W3 = 0 \times 003D$	
After Instruction:	
$W2 = 0 \times 003D$	
$W3 = 0 \times 3499$	
IOR #0x44, W0	; Inclusive-OR 0x44 and W0
Before Instruction:	
$W0 = 0 \times 9C2E$	
After Instruction:	
WO = 0x9C6E	
SL W6, W7, W8	; Shift left W6 by W7, and store to W8
Before Instruction:	
$W6 = 0 \times 000C$	
$W7 = 0 \times 0008$	
$W8 = 0 \times 1234$	
After Instruction:	
$W6 = 0 \times 000 C$	
$W7 = 0 \times 0008$	
$W8 = 0 \times 0 C00$	

Example 4-3: Register Direct Addressing

## 4.1.3 Register Indirect Addressing

Register indirect addressing is used to access any location in data memory by treating the contents of a working register as an Effective Address (EA) to data memory. Essentially, the contents of the working register become a pointer to the location in data memory which is to be accessed by the instruction.

This Addressing mode is powerful, because it also allows one to modify the contents of the working register, either before or after the data access is made, by incrementing or decrementing the EA. By modifying the EA in the same cycle that an operation is being performed, register indirect addressing allows for the efficient processing of data that is stored sequentially in memory. The modes of indirect addressing supported by the 16-bit MCU and DSC devices are shown in Table 4-2.

Indirect Mode	Syntax	Function (Byte Instruction)	Function (Word Instruction)	Description
No Modification	[Wn]	EA = [Wn]	EA = [Wn]	The contents of Wn forms the EA.
Pre-Increment	[++\mathbb{W}n]	EA = [Wn + = 1]	EA = [Wn + = 2]	Wn is pre-incremented to form the EA.
Pre-Decrement	[Wn]	EA = [Wn - = 1]	EA = [Wn - = 2]	Wn is pre-decremented to form the EA.
Post-Increment	[Wn++]	EA = [Wn] + = 1	EA = [Wn] + = 2	The contents of Wn forms the EA, then Wn is post-incremented.
Post-Decrement	[Wn]	EA = [Wn] - = 1	EA = [Wn] - = 2	The contents of Wn forms the EA, then Wn is post-decremented.
Register Offset	[Wn+Wb]	EA = [Wn + Wb]	EA = [Wn + Wb]	The sum of Wn and Wb forms the EA. Wn and Wb are not modified.

 Table 4-2:
 Indirect Addressing Modes

Table 4-2 shows that four Addressing modes modify the EA used in the instruction, and this allows the following updates to be made to the working register: post-increment, post-decrement, pre-increment and pre-decrement. Since all EAs must be given as byte addresses, support is provided for Word mode instructions by scaling the EA update by 2. Namely, in Word mode, pre/post-decrements subtract 2 from the EA stored in the working register, and pre/post-increments add 2 to the EA. This feature ensures that after an EA modification is made, the EA will point to the next adjacent word in memory. Example 4-4 shows how indirect addressing may be used to update the EA.

Table 4-2 also shows that the Register Offset mode addresses data which is offset from a base EA stored in a working register. This mode uses the contents of a second working register to form the EA by adding the two specified working registers. This mode does not scale for Word mode instructions, but offers the complete offset range of 64 Kbytes. Note that neither of the working registers used to form the EA are modified. Example 4-5 shows how register offset indirect addressing may be used to access data memory.

**Note:** The MOV with offset instructions (see pages 285 and 286) provides a literal addressing offset ability to be used with indirect addressing. In these instructions, the EA is formed by adding the contents of a working register to a signed 10-bit literal. Example 4-6 shows how these instructions may be used to move data to and from the working register array.

#### Example 4-4: Indirect Addressing with Effective Address Update

MOV.B [W0++], [W13]	; byte move [W0] to [W13] ; post-inc W0, post-dec W13
Before Instruction:	
W0 = 0x2300 W13 = 0x2708 Data Memory 0x2300 = 0x7783 Data Memory 0x2708 = 0x904E	
After Instruction:	
W0 = 0x2301 W13 = 0x2707 Data Memory 0x2300 = 0x7783 Data Memory 0x2708 = 0x9083	
ADD W1, [W5], [++W8]	; pre-dec W5, pre-inc W8 ; add W1 to [W5], store in [W8]
Before Instruction:	
W1 = 0x0800 W5 = 0x2200 W8 = 0x2400 Data Memory 0x21FE = 0x7783 Data Memory 0x2402 = 0xAACC	
After Instruction:	
W1 = 0x0800 W5 = 0x21FE W8 = 0x2402 Data Memory 0x21FE = 0x7783 Data Memory 0x2402 = 0x7F83	

Indirect Addressing with Register Offset

Example 4-5:

```
MOV.B [W0+W1], [W7++]
                                    ; byte move [W0+W1] to W7, post-inc W7
Before Instruction:
   W0 = 0x2300
   W1 = 0 \times 01 FE
   W7 = 0x1000
   Data Memory 0x24FE = 0x7783
   Data Memory 0x1000 = 0x11DC
After Instruction:
   W0 = 0x2300
   W1 = 0 \times 01 FE
   W7 = 0x1001
   Data Memory 0x24FE = 0x7783
   Data Memory 0x1000 = 0x1183
          [W0+W8], A
   LAC
                                    ; load ACCA with [W0+W8]
                                    ; (sign-extend and zero-backfill)
Before Instruction:
   W0 = 0 \times 2344
   W8 = 0 \times 0008
   ACCA = 0 \times 00 7877 9321
   Data Memory 0x234C = 0xE290
After Instruction:
   W0 = 0x2344
   W8 = 0 \times 0008
   ACCA = 0xFF E290 0000
   Data Memory 0x234C = 0xE290
```

```
Example 4-6: Move with Literal Offset Instructions
```

```
MOV
            [W0+0x20], W1
                                   ; move [W0+0x20] to W1
Before Instruction:
   W0 = 0 \times 1200
   W1 = 0 \times 01 FE
   Data Memory 0x1220 = 0xFD27
After Instruction:
   W0 = 0x1200
   W1 = 0 \times FD27
   Data Memory 0x1220 = 0xFD27
   MOV
            W4, [W8-0x300] ; move W4 to [W8-0x300]
Before Instruction:
   W4 = 0x3411
   W8 = 0x2944
   Data Memory 0x2644 = 0xCB98
After Instruction:
   W4 = 0 \times 3411
   W8 = 0x2944
   Data Memory 0x2644 = 0x3411
```

### 4.1.3.1 REGISTER INDIRECT ADDRESSING AND THE INSTRUCTION SET

The Addressing modes presented in Table 4-2 demonstrate the Indirect Addressing mode capability of the 16-bit MCU and DSC devices. Due to operation encoding and functional considerations, not every instruction which supports indirect addressing supports all modes shown in Table 4-2. The majority of instructions which use indirect addressing support the No Modify, Pre-Increment, Pre-Decrement, Post-Increment and Post-Decrement Addressing modes. The MOV instructions, and several accumulator-based DSP instructions (dsPIC30F, dsPIC33F, and dsPIC33E devices only), are also capable of using the Register Offset Addressing mode.

Note: Instructions which use register indirect addressing use the operand symbols Wd and Ws in the summary tables of Section 3. "Instruction Set Overview".

# 4.1.3.2 DSP MAC INDIRECT ADDRESSING MODES (dsPIC30F, dsPIC33F, AND dsPIC33E DEVICES)

A special class of Indirect Addressing modes is utilized by the DSP MAC instructions. As is described later in **Section 4.14** "**DSP MAC Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)**", the DSP MAC class of instructions are capable of performing two fetches from memory using effective addressing. Since DSP algorithms frequently demand a broader range of address updates, the Addressing modes offered by the DSP MAC instructions provide greater range in the size of the effective address update which may be made. Table 4-3 shows that both X and Y prefetches support Post-Increment and Post-Decrement Addressing modes, with updates of 2, 4 and 6 bytes. Since DSP instructions only execute in Word mode, no provisions are made for odd sized EA updates.

		1
Addressing Mode	X Memory	Y Memory
Indirect with no modification	EA = [Wx]	EA = [Wy]
Indirect with Post-Increment by 2	EA = [Wx] + = 2	EA = [Wy] + = 2
Indirect with Post-Increment by 4	EA = [Wx] + = 4	EA = [Wy] + = 4
Indirect with Post-Increment by 6	EA = [Wx] + = 6	EA = [Wy] + = 6
Indirect with Post-Decrement by 2	EA = [Wx] - = 2	EA = [Wy] - = 2
Indirect with Post-Decrement by 4	EA = [Wx] - = 4	EA = [Wy] - = 4
Indirect with Post-Decrement by 6	EA = [Wx] - = 6	EA = [Wy] - = 6
Indirect with Register Offset	EA = [W9 + W12]	EA = [W11 + W12]

Table 4-3: DSP MAC Indirect Addressing Modes

Note: As described in Section 4.14 "DSP MAC Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)", only W8 and W9 may be used to access X Memory, and only W10 and W11 may be used to access Y Memory.

# 4.1.3.3 MODULO AND BIT-REVERSED ADDRESSING MODES (dsPIC30F, dsPIC33F, AND dsPIC33E DEVICES)

The 16-bit DSC architecture provides support for two special Register Indirect Addressing modes, which are commonly used to implement DSP algorithms. Modulo (or circular) addressing provides an automated means to support circular data buffers in X and/or Y memory. Modulo buffers remove the need for software to perform address boundary checks, which can improve the performance of certain algorithms. Similarly, bit-reversed addressing allows one to access the elements of a buffer in a nonlinear fashion. This Addressing mode simplifies data re-ordering for radix-2 FFT algorithms and provides a significant reduction in FFT processing time.

Both of these Addressing modes are powerful features of the dsPIC30F, dsPIC33F, and dsPIC33E architectures, which can be exploited by any instruction that uses indirect addressing. Refer to the specific device family reference manual for details on using modulo and bit-reversed addressing.

Instruction Set

S

© 2005-2011 Microchip Technology Inc.

## 4.1.4 Immediate Addressing

In immediate addressing, the instruction encoding contains a predefined constant operand, which is used by the instruction. This Addressing mode may be used independently, but it is more frequently combined with the File Register, Direct and Indirect Addressing modes. The size of the immediate operand which may be used varies with the instruction type. Constants of size 1-bit (#lit1), 4-bit (#bit4, #lit4 and #Slit4), 5-bit (#lit5), 6-bit (#Slit6), 8-bit (#lit8), 10-bit (#lit10 and #Slit10), 14-bit (#lit14) and 16-bit (#lit16) may be used. Constants may be signed or unsigned and the symbols #Slit4, #Slit6 and #Slit10 designate a signed constant. All other immediate constants are unsigned. Table 4-4 shows the usage of each immediate operand in the instruction set.

**Note:** The 6-bit (#Slit6) operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

Operand	Instruction Usage
#lit1	PWRSAV
#bit4	BCLR, BSET, BTG, BTSC, BTSS, BTST, BTST.C, BTST.Z, BTSTS, BTSTS.C, BTSTS.Z
#lit4	ASR, LSR, SL
#Slit4	ADD, LAC, SAC, SAC.R
#lit5	ADD, ADDC, AND, CP <sup>(5)</sup> , CPB <sup>(5)</sup> , IOR, MUL.SU, MUL.UU, SUB, SUBB, SUBBR, SUBR, XOR
#Slit6 <b>(1)</b>	SFTAC
#lit8	MOV.B, CP <sup>(4)</sup> , CPB <sup>(4)</sup>
#lit10	ADD, ADDC, AND, CP, CPB, IOR, RETLW, SUB, SUBB, XOR
#Slit10	MOV
#lit14	DISI, DO <sup>(2)</sup> , LNK, REPEAT <sup>(5)</sup>
#lit15	DO <sup>(3)</sup> , REPEAT <sup>(4)</sup>
#lit16	MOV

Table 4-4: Immediate Operands in the Instruction Set

**Note 1:** This operand or instruction is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

- 2: This operand or instruction is only available in dsPIC30F and dsPIC33F devices.
- 3: This operand or instruction is only available in dsPIC33E devices.
- 4: This operand or instruction is only available in dsPIC33E and PIC24E devices.
- **5:** This operand or instruction is only available in dsPIC30F, dsPIC33F, PIC24F, and PIC24H devices.

The syntax for immediate addressing requires that the number sign (#) must immediately precede the constant operand value. The "#" symbol indicates to the assembler that the quantity is a constant. If an out-of-range constant is used with an instruction, the assembler will generate an error. Several examples of immediate addressing are shown in Example 4-7.

Example 4-7: Immediate Addressing

PWRSAV #1	; Enter IDLE mode
ADD.B #0x10, WO	; Add 0x10 to W0 (byte mode)
Before Instruction:	
$W0 = 0 \times 12 A9$	
After Instruction:	
$W0 = 0 \times 12B9$	
XOR W0, #1, [W1++]	; Exclusive-OR W0 and 0x1 ; Store the result to [W1] ; Post-increment W1
Before Instruction:	
W0 = 0xFFFF W1 = 0x0890 Data Memory 0x0890 = 0x0032	
After Instruction:	
W0 = 0xFFFF W1 = 0x0892 Data Memory 0x0890 = 0xFFFE	

# 4.1.5 Data Addressing Mode Tree

The Data Addressing modes of the PIC24F, PIC24H, and PIC24E families are summarized in Figure 4-1.

	Immediate	
	File Register	No Modification
Data Addressing Modes	Modes Direct	Pre-Increment
	Direct	Pre-Decrement
	Indirect	Post-Increment
		Post-Decrement
		Literal Offset
		Register Offset

Figure 4-1: Data Addressing Mode Tree (PIC24F, PIC24H, and PIC24E)

The Data Addressing modes of the dsPIC30F, dsPIC33F, and dsPIC33E are summarized in Figure 4-2.

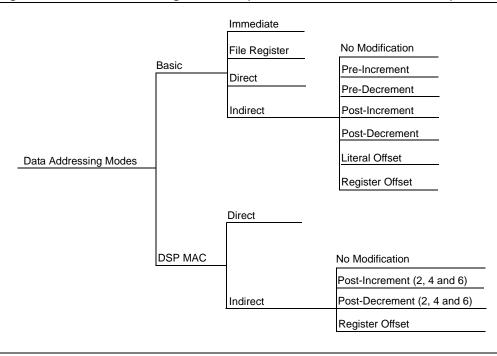


Figure 4-2: Data Addressing Mode Tree (dsPIC30F, dsPIC33F, and dsPIC33E)

# 4.2 PROGRAM ADDRESSING MODES

The 16-bit MCU and DSC devices have a 24-bit Program Counter (PC). The PC addresses the 24-bit wide program memory to fetch instructions for execution, and it may be loaded in several ways. For byte compatibility with the table read and table write instructions, each instruction word consumes two locations in program memory. This means that during serial execution, the PC is loaded with PC + 2.

Several methods may be used to modify the PC in a non-sequential manner, and both absolute and relative changes may be made to the PC. The change to the PC may be from an immediate value encoded in the instruction, or a dynamic value contained in a working register. In dsPIC30F, dsPIC33F, and dsPIC33E devices, when Do looping is active, the PC is loaded with the address stored in the DOSTART register, after the instruction at the DOEND address is executed. For exception handling, the PC is loaded with the address of the exception handler, which is stored in the interrupt vector table. When required, the software stack is used to return scope to the foreground process from where the change in program flow occurred.

Table 4-5 summarizes the instructions which modify the PC. When performing function calls, it is recommended that RCALL be used instead of CALL, since RCALL only consumes 1 word of program memory.

Condition/Instruction	PC Modification	Software Stack Usage
Sequential Execution	PC = PC + 2	None
BRA Expr <sup>(1)</sup> (Branch Unconditionally)	PC = PC + 2*Slit16	None
BRA Condition, Expr <sup>(1)</sup> (Branch Conditionally)	PC = PC + 2 (condition false) PC = PC + 2 * Slit16 (condition true)	None
CALL Expr <sup>(1)</sup> (Call Subroutine)	PC = lit23	PC + 4 is PUSHed on the stack <sup>(2)</sup>
CALL Wn (Call Subroutine Indirect)	PC = Wn	PC + 2 is PUSHed on the stack <sup>(2)</sup>
CALL.L Wn <sup>(5)</sup> (Call Indirect Subroutine Long)	PC = {Wn+1:Wn}	None
GOTO Expr <sup>(1)</sup> (Unconditional Jump)	PC = lit23	None
GOTO Wn (Unconditional Indirect Jump)	PC = Wn	None
GOTO.L Wn <sup>(5)</sup> (Unconditional Indirect Long Jump)	PC = {Wn+1:Wn}	None
RCALL Expr(1) (Relative Call)	PC = PC + 2 * Slit16	PC + 2 is PUSHed on the stack <sup>(2)</sup>
RCALL Wn (Computed Relative Call)	PC = PC + 2 * Wn	PC + 2 is PUSHed on the stack <sup>(2)</sup>
Exception Handling	PC = address of the exception handler (read from vector table)	PC + 2 is PUSHed on the stack <sup>(3)</sup>
PC = Target REPEAT instruction (REPEAT Looping)	PC not modified (if REPEAT active)	None
PC = DOEND address <sup>(4)</sup> (DO Looping)	PC = DOSTART (if DO active)	None

### Table 4-5: Methods of Modifying Program Flow

Note 1: For BRA, CALL and GOTO, the Expr may be a label, absolute address, or expression, which is resolved by the linker to a 16-bit or 23-bit value (Slit16 or lit23). See Section 5. "Instruction Descriptions" for details.

2: After CALL or RCALL is executed, RETURN or RETLW will POP the Top-of-Stack (TOS) back into the PC.

3: After an exception is processed, RETFIE will POP the Top-of-Stack (TOS) back into the PC.

4: This condition/instruction is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

5: This condition instruction is only available in dsPIC33E and PIC24E devices.

# 4.3 INSTRUCTION STALLS

In order to maximize the data space EA calculation and operand fetch time, the X data space read and write accesses are partially pipelined. A consequence of this pipelining is that address register data dependencies may arise between successive read and write operations using common registers.

'Read After Write' (RAW) dependencies occur across instruction boundaries and are detected by the hardware. An example of a RAW dependency would be a write operation that modifies W5, followed by a read operation that uses W5 as an Address Pointer. The contents of W5 will not be valid for the read operation until the earlier write completes. This problem is resolved by stalling the instruction execution for one instruction cycle, which allows the write to complete before the next read is started.

# 4.3.1 RAW Dependency Detection

During the instruction pre-decode, the core determines if any address register dependency is imminent across an instruction boundary. The stall detection logic compares the W register (if any) used for the destination EA of the instruction currently being executed with the W register to be used by the source EA (if any) of the prefetched instruction. When a match between the destination and source registers is identified, a set of rules are applied to decide whether or not to stall the instruction by one cycle. Table 4-6 lists various RAW conditions which cause an instruction execution stall.

Destination Address Mode Using Wn	Source Address Mode Using Wn	Stall Required?	Examples <sup>(2)</sup> (Wn = W2)
Direct	Direct	No Stall	ADD.W W0, W1, W2 MOV.W W2, W3
Indirect	Direct	No Stall	ADD.W W0, W1, [W2] MOV.W W2, W3
Indirect	Indirect	No Stall	ADD.W W0, W1, [W2] MOV.W [W2], W3
Indirect	Indirect with pre/post-modification	No Stall	ADD.W W0, W1, [W2] MOV.W [W2++], W3
Indirect with pre/post-modification	Direct	No Stall	ADD.W W0, W1, [W2++] MOV.W W2, W3
Direct	Indirect	Stall <sup>(1)</sup>	ADD.W W0, W1, W2 MOV.W [W2], W3
Direct	Indirect with pre/post-modification	Stall <sup>(1)</sup>	ADD.W W0, W1, W2 MOV.W [W2++], W3
Indirect	Indirect	Stall <sup>(1)</sup>	ADD.W W0, W1, [W2](2) MOV.W [W2], W3(2)
Indirect	Indirect with pre/post-modification	Stall <sup>(1)</sup>	ADD.W W0, W1, [W2](2) MOV.W [W2++], W3(2)
Indirect with pre/post-modification	Indirect	Stall <sup>(1)</sup>	ADD.W W0, W1, [W2++] MOV.W [W2], W3
Indirect with pre/post-modification	Indirect with pre/post-modification	Stall <sup>(1)</sup>	ADD.W W0, W1, [W2++] MOV.W [W2++], W3

Table 4-6: Raw Dependency Rules (Detection By Hardware)

**Note 1:** When stalls are detected, one cycle is added to the instruction execution time.

2: For these examples, the contents of W2 = the mapped address of W2 (0x0004).

## 4.3.2 Instruction Stalls and Exceptions

In order to maintain deterministic operation, instruction stalls are allowed to happen, even if they occur immediately prior to exception processing.

## 4.3.3 Instruction Stalls and Instructions that Change Program Flow

CALL and RCALL write to the stack using W15 and may, therefore, be subject to an instruction stall if the source read of the subsequent instruction uses W15.

GOTO, RETFIE and RETURN instructions are never subject to an instruction stall because they do not perform write operations to the working registers.

### 4.3.4 Instruction Stalls and DO/REPEAT Loops

Instructions operating in a DO or REPEAT loop are subject to instruction stalls, just like any other instruction. Stalls may occur on loop entry, loop exit and also during loop processing.

Note: DO loops are only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

## 4.3.5 Instruction Stalls and PSV

Instructions operating in PSV address space are subject to instruction stalls, just like any other instruction. Should a data dependency be detected in the instruction immediately following the PSV data access, the second cycle of the instruction will initiate a stall. Should a data dependency be detected in the instruction immediately before the PSV data access, the last cycle of the previous instruction will initiate a stall.

**Note:** Refer to the specific device family reference manual for more detailed information about RAW instruction stalls.

# 4.4 BYTE OPERATIONS

Since the data memory is byte addressable, most of the base instructions may operate in either Byte mode or Word mode. When these instructions operate in Byte mode, the following rules apply:

- All direct working register references use the Least Significant Byte of the 16-bit working register and leave the Most Significant Byte (MSB) unchanged
- All indirect working register references use the data byte specified by the 16-bit address stored in the working register
- · All file register references use the data byte specified by the byte address
- The STATUS Register is updated to reflect the result of the byte operation

It should be noted that data addresses are always represented as **byte** addresses. Additionally, the native data format is little-endian, which means that words are stored with the Least Significant Byte at the lower address, and the Most Significant Byte at the adjacent, higher address (as shown in Figure 4-3). Example 4-8 shows sample byte move operations and Example 4-9 shows sample byte math operations.

**Note:** Instructions that operate in Byte mode must use the ".b" or ".B" instruction extension to specify a byte instruction. For example, the following two instructions are valid forms of a byte clear operation:

- CLR.b W0
- CLR.B W0

### Example 4-8: Sample Byte Move Operations

```
MOV.B
           #0x30, W0
                           ; move the literal byte 0x30 to W0
Before Instruction:
   W0 = 0x5555
After Instruction:
   W0 = 0x5530
   MOV.B 0x1000, W0
                          ; move the byte at 0x1000 to W0
Before Instruction:
   W0 = 0x5555
   Data Memory 0x1000 = 0x1234
After Instruction:
   W0 = 0x5534
   Data Memory 0x1000 = 0x1234
   MOV.B W0, 0x1001 ; byte move W0 to address 0x1001
Before Instruction:
   W0 = 0 \times 1234
   Data Memory 0x1000 = 0x5555
After Instruction:
   W0 = 0 \times 1234
   Data Memory 0x1000 = 0x3455
   MOV.B W0, [W1++]
                          ; byte move W0 to [W1], then post-inc W1
Before Instruction:
   W0 = 0x1234
   W1 = 0x1001
   Data Memory 0x1000 = 0x5555
After Instruction:
   W0 = 0x1234
   W1 = 0 \times 1002
   Data Memory 0x1000 = 0x3455
```

Example 4-9: Sample Byte Math Operations

```
CLR.B
           [W6--]
                                ; byte clear [W6], then post-dec W6
Before Instruction:
    W6 = 0 \times 1001
    Data Memory 0x1000 = 0x5555
After Instruction:
    W6 = 0 \times 1000
    Data Memory 0x1000 = 0x0055
    SUB.B W0, #0x10, W1
                               ; byte subtract literal 0x10 from W0
                                ; and store to Wl
Before Instruction:
    W0 = 0x1234
   W1 = 0 \times FFFF
After Instruction:
   W0 = 0x1234
   W1 = 0xFF24
   ADD.B W0, W1, [W2++] ; byte add W0 and W1, store to [W2]
                                ; and post-inc W2
Before Instruction:
    W0 = 0x1234
   W1 = 0x5678
   W2 = 0 \times 1000
   Data Memory 0x1000 = 0x5555
After Instruction:
   W0 = 0x1234
   W1 = 0x5678
   W2 = 0 \times 1001
   Data Memory 0x1000 = 0x55AC
```

# 4.5 WORD MOVE OPERATIONS

Even though the data space is byte addressable, all move operations made in Word mode must be word-aligned. This means that for all source and destination operands, the Least Significant address bit must be '0'. If a word move is made to or from an odd address, an address error exception is generated. Likewise, all double words must be word-aligned. Figure 4-3 shows how bytes and words may be aligned in data memory. Example 4-10 contains several legal word move operations.

When an exception is generated due to a misaligned access, the exception is taken after the instruction executes. If the illegal access occurs from a data read, the operation will be allowed to complete, but the Least Significant bit of the source address will be cleared to force word alignment. If the illegal access occurs during a data write, the write will be inhibited. Example 4-11 contains several illegal word move operations.

0x1001		b0	0x1000	
0x1003	b1		0x1002	
0x1005	b3	b2	0x1004	
0x1007	b5	b4	0x1006	
0x1009	b7	b6	0x1008	
0x100B		b8	0x100A	
	Legend:		I	
	b0 – byte stored at 0x1000			
	b1 – byte stored at 0x1003			
	b3:b2 – word stored at 0x100	,		
	b7:b4 – double word stored at 0x1009:0x1006 (b4 is LSB)			
	b8 – byte stored at 0x100A			
	•	mode are not required to use		
	extension. However, they may be specified with an optional ".w" or ".W" extension,			
if do	if desired. For example, the following instructions are valid forms of a word clear			

Figure 4-3: Data Alignment in Memory

Note:	Instructions that operate in Word mode are not required to use an instruction extension. However, they may be specified with an optional ".w" or ".w" extension, if desired. For example, the following instructions are valid forms of a word clear operation:
	• CLR W0 • CLR.w W0

CLR.W W0

Example 4-10: Legal Word Move Operations

```
MOV
            #0x30, W0
                                 ; move the literal word 0x30 to W0
Before Instruction:
    W0 = 0x5555
After Instruction:
    W0 = 0 \times 0030
    MOV
            0x1000, W0
                           ; move the word at 0x1000 to W0
Before Instruction:
    W0 = 0x5555
    Data Memory 0 \times 1000 = 0 \times 1234
After Instruction:
    W0 = 0x1234
   Data Memory 0x1000 = 0x1234
            [WO], [W1++]
                               ; word move [W0] to [W1],
    MOV
                                ; then post-inc W1
Before Instruction:
    W0 = 0x1234
   W1 = 0 \times 1000
   Data Memory 0x1000 = 0x5555
   Data Memory 0x1234 = 0xAAAA
After Instruction:
    W0 = 0x1234
    W1 = 0 \times 1002
   Data Memory 0x1000 = 0xAAAA
    Data Memory 0x1234 = 0xAAAA
```

```
Example 4-11: Illegal Word Move Operations
            0x1001, W0
                                 ; move the word at 0x1001 to W0
    MOV
Before Instruction:
W0 = 0x5555
Data Memory 0x1000 = 0x1234
Data Memory 0x1002 = 0x5678
After Instruction:
W0 = 0x1234
Data Memory 0x1000 = 0x1234
Data Memory 0 \times 1002 = 0 \times 5678
ADDRESS ERROR TRAP GENERATED
(source address is misaligned, so MOV is performed)
    MOV
            W0, 0x1001
                                ; move W0 to the word at 0x1001
Before Instruction:
W0 = 0 \times 1234
Data Memory 0x1000 = 0x5555
Data Memory 0x1002 = 0x6666
After Instruction:
W0 = 0 \times 1234
Data Memory 0 \times 1000 = 0 \times 5555
Data Memory 0x1002 = 0x6666
ADDRESS ERROR TRAP GENERATED
(destination address is misaligned, so MOV is not performed)
                               ; word move [W0] to [W1],
    MOV
            [WO], [W1++]
                                 ; then post-inc W1
Before Instruction:
W0 = 0x1235
W1 = 0 \times 1000
Data Memory 0x1000 = 0x1234
Data Memory 0x1234 = 0xAAAA
Data Memory 0x1236 = 0xBBBB
After Instruction:
W0 = 0 \times 1235
W1 = 0 \times 1002
Data Memory 0x1000 = 0xAAAA
Data Memory 0x1234 = 0xAAAA
Data Memory 0x1236 = 0xBBBB
ADDRESS ERROR TRAP GENERATED
(source address is misaligned, so MOV is performed)
```

# 4.6 USING 10-BIT LITERAL OPERANDS

Several instructions that support Byte and Word mode have 10-bit operands. For byte instructions, a 10-bit literal is too large to use. So when 10-bit literals are used in Byte mode, the range of the operand must be reduced to 8 bits or the assembler will generate an error. Table 4-7 shows that the range of a 10-bit literal is 0:1023 in Word mode and 0:255 in Byte mode.

Instructions which employ 10-bit literals in Byte and Word mode are: ADD, ADDC, AND, IOR, RETLW, SUB, SUBB, and XOR. Example 4-12 shows how positive and negative literals are used in Byte mode for the ADD instruction.

Literal Value	Word Mode kk kkkk kkkk	Byte Mode kkkk kkkk
0	00 0000 0000	0000 0000
1	00 0000 0001	0000 0001
2	00 0000 0010	0000 0010
127	00 0111 1111	0111 1111
128	00 1000 0000	1000 0000
255	00 1111 1111	1111 1111
256	01 0000 0000	N/A
512	10 0000 0000	N/A
1023	11 1111 1111	N/A

Table 4-7: 10-bit Literal Coding

### Example 4-12: Using 10-bit Literals for Byte Operands

	•••••••••••••••••••••••••••••••••••••••	··· ··· -······ -·· - ··· - ··· - ···	
ADD.B	#0x80, W0	; add 128 (or -128) to WO	
ADD.B	#0x380, W0	; ERROR Illegal syntax for byte mode	
ADD.B	#0xFF, W0	; add 255 (or -1) to W0	
ADD.B	#0x3FF, W0	; ERROR Illegal syntax for byte mode	
ADD.B	#0xF, W0	; add 15 to WO	
ADD.B	#0x7F, W0	; add 127 to W0	
ADD.B	#0x100, W0	; ERROR Illegal syntax for byte mode	
L			

**Note:** Using a literal value greater than 127 in Byte mode is functionally identical to using the equivalent negative two's complement value, since the Most Significant bit of the byte is set. When operating in Byte mode, the Assembler will accept either a positive or negative literal value (i.e., #-10).

# 4.7 SOFTWARE STACK POINTER AND FRAME POINTER

## 4.7.1 Software Stack Pointer

The 16-bit MCU and DSC devices feature a software stack which facilitates function calls and exception handling. W15 is the default Stack Pointer (SP) and after any Reset, it is initialized to 0x0800 (0x1000 for PIC24E and dsPIC33E devices). This ensures that the SP will point to valid RAM and permits stack availability for exceptions, which may occur before the SP is set by the user software. The user may reprogram the SP during initialization to any location within data space.

The SP always points to the first available free word (Top-of-Stack) and fills the software stack, working from lower addresses towards higher addresses. It pre-decrements for a stack POP (read) and post-increments for a stack PUSH (write).

The software stack is manipulated using the PUSH and POP instructions. The PUSH and POP instructions are the equivalent of a MOV instruction, with W15 used as the destination pointer. For example, the contents of W0 can be PUSHed onto the Top-of-Stack (TOS) by:

PUSH WO

This syntax is equivalent to:

MOV W0,[W15++]

The contents of the TOS can be returned to W0 by:

POP WO

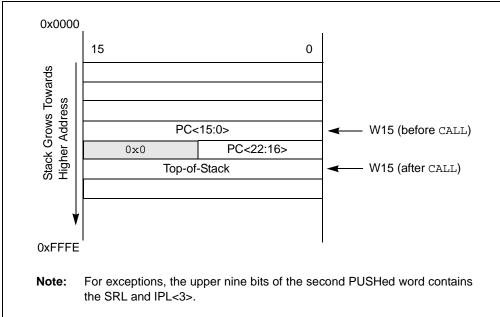
This syntax is equivalent to:

MOV [--W15],W0

During any CALL instruction, the PC is PUSHed onto the stack, such that when the subroutine completes execution, program flow may resume from the correct location. When the PC is PUSHed onto the stack, PC<15:0> is PUSHed onto the first available stack word, then PC<22:16> is PUSHed. When PC<22:16> is PUSHed, the Most Significant 7 bits of the PC are zero-extended before the PUSH is made, as shown in Figure 4-4. During exception processing, the Most Significant 7 bits of the PC are concatenated with the lower byte of the STATUS register (SRL) and IPL<3>, CORCON<3>. This allows the primary STATUS register contents and CPU Interrupt Priority Level to be automatically preserved during interrupts.

Note: In order to protect against misaligned stack accesses, W15<0> is always clear.

Figure 4-4: Stack Operation for CALL Instruction



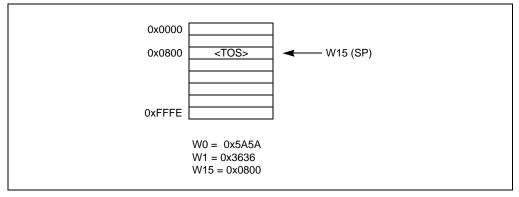
### 4.7.1.1 STACK POINTER EXAMPLE

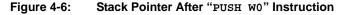
Figure 4-5 through Figure 4-8 show how the software stack is modified for the code snippet shown in Example 4-13. Figure 4-5 shows the software stack before the first PUSH has executed. Note that the SP has the initialized value of 0x0800. Furthermore, the example loads 0x5A5A and 0x3636 to W0 and W1, respectively. The stack is PUSHed for the first time in Figure 4-6 and the value contained in W0 is copied to TOS. W15 is automatically updated to point to the next available stack location, and the new TOS is 0x0802. In Figure 4-7, the contents of W1 are PUSHed onto the stack, and the new TOS becomes 0x0804. In Figure 4-8, the stack is POPped, which copies the last PUSHed value (W1) to W3. The SP is decremented during the POP operation, and at the end of the example, the final TOS is 0x0802.

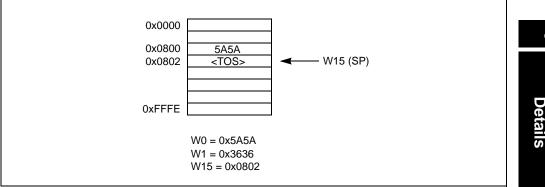
#### Example 4-13: Stack Pointer Usage

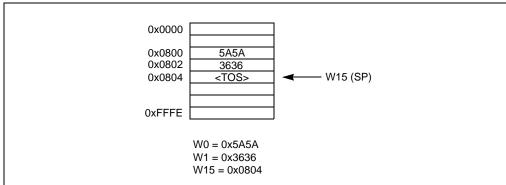
-		-	
MOV	#0x5A5A, W0	; Load W0 with 0x5A5A	
MOV	#0x3636, W1	; Load W1 with 0x3636	
PUSH	WO	; Push WO to TOS (see Figure 4-5)	
PUSH	Wl	; Push W1 to TOS (see Figure 4-7)	
POP	W3	; Pop TOS to W3 (see Figure 4-8)	



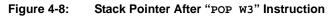


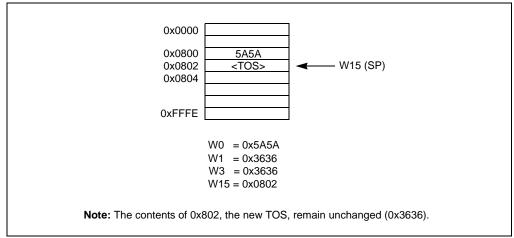












## 4.7.2 Software Stack Frame Pointer

A Stack Frame is a user-defined section of memory residing in the software stack. It is used to allocate memory for temporary variables which a function uses, and one Stack Frame may be created for each function. W14 is the default Stack Frame Pointer (FP) and it is initialized to 0x0000 on any Reset. If the Stack Frame Pointer is not used, W14 may be used like any other working register.

The link (LNK) and unlink (ULNK) instructions provide Stack Frame functionality. The LNK instruction is used to create a Stack Frame. It is used during a call sequence to adjust the SP, such that the stack may be used to store temporary variables utilized by the called function. After the function completes execution, the ULNK instruction is used to remove the Stack Frame created by the LNK instruction. The LNK and ULNK instructions must always be used together to avoid stack overflow.

#### 4.7.2.1 STACK FRAME POINTER EXAMPLE

Figure 4-9 through Figure 4-11 show how a Stack Frame is created and removed for the code snippet shown in Example 4-14. This example demonstrates how a Stack Frame operates and is not indicative of the code generated by the compiler. Figure 4-9 shows the stack condition at the beginning of the example, before any registers are pushed to the stack. Here, W15 points to the first free stack location (TOS) and W14 points to a portion of stack memory allocated for the routine that is currently executing.

Before calling the function "COMPUTE", the parameters of the function (W0, W1 and W2) are PUSHed on the stack. After the "CALL COMPUTE" instruction is executed, the PC changes to the address of "COMPUTE" and the return address of the function "TASKA" is placed on the stack (Figure 4-10). Function "COMPUTE" then uses the "LNK #4" instruction to PUSH the calling routine's Frame Pointer value onto the stack and the new Frame Pointer will be set to point to the current Stack Pointer. Then, the literal 4 is added to the Stack Pointer address in W15, which reserves memory for two words of temporary data (Figure 4-11).

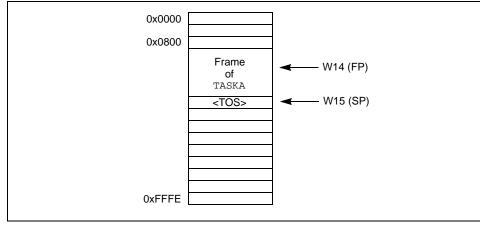
Inside the function "COMPUTE", the FP is used to access the function parameters and temporary (local) variables. [W14 + n] will access the temporary variables used by the routine and [W14 - n] is used to access the parameters. At the end of the function, the ULNK instruction is used to copy the Frame Pointer address to the Stack Pointer and then POP the calling subroutine's Frame Pointer back to the W14 register. The ULNK instruction returns the stack back to the state shown in Figure 4-10.

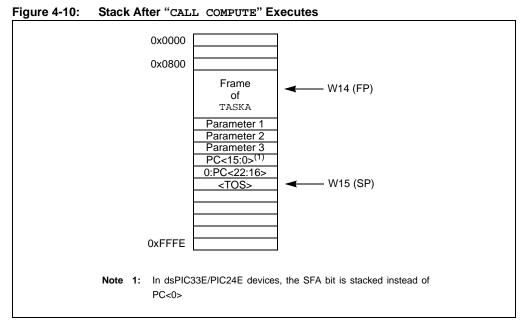
A RETURN instruction will return to the code that called the subroutine. The calling code is responsible for removing the parameters from the stack. The RETURN and POP instructions restore the stack to the state shown in Figure 4-9.

Example 4-14: Frame Pointer Usage

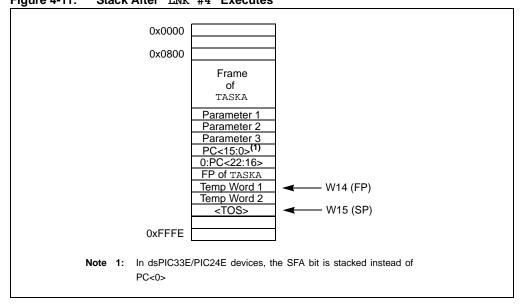
-		-
TASKA:		
PUSH	WO	; Push parameter 1
PUSH	Wl	; Push parameter 2
PUSH	W2	; Push parameter 3
CALL	COMPUTE	; Call COMPUTE function
POP	W2	; Pop parameter 3
POP	W1	; Pop parameter 2
POP	WO	; Pop parameter 1
COMPUTE:		
	щ л	· Charle ED, allocate 4 botton for local conviction
LNK	#4	; Stack FP, allocate 4 bytes for local variables
ULNK		; Free allocated memory, restore original FP
RETURN	ſ	; Return to TASKA











## 4.7.3 Stack Pointer Overflow

There is a Stack Limit register (SPLIM) associated with the Stack Pointer that is reset to 0x0000. SPLIM is a 16-bit register, but SPLIM<0> is fixed to '0', because all stack operations must be word-aligned.

The stack overflow check will not be enabled until a word write to SPLIM occurs, after which time it can only be disabled by a device Reset. All effective addresses generated using W15 as a source or destination are compared against the value in SPLIM. Should the effective address be greater than the contents of SPLIM, then a stack error trap is generated.

If stack overflow checking has been enabled, a stack error trap will also occur if the W15 effective address calculation wraps over the end of data space (0xFFFF).

Refer to the specific device family reference manual for more information on the stack error trap.

## 4.7.4 Stack Pointer Underflow

The stack is initialized to 0x0800 during Reset (0x1000 for PIC24E and dsPIC33E devices). A stack error trap will be initiated should the Stack Pointer address ever be less than 0x0800 (0x1000 for PIC24E and dsPIC33E devices).

**Note:** Locations in data space between 0x0000 and 0x07FF (0x0FFF for PIC24E and dsPIC33E devices) are, in general, reserved for core and peripheral Special Function Registers (SFRs).

# 4.7.5 Stack Frame Active (SFA) Control (dsPIC33E and PIC24E Devices)

W15 is never subject to paging and is therefore restricted to address range 0x000000 to 0x00FFFF. However, the Stack Frame Pointer (W14) for any user software function is only dedicated to that function when a stack frame addressed by W14 is active (i.e., after a LNK instruction). Therefore, it is desirable to have the ability to dynamically switch W14 between use as a general purpose W register, and use as a Stack Frame Pointer. The SFA Status bit (CORCON<2>) achieves this function without additional software overhead.

When the SFA bit is clear, W14 may be used with any page register. When SFA is set, W14 is not subject to paging and is locked into the same address range as W15 (0x000000 to 0x00FFFF). Operation of the SFA register lock is as follows:

- The LNK instruction sets SFA (and creates a stack frame)
- The ULNK instruction clears SFA (and deletes the stack frame)
- The CALL, CALL. L, and RCALL instructions also stack the SFA bit (placing it in the LSb of the stacked PC), and clear the SFA bit after the stacking operation is complete. The called procedure is now free to either use W14 as a general purpose register, or create another stack frame using the LNK instruction.
- The RETURN, RETLW and RETFIE instructions all restore the SFA bit from its previously stacked value

The SFA bit is a read-only bit. It can only be set by execution of the LNK instruction, and cleared by the ULNK, CALL, CALL, L, and RCALL instructions.

Note: In dsPIC33E and PIC24E devices, the SFA bit is stacked instead of PC<0>.

© 2005-2011 Microchip Technology Inc.

#### 4.8 CONDITIONAL BRANCH INSTRUCTIONS

Conditional branch instructions are used to direct program flow, based on the contents of the STATUS register. These instructions are generally used in conjunction with a Compare class instruction, but they may be employed effectively after any operation that modifies the STATUS register.

The compare instructions CP, CPO and CPB, perform a subtract operation (minuend - subtrahend), but do not actually store the result of the subtraction. Instead, compare instructions just update the flags in the STATUS register, such that an ensuing conditional branch instruction may change program flow by testing the contents of the updated STATUS register. If the result of the STATUS register test is true, the branch is taken. If the result of the STATUS register test is false, the branch is not taken.

The conditional branch instructions supported by the dsPIC30F, dsPIC33F, and dsPIC33E devices are shown in Table 4-8. This table identifies the condition in the STATUS register which must be true for the branch to be taken. In some cases, just a single bit is tested (as in BRA C), while in other cases, a complex logic operation is performed (as in BRA GT). For dsPIC30F, dsPIC33F, and dsPIC33E devices, it is worth noting that both signed and unsigned conditional tests are supported, and that support is provided for DSP algorithms with the OA, OB, SA and SB condition mnemonics.

	Status Test
Carry (not Borrow)	С
Signed greater than or equal	(N&&OV)    (N&&OV)
Unsigned greater than or equal	С
Signed greater than	$(\overline{Z}\&\&\overline{N}\&\&\overline{OV}) \parallel (\overline{Z}\&\&N\&\&OV)$
Unsigned greater than	C&&Z
Signed less than or equal	Z    (N&&OV)    (N&&OV)
Unsigned less than or equal	<u>C</u>    z
Signed less than	(N&&OV)    (N&&OV)
Unsigned less than	С
Negative	N
Not Carry (Borrow)	С
Not Negative	N
Not Overflow	OV
Not Zero	Z
Accumulator A overflow	OA
Accumulator B overflow	OB
Overflow	OV
Accumulator A saturate	SA
Accumulator B saturate	SB
Zero	Z
	Signed greater than or equal Unsigned greater than or equal Signed greater than Unsigned greater than Signed less than or equal Unsigned less than or equal Signed less than Unsigned less than Unsigned less than Negative Not Carry (Borrow) Not Carry (Borrow) Not Negative Not Overflow Not Zero Accumulator A overflow Accumulator B overflow Overflow Accumulator A saturate Accumulator B saturate

Table 4-8: **Conditional Branch Instructions** 

**Note 1:** Instructions are of the form: BRA mnemonic, Expr.

2: GEU is identical to C and will reverse assemble to BRA C, Expr.

3: LTU is identical to NC and will reverse assemble to BRA NC, Expr.

This condition is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices. 4:

Note: The "Compare and Skip" instructions (CPBEQ, CPBGT, CPBLT, CPBNE, CPSEQ, CPSGT, CPSLT, and CPSNE) do not modify the STATUS register.

## 4.9 Z STATUS BIT

The Z Status bit is a special zero Status bit that is useful for extended precision arithmetic. The Z bit functions like a normal Z flag for all instructions, except those that use the Carry/Borrow input (ADDC, CPB, SUBB and SUBBR). For the ADDC, CPB, SUBB and SUBBR instructions, the Z bit can only be cleared and never set. If the result of one of these instructions is non-zero, the Z bit will be cleared and will remain cleared, regardless of the result of subsequent ADDC, CPB, SUBB or SUBBR operations. This allows the Z bit to be used for performing a simple zero check on the result of a series of extended precision operations.

A sequence of instructions working on multi-precision data (starting with an instruction with no Carry/Borrow input), will automatically logically AND the successive results of the zero test. All results must be zero for the Z flag to remain set at the end of the sequence of operations. If the result of the ADDC, CPB, SUBB or SUBBR instruction is non-zero, the Z bit will be cleared and remain cleared for all subsequent ADDC, CPB, SUBB or SUBBR instructions. Example 4-15 shows how the Z bit operates for a 32-bit addition. It shows how the Z bit is affected for a 32-bit addition implemented with an ADD/ADDC instruction sequence. The first example generates a zero result for only the most significant word, and the second example generates a zero result for both the least significant word and most significant word.

### Example 4-15: 'Z' Status bit Operation for 32-bit Addition

; Add two doubles (W0:W1 and W2:W3)	
; Store the result in W5:W4	
ADD W0, W2, W4 ; Add LSWord and store to W4	
ADDC W1, W3, W5 ; Add MSWord and store to W5	
Before 32-bit Addition (zero result for the most significant word):	
$W0 = 0 \times 2342$	
W1 = 0xFFF0	
W2 = 0x39AA	
$W3 = 0 \times 0010$	
$W4 = 0 \times 0000$	
$W5 = 0 \times 0000$	
$SR = 0 \times 0000$	
After 32-bit Addition:	
$W0 = 0 \times 2342$	
W1 = 0xFFF0	
W2 = 0x39AA	
$W3 = 0 \times 0010$	
$W4 = 0 \times 5 CEC$	
$W5 = 0 \times 0000$	
SR = 0x0201 (DC, C=1)	
Before 32-bit Addition (zero result for the least significant word and most s	ignificant word):
W0 = 0xB76E	
W1 = 0xFB7B	
W2 = 0x4892	
$W3 = 0 \times 0484$	
$W4 = 0 \times 0000$	
$W5 = 0 \times 0000$	
$SR = 0 \times 0000$	
After 32-bit Addition:	
$W0 = 0 \times B76E$	
W1 = 0xFB7B	
W2 = 0x4892	
$W3 = 0 \times 0485$	
$W4 = 0 \times 00000$	
$W5 = 0 \times 0000$	
$SR = 0 \times 0103 \ (DC, Z, C=1)$	

## 4.10 ASSIGNED WORKING REGISTER USAGE

The 16 working registers of the 16-bit MCU and DSC devices provide a large register set for efficient code generation and algorithm implementation. In an effort to maintain an instruction set that provides advanced capability, a stable run-time environment and backwards compatibility with earlier Microchip processor cores, some working registers have a preassigned usage. Table 4-9 summarizes these working register assignments. For the dsPIC30F, dsPIC33F, and dsPIC33E, additional details are provided in subsections Section 4.10.1 "Implied DSP Operands (dsPIC30F, dsPIC33F and dsPIC33E Devices)" through Section 4.10.3 "PIC<sup>®</sup> Microcontroller Compatibility".

Register	Special Assignment	
W0	Default WREG, Divide Quotient	
W1	Divide Remainder	
W2	"MUL f" Product least significant word	
W3	"MUL f" Product most significant word	
W4	MAC Operand <sup>(1)</sup>	
W5	MAC Operand <sup>(1)</sup>	
W6	MAC Operand <sup>(1)</sup>	
W7	MAC Operand <sup>(1)</sup>	
W8	AC Prefetch Address (X Memory) <sup>(1)</sup>	
W9	MAC Prefetch Address (X Memory) <sup>(1)</sup>	
W10	MAC Prefetch Address (Y Memory) <sup>(1)</sup>	
W11	MAC Prefetch Address (Y Memory) <sup>(1)</sup>	
W12	MAC Prefetch Offset <sup>(1)</sup>	
W13	MAC Write Back Destination <sup>(1)</sup>	
W14	Frame Pointer	
W15	Stack Pointer	

 Table 4-9:
 Special Working Register Assignments

Note 1: This assignment is only applicable in dsPIC30F, dsPIC33F, and dsPIC33E devices.

# 4.10.1 Implied DSP Operands (dsPIC30F, dsPIC33F and dsPIC33E Devices)

To assist instruction encoding and maintain uniformity among the DSP class of instructions, some working registers have pre-assigned functionality. For all DSP instructions which have prefetch ability, the following 10 register assignments must be adhered to:

- W4-W7 are used for arithmetic operands
- W8-W11 are used for prefetch addresses (pointers)
- · W12 is used for the prefetch register offset index
- W13 is used for the accumulator Write Back destination

These restrictions only apply to the DSP MAC class of instructions, which utilize working registers and have prefetch ability (described in Section 4.15 "DSP Accumulator Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)"). These instructions are CLR, ED, EDAC, MAC, MOVSAC, MPY, MPY . N and MSC.

In dsPIC33E devices, mixed-sign DSP multiplication operations are supported without the need to dynamically modify the US<1:0> bits. In this mode (US<1:0> = '10'), each input operand is treated as unsigned or signed based on which register is being used for that operand. W4 and W6 are always unsigned operand, whereas W5 and W7 are always signed operands. This feature can be used to efficiently execute extended-precision DSP multiplications.

The DSP Accumulator class of instructions (described in **Section 4.15 "DSP Accumulator Instructions (dsPIC30F, dsPIC33F and dsPIC33E Devices)**") are not required to follow the working register assignments in Table 4-9 and may freely use any working register when required.

## 4.10.2 Implied Frame and Stack Pointer

To accommodate software stack usage, W14 is the implied Frame Pointer (used by the LNK and ULNK instructions) and W15 is the implied Stack Pointer (used by the CALL, LNK, POP, PUSH, RCALL, RETFIE, RETLW, RETURN, TRAP and ULNK instructions). Even though W14 and W15 have this implied usage, they may still be used as generic operands in any instruction, with the exceptions outlined in Section 4.10.1 "Implied DSP Operands (dsPIC30F, dsPIC33F and dsPIC33E Devices)". If W14 and W15 must be used for other purposes (it is strongly advised that they remain reserved for the Frame and Stack Pointer), extreme care must be taken such that the run-time environment is not corrupted.

### 4.10.3 PIC<sup>®</sup> Microcontroller Compatibility

### 4.10.3.1 DEFAULT WORKING REGISTER WREG

To ease the migration path for users of the Microchip 8-bit PIC MCU families, the 16-bit MCU and DSC devices have matched the functionality of the PIC MCU instruction sets as closely as possible. One major difference between the 16-bit MCU and DSC and the 8-bit PIC MCU processors is the number of working registers provided. The 8-bit PIC MCU families only provide one 8-bit working register, while the 16-bit MCU and DSC families provide sixteen, 16-bit working registers. To accommodate for the one working register of the 8-bit PIC MCU, the 16-bit MCU and DSC device instruction set has designated one working register to be the default working register for all legacy file register instructions. The default working register is set to W0, and it is used by all instructions which use file register addressing.

Additionally, the syntax used by the 16-bit MCU and DSC device assembler to specify the default working register is similar to that used by the 8-bit PIC MCU assembler. As shown in the detailed instruction descriptions in **Section 5. "Instruction Descriptions"**, "WREG" must be used to specify the default working register. Example 4-16 shows several instructions that use WREG.

#### Example 4-16: Using the Default Working Register WREG

ADD	RAM100	; add RAM100 and WREG, store in RAM100
ASR	RAM100, WREG	; shift RAM100 right, store in WREG
CLR	.B WREG	; clear the WREG LS Byte
DEC	RAM100, WREG	; decrement RAM100, store in WREG
MOV	WREG, RAM100	; move WREG to RAM100
SET	M WREG	; set all bits in the WREG
XOR	RAM100	; XOR RAM100 and WREG, store in RAM100
1		

### 4.10.3.2 PRODH:PRODL REGISTER PAIR

Another significant difference between the Microchip 8-bit PIC MCU and 16-bit MCU and DSC architectures is the multiplier. Some PIC MCU families support an 8-bit x 8-bit multiplier, which places the multiply product in the PRODH:PRODL register pair. The 16-bit MCU and DSC devices have a 17-bit x 17-bit multiplier, which may place the result into any two successive working registers (starting with an even register), or an accumulator.

Despite this architectural difference, the 16-bit MCU and DSC devices still support the legacy file register multiply instruction (MULWF) with the "MUL{.B} f" instruction (described on page 303). Supporting the legacy MULWF instruction has been accomplished by mapping the PRODH:PRODL registers to the working register pair W3:W2. This means that when "MUL{.B} f" is executed in Word mode, the multiply generates a 32-bit product which is stored in W3:W2, where W3 has the most significant word of the product and W2 has the least significant word of the product. When "MUL{.B} f" is executed in Byte mode, the 16-bit product is stored in W2, and W3 is unaffected. Examples of this instruction are shown in Example 4-17.

-	-				,
MUL.B 0x100	;	(0x100)*WREG	(byte mode),	store t	to W2
Before Instruction:					
W0 (WREG) = 0x' W2 = 0x1235 W3 = 0x1000 Data Memory 0x0		0x1255			
After Instruction:					
W0 (WREG) = 0x' W2 = 0x01A9 W3 = 0x1000	7705				
Data Memory 0x	0100 =	0x1255			
MUL 0x100	;	(0x100)*WREG	(word mode),	store t	co W3:W2
Before Instruction:					
W0 (WREG) = 0x' W2 = 0x1235 W3 = 0x1000					
Data Memory 0x0	0100 =	0x1255			
After Instruction:					
W0 (WREG) = 0x' W2 = 0xDEA9 W3 = 0x0885					
Data Memory 0x	0100 =	0x1255			

#### Example 4-17: Unsigned f and WREG Multiply (Legacy MULWF Instruction)

### 4.10.3.3 MOVING DATA WITH WREG

The "MOV{.B} f {, WREG}" instruction (described on page 279) and "MOV{.B} WREG, f" instruction (described on page 280) allow for byte or word data to be moved between file register memory and the WREG (working register W0). These instructions provide equivalent functionality to the legacy Microchip PIC MCU MOVF and MOVWF instructions.

The "MOV $\{.B\}$  f  $\{,WREG\}$ " and "MOV $\{.B\}$  WREG, f" instructions are the only MOV instructions which support moves of byte data to and from file register memory. Example 4-18 shows several MOV instruction examples using the WREG.

**Note:** When moving word data between file register memory and the working register array, the "MOV Wns, f" and "MOV f, Wnd" instructions allow any working register (W0:W15) to be used as the source or destination register, not just WREG.

#### Example 4-18: Moving Data with WREG

MOV.B	0x1001, WREG	; move the byte stored at location 0x1001 to W0
MOV	0x1000, WREG	; move the word stored at location 0x1000 to W0
MOV.B	WREG, TBLPAG	; move the byte stored at WO to the TBLPAG register $% \left[ {{\left[ {{{\rm{TBLPAG}}} \right]_{\rm{TBLPAG}}}} \right]$
MOV	WREG, 0x804	; move the word stored at W0 to location $0 \times 804$

## 4.11 DSP DATA FORMATS (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

## 4.11.1 Integer and Fractional Data

The dsPIC30F, dsPIC33F, and dsPIC33E devices support both integer and fractional data types. Integer data is inherently represented as a signed two's complement value, where the Most Significant bit is defined as a sign bit. Generally speaking, the range of an N-bit two's complement integer is  $-2^{N-1}$  to  $2^{N-1} - 1$ . For a 16-bit integer, the data range is -32768 (0x8000) to 32767 (0x7FFF), including '0'. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to 2,147,483,647 (0x7FFF FFFF).

Fractional data is represented as a two's complement number, where the Most Significant bit is defined as a sign bit, and the radix point is implied to lie just after the sign bit. This format is commonly referred to as 1.15 (or Q15) format, where 1 is the number of bits used to represent the integer portion of the number, and 15 is the number of bits used to represent the fractional portion. The range of an N-bit two's complement fraction with this implied radix point is -1.0 to  $(1 - 2^{1-N})$ . For a 16-bit fraction, the 1.15 data range is -1.0 (0x8000) to 0.999969482 (0x7FFF), including 0.0 and it has a precision of  $3.05176 \times 10^{-5}$ . In Normal Saturation mode, the 32-bit accumulators use a 1.31 format, which enhances the precision to  $4.6566 \times 10^{-10}$ .

The dynamic range of the accumulators can be expanded by using the 8 bits of the Upper Accumulator register (ACCxU) as guard bits. Guard bits are used if the value stored in the accumulator overflows beyond the  $32^{nd}$  bit, and they are useful for implementing DSP algorithms. This mode is enabled when the ACCSAT bit (CORCON<4>) is set to '1' and it expands the accumulators to 40 bits. The guard bits are also used when the accumulator saturation is disabled. The accumulators then support an integer range of -5.498x10<sup>11</sup> (0x80 0000 0000) to 5.498x10<sup>11</sup> (0x7F FFFF FFFF). In Fractional mode, the guard bits of the accumulator do not modify the location of the radix point and the 40-bit accumulators use a 9.31 fractional format. Note that all fractional operation results are stored in the 40-bit Accumulator, justified with a 1.31 radix point. As in Integer mode, the guard bits merely increase the dynamic range of the accumulator. 9.31 fractions have a range of -256.0 (0x80 0000 0000) to (256.0 – 4.65661x10<sup>-10</sup>) (0x7F FFFF FFFF). Table 4-10 identifies the range and precision of integers and fractions on the dsPIC30F/33F/33E devices for 16-bit, 32-bit and 40-bit registers.

It should be noted that, with the exception of DSP multiplies, the ALU operates identically on integer and fractional data. Namely, an addition of two integers will yield the same result (binary number) as the addition of two fractional numbers. The only difference is how the result is interpreted by the user. However, multiplies performed by DSP operations are different. In these instructions, data format selection is made by the IF bit (CORCON<0>), and it must be set accordingly ('0' for Fractional mode, '1' for Integer mode). This is required because of the implied radix point used by dsPIC30F/33F/33E fractional numbers. In Integer mode, multiplying two 16-bit integers produces a 32-bit integer result. However, multiplying two 1.15 values generates a 2.30 result. Since the dsPIC30F, dsPIC33F, and dsPIC33E devices use a 1.31 format for the accumulators, a DSP multiply in Fractional mode also includes a left shift of one bit to keep the radix point properly aligned. This feature reduces the resolution of the DSP multiplier to  $2^{-30}$ , but has no other effect on the computation (e.g.,  $0.5 \times 0.5 = 0.25$ ).

Register Size	Integer Range	Fraction Range	Fraction Resolution
16-bit	-32768 to 32767	-1.0 to (1.0 – 2 <sup>-15</sup> )	3.052 x 10 <sup>-5</sup>
32-bit	-2,147,483,648 to 2,147,483,647	-1.0 to (1.0 – 2 <sup>-31</sup> )	4.657 x 10 <sup>-10</sup>
40-bit	-549,755,813,888 to 549,755,813,887	-256.0 to (256.0 – 2 <sup>-31</sup> )	4.657 x 10 <sup>-10</sup>

Table 4-10: dsPIC30F/33F/33E Data Ranges

## 4.11.2 Integer and Fractional Data Representation

Having a working knowledge of how integer and fractional data are represented on the dsPIC30F, dsPIC33F, and dsPIC33E is fundamental to working with the device. Both integer and fractional data treat the Most Significant bit as a sign bit, and the binary exponent decreases by one as the bit position advances toward the Least Significant bit. The binary exponent for an N-bit integer starts at (N-1) for the Most Significant bit, and ends at '0' for the Least Significant bit. For an N-bit fraction, the binary exponent starts at '0' for the Most Significant bit, and ends at (1-N) for the Least Significant bit (as shown in Figure 4-12 for a positive value and in Figure 4-13 for a negative value).

Conversion between integer and fractional representations can be performed using simple division and multiplication. To go from an N-bit integer to a fraction, divide the integer value by  $2^{N-1}$ . Similarly, to convert an N-bit fraction to an integer, multiply the fractional value by  $2^{N-1}$ .

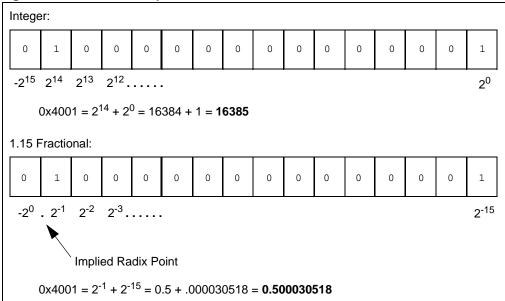
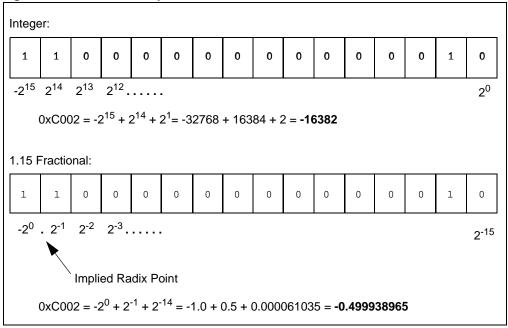


Figure 4-12: Different Representations of 0x4001

Figure 4-13: Different Representations of 0xC002



## 4.12 ACCUMULATOR USAGE (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

Accumulators A and B are utilized by DSP instructions to perform mathematical and shifting operations. Since the accumulators are 40 bits wide and the X and Y data paths are only 16 bits, the method to load and store the accumulators must be understood.

Item A in Figure 4-14 shows that each 40-bit Accumulator (ACCA and ACCB) consists of an 8-bit Upper register (ACCxU), a 16-bit High register (ACCxH) and a 16-bit Low register (ACCxL). To address the bus alignment requirement and provide the ability for 1.31 math, ACCxH is used as a destination register for loading the accumulator (with the LAC instruction), and also as a source register for storing the accumulator (with the SAC.R instruction). This is represented by Item B, Figure 4-14, where the upper and lower portions of the accumulator are shaded. In reality, during accumulator loads, ACCxL is zero backfilled and ACCxU is sign-extended to represent the sign of the value loaded in ACCxH.

When Normal (31-bit) Saturation is enabled, DSP operations (such as ADD, MAC, MSC, etc.) utilize solely ACCxH:ACCxL (Item C in Figure 4-14) and ACCxU is only used to maintain the sign of the value stored in ACCxH:ACCxL. For instance, when a MPY instruction is executed, the result is stored in ACCxH:ACCxL, and the sign of the result is extended through ACCxU.

When Super Saturation is enabled, or when saturation is disabled, all registers of the accumulator may be used (Item D in Figure 4-14) and the results of DSP operations are stored in ACCxU:ACCxH:ACCxL. The benefit of ACCxU is that it increases the dynamic range of the accumulator, as described in **Section 4.11.1 "Integer and Fractional Data**". Refer to Table 4-10 to see the range of values which may be stored in the accumulator when in Normal and Super Saturation modes.

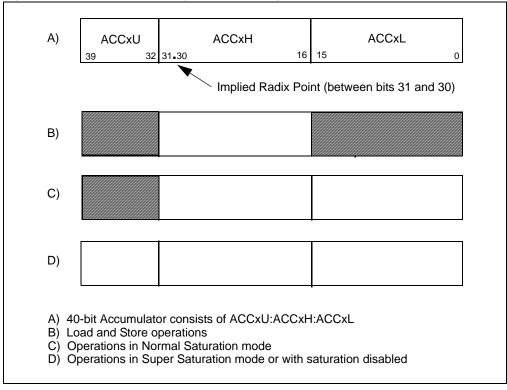


Figure 4-14: Accumulator Alignment and Usage

## 4.13 ACCUMULATOR ACCESS (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The six registers of Accumulator A and Accumulator B are memory mapped like any other Special Function Register. This feature allows them to be accessed with file register or indirect addressing, using any instruction which supports such addressing. However, it is recommended that the DSP instructions LAC, SAC and SAC.R be used to load and store the accumulators, since they provide sign-extension, shifting and rounding capabilities. LAC, SAC and SAC.R instruction details are provided in Section 5. "Instruction Descriptions".

- **Note 1:** For convenience, ACCAU and ACCBU are sign-extended to 16 bits. This provides the flexibility to access these registers using either Byte or Word mode (when file register or indirect addressing is used).
  - 2: The OA, OB, SA or SB bit cannot be set by writing overflowed values to the memory mapped accumulators using MOV instructions, as these status bits are only affected by DSP operations.

## 4.14 DSP MAC INSTRUCTIONS (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The DSP Multiply and Accumulate (MAC) operations are a special suite of instructions which provide the most efficient use of the dsPIC30F, dsPIC33F, and dsPIC33E architectures. The DSP MAC instructions, shown in Table 4-11, utilize both the X and Y data paths of the CPU core, which enables these instructions to perform the following operations all in one cycle:

- two reads from data memory using prefetch working registers (MAC Prefetches)
- two updates to prefetch working registers (MAC Prefetch Register Updates)
- one mathematical operation with an accumulator (MAC Operations)

In addition, four of the ten DSP MAC instructions are also capable of performing an operation with one accumulator, while storing out the rounded contents of the alternate accumulator. This feature is called accumulator Write Back (WB) and it provides flexibility for the software developer. For instance, the accumulator WB may be used to run two algorithms concurrently, or efficiently process complex numbers, among other things.

Instruction	Description	Accumulator WB?
CLR	Clear accumulator	Yes
ED	Euclidean distance (no accumulate)	No
EDAC	Euclidean distance	No
MAC	Multiply and accumulate	Yes
MAC	Square and accumulate	No
MOVSAC	Move from X and Y bus	Yes
MPY	Multiply to accumulator	No
MPY	Square to accumulator	No
MPY.N	Negative multiply to accumulator	No
MSC	Multiply and subtract	Yes

Table 4-11: DSP MAC Instructions

### 4.14.1 MAC Prefetches

Prefetches (or data reads) are made using the effective address stored in the working register. The two prefetches from data memory must be specified using the working register assignments shown in Table 4-9. One read must occur from the X data bus using W8 or W9, and one read must occur from the Y data bus using W10 or W11. The allowed destination registers for both prefetches are W4-W7.

As shown in Table 4-3, one special Addressing mode exists for the MAC class of instructions. This mode is the Register Offset Addressing mode and utilizes W12. In this mode, the prefetch is made using the effective address of the specified working register, plus the 16-bit signed value stored in W12. Register Offset Addressing may only be used in the X space with W9, and in the Y-space with W11.

#### **MAC Prefetch Register Updates** 4.14.2

After the MAC prefetches are made, the effective address stored in each prefetch working register may be modified. This feature enables efficient single-cycle processing for data stored sequentially in X and Y memory. Since all DSP instructions execute in Word mode, only even numbered updates may be made to the effective address stored in the working register. Allowable address modifications to each prefetch register are -6, -4, -2, 0 (no update), +2, +4 and +6. This means that effective address updates may be made up to 3 words in either direction.

When the Register Offset Addressing mode is used, no update is made to the base prefetch register (W9 or W11), or the offset register (W12).

#### **MAC Operations** 4.14.3

The mathematical operations performed by the MAC class of DSP instructions center around multiplying the contents of two working registers and either adding or storing the result to either Accumulator A or Accumulator B. This is the operation of the MAC, MPY, MPY.N and MSC instructions. Table 4-9 shows that W4-W7 must be used for data source operands in the MAC class of instructions. W4-W7 may be combined in any fashion, and when the same working register is specified for both operands, a square or square and accumulate operation is performed.

For the ED and EDAC instructions, the same multiplicand operand must be specified by the instruction, because this is the definition of the Euclidean Distance operation. Another unique feature about this instruction is that the values prefetched from X and Y memory are not actually stored in W4-W7. Instead, only the difference of the prefetched data words is stored in W4-W7.

The two remaining MAC class instructions, CLR and MOVSAC, are useful for initiating or completing a series of MAC or EDAC instructions and do not use the multiplier. CLR has the ability to clear Accumulator A or B, prefetch two values from data memory and store the contents of the other accumulator. Similarly, MOVSAC has the ability to prefetch two values from data memory and store the contents of either accumulator.

#### 4.14.4 MAC Write Back

The write back ability of the MAC class of DSP instructions facilitates efficient processing of algorithms. This feature allows one mathematical operation to be performed with one accumulator, and the rounded contents of the other accumulator to be stored in the same cycle. As indicated in Table 4-9, register W13 is assigned for performing the write back, and two Addressing modes are supported: Direct and Indirect with Post-Increment.

The CLR, MOVSAC and MSC instructions support accumulator Write Back, while the ED, EDAC, MPY and MPY. N instructions do not support accumulator Write Back. The MAC instruction, which multiplies two working registers which are not the same, also supports accumulator Write Back. However, the square and accumulate MAC instruction does not support accumulator Write Back (see Table 4-11).

#### 4.14.5 MAC Syntax

The syntax of the MAC class of instructions can have several formats, which depend on the instruction type and the operation it is performing, with respect to prefetches and accumulator Write Back. With the exception of the CLR and MOVSAC instructions, all MAC class instructions must specify a target accumulator along with two multiplicands, as shown in Example 4-19.

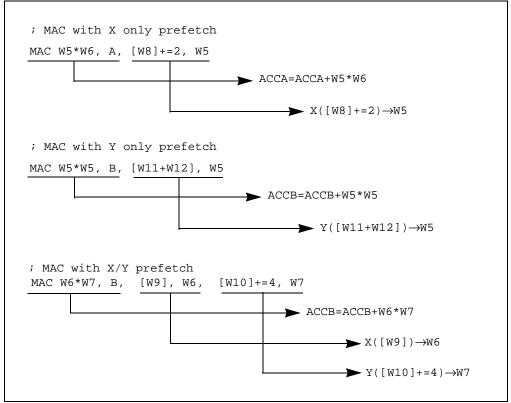
#### Example 4-19: Base MAC Syntax

; MAC with MAC W4*W5,	prefetch						
; MAC with MAC W7*W7,	prefetch						
		->	Multiply	W7*W7,	Accumulate	to	ACCB

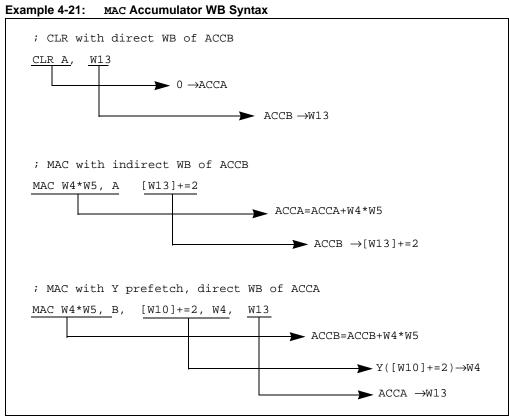
S

If a prefetch is used in the instruction, the assembler is capable of discriminating between the X or Y data prefetch based on the register used for the effective address. [W8] or [W9] specifies the X prefetch and [W10] or [W11] specifies the Y prefetch. Brackets around the working register are required in the syntax, and they designate that indirect addressing is used to perform the prefetch. When address modification is used, it must be specified using a minus-equals or plus-equals "C"-like syntax (i.e., "[W8] – = 2" or "[W8] + = 6"). When Register Offset Addressing is used for the prefetch, W12 is placed inside the brackets ([W9 + W12] for X prefetches and [W11 + W12] for Y prefetches). Each prefetch operation must also specify a prefetch destination register (W4-W7). In the instruction syntax, the destination register appears before the prefetch register. Legal forms of prefetch are shown in Example 4-20.



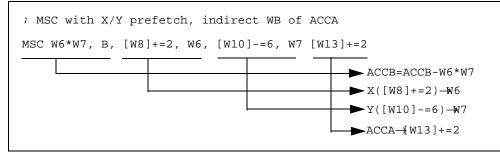


If an accumulator Write Back is used in the instruction, it is specified last. The Write Back must use the W13 register, and allowable forms for the Write Back are "W13" for direct addressing and "[W13] + = 2" for indirect addressing with post-increment. By definition, the accumulator not used in the mathematical operation is stored, so the Write Back accumulator is not specified in the instruction. Legal forms of accumulator Write Back (WB) are shown in Example 4-21.



Putting it all together, an MSC instruction which performs two prefetches and a write back is shown in Example 4-22.

#### Example 4-22: MSC Instruction with Two Prefetches and Accumulator Write Back



# 4.15 DSP ACCUMULATOR INSTRUCTIONS (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The DSP Accumulator instructions do not have prefetch or accumulator WB ability, but they do provide the ability to add, negate, shift, load and store the contents of either 40-bit Accumulator. In addition, the ADD and SUB instructions allow the two accumulators to be added or subtracted from each other. DSP Accumulator instructions are shown in Table 4-12 and instruction details are provided in Section 5. "Instruction Descriptions".

Instruction	Description	Accumulator WB?
ADD	Add accumulators	No
ADD	16-bit signed accumulator add	No
LAC	Load accumulator	No
NEG	Negate accumulator	No
SAC	Store accumulator	No
SAC.R	Store rounded accumulator	No
SFTAC	Arithmetic shift accumulator by Literal	No
SFTAC	Arithmetic shift accumulator by (Wn)	No
SUB	Subtract accumulators	No

Table 4-12: DSP Accumulator Instructions

# 4.16 SCALING DATA WITH THE FBCL INSTRUCTION (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

To minimize quantization errors that are associated with data processing using DSP instructions, it is important to utilize the complete numerical result of the operations. This may require scaling data up to avoid underflow (i.e., when processing data from a 12-bit ADC), or scaling data down to avoid overflow (i.e., when sending data to a 10-bit DAC). The scaling, which must be performed to minimize quantization error, depends on the dynamic range of the input data which is operated on, and the required dynamic range of the output data. At times, these conditions may be known beforehand and fixed scaling may be employed. In other cases, scaling conditions may not be fixed or known, and then dynamic scaling must be used to process data.

The FBCL instruction (Find First Bit Change Left) can efficiently be used to perform dynamic scaling, because it determines the exponent of a value. A fixed point or integer value's exponent represents the amount which the value may be shifted before overflowing. This information is valuable, because it may be used to bring the data value to "full scale", meaning that its numeric representation utilizes all the bits of the register it is stored in.

The FBCL instruction determines the exponent of a word by detecting the first bit change starting from the value's sign bit and working towards the LSB. Since the dsPIC DSC device's barrel shifter uses negative values to specify a left shift, the FBCL instruction returns the negated exponent of a value. If the value is being scaled up, this allows the ensuing shift to be performed immediately with the value returned by FBCL. Additionally, since the FBCL instruction only operates on signed quantities, FBCL produces results in the range of -15:0. When the FBCL instruction returns '0', it indicates that the value is already at full scale. When the instruction returns -15, it indicates that the value cannot be scaled (as is the case with 0x0 and 0xFFFF). Table 4-13 shows word data with various dynamic ranges, their exponents, and the value after scaling each data to maximize the dynamic range. Example 4-23 shows how the FBCL instruction may be used for block processing.

Word Value	Exponent	Full Scale Value (Word Value << Exponent)
0x0001	14	0x4000
0x0002	13	0x4000
0x0004	12	0x4000
0x0100	6	0x4000
0x01FF	6	0x7FC0
0x0806	3	0x4030
0x2007	1	0x400E
0x4800	0	0x4800
0x7000	0	0x7000
0x8000	0	0x8000
0x900A	0	0x900A
0xE001	2	0x8004
0xFF07	7	0x8380

Table 4-13: Scaling Examples

Note: For the word values 0x0000 and 0xFFFF, the FBCL instruction returns -15.

As a practical example, assume that block processing is performed on a sequence of data with very low dynamic range stored in 1.15 fractional format. To minimize quantization errors, the data may be scaled up to prevent any quantization loss which may occur as it is processed. The FBCL instruction can be executed on the sample with the largest magnitude to determine the optimal scaling value for processing the data. Note that scaling the data up is performed by left shifting the data. This is demonstrated with the code snippet below.

#### Example 4-23: Scaling with FBCL

; assume WO contains the largest absolute value of the data block
; assume W4 points to the beginning of the data block
; assume the block of data contains BLOCK_SIZE words
A determine the encount to use for another
; determine the exponent to use for scaling
FBCL W0, W2 ; store exponent in W2
; scale the entire data block before processing
DO #(BLOCK_SIZE-1), SCALE
LAC [W4], A ; move the next data sample to ACCA
SFTAC A, W2 ; shift ACCA by W2 bits
SCALE:
SAC A, [W4++] ; store scaled input (overwrite original)
; now process the data
; (processing block goes here)
, (processing stock goes here,

## 4.17 NORMALIZING THE ACCUMULATOR WITH THE FBCL INSTRUCTION (dsPIC30F, dsPIC33F AND dsPIC33E DEVICES)

The process of scaling a quantized value for its maximum dynamic range is known as normalization (the data in the third column in Table 4-13 contains normalized data). Accumulator normalization is a technique used to ensure that the accumulator is properly aligned before storing data from the accumulator, and the FBCL instruction facilitates this function.

The two 40-bit accumulators each have 8 guard bits from the ACCxU register, which expands the dynamic range of the accumulators from 1.31 to 9.31, when operating in Super Saturation mode (see Section 4.11.1 "Integer and Fractional Data"). However, even in Super Saturation mode, the Store Rounded Accumulator (SAC.R) instruction only stores 16-bit data (in 1.15 format) from ACCxH, as described in Section 4.12 "Accumulator Usage (dsPIC30F, dsPIC33F and dsPIC33E Devices)". Under certain conditions, this may pose a problem.

Proper data alignment for storing the contents of the accumulator may be achieved by scaling the accumulator down if ACCxU is in use, or scaling the accumulator up if all of the ACCxH bits are not being used. To perform such scaling, the FBCL instruction must operate on the ACCxU byte and it must operate on the ACCxH word. If a shift is required, the ALU's 40-bit shifter is employed, using the SFTAC instruction to perform the scaling. Example 4-24 contains a code snippet for accumulator normalization.

#### Example 4-24: Normalizing with FBCL

; assume ar	n operation in A	CCA ha	s just completed (SR intact)
; assume th	ne processor is	in sup	er saturation mode
; assume AC	CCAH is defined	to be	the address of ACCAH (0x24)
MOV	#ACCAH, W5	;	W5 points to ACCAH
BRA	OA, FBCL_GUARD	;	if overflow we right shift
FBCL_HI:			
FBCL	[W5], WO	; ;	extract exponent for left shift
BRA	SHIFT_ACC	; ]	branch to the shift
FBCL_GUARD	:		
FBCL	[++W5], WO	; ;	extract exponent for right shift
ADD.B	WO, #15, WO	; ;	adjust the sign for right shift
SHIFT_ACC:			
SFTAC	A, W0	;	shift ACCA to normalize

# 4.18 EXTENDED-PRECISON ARITHMETIC USING MIXED-SIGN MULTIPLICATIONS (dsPIC33E ONLY)

Many DSP algorithms utilize extended-precision arithmetic operations (operations with 32-bit or 64-bit operands and results) to enhance the resolution and accuracy of computations. These can be implemented using 16-bit signed or unsigned multiplications; however, this would require some additional processing and shifting of the data to obtain the correct results. To enable such extended-precision algorithms to be computed faster, dsPIC33E devices support an optional implicit mixed-sign multiplication mode, which is selected by setting US<1:0> (CORCON<13:12>) = '10'.

In this mode, mixed-sign (unsigned x signed and signed x unsigned) multiplications can be performed without the need to dynamically reconfigure the US<1:0> bits and shift data to account for the difference in operand formats. Moreover, signed x signed and unsigned x unsigned multiplications can also be performed without changing the multiplication mode. Each input operand is implicitly treated as an unsigned number if the working register being used to specify the operand is either W4 or W6. Similarly, an operand is treated as a signed number if the register used is either W5 or W7. The DSP Engine selects the type of multiplication to be performed based on the operand registers used, thereby eliminating the need for the user software to modify the US<1:0> bits.

The execution time reductions provided by the implicit mixed-sign multiplication feature is illustrated in the following code example, where the instruction cycle count for performing a 32-bit multiplication is reduced from 7 cycles to 4 cycles when the mixed-sign multiplication mode is enabled.

Example 4-25:	32-bit Signed Multi	plication using Im	plicit Mixed-Sign Mode

Case A	- Mixed-Sign Multiplication Mode Not Enabled
	W5, W6, W0; Wordl (signed) x Word2 (unsigned) W4, W7, W2; Word0 (unsigned) x Word3 (signed)
CLR	B ; Clear Accumulator B
ADD	W1, B
ADD	W3, В
SFTAC	B, #15 ; Shift right by 15 bits to align for Q31 format
MAC	W5*W7, B; Word1 (signed) x Word 3 (signed)
Case B	- Mixed-Sign Multiplication Mode Enabled
MPY	W5*W6, B; Word1 (signed) x Word2 (unsigned)
MAC	W4*W7, B; Word0 (unsigned) x Word3 (signed)
SFTAC	B, #15 ; Shift right by 15 bits to align for Q31 format
MAC	W5*W7, B; Word1 (signed) x Word 3 (signed)

Besides DSP instructions, MCU multiplication (MUL) instructions can also utilize Accumulator A or Accumulator B as a result destination, which enables faster extended-precision arithmetic even when not using DSP multiplication instructions such as MPY or MAC.



# **Section 5. Instruction Descriptions**

## HIGHLIGHTS

This section of the manual contains the following major topics:

5.1	Instruction Symbols	94
5.2	Instruction Encoding Field Descriptors Introduction	94
5.3	Instruction Description Example	98
5.4	Instruction Descriptions	99

## 5.1 INSTRUCTION SYMBOLS

All the symbols used in **Section 5.4 "Instruction Descriptions**" are listed in Table 1-2.

## 5.2 INSTRUCTION ENCODING FIELD DESCRIPTORS INTRODUCTION

All instruction encoding field descriptors used in **Section 5.4** "Instruction Descriptions" are shown in Table 5-2 through Table 5-12.

 Table 5-1:
 Instruction Encoding Field Descriptors

Field	Description
A <sup>(1)</sup>	Accumulator selection bit: 0 = ACCA; 1 = CCB
aa <sup>(1)</sup>	Accumulator Write Back mode (see Table 5-12)
В	Byte mode selection bit: 0 = word operation; 1 = byte operation
bbbb	4-bit bit position select: 0000 = LSB; 1111 = MSB
D	Destination address bit: 0 = result stored in WREG;
	1 = result stored in file register
dddd	Wd destination register select: 0000 = W0; 1111 = W15
f ffff ffff ffff	13-bit register file address (0x0000 to 0x1FFF)
fff ffff ffff ffff	15-bit register file word address (implied 0 LSB) (0x0000 to 0xFFFE)
ffff ffff ffff ffff	16-bit register file byte address (0x0000 to 0xFFFF)
aaa	Register Offset Addressing mode for Ws source register (see Table 5-4)
hhh	Register Offset Addressing mode for Wd destination register (see Table 5-5)
iiii(1)	Prefetch X Operation (see Table 5-6)
jjjj <sup>(1)</sup>	Prefetch Y Operation (see Table 5-8)
k	1-bit literal field, constant data or expression
kkkk	4-bit literal field, constant data or expression
kk kkkk	6-bit literal field, constant data or expression
kkkk kkkk	8-bit literal field, constant data or expression
kk kkkk kkkk	10-bit literal field, constant data or expression
kk kkkk kkkk kkkk	14-bit literal field, constant data or expression
kkkk kkkk kkkk kkkk	16-bit literal field, constant data or expression
mm	Multiplier source select with same working registers (see Table 5-10)
mmm	Multiplier source select with different working registers (see Table 5-11)
nnnn nnnn nnnn nnn0	23-bit program address for CALL and GOTO instructions
nnn nnnn	
nnnn nnnn nnnn nnnn	16-bit program offset field for relative branch/call instructions
qqq	Addressing mode for Ws source register (see Table 5-2)
qqq	Addressing mode for Wd destination register (see Table 5-3)
rrrr	Barrel shift count
SSSS	Ws source register select: 0000 = W0; 1111 = W15
tttt	Dividend select, most significant word
VVVV	Dividend select, least significant word
W	Double Word mode selection bit: 0 = word operation;
	1 = double word operation
WWWW (1)	Wb base register select: 0000 = W0; 1111 = W15
xx <sup>(1)</sup>	Prefetch X Destination (see Table 5-7)
xxxx xxxx xxxx xxxx (1)	16-bit unused field (don't care)
уу <sup>(1)</sup>	Prefetch Y Destination (see Table 5-9)
Z	Bit test destination: 0 = C flag bit; 1 = Z flag bit

Note 1: This field is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

ppp	Addressing Mode	Source Operand
000	Register Direct	Ws
001	Indirect	[Ws]
010	Indirect with Post-Decrement	[Ws]
011	Indirect with Post-Increment	[Ws++]
100	Indirect with Pre-Decrement	[Ws]
101	Indirect with Pre-Increment	[++Ws]
11x	Unused	

 Table 5-2:
 Addressing Modes for Ws Source Register

### Table 5-3: Addressing Modes for Wd Destination Register

qqq	Addressing Mode	Destination Operand
000	Register Direct	Wd
001	Indirect	[Wd]
010	Indirect with Post-Decrement	[Wd]
011	Indirect with Post-Increment	[Wd++]
100	Indirect with Pre-Decrement	[Wd]
101	Indirect with Pre-Increment	[++Wd]
11x	Unused (an attempt to use this Addressing	mode will force a RESET instruction)

### Table 5-4: Offset Addressing Modes for Ws Source Register (with Register Offset)

aaa	Addressing Mode	Source Operand
000	Register Direct	Ws
001	Indirect	[Ws]
010	Indirect with Post-Decrement	[Ws]
011	Indirect with Post-Increment	[Ws++]
100	Indirect with Pre-Decrement	[Ws]
101	Indirect with Pre-Increment	[++Ws]
11x	Indirect with Register Offset	[Ws+Wb]

#### Table 5-5: Offset Addressing Modes for Wd Destination Register (with Register Offset)

hhh	Addressing Mode	Source Operand
000	Register Direct	Wd
001	Indirect	[Wd]
010	Indirect with Post-Decrement	[Wd]
011	Indirect with Post-Increment	[Wd++]
100	Indirect with Pre-Decrement	[Wd]
101	Indirect with Pre-Increment	[++Wd]
11x	Indirect with Register Offset	[Wd+Wb]

## 16-bit MCU and DSC Programmer's Reference Manual

iiii	Operation
0000	Wxd = [W8]
0001	Wxd = [W8], W8 = W8 + 2
0010	Wxd = [W8], W8 = W8 + 4
0011	Wxd = [W8], W8 = W8 + 6
0100	No Prefetch for X Data Space
0101	Wxd = [W8], W8 = W8 - 6
0110	Wxd = [W8], W8 = W8 - 4
0111	Wxd = [W8], W8 = W8 - 2
1000	Wxd = [W9]
1001	Wxd = [W9], W9 = W9 + 2
1010	Wxd = [W9], W9 = W9 + 4
1011	Wxd = [W9], W9 = W9 + 6
1100	Wxd = [W9 + W12]
1101	Wxd = [W9], W9 = W9 - 6
1110	Wxd = [W9], W9 = W9 - 4
1111	Wxd = [W9], W9 = W9 - 2

## Table 5-6: X Data Space Prefetch Operation (dsPIC30F, dsPIC33F and dsPIC33E)



xx	Wxd
00	W4
01	W5
10	W6
11	W7

Table 5-8:	Y Data Space Prefetch Operation (dsPIC30F, dsPIC33F and dsPIC3	3E)
------------	--	-----

jjjj	Operation
0000	Wyd = [W10]
0001	Wyd = [W10], W10 = W10 + 2
0010	Wyd = [W10], W10 = W10 + 4
0011	Wyd = [W10], W10 = W10 + 6
0100	No Prefetch for Y Data Space
0101	Wyd = [W10], W10 = W10 - 6
0110	Wyd = [W10], W10 = W10 - 4
0111	Wyd = [W10], W10 = W10 - 2
1000	Wyd = [W11]
1001	Wyd = [W11], W11 = W11 + 2
1010	Wyd = [W11], W11 = W11 + 4
1011	Wyd = [W11], W11 = W11 + 6
1100	Wyd = [W11 + W12]
1101	Wyd = [W11], W11 = W11 - 6
1110	Wyd = [W11], W11 = W11 - 4
1111	Wyd = [W11], W11 = W11 - 2

#### Table 5-9: Y Data Space Prefetch Destination (dsPIC30F, dsPIC33F and dsPIC33E)

уу	Wyd
00	W4
01	W5
10	W6
11	W7

## Table 5-10: MAC or MPY Source Operands (Same Working Register) (dsPIC30F, dsPIC33F and dsPIC33E)

mm	Multiplicands
00	W4 * W4
01	W5 * W5
10	W6 * W6
11	W7 * W7

## Table 5-11: MAC or MPY Source Operands (Different Working Register) (dsPIC30F, dsPIC33F and dsPIC33E)

mmm	Multiplicands
000	W4 * W5
001	W4 * W6
010	W4 * W7
011	Invalid
100	W5 * W6
101	W5 * W7
110	W6 * W7
111	Invalid

## Table 5-12: MAC Accumulator Write Back Selection (dsPIC30F, dsPIC33F and dsPIC33E)

aa	Write Back Selection
00	W13 = Other Accumulator (Direct Addressing)
01	[W13] + = 2 = Other Accumulator (Indirect Addressing with Post-Increment)
10	No Write Back
11	Invalid

#### Table 5-13: MOVPAG Destination Selection

PP	Target Page Register
00	DSRPAG
01	DSWPAG
10	TBLPAG
11	Reserved – do not use

### Table 5-14: Accumulator Selection

Α	Target Accumulator
0	Accumulator A
1	Accumulator B

## 5.3 INSTRUCTION DESCRIPTION EXAMPLE

The example description below is for the fictitious instruction FOO. The following example instruction was created to demonstrate how the table fields (syntax, operands, operation, etc.) are used to describe the instructions presented in Section 5.4 "Instruction Descriptions".

FOO	The Header field summarizes what the instruction does							
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
				Х	Х	Х		
	Cells marke device fami	ed with an 'X' ly.	indicate the i	nstruction is in	mplemented	for that		
Syntax:	The Syntax field consists of an optional label, the instruction mnemonic, any optional extensions which exist for the instruction and the operands for the instruction. Most instructions support more than one operand variant to support the various Addressing modes. In these circumstances, all possible instruction operands are listed beneath each other and are enclosed in braces.							
Operands:	may take. C	The Operands field describes the set of values which each of the operands may take. Operands may be accumulator registers, file registers, literal constants (signed or unsigned), or working registers.						
Operation:	The Operat	ion field sumr	marizes the o	peration perfo	ormed by the	instruction.		
Status Affected:		Affected field the instruction order.						
Encoding:	fields are ex	The Encoding field shows how the instruction is bit encoded. Individual bit fields are explained in the Description field, and complete encoding details are provided in Table 5.2.						
Description:	The Description field describes in detail the operation performed by the instruction. A key for the encoding bits is also provided.							
Words:	The Words field contains the number of program words that are used to store the instruction in memory.							
Cycles:	The Cycles field contains the number of instruction cycles that are required to execute the instruction.							
Examples:	operates. "E	les field conta Before" and "/ clearly unders	After" register	snapshots ar	e provided, v	which allow		

## 5.4 INSTRUCTION DESCRIPTIONS

ADD		Add f to WR	EG					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	ADD{.B}	f	{,WREG}				
Operands:	f ∈ [0 81	91]						
Operation:	(f) + (WREG) $\rightarrow$ destination designated by D							
Status Affected:	DC, N, OV,	Z, C						
Encoding:	1011	0100	OBDf	ffff	ffff	ffff		
	optional W specified, t result is sto The 'B' bit s	ster, and plac REG operand he result is sto ored in the file selects byte o	l determines ored in WRE register. r word opera	the destination G. If WREG in tion ('0' for w	on register. If is not specifie vord, '1' for b	WREG is ed, the yte).		
		selects the de select the add	•		1' for file regi	ister).		
		The extensior rather than a denote a word The WREG is	word operati d operation, I	on. You may out it is not re	use a .w ext equired.	-		
Words:	1							
Cycles:	1 <sup>(1)</sup>							
read-moo details, se	lify-write ope ee <b>Note 3</b> in	24E devices, erations on no Section 3.2.7	n-CPU Speci I "Multi-Cyc	al Function R	Registers. For ns".	more		
	50.0	IU II II O O	, naa i		100 (Dyce i	lioue)		
WRE RAM10 S	0 FFC0	WR RAM <sup>2</sup>		n ] (OV, C = 1)				
Example 2: AD		AM200, WREG		AM200 to WR	EG (Word m	ode)		
WREG RAM200 SF	FFC0	WRE RAM20 S	00 FFC0	( <b>C</b> = 1)				

ADD		Add Literal	to Wn					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	ADD{.B}	#lit10,	Wn				
Operands:	lit10 $\in$ [0 255] for byte operation lit10 $\in$ [0 1023] for word operation Wn $\in$ [W0 W15]							
Operation:	lit10 + (Wn)	) →Wn						
Status Affected:	DC, N, OV,	Z, C						
Encoding:	1011	0000	0Bkk	kkkk	kkkk	dddd		
Description:		-bit unsigned l n, and place th						
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'k' bits specify the literal operand. The 'd' bits select the address of the working register.							
	<b>2:</b>	rather than a v denote a word For byte opera value [0:255]. for informatior	d operation, b ations, the lite See <mark>Section</mark>	ut it is not rec eral must be s 4.6 "Using 1	quired. specified as a <b>0-bit Literal (</b>	n unsigned <b>Dperands"</b>		
Words:	1		U	·				
Cycles:	1							
Example 1:	ADD.B	#0xFF, W7	; Add	-1 to W7 (	Byte mode)			
	Before Instructio W7 12C0 SR 0000	on )	After Instructio W7 12BF SR 0009					
Example 2:	ADD	#0xFF, W1	; Add	255 to W1	(Word mode)			
	Before Instructio W1 12C0 SR 0000	on I	After Instructio W1 13BF SR 0000					

ADD		Add Wb to	Short Litera	I		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	ADD{.B}	Wb,	#lit5,	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	(Wb) + lit5	→Wd				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	0100	0www	wBqq	qddd	d11k	kkkk
	operand, a direct addr addressing	Intents of the b and place the r ressing must b g may be used s select the ac	result in the o be used for V I for Wd.	destination re Vb. Either reg	gister Wd. Re lister direct of	egister
	The 'B' bit The 'q' bits The 'd' bits	s select the ac selects byte c s select the de s select the de provide the li	or word opera stination Ado stination reg	ation ('0' for w dress mode. ister.	vord, '1' for b	
	Note:	The extension rather than a denote a wor	word opera	tion. You ma	yusea.we	
Words:	1		· ·		I	
Cycles:	1					
Example 1:	ADD.B	W0, #0x1F,		Add W0 and Store the r	—	
	Before Instruction W0 2290	ר 	After Instructic W0 2290	n T		

Data 0F Data 10	000 DDEE	Data 01 Data 1	After Instruction W3 6006 W4 0FFE FFE 600C 000 DDEE		result in [	W4 ]	
ADD	SR 0000	Add Wb to	SR 0000 Ws				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
-	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	ADD{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]		
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]					
Operation:	(Wb) + (W	∕s) →Wd					
Status Affected:	DC, N, O\	/, Z, C					
Encoding:	0100	0www	wBqq	qddd	dppp	SSSS	
Description:	register W direct add addressin The 'w' bit The 'B' bit	ontents of the b, and place t ressing must t g may be used s select the ad selects byte o s select the de	he result in the used for \ d for Ws and ddress of the pr word oper	the destinatio Wb. Either reg Wd. base registe ation ('0' for v	n register Wo gister direct o er.	d. Register r indirect	
	The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.						
	Note:	rathjer than a	a word opera	instruction d ation. You ma , but it is not i	ayusea.we		
Words:	1						
Cycles:	1(1)						

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	ADD.	.в и	15, W6,	W7		W5 to W6, te mode)	store resu	lt in W7
	W5 W6 W7 SR	Before Instruction AB00 0030 FFFF 0000		W W SI	6 0030 7 FF30			
Example 2:	ADD	7	V5, W6,	W7		W5 to W6, rd mode)	store resul	lt in W7
	W5 W6 W7 SR	Before nstruction AB00 0030 FFFF 0000		W5 W6 W7 SR	6 0030 AB30	l = 1)		
ADD					ulators			
Implemented in	:	PIC24F	PIC	24H	PIC24E	dsPIC30F X	dsPIC33F X	dsPIC33E X
Syntax: Operands: Operation:		{label:} Acc ∈ [A,Ⅰ I <u>f (Acc =</u> A			Асс			
		(ACCA) - <u>Else:</u> (ACCA) -	+ (ACCI	B) →A(	ССВ			
Status Affected:	: г	OA, OB, C	-		1		Γ	T 1
Encoding:	L	1100		of Acc		0000		0000
Description:			result in			o the contents nulator. This i		
		The 'A' bit	specifie	es the o	destination a	ccumulator.		
Words:		1						
Cycles:		1						
Example 1:	ADD		A		; Add A	CCB to ACCA		
	add ACCA	Befo	ore	٦	; Add Ad	CCB to ACCA After Instructio 00 1855 7	n	
ŀ		Befo Instru 00 002 00 183	ore ction			After Instructio 00 1855 7 00 1833 4	n 858	

Example 2:	ADD	В	; Assume	CA to ACCB Super Satu 5 = 1, SATA			
	Instru CA 00 E	fore uction 111 2222 54 3210 0000	ACCA ACCB SR	After Instruction 00 E111 22 01 5765 54 48		B = 1)	
ADD		16-bit Signe	d Add to Acc	umulator			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
				Х	Х	X	
Syntax:	{label:}	ADD	Ws, [Ws], [Ws++], [Ws], [Ws], [++Ws], [Ws+Wb],	{#Slit4,}	Acc		
Operands:	$\label{eq:ws} \begin{array}{l} Ws \in [W0\ .\\ Wb \in [W0\ .\\ Slit4 \in [-8\\\ Acc \in [A,B] \end{array}$	W15] . +7]					
Operation:	Shift <sub>Slit4</sub> (Ext	end(Ws)) + (A	Асс) →Асс				
Status Affected:	OA, OB, OA	B, SA, SB, SA	AB				
Encoding:	1100	1001	Awww	wrrr	rggg	SSSS	
Description:	Add a 16-bit value specified by the source working register to the most significant word of the selected accumulator. The source operand may specify the direct contents of a working register or an effective address. The value specified is added to the most significant word of the accumulator by sign-extending and zero backfilling the source operand prior to the operation. The value added to the accumulator may also be shifted by a 4-bit signed literal before the addition is made.						
	The 'w' bits The 'r' bits e The 'g' bits	pecifies the de specify the off encode the op select the sou specify the sou	set register W tional shift. rce Address n	/b. node.			
	ä	Positive values and negative v eft. The conte	alues of oper	and Slit4 rep	resent an ari	ithmetic shift	

ADD	16-bit Sig	gned Add to Accumulator
Words:	1	
Cycles:	1 <sup>(1)</sup>	
r c	ead-modify-write operations	vices, the listed cycle count does not apply to read and on non-CPU Special Function Registers. For more n 3.2.1 "Multi-Cycle Instructions".
Example 1:	ADD W0, #2, A	; Add W0 right-shifted by 2 to ACCA
	Before	After
	Instruction	Instruction
	W0 8000	W0 8000
	ACCA 00 7000 0000	ACCA 00 5000 0000
	SR 0000	SR 0000
Example 2:	ADD [W5++], A	; Add the effective value of W5 to ACCA ; Post-increment W5
	Before	After

Before		After
Instruction		Instruction
2000	W5	2002
00 0067 2345	ACCA	00 5067 2345
5000	Data 2000	5000
0000	SR	0000
	Instruction 2000 00 0067 2345 5000	Instruction           2000         W5           00 0067 2345         ACCA           5000         Data 2000

5

	Add f to WREG with Carry							
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	ADDC{.B}	f	{,WREG}				
Operands:	f ∈ [0 81	91]						
Operation:	-	G) + (C) →de	stination de	signated by I	C			
Status Affected:	DC, N, OV,	, , ,		0 ,				
Encoding:	1011	0100	1BDf	ffff	ffff	ffff		
	Add the contents of the default working register WREG, the contents of the file register and the Carry bit and place the result in the destination register. The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register. The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'D' bit selects the destination ('0' for WREG, '1' for file register).							
	<ul> <li>The 'f' bits select the address of the file register.</li> <li>Note 1: The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . W extension to denote a word operation, but it is not required.</li> <li>2: The WREG is set to working register W0.</li> <li>3: The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These</li> </ul>							
		instructions c	an only clea	r Z.				
Words: Cycles:	1 <sub>1</sub> (1)							
		24E devices,			s not apply t			
details, se	ee Note 3 in	Section 3.2.	l "Multi-Cyc	<b>le Instructio</b>	Registers. Fo ons".	r more		
details, se <u>Example 1:</u> AD WREG RAM100 SF	Before Instruction CC60 8006 0001 ((	Section 3.2. AM100 WRE RAM1	1 "Multi-Cyc ; Add W ; (Byte After Instruction EG CC60 00 8067 SR 0000	<b>le Instructic</b> REG and C e mode)	Registers. Fo	r more		

ADDC		Add Literal to Wn with Carry							
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	ADDC{.B}	#lit10,	Wn					
Operands:	lit10 ∈ [0	lit10 ∈ $[0 \dots 255]$ for byte operation lit10 ∈ $[0 \dots 1023]$ for word operation Wn ∈ $[W0 \dots W15]$							
Operation:	lit10 + (Wn	lit10 + (Wn) + (C) →Wn							
Status Affected:	DC, N, OV,								
Encoding:	1011	0000	1Bkk	kkkk	kkkk	dddd			
Description:	register Wr register Wr		ry bit, and pla	ace the result	t back into th	ne working			
	The 'k' bits	selects byte or s specify the lite s select the add	teral operand	d.		yte).			
		The extension rather than a v denote a word	word operation	ion. You may	use a .w ext	-			
	2:	For byte opera unsigned valu <b>Operands</b> " fo Byte mode.	rations, the lit ue [0:255]. Se	teral must be ee <b>Section 4.</b>	specified as .6 "Using 10	)-bit Literal			
	3:	The Z flag is " These instruct			SUBB and ST	UBBR.			
Words:	1								
Cycles:	1								
Example 1:	ADDC.B #0	OxFF, W7	; Add -1	and C bit t	to W7 (Byte	mode)			
	Before		After						
V	Instruction N7 12C0	W	Instruction 7 12BF						
		VV C = 0) SF		N,C = 1)					
			`	·,- ,					
Example 2: A	ADDC #0	OxFF, W1	; Add 255	5 and C bit	to Wl (Wor	cd mode)			
v	Before Instruction V1 12C0	W1	After Instruction						

ADDC	Add Wb to Short Literal with Carry							
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	ADDC{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]			
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]			[			
Operation:	(Wb) + lit5	+ (C) $\rightarrow$ Wd						
Status Affected:	DC, N, OV	, Z, C						
Encoding:	0100	lwww	wBqq	qddd	d11k	kkkk		
	Wd. Regis indirect ad The 'w' bita The 'B' bit The 'q' bits The 'd' bits	nd the Carry b ter direct addr dressing may s select the ac selects byte o s select the de s select the de s provide the li	tessing must be used for V ddress of the or word opera stination Add stination regi	be used for W Wd. base register tion ('0' for w Iress mode. ster.	/b. Register o ord, '1' for by	direct or te).		
	Note 1: 2:	The extension rather than a denote a word The Z flag is ' instructions ca	word operation, b d operation, b "sticky" for AL	on. You may u out it is not re- DDC, CPB, St	use a .w exte quired.	nsion to		
Words:	1							
Cycles:	1							
Example 1:	ADDC.B	W0, #0x1F,		dd W0, 31 am core the rea				
۱ Data 12	Before           Instruction           W0         CC80           W7         12C0           C0         B000           SR         0000		a 12C0 B0	ction 80 C0				

Example 2: ADI	C	W3, #0x	6, [W4]		, 6 and C bit (Word mode) the result in [W4]
	Before			After	
	Instructio	n		Instruction	
W3	6006		W3	6006	
W4	1000		W4	0FFE	
Data 0FFE	DDEE		Data 0FFE	600D	
Data 1000	DDEE		Data 1000	DDEE	
SR	0001	(C = 1)	SR	0000	

5

Instruction Descriptions

ADDC		Add Wb to Ws with Carry								
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E				
	Х	Х	X	Х	Х	Х				
Syntax:	{label:}	ADDC{.B}	Wb,	Ws,	Wd					
				[Ws],	[Wd]					
				[Ws++],	[Wd++]					
				[Ws],	[Wd]					
				[++Ws],	[++Wd]					
				[Ws],	[Wd]					
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]								
Operation:	(Wb) + (Ws	s) + (C) →Wd								
Status Affected:	DC, N, OV	Z, C								
Encoding:	0100	lwww	wBqq	qddd	dppp	SSSS				
Description:	register Wi register Wo	ntents of the s and the Carr d. Register dir ect or indirect	ry bit, and pla ect addressir	ice the result	in the destir sed for Wb.	nation Either				
	The 'B' bit The 'q' bits The 'd' bits The 'p' bits	s select the ac selects byte o select the de select the de select the so select the so	r word opera stination Add stination regi urce Address	tion ('0' for w ress mode. ster. s mode.		yte).				
		<b>Note 1:</b> The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . w extension to denote a word operation, but it is not required.								
	2:	The Z flag is " instructions ca	•		JBB and SUE	BR. These				
Words:	1									
Cycles:	1 <sup>(1)</sup>									

Note 1:	In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and
	read-modify-write operations on non-CPU Special Function Registers. For more
	details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

; Add W0, [W1] and C bit (Byte mode)

			; Store the result in [W2] ; Post-increment W1, W2
W0 W1	Before nstructio CC20 0800	W0 W1	0801
W2 Data 0800 Data 1000 SR	1000 AB25 FFFF 0001	W2 Data 0800 Data 1000 (C = 1) SR	AB25 FF46
Example 2: ADDO		W3,[W2++],[W1++]	] ; Add W3, [W2] and C bit (Word mode) ; Store the result in [W1] ; Post-increment W1, W2
1	Before nstructio	n l	After
W1	1000	W1	1002
W2	2000	W2	2002
W3 Data 1000	0180 8000	W3 Data 1000	0180 2681
Data 2000	2500	Data 2000	2500

SR

0000

W0,[W1++],[W2++]

0001 (C = 1)

SR

5

Example 1:

ADDC.B

AND		AND f and V	WREG						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	AND{.B}	f	{,WREG}					
Operands:	f∈ [0 81	91]							
Operation:	(f).AND.(W	REG) →desti	nation desigi	nated by D					
Status Affected:	N, Z		-						
Encoding:	1011	0110	OBDf	ffff	fff	ffff			
Description:	register WF the destina destination	ne logical ANI REG and the of tion register. register. If W not specified	contents of th The optional REG is spec	e file register WREG opera ified, the resu	r, and place t and determin ult is stored ir	he result ir es the n WREG.			
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'D' bit selects the destination ('0' for WREG, '1' for file register). The 'f' bits select the address of the file register.								
		The extension rather than a denote a wore The WREG is	word operati d operation, l	on. You may out it is not re	use a .w ext equired.	-			
Words:	1			.99	•				
Cycles:	1 <sup>(1)</sup>								
read-mo details, s	dify-write ope	24E devices, erations on no Section 3.2.	on-CPU Spec	ial Function R	Registers. For				
Example 1: AI	ND.B RAM10	0	; AND W	REG to RAM1	.00 (Byte m	ode)			
WRE RAM10 S		WRI RAM1	00 FF80	(N = 1)					
Example 2: A	ND RAM200,		; AND R						

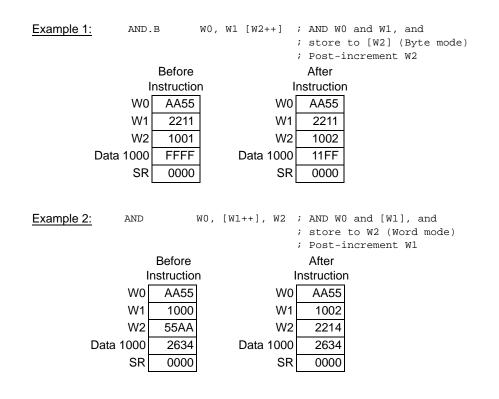
I	Before nstructior	n Ir	After Instruction
WREG	CC80	WREG	0080
RAM200	12C0	RAM200	12C0
SR	0000	SR	0000

AND		AND Litera	l and Wn						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	AND{.B}	#lit10,	Wn					
Operands:		. 255] for byte . 1023] for wo W15]							
Operation:	lit10.AND.(	Wn) →Wn							
Status Affected:	N, Z								
Encoding:	1011	0010	0Bkk	kkkk	kkkk	dddd			
Description:	contents of	ne logical ANI the working r gister Wn. Re	egister Wn a	nd place the	result back i	nto the			
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'k' bits specify the literal operand. The 'd' bits select the address of the working register.								
	<b>Note 1:</b> The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . W extension to denote a word operation, but it is not required.								
	2:	For byte oper unsigned valu Operands" fo Byte mode.	ations, the lite ie [0:255]. Se	eral must be e <b>Section 4.</b>	specified as 6 "Using 10	-bit Litera			
Words:	1								
Cycles:	1								
Example 1:	AND.B #0x83	8, W7	; AND 02	x83 to W7 (	Byte mode)				
	Before Instruction W7 12C0 SR 0000		After Instruction V7 1280 SR 0008	(N = 1)					
Example 2:	AND #0x333,	Wl	; AND 02	x333 to W1	(Word mode	)			
	Before Instruction W1 12D0 SR 0000	-	After Instruction V1 0210 SR 0000						

AND		AND Wb an	d Short Lite	ral		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	AND{.B}	Wb,	#lit5,	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb ∈ [W0					
	lit5 ∈ [0 Wd ∈ [W0	-				
Operation:	(Wb).AND.	-				
Status Affected:	N, Z					
Encoding:	0110	0www	wBqq	qddd	d11k	kkkk
Description:	Compute tl	ne logical ANI	D operation c	of the content	s of the base	register
		e 5-bit literal a				
		rect addressii dressing may			ther registe	r direct or
		select the ac	dress of the	base register	:	
			or word opera		ord, '1' for by	yte).
	The 'q' bits	selects byte c select the de select the de	stination Add	lress mode.	ord, '1' for by	yte).
	The 'q' bits The 'd' bits	select the de	stination Add stination regi	lress mode. ster.		
	The 'q' bits The 'd' bits	select the de select the de provide the li The extension rather than a	stination Add stination regi teral operand on .B in the word opera	Iress mode. ster. I, a five-bit in instruction de tion. You may	teger numbe enotes a byt y use a . w e	r. e operatior
Words:	The 'q' bits The 'd' bits The 'k' bits	select the de select the de provide the li The extensio	stination Add stination regi teral operand on .B in the word opera	Iress mode. ster. I, a five-bit in instruction de tion. You may	teger numbe enotes a byt y use a . w e	r. e operatior
Words: Cycles:	The 'q' bits The 'd' bits The 'k' bits <b>Note:</b>	select the de select the de provide the li The extension rather than a	stination Add stination regi teral operand on .B in the word opera	Iress mode. ster. I, a five-bit in instruction de tion. You may	teger numbe enotes a byt y use a . w e	r. e operatior
Cycles:	The 'q' bits The 'd' bits The 'k' bits <b>Note:</b> 1	select the de select the de provide the li The extension rather than a	stination Add stination regi teral operand on . B in the word opera d operation,	Iress mode. ster. I, a five-bit ini instruction de tion. You may but it is not re	teger numbe enotes a byt y use a .w e equired.	r. e operatior
Cycles:	The 'q' bits The 'd' bits The 'k' bits <b>Note:</b> 1	select the de select the de provide the li The extension rather than a denote a wor	stination Add stination regi teral operand on . B in the word operation, d operation, ] ; AND W ; Store	Iress mode. ster. I, a five-bit ini instruction de tion. You may but it is not re	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatior
Cycles:	The 'q' bits The 'd' bits The 'k' bits <b>Note:</b> 1 1 AND.B W0 Before	select the de select the de provide the li The extension rather than a denote a work ,#0x3, [W1++	stination Add stination regi teral operand on . B in the word operation, ] ; AND W ; Store ; Post- After	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatio
Cycles: <u>Example 1:</u>	The 'q' bits The 'd' bits The 'k' bits <b>Note:</b> 1 1 AND.B W0 Before Instruction	select the de select the de provide the li The extensio rather than a denote a wor	stination Add stination regi teral operand on . B in the word operation, ] ; AND W ; Store ; Post- After Instructio	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatio
Cycles: Example 1:	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 AND.B W0 Before Instruction W0 23A5	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++	stination Add stination regi teral operand on . B in the word operation, ) ; AND W ; Store ; Post- After Instructio W0 23A5	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatio
Cycles: Example 1:	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND.B W0 Before Instruction W0 23A5 W1 2211	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++	stination Add stination regi teral operand on . B in the word operation, ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatio
Cycles: Example 1:	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND.B W0 Before Instruction W0 23A5 W1 2211	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++	stination Add stination regi teral operand on . B in the word operation, ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operatio
Cycles: Example 1:	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND . B W0 Before Instruction W0 23A5 W1 2211 210 9999	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++	stination Add stination regi teral operand on . B in the word operation, ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212 210 0199 SR 0000 ; AND W	Iress mode. ster. I, a five-bit initian instruction detion. You may but it is not re 0 and 0x3 ( to [W1] increment W	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operation extension to
Cycles: <u>Example 1:</u> Data 2	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND.B W0 Before Instruction W0 23A5 W1 2211 210 9999 SR 0000 AND Before	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++ Data 2 W0, #0x1F, W1	stination Add stination regi teral operand on . B in the word operation, ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212 210 0199 SR 0000 ; AND W ; Store After	Iress mode. ster. I, a five-bit ind instruction de- tion. You may but it is not re 0 and 0x3 ( to [W1] increment W n	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operation extension to
Cycles: <u>Example 1:</u> Data 2	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND.B W0 Before Instruction W0 23A5 W1 2211 210 9999 SR 0000 AND Before Instruction	select the de select the de provide the li The extension rather than a denote a word ,#0x3, [W1++ Data 2: W0, #0x1F, W1	stination Add stination regi teral operand on . B in the word operation, } ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212 210 0199 SR 0000 ; AND W ; Store After Instructior	Iress mode. ster. I, a five-bit ind instruction de- tion. You may but it is not re 0 and 0x3 ( to [W1] increment W n	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operation extension to
Cycles: <u>Example 1:</u> Data 2 <u>Example 2:</u>	The 'q' bits The 'd' bits The 'k' bits Note: 1 1 1 AND.B W0 Before Instruction W0 23A5 W1 2211 210 9999 SR 0000 AND Before	select the de select the de provide the li The extensio rather than a denote a wor ,#0x3, [W1++ Data 2 W0, #0x1F, W1	stination Add stination regi teral operand on . B in the word operation, ; AND W ; Store ; Post- After Instructio W0 23A5 W1 2212 210 0199 SR 0000 ; AND W ; Store After	Iress mode. ster. I, a five-bit ind instruction de- tion. You may but it is not re 0 and 0x3 ( to [W1] increment W n	teger numbe enotes a byt y use a .w e equired. Byte mode)	r. e operation extension to

AND			And Wb and	d Ws			
Implement	ted in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	AND{.B}	Wb,	Ws,	Wd	
					[Ws],	[Wd]	
					[Ws++],	[Wd++]	
					[Ws],	[Wd]	
					[++Ws],	[++Wd]	
					[Ws],	[Wd]	
Operands:	:	$ \begin{array}{l} Wb \in \; [W0 \\ Ws \in \; [W0 \\ Wd \in \; [W0 \end{array} \end{array} $	W15]				
Operation:	:	(Wb).AND.	(Ws) →Wd				
Status Affe	ected:	N, Z					
Encoding:		0110	0www	wBqq	qddd	dppp	SSSS
Description	n:	Ws and the destination	ne logical ANI e contents of t register Wd. ster direct or i	he base regis Register dire	ster Wb, and ct addressing	place the res g must be use	ult in the ed for Wb.
		The 'B' bits The 'q' bits The 'd' bits The 'p' bits	ldress of the l r word operat stination Add stination regis urce Address urce register.	tion ('0' for w ress mode. ster.		rte).	
		Note:	The extensio rather than a denote a wor	word operat	ion. You may	yuse a .w e	
		1					
Words:							

### 16-bit MCU and DSC Programmer's Reference Manual



ASR	DICO4E	Arithmetic S	PIC24E			
Implemented in:	PIC24F	PIC24H		dsPIC30F	dsPIC33F	dsPIC33E
	X	X	Х	X	Х	Х
Syntax:	{label:}	ASR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f<7>) → (f<6:1>) (f<0>) → <u>For word o</u> (f<15>) → (f<15>) →	Dest<7> Dest<6> →Dest<5:0> C <u>peration:</u> →Dest<15> →Dest<14> ) →Dest<13:0	>			
Status Affected:	N, Z, C					
Encoding:	1101	0101	1BDf	ffff	ffff	ffff
Description:	in the desti shifted into performed, determines stored in W register. The 'B' bit The 'D' bit	ntents of the fination register the Carry bit of the result is s the destination (REG. If WRE) selects byte of selects the de select the add	r. The Least of the STATL ign-extended on register. If G is not spec r word opera stination ('0'	Significant bi JS Register. / d. The option WREG is sp cified, the res tion ('0' for w for WREG, '2	t of the file re After the shif al WREG op ecified, the r ult is stored ord, '1' for by	egister is t is erand esult is in the file yte).
	Note 1:	The extension rather than a v denote a word The WREG is	B in the in word operation, b l operation, b	struction den on. You may out it is not re	use a . w ext quired.	
Words:	1					

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

ASR.B RAM400, WREG

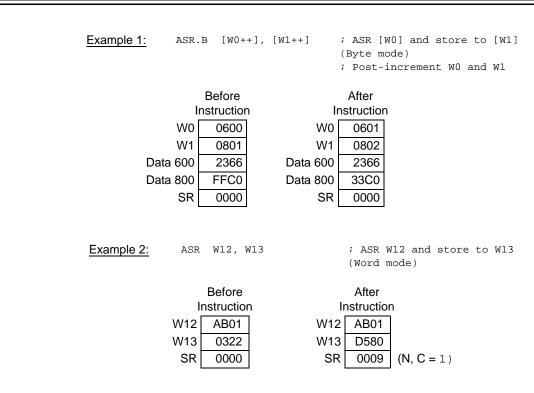
Example 1:

<b>_</b>		; (1	Byte mode)
	Before struction 0600 0823 0000	Instru WREG 0 RAM400 0	iter Jaction 1611 1823 1001 (C = 1)
Example 2: ASR	RAM200	; 2	ASR RAM200 (Word mode)
	Before struction		fter
RAM200	8009	RAM200 C	004
SR	0000	SR 0	0009 (N, C = 1)

; ASR RAM400 and store to WREG

ASR		Arithmetic S	210045			1			
Implemented in:	PIC24F X	PIC24H X	PIC24E X	dsPIC30F X	dsPIC33F X	dsPIC33E X			
	-				-				
Syntax:	{label:}	ASR{.B}	Ws,	Wd					
			[Ws],	[Wd]					
			[Ws++],	[Wd++]					
			[Ws],	[Wd]					
			[++Ws],	[++Wd]					
			[Ws],	[Wd]					
Operands:	Ws ∈ [W0 Wd ∈ [W0								
Operation:	(Ws<7>) (Ws<6:1: (Ws<0>) For word op (Ws<15> (Ws<15>	)							
	(Ws<0>)	,	0.						
	- <u> </u>								
Status Affected:	N, Z, C		<del></del>	<del></del>	<del></del>	- <b>I</b>			
Encoding:	1101	0001	1Bqq	qddd	dppp	SSSS			
Description:	result in the shifted into the result is be used for The 'B' bit s The 'q' bits The 'd' bits	e destination ro the Carry bit of s sign-extende or Ws and Wd. selects byte of s select the des s select the des	register Wd. <sup>-</sup> of the STATU ed. Either reg or word opera estination Add estination regi	jister.	nificant bit of ter the shift is indirect addre	Ws is performed essing may			
	The 's' bits	The 'p' bits select the source Address mode. The 's' bits select the source register.							
		rather than a	a word opera	e instruction de ation. You may but it is not ree	yusea.we				
Words:	1								
Cycles:	1 <sup>(1)</sup>								
read-m	modify-write op	perations on n	non-CPU Spe	cycle count doe: ecial Function F y <b>cle Instructio</b>	Registers. For				

### 16-bit MCU and DSC Programmer's Reference Manual



Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	ASR	Wb,	#lit4,	Wnd	
Operands:	Wb ∈ [W0 lit4 ∈ [01 Wnd ∈ [W0	5]				
Operation:	lit4<3:0> <i>→</i> Wb<15> <i>→</i>	-				
Status Affected:	N, Z					
Encoding:	1101	1110	lwww	wddd	d100	kkkk
Description:	unsigned ling the shift is	shift right the iteral, and stor performed, the r Wb and Wno	re the result ir ne result is sign	n the destina	tion register	Wnd. After
	The 'd' bits The 'k' bits	s select the ad s select the de s provide the lit	estination regis	ster.		
	Note:	This instruction	on operates ir	ו Word mode	e only.	
Words:	1					
Cycles:	1					
Example 1:	ASR W0, #0x4	4, Wl		by 4 and s	store to Wl	
	Before Instruction W0 060F W1 1234 SR 0000	V	After Instruction W0 060F W1 0060 SR 0000			
Example 2:	ASR W0, #0x6	5, Wl	; ASR WO	by 6 and s	store to Wl	
١	Before           Instruction           W0         80FF           W1         0060           SR         0000	V	After Instruction V0 80FF V1 FE03 SR 0008 (1	N = 1)		
Example 3:	ASR W0, #0x1	F, Wl	; ASR WO	by 15 and	store to W	11
	Before Instruction W0 70FF W1 CC26 SR 0000	V	After Instruction W0 70FF W1 0000 SR 0002	(Z = 1)		

ASR		Arithmetic	Shift Right b	oy Wns		
Implemented in	: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	ASR	Wb,	Wns,	Wnd	
Operands:	Wb ∈ [W0 Wns ∈ [W0 Wnd ∈ [W0	W15]				
Operation:	Wns<3:0> Wb<15> $\rightarrow$	→Shift_Val Wnd<15:15-	Shift_Val + 1: d<15-Shift_V			
Status Affected	: N, Z					
Encoding:	1101	1110	lwww	wddd	d000	SSSS
	sign-extend The 'w' bits The 'd' bits The 's' bits <b>Note 1:</b> 2:	led. Direct ac select the ac select the de select the so This instruction of Wns is great	ddressing mu ddress of the estination reg urce register on operates i ater than 15,	n Word mode Wnd = 0x0 if	r Wb, Wns ar r. e only.	nd Wnd.
		Wnd = 0xFFF	F if Wb is ne	egative.		
Words:	1					
Cycles:	1					
Example 1:	ASR W0, W5,	W6	; ASR W	0 by W5 and	store to W	16
	Before		After			
	Instruction		Instructio	n 1		
	W0 80FF		W0 80FF	-		
	W5 0004 W6 2633		W5 0004 W6 F80F	-		
	SR 0000		SR 0000			
Example 2:	ASR W0, W5,			10 by W5 and	l store to 1	W6
	Before		After			
	Instruction		Instructio	'n		
	W0 6688		W0 6688	_		
	W5 000A		W5 000A			
	W6 FF00 SR 0000		W6 0019 SR 0000	_		
Example 3:	ASR W11, W1	2, W13		] 11 by W12 a	and store to	o W13
<u> </u>	Poforo		Aftor			
	Before Instruction		After Instructio	n		
	W11 8765	V	V11 8765	]		
	W12 88E4	V	V12 88E4	1		
	W13 A5A5	V	V13 F876			
	SR 0000		SR 0008	(N = 1)		

BCLR		Bit Clear f						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	BCLR{.B}	f,	#bit4				
Operands:	f ∈ [0 81 bit4 ∈ [0	$f \in [0 \dots 8191]$ for byte operation $f \in [0 \dots 8190]$ (even only) for word operation $bit4 \in [0 \dots 7]$ for byte operation $bit4 \in [0 \dots 15]$ for byte operation						
Operation:	0 →f <bit4></bit4>							
Status Affected:	None							
Encoding:	1010	1001	bbbf	ffff	ffff	fffb		
Description:	with the Lea	it in the file reg ast Significant byte operatio	bit (bit 0) and	d advances t	o the Most S			
		select value b select the add			e cleared.			
	Note 1: The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . W extension to denote a word operation, but it is not required.							
		When this inst address must			mode, the file	e register		
		When this inst petween 0 and		ites in Byte n	node, 'bit4' m	nust be		
Words:	1							
Cycles:	1 <sup>(1)</sup>							

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 "Multi-Cycle Instructions**".

Example 1:	BCLR.B 0x800, ‡	#0x7 ;	Clear bit 7 in 0x800
Data	Before Instruction 0800 66EF SR 0000	lr Data 0800 SR [	After nstruction 666F 0000
Example 2:	BCLR 0x400, ‡	#0x9	; Clear bit 9 in 0x400
Data	Before Instruction 0400 AA55 SR 0000	Ir Data 0400 [ SR [	After nstruction A855 0000

BCLR		Bit Clear in	Ws				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BCLR{.B}	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	#bit4			
Operands:		W15] 7] for byte op 15] for word o					
Operation:	0 <b>→Ws<bi< b=""></bi<></b>	4>					
Status Affected:	None						
Encoding:	1010	0001	bbbb	0800	0ppp	SSSS	
Description:	the Least S 7 for byte o	it in register W ignificant bit (I perations, bit Iressing may	bit 0) and adv 15 for word o	vances to the operations).	Most Signifi	cant bit (bit	
	The 'b' bits select value bit4 of the bit position to be cleared. The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 's' bits select the source/destination register. The 'p' bits select the source Address mode.						
	<b>Note 1:</b> The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a .w extension to denote a word operation, but it is not required.						
	<ol> <li>When this instruction operates in Word mode, the source register address must be word-aligned.</li> </ol>						
	<ol> <li>When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.</li> </ol>						
	I	n dsPIC33E a DSRPAG regi Data Space.					
Words:	1						
Cycles:	1 <sup>(1)</sup>						

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more

Example 1:	BCLR.B W2, #0x2	; Clear bit 3 in W2
Eugenela ()	Before Instruction W2 F234 SR 0000	After Instruction W2 F230 SR 0000
Example 2:	BCLR [W0++], #0x0	; Clear bit 0 in [W0] ; Post-increment W0
	Before Instruction	After Instruction
	W0 2300	W0 2302
Data	a 2300 5607 D	Data 2300 5606
	SR 0000	SR 0000

5

Instruction Descriptions

BRA		Branch Unc	onditionally			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BRA	Expr			
Operands:		e a label, abso lved by the lin				+32767].
Operation:	· · ·	2 * Slit16 → PC				
Status Affected:	None					
Encoding:	0011	0111	nnnn	nnnn	nnnn	nnnn
	branches up resolved by expression. Slit16, since	h is the two's to 32K instru- the linker fror After the brane the PC will h are a signed line PC + 2).	ictions forward in the supplied ich is taken, th ave incremen	d or backward I label, absolu ne new addre ted to fetch th	d. The Slit16 ute address o ss will be (P0 ne next instru	value is or (2 + 2) + 2 * oction.
Words:	1					
Cycles:	2 (PIC24F, F 4 (PIC24E, 1	PIC24H, dsPl( dsPIC33E)	C30F, dsPIC3	3F)		
Example 1:	002000 HERE 002002 002004 002006 002008 00200A THERI 00200C	· · · · · · · · · · ·	RE	; B.	ranch to TH	ERE
	Before Instructi PC 00 20 SR 00	on	PC SR	After Instruction 00 200A 0000		
Example 2:	002000 HERE: 002002 002004 002006 002008 00200A THERE 00200C	BRA THEN	RE+0x2	; Br	anch to THE	RE+0x2
	Before Instructio PC 00 20 SR 00	00	PC SR	After Instruction 00 200C 0000		

Example 3:	002000
<u> </u>	002002
	002004

PC

SR

0 HERE: BRA 0x1366 2 . . . 4 . . .

Before Instruction

00 2000

0000

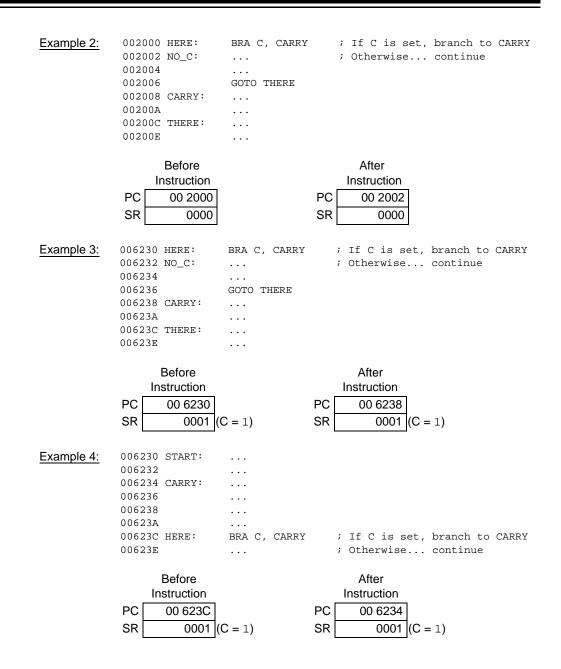
; Branch to 0x1366

	After			
	Instruction			
PC	00 1366			
SR	0000			

BRA			Computed E	Branch			
Implemented in:	:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х	Х		Х	Х	
Syntax:	{	abel:}	BRA	Wn			
Operands:	V	√n∈ [W0	W15]				
Operation:	· ·	,	(2 * Wn) →PC uction Registe				
Status Affected:	Ν	lone					
Encoding:		0000	0001	0110	0000	0000	SSSS
	e ir	xecutes, t ncremente	p to 32K instru he new PC wi d to fetch the	II be (PC + 2) next instructi	+ 2 * Wn, si		
	Т	he 's' bits	select the sou	urce register.			
Words:	1						
Cycles:	2						
Example 1:	00200	8 A TABLE7	BRA W7	;	Branch for	ward (2+2*	W7)
	PC W7 SR	Before Instruction 00 200 008 000	0 4	 PC W7   SR	After nstruction 00 210A 0084 0000		

BRA			Computed E	Branch			
Implemented in	: PIC	C24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х			Х
Syntax:	{labe	el:}	BRA	Wn			
Operands:	Wn e	≣ [W0 .	W15]				
Operation:			2 * Wn) →PC uction Registe				
Status Affected:	None	e	-				
Encoding:	0	000	0001	0000	0110	0000	SSSS
	bran exec	ches up utes, th	th is the sign- to 32K instru- ne new PC wi d to fetch the	uctions forwa II be (PC + 2)	rd or backwa ) + 2 * Wn, si	rd. After this	instruction
	The	's' bits	select the sou	rce register.			
Words:	1						
Cycles:	4						
Example 1:	002000 H 002002  002108 00210A T 00210C		BRA W7	;	Branch for	ward (2+2*	W7)
	Inst	efore ruction 00 2000 0084 0000	D 4	 PC W7   SR	After nstruction 00 210A 0084 0000		

BRA C	<b>BIOME</b>	Branch if Ca	-			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	Х	Х	X	X	X
Syntax:	{label:}	BRA	С,	Expr		
Operands:		e a label, abso lved by the lin				+32767].
Operation:		-				
Status Affected:	None					
Encoding:	0011	0001	nnnn	nnnn	nnnn	nnnn
	expression. If the branch PC will have	blved by the lin is taken, the incremented two-cycle instr	new address to fetch the n	will be (PC +	2) + 2 * Slit16 1. The instruc	6, since the ction then
	PC will have		to fetch the n	ext instruction	n. The instruc	ction then
	The 'n' bits a instruction w	are a 16-bit siç vords.	gned literal that	at specify the	offset from (	PC + 2) in
Words:	1					
Cycles:	1 (2 if branc	h taken) – PIC	24F, PIC24H	, dsPIC30F, c	IsPIC33F	
	1 (4 if branc	h taken) – PIC	24E, dsPIC3	3E		
C C C C C C C C C C C C C C C C C C C	002000 HERE: 002002 NO_C: 002004 002006 002008 CARRY: 00200A 00200C THERE: 00200E	BRA C, (  GOTO THI  	;	If C is set Otherwise.		
	Before Instruction PC 00 200 SR 000		PC SR	After nstruction 00 2008 0001 (0	C = 1)	



BRA G	E	Branch if Si	igned Greate	er Than or E	qual		
Implemented in	n: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BRA	GE,	Expr			
Operands:	Expr is res	be a label, abs olved by the l 2768 +327	inker to a Slit		ion.		
Operation:	lf (Conditio (PC + 2)	Condition = (N&&OV)  (!N&&!OV) f (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affected	d: None						
Encoding:	0011	1101	nnnn	nnnn	nnnn	nnnn	
complement number '2 * Slit16', which supports branches up to 32 instructions forward or backward. The Slit16 value is resolved by th linker from the supplied label, absolute address or expression. If the branch is taken, the new address will be (PC + 2) + 2 * Slit16 the PC will have incremented to fetch the next instruction. The instruction becomes a two-cycle instruction, with a NOP executed in the s cycle.					lit16, since instruction		
	The 'n' bits in instructio	are a 16-bit s on words.	signed literal	that specify t	he offset fror	n (PC + 2)	
	Note:	The assembl be used.	er will conve	rt the specifie	ed label into t	he offset to	
Words:	1						
Cycles:	1 (2 if bran	ch taken) – P	IC24F, PIC24	4H, dsPIC30F	F, dsPIC33F		
	1 (4 if bran	ch taken) – P	IC24E, dsPI0	C33E			
Example 1:	007600 LOOP: 007602 007604 007606 007608 HERE: 00760A NO_GE:	  BRA GE, L	00P		branch to		
	Before Instruction PC 00 7608 SR 0000	_	PC SR	After Instruction 00 7600 0000			

Example 2:	007600 LOOP: 007602 007604 007606	· · · · · · · · · · ·	
	007608 HERE: 00760A NO_GE:	BRA GE, LOOP	; If GE, branch to LOOP ; Otherwise continue
	Before Instruction		After Instruction
	PC 00 7608		PC 00 760A
	SR 0008	(N = 1)	SR 0008 (N = 1)

BRA G	EU		Branch if U	nsigned Gr	eater Than	or Equal			
Implemented in	:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
		Х	Х	X	X	Х	Х		
Syntax:		{label:}	BRA	GEU,	Expr				
Operands:		Expr is res	be a label, at solved by the 32768 +32	linker to a S	lit16 offset th		an offset		
Operation:		If (Condition (PC + 2	Condition = C If (Condition) (PC + 2) + 2 * Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register						
Status Affected	:	None							
Encoding:		0011	0001	nnnn	nnnn	nnnn	nnnn		
		Slit16', which supports branches up to 32K instructions forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression. If the branch is taken, the new address will be $(PC + 2) + 2 *$ Slit16, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second cycle.							
		The 'n' bits are a 16-bit signed literal that specify the offset from (PC + 2) in instruction words.							
		Note:		iction and ha	tical to the I as the same t16.				
Words:		1							
Cycles:		1 (2 if brar	nch taken) – F	PIC24F, PIC	24H, dsPIC3	0F, dsPIC33	F		
		1 (4 if brar	nch taken) – F	PIC24E, dsP	PIC33E				
Example 1:	002002 002004 002008 002008 002008	5 3 A C BYPASS:	BRA GEU,   GOTO THEF 		; to :	C is set, 1 BYPASS erwise (			
	I PC SR	Before nstruction 00 2000 0001	(C = 1)	Ir PC SR	After Instruction 00 200C 0001 (C	C = 1)			

BRA G	Т	Branch if S	igned Great	er Than			
Implemented in	: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BRA	GT,	Expr			
Operands:	Expr is re	be a label, ab solved by the -32768 +327	linker to a Sli	•	ion.		
Operation:	If (Condit (PC +	Condition = (!Z&&N&&OV)  (!Z&&!N&&!OV) If (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affected	: None						
Encoding:	0011	1100	nnnn	nnnn	nnnn	nnnn	
	instructio linker fror If the bra the PC w	<ul> <li>program will branch relative to the next PC. The offset of the branch is th two's complement number '2 * Slit16', which supports branches up to 32 instructions forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.</li> <li>If the branch is taken, the new address will be (PC + 2) + 2 * Slit16, sinc the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second.</li> </ul>					
	cycle. The 'n' bi	ts are a 16-bit tion words.					
Words:	1						
Cycles:		ınch taken) – F ınch taken) – F			F, dsPIC33F		
Example 1:	002000 HERE: 002002 NO_GT 002004 002006 002008 002008 00200A 00200C BYPAS 00200E	  GOTO TH			ST, branch erwise c		
	Before Instructio PC 00 20 SR 00	on	Ir PC SR	After Instruction 00 200C 0001 (C	= 1)		

BRA G	TU		Branch if U	nsigned Gre	ater Than			
Implemented in	n:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
		Х	Х	Х	Х	Х	Х	
Syntax:	{	label:}	BRA	GTU,	Expr			
Operands:	E	Expr is reso	e a label, abs blved by the li 2768 +327	inker to a Slit		ion.		
Operation:								
Status Affected	۱ :t	None						
Encoding:	Γ	0011	1110	nnnn	nnnn	nnnn	nnnn	
	f s li t	number '2 * Slit16', which supports branches up to 32K instructions forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression. If the branch is taken, the new address will be (PC + 2) + 2 * Slit16, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second						
	с Г	cycle. The 'n' bits	are a signed					
Words:	C 1	offset from	(PC + 2).					
Cycles:	-		ch taken) – P		1H dePIC30	- dePIC33F		
Cycles.			ch taken) – P			, นอก 10000		
<u>Example 1:</u>	002002 002004 002006 002008 00200A	BYPASS:	BRA GTU,   GOTO THEN 			J, branch t vise con		
	Ir PC SR	Before Instruction 00 2000 0001	(C = 1)	Inst	After cruction 00 200C 0001 (C =	-1)		

#### :4 1 1

BRA L	E	Branch if S	igned Less	Than or Equa	al			
Implemented	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	BRA	LE,	Expr				
Operands:	Expr is res	be a label, abs solved by the l 32768 +327	inker to a Slit		ion.			
Operation:	If (Conditio (PC + 2	ondition = Z  (N&&!OV)  (!N&&OV) (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register						
Status Affecte	d: None							
Encoding:	0011	0100	nnnn	nnnn	nnnn	nnnn		
instructions forward or backward. The Slit16 value is resolved by the li from the supplied label, absolute address or expression. If the branch is taken, the new address will be (PC + 2) + 2 * Slit16, s the PC will have incremented to fetch the next instruction. The instruc- then becomes a two-cycle instruction, with a NOP executed in the sec								
	cycle. The 'n' bits offset from	s are a signed (PC + 2).	literal that sp	pecifies the n	umber of inst	ructions		
Words:	1	(						
Cycles:	1 (2 if brar	nch taken) – P	IC24F, PIC24	1H, dsPIC30F	, dsPIC33F			
	1 (4 if brar	nch taken) – P	IC24E, dsPI0	C33E				
Example 1:	002000 HERE: 002002 NO_LE: 002004 002006 002008 00200A 00200C BYPASS 00200E	BRA LE,  GOTO THE 			LE, branch herwise			
	Before Instruction PC 00 200 SR 000		PC SR	After Instruction 00 2002 0001 (1	C = 1)			

# 5

BRA LEU		Branch if U	nsigned Les	s Than or E	qual		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BRA	LEU,	Expr			
Operands:	Expr is reso	e a label, abs blved by the li 2768 +327	inker to a Slit		ion.		
Operation:	If (Condition (PC + 2)	Condition = !C  Z f (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affected:	None						
Encoding:	0011	0110	nnnn	nnnn	nnnn	nnnn	
supplied label, absolute address or expression. If the branch is taken, the new address will be (PC + 2) + 2 * Slit the PC will have incremented to fetch the next instruction. The in then becomes a two-cycle instruction, with a NOP executed in th cycle.						instruction	
	cycle. The 'n' bits	are a signed	literal that sp	pecifies the n	umber of inst	tructions	
	offset from	(PC + 2).					
Words:	1						
Cycles:		ch taken) – P			F, dsPIC33F		
	1 (4 if brand	ch taken) – P	IC24E, dsPI0	C33E			
0020 0020 0020 0020 0020 0020	006 008 00A 00C BYPASS:	BRA LEU,   GOTO THE 			EU, branch rwise cc		
	Before			After			
50	Instruction			struction			
PC SR	00 2000		PC SR	00 200C 0001 (C	= 1)		

## BRALEU Branch if Unsigned Less Than or Equal

BRA L	Г		Branch if Si	gned Less	Than		
Implemented i	n:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	BRA	LT,	Expr		
Operands:		Expr is res	be a label, abs olved by the l 2768 +327	inker to a Sli		ion.	
Operation:		If (Conditio (PC + 2)	Condition = (N&&!OV)  (!N&&OV) If (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register				
Status Affected	d:	None					
Encoding:		0011	0101	nnnn	nnnn	nnnn	nnnn
	complement number '2 * Slit16', which supports branches up to 32K instructions forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression. If the branch is taken, the new address will be $(PC + 2) + 2$ * Slit16, si the PC will have incremented to fetch the next instruction. The instruct then becomes a two-cycle instruction, with a NOP executed in the second cycle.					lit16, since instruction he second	
		The 'n' bits offset from	are a signed (PC + 2).	literal that sp	pecifies the n	umber of inst	ructions
Words:		1					
Cycles:			ch taken) – P ch taken) – P			<sup>F</sup> , dsPIC33F	
Example 1:	0020 0020 0020 0020 0020	006 008 00A 00C BYPASS	BRA LT,    GOTO THU 			I, branch t rwise cc	
	PC SR	Before Instruction 00 200 000	0	PC SR	After struction 00 2002 0001 (C	= 1)	

BRA LT	U	Branch if U	nsigned Les	s Than			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BRA	LTU,	Expr			
Operands:	Expr is res	be a label, absolved by the l 32768 +327	inker to a Sli		ion.		
Operation:	If (Condition (PC + 2	Condition = !C f (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affected:	None						
Encoding:	0011	1001	nnnn	nnnn	nnnn	nnnn	
Description.	escription: If the Carry flag is '0', then the program will branch relative to the next PC. The offset of the branch is the two's complement number '2 * Slit16', which supports branches up to 32K instructions forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.						
	the PC wil	If the branch is taken, the new address will be $(PC + 2) + 2 * Slit16$ , since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second cycle.					
	The 'n' bits offset from	s are a signed n (PC + 2).	literal that sp	pecifies the n	umber of ins	ructions	
	Note :	This instruction Carry) instruction assemble as	tion and has	the same en			
Words:	1						
Cycles:	1 (2 if brar	nch taken) – P	IC24F, PIC24	4H, dsPIC30I	<del>-</del> , dsPIC33F		
	1 (4 if brar	nch taken) – P	IC24E, dsPI	C33E			
	002000 HERE: 002002 NO_LTU 002004 002006 002008 00200A 00200A 00200C BYPASS 00200E	  GOTO THE			U, branch wise co		
	Before Instruction PC 00 2000 SR 000			After struction 00 2002 0001 (C	= 1)		

BRA N		Branch if N	egative			
Implemented in	n: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BRA	N,	Expr		
Operands:	Expr is res	be a label, abs solved by the l 32768 +327	inker to a Slit		ion.	
Operation:	If (Condition (PC + 2	Condition = N If (Condition) (PC + 2) + 2 * Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register.				
Status Affected	d: None					
Encoding:	0011	0011	nnnn	nnnn	nnnn	nnnn
which supports branches up to 32K instructions forward or backwa Slit16 value is resolved by the linker from the supplied label, absol address or expression. If the branch is taken, the new address will be (PC + 2) + 2 * Slit16 the PC will have incremented to fetch the next instruction. The inst then becomes a two-cycle instruction, with a NOP executed in the s cycle.					osolute lit16, since instruction	
	The 'n' bits offset from	s are a signed ı (PC + 2).	literal that sp	pecifies the n	umber of inst	ructions
Words:	1	, , , , , , , , , , , , , , , , , , ,				
Cycles:	1 (2 if brar	nch taken) – P	IC24F, PIC24	1H, dsPIC30F	F, dsPIC33F	
	1 (4 if brar	nch taken) – P	IC24E, dsPI0	C33E		
Example 1:	002000 HERE: 002002 NO_N: 002004 002006 002008 00200A 00200C BYPASS 00200E	BRA N, I   GOTO THI 			N, branch t erwise c	
	Before Instruction PC 00 200 SR 000		Ir PC SR	After Instruction 00 200C 0008 (N	l = 1)	

BRA N	C		Branch if N	ot Carry				
Implemented	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
		Х	Х	Х	Х	Х	Х	
Syntax:		{label:}	BRA	NC,	Expr			
Operands:		Expr is res	be a label, ab solved by the l 32768 +327	inker to a Sli		ion.		
Operation:		lf (Conditio (PC + 2	Condition = !C f (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affecte	d:	None						
Encoding:	1	0011	1001	nnnn	nnnn	nnnn	nnnn	
		expression If the bran the PC will then beco	solved by the n. ch is taken, th I have increm mes a two-cyc	e new addre ented to fetch	ss will be (PC n the next ins	C + 2) + 2 * S truction. The	lit16, since instruction	
			s are a signed n (PC + 2).	literal that sp	pecifies the n	umber of inst	ructions	
Words:		1	· · · ·					
Cycles:		1 (2 if brar	nch taken) – P	IC24F, PIC2	4H, dsPIC30F	F, dsPIC33F		
		1 (4 if brar	nch taken) – P	IC24E, dsPl	C33E			
Example 1:	00200 00200 00200 00200 00200	)6 )8 )A )C BYPASS	BRA NC,   GOTO THE 			, branch to wise com		
	PC SR	Before Instruction 00 200 000	0		After struction 00 2002 0001 (C :	= 1)		

BRA N	IN		Branch if N	ot Negative				
Implemented	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33	
		Х	Х	Х	Х	Х	Х	
Syntax:		{label:}	BRA	NN,	Expr			
Operands:		Expr is res	be a label, ab olved by the l 32768 +327	inker to a Slit		ion.		
Operation:		Condition = $!N$ If (Condition) (PC + 2) + 2 * Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register						
Status Affecte	ed:	None						
Encoding:		0011	1011	nnnn	nnnn	nnnn	nnnn	
		Slit16 value is resolved by the linker from the supplied label, absolute address or expression. If the branch is taken, the new address will be $(PC + 2) + 2 * Slit16$ , since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second						
		cycle. The 'n' bits are a signed literal that specifies the number of instructions						
Words:		offset from 1	(PC + 2).					
Cycles:		•	ch taken) – P			E dePIC33E		
Cycles.			ch taken) – P			, usi 10001		
Example 1:	00200 00200 00200 00200 00200	6 8 A C BYPASS:	BRA NN, 1			branch to ise cont		
	PC SR	Before Instruction 00 2000 0000		Instr	fter uction 0 200C 0000			

# 5

BRA N	OV	Branch if N	ot Overflow				
Implemented	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BRA	NOV,	Expr			
Operands:	Expr is res	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].					
Operation:	If (Condition (PC + 2	Condition = !OV If (Condition) (PC + 2) + 2 * Slit16 →PC NOP →Instruction Register					
Status Affecte	d: None						
Encoding:	0011	1000	nnnn	nnnn	nnnn	nnnn	
	the PC wil then beco	If the branch is taken, the new address will be $(PC + 2) + 2 * Slit16$ , since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second cycle.					
	cycle.	-					
	offset from	offset from (PC + 2).					
Vords: 1							
Cycles:		1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F					
	1 (4 if brar	nch taken) – P	IC24E, dsPI0	C33E			
Example 1:	002000 HERE: 002002 NO_NOV 002004 002006 002008 00200A 00200C BYPASS 00200E	  Goto the			V, branch t wise cor		
	Before Instruction PC 00 200 SR 000		Ins	After truction 00 200C 0008 (N =	= 1)		

### BRANOV Branch if Not Overfl

Implemented	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	BRA	NZ,	Expr		
Operands:		Expr is res	be a label, ab olved by the l 2768 +327	linker to a Sli	ss or express t16, where	ion.	
Operation:		• • •					
Status Affecte	ed:	None					
Encoding:		0011	1010	nnnn	nnnn	nnnn	nnnn
		value is res expression If the branc	olved by the ch is taken, th	linker from th he new addre	ions forward ( le supplied lal ss will be (PC	oel, absolute c + 2) + 2 * S	address or lit16, since
		the PC will then becon	have increm	ented to fetcl	n the next ins	truction. The	instruction
		cycle. The 'n' bits offset from	-	l literal that s	pecifies the n	umber of inst	ructions
Words:		1					
Cycles:		1 (2 if bran	ch taken) – F	PIC24F, PIC2	4H, dsPIC30F	F, dsPIC33F	
		1 (4 if bran	ch taken) – F	PIC24E, dsPI	C33E		
Example 1:	00200 00200 00200 00200 00200	)6 )8 )A )C BYPASS:	BRA NZ,   GOTO THE 			branch to	
	PC SR	Before Instruction 00 2000 0002	(Z = 1)	Ins	After truction 00 2002 0002 (Z =	1)	

. . .....

.

-

Implemented i	n: F	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
					Х	Х	Х		
Syntax:	{la	bel:}	BRA	OA,	Expr				
Operands:	Ex	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].							
Operation:	lf (	Condition = OA If (Condition) $(PC + 2) + 2 * Slit16 \rightarrow PC$ NOP $\rightarrow$ Instruction Register							
Status Affecte	d: No	one							
Encoding:		0000	1100	nnnn	nnnn	nnnn	nnnn		
	nu for su	mber '2 ' ward or I pplied Ial	he next PC. T Slit16', whic backward. Th bel, absolute	h supports bi e Slit16 valu address or e	ranches up to e is resolved xpression.	o 32K instruc by the linker	tions from the		
	the the	e PC will	h is taken, th have increments a two-cyc	ented to fetch	the next inst	truction. The	instruction		
			are a signed (PC + 2).	literal that sp	pecifies the n	umber of ins	tructions		
	I		The assembl be used.	er will conve	rt the specifie	ed label into t	the offset to		
Words:	1								
Cycles:	1 (	2 if brand	ch taken) – d	sPIC30F, dsF	PIC33F				
	1 (	4 if brand	ch taken) – d	sPIC33E					
Example 1:	002000 2 002002 2 002004 002006 002008 002008 00200C 2 00200E	NO_OA:	BRA OA,   GOTO THE 			, branch to wise con			
		Before struction 00 2000 8800	-	Ins PC (	After truction 00 200C 8800 (OA	N, OAB = 1)			

<b>BRA O</b>	B		Branch if O	verflow Aco	cumulator B		
Implemented i	n:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	[				Х	Х	Х
Syntax:		{label:}	BRA	OB,	Expr		
Operands:		Expr is res	be a label, ab olved by the l 32768 +327	linker to a SI	ess or express it16, where	iion.	
Operation:			-				
Status Affecte	d:	None					
Encoding:		0000	1101	nnnn	nnnn	nnnn	nnnn
		If the brand the PC will then becor	have increm	e new addre ented to fetc	expression. ess will be (PC h the next ins n, with a NOP	truction. The	instruction
		cycle.	-		n, with a NOP		
		offset from	-				
Words:		1					
Cycles:			ch taken) – d		PIC33F		
		1 (4 if bran	ch taken) – d	sPIC33E			
Example 1:	00200 00200 00200 00200 00200	6 8 A C BYPASS:	BRA OB,   GOTO THE 			8, branch t wise co	
	PC SR	Before Instruction 00 2000 8800	) )(OA, OAB =	PC	After struction 00 2002 8800 (OA	A, OAB = 1)	

.

BRA C	V	Branch if C	verflow			
Implemented	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BRA	OV,	Expr		
Operands:	Expr is res	be a label, ab solved by the l 32768 +327	inker to a Sli	•	ion.	
Operation:		-				
Status Affecte	d: None					
Encoding:	0011	0000	nnnn	nnnn	nnnn	nnnn
	address o If the bran the PC wil	ie is resolved r expression. ch is taken, th I have increme mes a two-cyc	e new addre ented to fetcl	ss will be (PC n the next inst	C + 2) + 2 * S truction. The	it16, since
	The 'n' bit	s are a signed ı (PC + 2).	literal that s	pecifies the n	umber of inst	ructions
Words:	1	(				
Cycles:	1 (2 if brai	nch taken) – P	IC24F, PIC2	4H, dsPIC30F	F, dsPIC33F	
	1 (4 if brar	nch taken) – P	IC24E, dsPl	C33E		
Example 1:	002000 HERE: 002002 NO_OV 002004 002006 002008 00200A 00200C BYPASS 00200E	BRA OV,   GOTO THE			7, branch t wise co:	
	Before Instruction PC 00 200 SR 000	_	In: PC SR	After struction 00 2002 0002 (Z =	= 1)	

BRA S	A		Branch if S	aturation Ac	cumulator A	\	
Implemented	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
					Х	Х	Х
Syntax:		{label:}	BRA	SA,	Expr		
Operands:		Expr is res	be a label, ab solved by the l 32768 +327	inker to a Sli		ion.	
Operation:			-				
Status Affecte	ed:	None					
Encoding:		0000	1110	nnnn	nnnn	nnnn	nnnn
		supplied la If the brand the PC will then becord cycle.	backward. Th abel, absolute ch is taken, th I have increme mes a two-cyc	address or e e new addre ented to fetcl cle instructior	expression. less will be (PC h the next ins h, with a NOP	C + 2) + 2 * S truction. The executed in t	lit16, since instruction he second
		The 'n' bits offset from	s are a signed (PC + 2).	literal that s	pecifies the n	umber of inst	tructions
Words:		1	. ,				
Cycles:		1 (2 if bran	nch taken) – d	sPIC30F, dsl	PIC33F		
		1 (4 if bran	nch taken) – d	sPIC33E			
Example 1:	00200 00200 00200 00200 00200	6 8 A C BYPASS:	BRA SA,   GOTO THE 			A, branch t wise co	
	PC SR	Before Instruction 00 2000 2400		PC	After struction 00 200C 2400 (SA	A, SAB = 1)	

BRA S	В		Branch if Sa	aturation Ac	cumulator E	3	
Implemented	in: Pl	IC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
					Х	Х	Х
Syntax:	{lab	el:}	BRA	SB,	Expr		
Operands:	Exp	or is reso	be a label, abs blved by the li 2768 +327	inker to a Slit		ion.	
Operation:	if (C		-				
Status Affecte	d: Nor	ne					
Encoding:	C	0000	1111	nnnn	nnnn	nnnn	nnnn
	the ther cyc	PC will n becon le.	h is taken, th have increme nes a two-cyc	ented to fetch le instruction	the next inst , with a NOP	truction. The executed in t	instruction he second
	-		are a signed	literal that sp	ecifies the n	umber of inst	ructions
		et from	(PC + 2).				
Words:	1						
Cycles:			ch taken) – d: ch taken) – d:		1C33F		
Example 1:	002000 H 002002 N 002004 002006 002008 00200A 00200C B 00200E	IO_SB:	BRA SB,   GOTO THE 			branch to ise cont	
	Inst	efore ruction 00 2000 0000		Instr	fter ruction 0 2002 0000		

BRA Z		Branch if Z	ero						
Implemented in	: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	BRA	Ζ,	Expr					
Operands:	Expr is reso	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].							
Operation:	if (Condition (PC + 2)	Condition = Z if (Condition) $(PC + 2) + 2 * Slit16 \rightarrow PC$ NOP $\rightarrow$ Instruction Register							
Status Affected:	None								
Encoding:	0011	0010	nnnn	nnnn	nnnn	nnnn			
	the PC will	have increme	ented to fetch	ss will be (PC a the next inst a, with a NOP e	ruction. The	instruction			
			literal that sp	pecifies the nu	umber of inst	ructions			
Words:	1	(1012).							
Cycles:		-	IC24F, PIC24 IC24E, dsPI0	4H, dsPIC30F C33E	, dsPIC33F				
Example 1:	002000 HERE: 002002 NO_Z: 002004 002006 002008 00200A 00200A 00200C BYPASS 00200E	  Goto th	BYPASS HERE		Z, branch a				
	Before Instruction PC 00 200 SR 000		PC SR	After Instruction 00 200C 0002 (/					

BSET		Bit Set f				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BSET{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0	91] for byte o 90] (even on 7] for byte o 15] for word	ly) for word operation	peration		
Operation:	1 →f <bit4></bit4>					
Status Affected:	None					
Encoding:	1010	1000	bbbf	ffff	ffff	fffb
	bit (bit 7 for The 'b' bits	ast Significar r byte operati select value	ons, bit 15 fo bit4 of the bi	r word opera t position to t	tions).	Significant
	The 'f' bits	select the ad	dress of the f	file register.		
		The extensio rather than a denote a wor When this ins	word operati d operation,	ion. You may but it is not re	use a .w ex equired.	tension to
		address mus				
		When this ins between 0 ar	-	rates in Byte	mode, 'bit4'	must be
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mod	lify-write ope	24E devices, rations on no Section 3.2.1	n-CPU Speci	al Function R	egisters. For	
Example 1: BSE	ET.B 0x601	, #0x3	; Set bi	t 3 in 0x6	01	
Data 0600 SR		Data 060	After Instruction 00 FA34 6R 0000			

I	Before nstructior	ו ו	After nstruction	
Data 0444	5604	Data 0444	D604	
SR	0000	SR	0000	

		Bit Set in W	ls			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BSET{.B}	Ws, [Ws],	#bit4		
			[WS], [WS++],			
			[WS],			
			[++Ws],			
			[Ws],			
Operands:		W15] . 7] for byte op . 15] for word				
Operation:	1 →Ws <bit< td=""><td>4&gt;</td><td></td><td></td><td></td><td></td></bit<>	4>				
Status Affected:	None					
Encoding:	1010	0000	bbbb	0B00	0ppp	SSSS
Description:	Least Sign for byte op	in register Ws ificant bit (bit erations, bit 1 may be used	0) and advan 5 for word op	ces to the Mo	nbering begi ost Significan	ns with the t bit (bit 7
Description:	Least Sign for byte op addressing The 'b' bits The 'B' bit The 'p' bits	ificant bit (bit erations, bit 1	0) and advan 5 for word op I for Ws. bit4 of the bit r word opera urce Address	ces to the Mo erations). Re position to be tion ('0' for w s mode.	nbering begi ost Significan gister direct e cleared.	ns with the It bit (bit 7 or indirect
Description:	Least Sign for byte op addressing The 'b' bits The 'B' bits The 'p' bits The 's' bits	ificant bit (bit ) erations, bit 1 may be used select value selects byte of select the so select the so The extension rather than a	0) and advan 5 for word op I for Ws. bit4 of the bit r word opera urce Address urce/destinat n . B in the in word operatio	ces to the Mo erations). Re position to be tion ('0' for w mode. ion register. struction den- on. You may	nbering begi ost Significan gister direct e cleared. ord, '1' for by otes a byte o use a .w exte	ns with the t bit (bit 7 or indirect /te).
Description:	Least Sign for byte op addressing The 'b' bits The 'B' bit The 'p' bits The 's' bits <b>Note 1:</b>	ificant bit (bit ) erations, bit 1 may be used select value selects byte of select the so select the so The extension	0) and advan 5 for word op I for Ws. bit4 of the bit r word opera urce Address urce/destinat n . B in the in: word operation, b truction opera	ces to the Mo erations). Re position to be tion ('0' for w mode. ion register. struction dene on. You may but it is not re ates in Word	nbering begi ost Significan gister direct e cleared. ord, '1' for by otes a byte o use a . w exte quired.	ns with the t bit (bit 7 or indirect /te). peration ension to
Description:	Least Sign for byte op addressing The 'b' bits The 'b' bits The 'p' bits The 's' bits <b>Note 1:</b> 2:	ificant bit (bit ) erations, bit 1 g may be used select value selects byte of select the so select the so The extension rather than a denote a work When this ins	0) and advan 5 for word op I for Ws. bit4 of the bit r word opera urce Address urce/destinat n . B in the ins word operation, b truction opera cess must be v truction opera	ces to the Mo erations). Re position to be tion ('0' for w mode. ion register. struction den- on. You may but it is not re ates in Word vord-aligned.	nbering begi ost Significan gister direct e cleared. ord, '1' for by otes a byte o use a .w exte quired. mode, the so	ns with the t bit (bit 7 or indirect /te). peration ension to purce
Description:	Least Sign for byte op addressing The 'b' bits The 'B' bits The 'p' bits The 's' bits <b>Note 1:</b> 2: 3:	ificant bit (bit ) erations, bit 1 g may be used select value selects byte of select the so The extension rather than a denote a worn When this ins register addre When this ins	0) and advan 5 for word op 1 for Ws. bit4 of the bit r word opera urce Address urce/destinat h . B in the in word operation, b truction opera truction opera d operation, pera d 7. and PIC24E	ces to the Mo erations). Re position to be tion ('0' for w mode. ion register. struction den- on. You may but it is not re ates in Word vord-aligned. ates in Byte r devices, this	nbering begi ost Significan gister direct e cleared. ord, '1' for by otes a byte o use a . w exter quired. mode, the so node, 'bit4' n	ns with the t bit (bit 7 or indirect /te). peration ension to burce hust be ses the
Description:	Least Sign for byte op addressing The 'b' bits The 'B' bits The 'p' bits The 's' bits <b>Note 1:</b> 2: 3:	ificant bit (bit ( erations, bit 1 may be used select value selects byte of select the so The extension rather than a denote a word When this ins register addre When this ins between 0 an In dsPIC33E DSRPAG reg	0) and advan 5 for word op 1 for Ws. bit4 of the bit r word opera urce Address urce/destinat h . B in the in word operation, b truction opera truction opera d operation, pera d 7. and PIC24E	ces to the Mo erations). Re position to be tion ('0' for w mode. ion register. struction den- on. You may but it is not re ates in Word vord-aligned. ates in Byte r devices, this	nbering begi ost Significan gister direct e cleared. ord, '1' for by otes a byte o use a . w exter quired. mode, the so node, 'bit4' n	ns with the t bit (bit 7 or indirect /te). peration ension to burce hust be ses the

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1** "**Multi-Cycle Instructions**".

Example 1:	BSET.	B W3, ‡	‡0x7	; Set b	oit 7 in W3
Evenale 2	W3 SR	Before struction 0026 0000	W3 SR	0000	]
Example 2:	BSET	[W4++],	#0×0		it 0 in [W4] increment W4
	-	efore ruction	Ir	After	ı
	W4	6700	W4	6702	
Data	6700	1734	Data 6700	1735	
	SR	0000	SR	0000	

BSW		Bit Write in	Ws			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BSW.C BSW.Z	Ws, [Ws], [Ws++], [Ws], [++Ws],	Wb		
			[Ws],			
Operands:	$Ws \in [W0]$ $Wb \in [W0]$					
Operation:	<u>For ".C" op</u> C →Ws<	oeration: <(Wb)> peration (defaul	<u>.lt):</u>			
Status Affected:	None	<u>.</u>	. <u></u>	<u> </u>		<u>    .                                </u>
Encoding:	1010	1101	Zwww	w000	0ppp	SSSS
	register. Or the destina Wb, and eit The 'Z' bit s The 'w' bits The 'p' bits	advances to the nly the four Lea ation bit number ither register di selects the C c s select the cou s select the sou	east Significan er. Register di direct, or indire or Z flag as so dress of the b urce Address	nt bits of Wb a lirect address ect addressin ource. bit select regis mode.	are used to c sing must be ng may be us	determine used for
		This instructic provided, the				extension is
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo details,	odify-write ope	C24E devices, t rerations on nor n <b>Section 3.2.</b> 1	on-CPU Specia 1 "Multi-Cycl ; Set b:	ial Function R	Registers. For ns".	r more
W2 W3	_		After Instruction W2 7234 W3 111F			

Example 2: ; Set bit W3 in W2 to the complement BSW.Z W2, W3 ; of the Z bit Before After Instruction Instruction W2 E235 W2 E234 W3 W3 0550 0550 SR 0002 (Z = 1, C = 0) SR0002 (Z = 1, C = 0)Example 3: BSW.C [++W0], W6 ; Set bit W6 in [W0++] to the value ; of the C bit Before After Instruction Instruction W0 1000 W0 1002 W6 34A3 W6 34A3 Data 1002 Data 1002 2380 2388 SR 0001 (Z = 0, C = 1)SR 0001 (Z = 0, C = 1)Example 4: BSW.Z [W1--], W5 ; Set bit W5 in [W1] to the ; complement of the Z bit ; Post-decrement W1 Before After Instruction Instruction W1 1000 W1 **0FFE** W5 888B W5 888B Data 1000 C4DD CCDD Data 1000 SR 0001 (C = 1)SR 0001 (C = 1)

BTG		Bit Toggle f	i			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BTG{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0 bit4 ∈ [0	191] for byte c 190] (even on 7] for byte o 15] for word	nly) for word opperation	operation		
Operation:	(f) <bit4> -</bit4>	→(f) <bit4></bit4>				
Status Affected:	None					
Encoding:	1010	1010	bbbf	ffff	ffff	fffb
Description:	operand, b advances t word opera	file register 'f bit numbering to the Most Si ration) of the b	begins with t Significant bit byte.	the Least Sig (bit 7 for byte	gnificant bit (b e operation, b	oit 0) and
		s select value s select the ad			ggle.	
	2:	rather than a denote a wor When this ins	a word operat rd operation, struction ope	nstruction der tion. You may , but it is not re erates in Worc	y use a .w ex required.	xtension to
		address mus When this ins between 0 ar	struction ope	igned. erates in Byte	mode, 'bit4'	must be
Words:	1	between o a.	nd 7.			
Cycles:	1 1(1)					
read-moo details, s	odify-write ope see <b>Note 3</b> in	C24E devices, erations on no a <b>Section 3.2.</b> 1	on-CPU Speci 1 "Multi-Cyc	ial Function R cle Instruction	Registers. For ns".	
Example 1: BTG	.B 0x100	01, #0x4	; Toggle	bit 4 in 0:	x1001	
Data 1000 SR		Data 100 SI				
Example 2: BTG	0x166	60, #0x8	; Toggle	bit 8 in RA	АМ660	
Data 1660 SR		Data 1660 SF				

BTG		Bit Toggle	in Ws						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	BTG{.B}	Ws, [Ws],	#bit4					
			[Ws++],						
			[Ws],						
			[++Ws],						
			[Ws],						
Operands: Operation:	bit4 ∈ [0	W15] . 7] for byte o . 15] for word - →Ws <bit4></bit4>	operation						
Status Affected:	None	·							
Encoding:	1010	0010	bbbb	0800	0ppp	SSSS			
Description:	bit number the Most S operations The 'b' bits	ing begins wi ignificant bit ). Register di s select value	th the Least (bit 7 for byte rect or indirect bit4, the bit p	mplemented) Significant bit operations, b t addressing position to tes ation ('0' for w	(bit 0) and a bit 15 for word may be used t.	dvances to d I for Ws.			
		select the so select the so		-					
	Note 1:								
	2:								
	3:								
	4.	In dsPIC33E	and PIC24E	devices, this		ses the			
	4:	DSRPAG reg Data Space.		ect address g	eneration in				
Words:	<b>4:</b> 1	DSRPAG reg		ect address g	eneration in				

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read an read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	BTG	W2, #0x0	; Toggle bit 0 in W2
	В	efore	After
	Ins	truction	Instruction
	W2	F234	W2 F235
	SR	0000	SR 0000

Example 2: BTG	[W0++],	#0x0	; Toggle bit 0 in [W0] ; Post-increment W0
	Before nstruction		After nstruction
W0	2300	W0	2302
Data 2300	5606	Data 2300	5607
SR	0000	SR	0000

BTSC		Bit Test f, S	Skip if Clear						
Implemented in	n: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
	Х	Х	Х	Х	Х	Х			
Syntax:	{label:}	BTSC{.B}	f,	#bit4					
Operands:	f ∈ [0 8 bit4 ∈ [0	191] for byte 190] (even or 7] for byte c 15] for worc	nly) for word opperation	operation					
Operation:	Test (f) <bi< td=""><td>t4&gt;, skip if cle</td><td>ear</td><td></td><td></td><td></td></bi<>	t4>, skip if cle	ear						
Status Affected	: None								
Encoding:	1010	1111	bbbf	ffff	ffff	fffb			
	and on the the next in the file reg begins wit	e next cycle, a struction is e jister are not h the Least S	a NOP is exect xecuted as n changed. For ignificant bit	nt instruction outed instead. ormal. In eith the bit4 ope (bit 0) and ad ns, bit 15 for	If the tested er case, the rand, bit num vances to the	bit is '1', contents of bering e Most			
		The 'b' bits select value bit4, the bit position to test. The 'f' bits select the address of the file register.							
	Note 1: 2: 3:	rather than a denote a wo When this in address mus	a word operat rd operation, struction ope st be word-ali struction ope	nstruction der ion. You may but it is not re rates in Word gned. rates in Byte	use a .w ex equired. d mode, the f	tension to ile register			
Words:	1								
Cycles:	1 (2 or 3) <sup>(*</sup>	1)							
read	sPIC33E and PIC d-modify-write op ails, see <b>Note 3</b> ir	erations on ne	on-CPU Spec	ial Function F	Registers. For				
Example 1:	002000 HERE: 002002 002004 002006 002008 BYPASS 00200A	GOTO B  	x1201, #2 YPASS	; If bit ; skip th	2 of 0x120 e GOTO	l is 0,			
	Before			After					
	Instruction			nstruction					
	PC 00 200		PC	00 2002					
Data 1			Data 1200 SR	264F					
	SR 000	0	SK	0000					

002 002 002	004 006 008 BYPASS:	BTSC GOTO   	0x804, #1 BYPASS		t 14 of 0x804 is 0, the GOTO
	Before Instruction			After Instruction	
PC	00 2000	]	PC	00 2006	
Data 0804	2647		Data 0804	2647	
SR	0000		SR	0000	

00 2000

264F

0000

PC

W0

SR

BTSC			Bit Test W	s, Skip if Cle	ar		
Implemented	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	BTSC	Ws,	#bit4		
				[Ws],			
				[Ws++],			
				[Ws],			
				[++Ws],			
				[Ws],			
Operands:		Ws∈ [W0					
		bit4 ∈ [0	. 15]				
Operation:		Test (Ws)<	bit4>, skip if	clear			
Status Affecte	d:	None				1	1
Encoding:		1010	0111	bbbb	0000	0ppp	SSSS
		(bit 0) and register dir The 'b' bits The 'p' bits	advances to ect or indirec select value select the se	t numbering b the Most Sign t addressing bit4, the bit p ource Addres ource register	nificant bit (bi may be used position to tes s mode.	t 15) of the w for Ws.	
		Note:	This instruct	tion operates	in Word mod	e only.	
Words:		1					
Cycles:		1 (2 or 3 if	the next inst	ruction is skip	ped) <sup>(1)</sup>		
rea	ad-mod	ify-write op	erations on n	, the listed cyo on-CPU Spec .1 "Multi-Cyc	ial Function F	Registers. For	
Example 1:	00200 00200 00200 00200	)4		WO, #0x0 BYPASS		bit 0 of W ip the GOTO	
	00200 00200	)8 BYPASS: )A	· · · ·				
		Before			After		
		Instruction			Instruction		

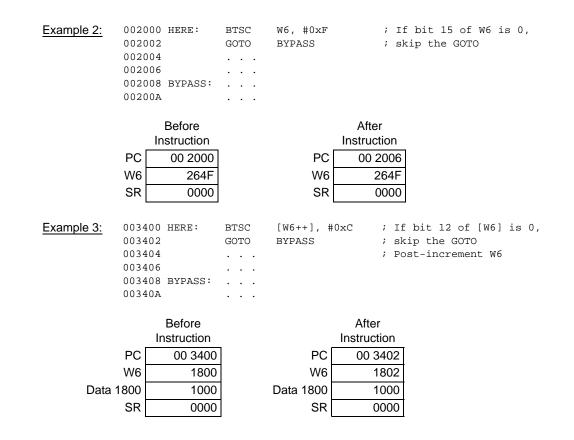
PC

W0

SR

00 2002

264F



Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BTSS{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0	91] for byte o 90] (even onl 7] for byte op 15] for word	y) for word operation	peration		
Operation:	Test (f) bit	4>, skip if set				
Status Affected:	None					
Encoding:	1010	1110	bbbf	ffff	ffff	fffb
Description:	instruction and on the	the file registe (fetched durir next cycle, a ction is execu	ng the current NOP is execu	t instruction e ted instead. I	execution) is fifthe tested b	discarded it is '0', the
Description:	instruction and on the next instruct file register with the Le bit (bit 7 for The 'b' bits	(fetched durir next cycle, a ction is execu- are not chan ast Significan byte operations select value	ng the current NOP is execu ted as norma ged. For the t bit (bit 0) ar on, bit 15 for bit4, the bit p	t instruction e ted instead. I I. In either ca bit4 operand ad advances word operation osition to tes	execution) is f the tested b use, the conte bit numberin to the Most S on).	discarded it is '0', the ents of the ng begins
Description:	instruction and on the next instruc- file register with the Le bit (bit 7 for The 'b' bits <b>Note 1:</b> 2: 3:	(fetched durir next cycle, a ction is execu- are not chan ast Significan byte operatio	ing the current NOP is execu- ted as norma- ged. For the t bit (bit 0) ar on, bit 15 for bit4, the bit p dress of the fin . B in the in word operation, b truction oper be word-alig truction oper	t instruction e ted instead. I I. In either ca bit4 operand ad advances word operation osition to tes ile register. struction den on. You may but it is not re ates in Word uned.	execution) is of f the tested b use, the content b the Most S on). t. otes a byte c use a . w ext quired. mode, the fil	discarded it is '0', the ents of the ng begins Significant peration ension to e register
Description:	instruction and on the next instruc- file register with the Le bit (bit 7 for The 'b' bits <b>Note 1:</b> 2: 3:	(fetched durir next cycle, a ction is execu- are not chan ast Significan byte operation select value the extension rather than a denote a word When this ins address must When this ins	ing the current NOP is execu- ted as norma- ged. For the t bit (bit 0) ar on, bit 15 for bit4, the bit p dress of the fin . B in the in word operation, b truction oper be word-alig truction oper	t instruction e ted instead. I I. In either ca bit4 operand ad advances word operation osition to tes ile register. struction den on. You may but it is not re ates in Word uned.	execution) is of f the tested b use, the content b the Most S on). t. otes a byte c use a . w ext quired. mode, the fil	discarded it is '0', the ents of the ng begins Significant peration ension to e register

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	007100 HERE: 007102 007104	BTSS.B 0x1401, #0x1 ; If bit 1 of 0x1401 is 1, CLR WREG ; don't clear WREG 
Data 1	Before           Instruction           PC         00 7100           400         0280           SR         0000	After Instruction PC 00 7104 Data 1400 0280 SR 0000
Example 2:	007100 HERE: 007102 007104 007106 BYPASS:	BTSS 0x890, #0x9 ; If bit 9 of 0x890 is 1, GOTO BYPASS ; skip the GOTO  
Data	Before Instruction PC 00 7100 0890 00FE SR 0000	After Instruction PC 00 7102 Data 0890 00FE SR 0000

Instruction

00 2000

264F

0000

PC

W0

SR

			Bit Test Ws	, Skip if Set			
Implemented i	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	BTSS	Ws,	#bit4		
				[Ws],			
				[Ws++],			
				[Ws],			
				[++Ws],			
				[Ws],			
Operands:		Ws ∈ [W0 bit4 ∈ [0					
Operation:		Test (Ws)<	bit4>, skip if :	set.			
Status Affecte	d:	None					
Encoding:	ſ	1010	0110	bbbb	0000	0ppp	SSSS
		(bit 0) and a register dire The 'b' bits The 's' bits	advances to ect or indirect select the va select the so	the Most Sigr t addressing		t 15) of the w for Ws.	
		-			in Word mod	e only.	
Words:		1					
Cycles:		1 (2 or 3 if 1	he next instr	uction is skip	ped) <sup>(1)</sup>		
rea	ad-modi	fy-write ope	erations on no	on-CPU Spec	cle count doe ial Function F <b>le Instructio</b>	Registers. For	
rea de	ad-modi tails, se 00200 00200 00200	fy-write ope e Note 3 in 0 HERE: 2 4	BTSS N	on-CPU Spec	ial Function F i <b>le Instructio</b> ; If	Registers. For	o is 1,
rea	ad-modi tails, se 00200 00200 00200 00200	fy-write ope e Note 3 in 0 HERE: 2 4 6 8 BYPASS:	BTSS W GOTO F	on-CPU Spec 1 <b>"Multi-Cyc</b> 10, #0x0	ial Function F i <b>le Instructio</b> ; If	Registers.For ns". bit 0 of W	o is 1,

Instruction

00 2006

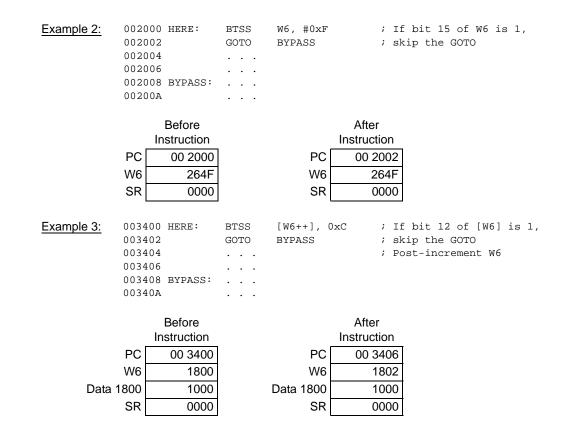
264F

0000

PC

W0

SR



## 5

Instruction Descriptions

BTST		Bit Test f						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	BTST{.B}	f,	#bit4				
Operands:	f ∈ [0 8 bit4 ∈ [0	191] for byte o 190] (even on . 7] for byte o . 15] for word	ly) for word o peration	operation				
Operation:	(f) <bit4> -</bit4>	→Z						
Status Affected:	Z							
Encoding:	1010	1011	bbbf	ffff	ffff	fffb		
Description:	stored to the register are the Least \$ (bit 7 for b	file register 'f he Z flag in th e not changed Significant bit yte operation,	e STATUS re d. For the bit- (bit 0) and a bit 15 for wo	egister. The c 4 operand, bit dvances to th ord operation)	ontents of the numbering t e Most Signi	e file begins with		
	The 'b' bits select value bit4, the bit position to be tested. The 'f' bits select the address of the file register.							
	2: 3:	rather than a denote a wor When this ins address mus When this ins between 0 ar	d operation, struction ope t be word-ali struction ope	but it is not re rates in Worc gned.	equired. mode, the f	ile register		
Words:	1							
Cycles:	1 <sup>(1)</sup>							
read-mo details, s	dify-write ope	24E devices, erations on no <b>Section 3.2.</b> 01, #0x3	on-CPU Spec 1 "Multi-Cyc ; Set Z	ial Function R	egisters. For ns".			
Data 120 SF		Data 12		(Z = 1)				
Example 2: BT	CST 0x13	02, #0x7		= compleme in 0x1302	nt of			
Data 130 S	Before Instruction 2 F7FF	Data 13	After Instructior 302 F7FF	1				

BTST	DICOAE	DICOALI	DICO4E				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	BTST.C	Ws,	#bit4			
		BTST.Z	[Ws],				
			[Ws++],				
			[Ws],				
			[++Ws],				
			[Ws],				
Operands:	Ws ∈ [W0 bit4 ∈ [0						
Operation:	For ".C" op (Ws) <bit For ".Z" op (Ws)<bit< td=""><td>4&gt; →C eration (defau</td><td><u>ılt):</u></td><td></td><td></td><td></td></bit<></bit 	4> →C eration (defau	<u>ılt):</u>				
Status Affected:	Z or C	.4>→∠					
Encoding:		0011	bbbb	7000	nad	gggg	
Description:	10100011bbbbZ0000pppssssBit 'bit4' in register Ws is tested. If the ". z" option of the instruction is specified, the complement of the tested bit is stored to the Zero flag in the STATUS register. If the ". c" option of the instruction is specified, the value of the tested bit is stored to the Carry flag in the STATUS register. In either case, the contents of Ws are not changed.						
	(bit 0) and register dir The 'b' bits The 'Z' bits The 'p' bits	t operand, bit advances to t ect or indirect select value selects the C select the so select the so	the Most Sign addressing r bit4, the bit p or Z flag as d ource Address	ificant bit (bit may be used osition to test lestination. mode.	15) of the wo		
	Note:	This instructi provided, the				extension is	
Words:	1						
Cycles:	1(1)						

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 "Multi-Cycle Instructions**".

Example 1:	BTST.C [W0++],	<pre>#0x3 ; Set C = bit 3 in [W0] ; Post-increment W0</pre>
Data	Before           Instruction           W0         1200           1200         FFF7           SR         0001         (C = 1)	After Instruction W0 1202 Data 1200 FFF7 L) SR 0000
Example 2:	BTST.Z W0, #0x7 Before Instruction W0 F234 SR 0000	After Instruction W0 F234 SR 0002 (Z = 1)

BTST	DI00/5	Bit Test in \	-			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BTST.C	Ws,	Wb		
		BTST.Z	[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:	Ws ∈ [W0 Wb ∈ [W0					
Operation:		Vb)> →C <u>eration (defa</u> t	<u>ult):</u>			
Status Affected:	Z or C	·				
Encoding:	1010	0101	Zwww	w000	0ppp	SSSS
Description:	specified, t STATUS re compleme	bit in register the value of the value of the value of the egister. If the fint of the tester either case, the tester of the case, the tester of tester	he tested bit i '. z" option of d bit is stored	s stored to th f the instruction d to the Zero	e Carry flag i on is specifie flag in the ST	n the d, the
	number. Bi advances f	our Least Sign it numbering to to the Most Si irect or indired	begins with th ignificant bit (	he Least Sign (bit 15) of the	ificant bit (bit working regi	0) and
	The 'w' bits The 'p' bits	selects the C s select the ac s select the sc s select the sc	ddress of the ource Address	bit select reg s mode.	ister.	
	Note:			ates in Word on is assume		extension is
Words:	1					
Cycles:	1 <sup>(1)</sup>					

**Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1** "**Multi-Cycle Instructions**".

Example 1: BTST.C W2, W3 ; Set C = bit W3 of W2 Before After Instruction Instruction F234 W2 F234 W2 2368 2368 W3 W3 0001 (C = 1) 0000 SR SR Example 2: BTST.Z [W0++], W1 ; Set Z = complement of ; bit W1 in [W0], ; Post-increment W0 Before After Instruction Instruction W0 1200 W0 1202 W1 CCC0 W1 CCC0 Data 1200 6243 Data 1200 6243 SR 0002 (Z = 1)SR 0000

BTSTS		Bit Test/Set	f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BTSTS{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0	91] for byte c 90] (even onl 7] for byte op 15] for word	y) for word operation	operation		
Operation:	(f) <bit4> → 1 →(f)<bit4< td=""><td></td><td></td><td></td><td></td><td></td></bit4<></bit4>					
Status Affected:	Z					
Encoding:	1010	1100	bbbf	ffff	ffff	fffb
	to '1' in the the Least S (bit 7 for by	ne Zero flag ir file register. Significant bit /te operations	For the bit4 ( (bit 0) and a , bit 15 for w	operand, bit i dvances to th ord operatio	numbering b ne Most Sign ns).	egins with
		select value select the ad			st/set.	
		The extension rather than a denote a wor	word operat	ion. You may	use a .wex	
						tension to
		When this ins address mus	truction ope	rates in Word	-	
	3:	When this ins address mus When this ins between 0 ar	truction ope be word-alig	rates in Word gned.	d mode, the f	ile register
	3:	address mus When this ins	truction ope be word-alig truction ope d 7.	rates in Word gned. rates in Byte	d mode, the f mode, 'bit4'	ile register must be
Words:	3:	address mus When this ins between 0 ar	truction ope be word-alig truction ope d 7.	rates in Word gned. rates in Byte	d mode, the f mode, 'bit4'	ile register must be

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

0002 (Z = 1)

SR

Example 1: BTSTS.B 0x1201, #0x3 ; Set Z = complement of bit 3 in 0x1201, ; then set bit 3 of 0x1201 = 1Before After Instruction Instruction Data 1200 F7FF Data 1200 FFFF SR 0000 SR 0002 (Z = 1)Example 2: BTSTS 0x808, #15 ; Set Z = complement of bit 15 in 0x808, ; then set bit 15 of 0x808 = 1Before After Instruction Instruction RAM300 8050 RAM300 8050

SR

	<b>B</b> 100 ( <b>F</b>		5100/5			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	BTSTS.C	Ws,	#bit4		
		BTSTS.Z	[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:	Ws ∈ [W0 bit4 ∈ [0					
Operation:	For ".C" op					
	(Ws) <bi 1 →Ws&lt;</bi 					
		eration (defa	<u>ult):</u>			
	(Ws) di 1 →Ws<					
Status Affected:	Z or C					
Encoding:	1010	0100	bbbb	Z000	0ppp	SSSS
Description:	specified, t STATUS re of the teste	he compleme egister. If the f ed bit is store	ent of the test '.C" option of	e ". z" option ed bit is store the instructio flag in the S	ed to the Zero n is specified	flag in the I, the value
	The 'b' bits	select the va	alue bit4, the	bit position to	test/set.	
			or Z flag as o			
			ource Addres			
	Note 1:	This instructi	on only opera	ates in Word r		xtension is
	2:		as a pointer, egister (SR).		ntain the add	Iress of the
	2: 3:	CPU Status r In dsPIC33E	egister (SR). and PIC24E		instruction us	ses the
Words:	2: 3:	CPU Status r In dsPIC33E DSRPAG reg	egister (SR). and PIC24E	devices, this	instruction us	ses the

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1: BTSTS.C [W0++], #0x3	<pre>; Set C = bit 3 in [W0] ; Set bit 3 in [W0] = 1 ; Post-increment W0</pre>
Before	After
Instruction	Instruction
W0 1200	W0 1202
Data 1200 FFF7 Data	a 1200 FFFF
SR 0001 (C = 1)	SR 0000
Example 2: BTSTS.Z W0, #0x7	; Set Z = complement of bit 7 ; in WO, and set bit 7 in WO = 1
Before	After
Instruction	Instruction
W0 F234 SR 0000	W0 F2BC SR 0002 (Z = 1)

CALL		Call Subrou	ıtine			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	CALL	Expr			
Operands:		e a label or e olved by the l				86061.
Operation:	$(PC) + 4 \rightarrow$ (PC<15:0>) (W15) + 2 - (PC<23:16: (W15) + 2 - $lit23 \rightarrow PC$	PC ) →(TOS) →W15 >) →(TOS)				
Status Affected:	None	action region				
Encoding:						
1st word	0000	0010	nnnn	nnnn	nnnn	nnn0
2nd word	0000	0000 broutine call	0000	0000	0nnn	nnnn
	(PC + 4) stacked, The 'n' bits <b>Note:</b>	range. Before is PUSHed c the 23-bit val form the targ The linker wi be used.	nto the stack ue 'lit23' is lo let address.	c. After the repart of the	eturn address e PC.	is
Words:	2					
Cycles:	2					
Example 1: 0260 0260	04	· · · · · · ·	Wl		l _FIR sub:	
0268 0268	44 _FIR: 46	MOV #0x4	00, W2	; _FI	R subrouti:	ne start
PC W15 Data A268 Data A26A SR	Before Instruction 02 6000 A268 FFFF FFFF 0000		PC W15 ata A268 ata A26A SR	After nstruction 02 6844 A26C 6004 0002 0000		

Example 2: 0720			G66 0, W1	; call ro	utine _G66	
0772 0772 0772		W	6, [₩7++]	; routine	start	
PC W15 Data 9004 Data 9006 SR	Before Instruction 07 2000 9004 FFFF FFFF 00000		 PC   W15   Data 9004   Data 9006   SR	After nstruction 07 7A28 9008 2004 0007 0000		
CALL		Call Subrou	utine			
Implemented in:	PIC24F	PIC24H	PIC24E X	dsPIC30F	dsPIC33F	dsPIC33E X
Syntax:	{label:}	CALL	Expr			
Operands:		be a label or e				
Operation:	(PC) + 4 → (PC<15:1> (W15) + 2 (PC<23:16 (W15) + 2 0 →SFA bir lit23 →PC	) →TOS<15: →W15 >) →TOS →W15	1>, SFA bit –		3 ∈ [0 838	88606].
Status Affected:	SFA	denon regis				
Encoding: 1st word	0000	0010	nnnn	nnnn	nnnn	nnn0
2nd word	0000	0000	0000	0000	0nnn	nnnn
Description:	memory (PC + 4) stacked,	Ibroutine call range. Before is PUSHed c the 23-bit va	e the CALL is onto the stack lue 'lit23' is lo	s made, the 2 k. After the re	4-bit return a turn address	address
		form the targ				
	Note:	The linker will be used.	ill resolve the	e specified ex	pression into	o the lit23 to
Words:	2					
Cycles:	4					

0000

SR

Example 1: 0260 0260	04 M	ALL _FIR NOV W0, W	1	; Call	_FIR subroutine
0268 0268	44 _FIR: M	 IOV #0x400 	, W2	; _FIR	subroutine start
	Before Instruction			After truction	
PC	02 6000		PC	02 6844	
W15	A268		W15	A26C	
Data A268	FFFF	Data	a A268	6004	
Data A26A	FFFF	Data	A26A	0002	
SR	0000		SR	0000	
	)04 M  A28 _G66: J	CALL _G66 MOV W0,  INC W6,		call rout	_
077 <i>1</i> 077 <i>1</i>		• • •			
	Before Instruction			After truction	
PC	07 2000		PC (	07 7A28	
W15	9004		W15	9008	
Data 9004	FFFF	Dat	a 9004	2004	
Data 9006	FFFF	Dat	a 9006	0007	

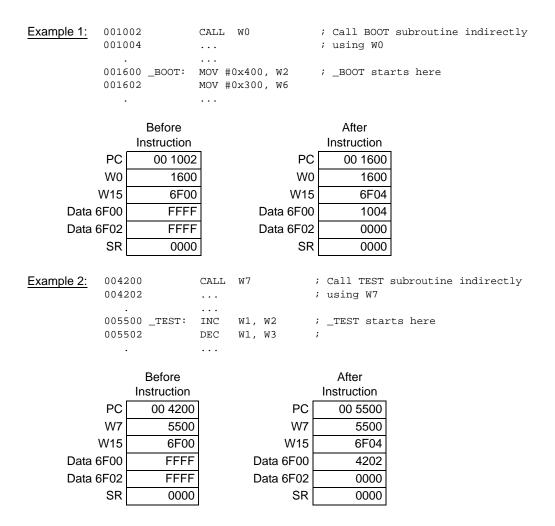
0000

SR

CALL							
Implemented	ın:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х	X		X	Х	
Syntax:		{label:}	CALL	Wn			
Operands:		Wn∈[W0	W15]				
Operation:			) →TOS →W15 >) →TOS →W15				
Status Affecte	ed:	None					
Encoding:		0000	0001				
Description:		Indirect sub Before the onto the sta	CALL is made ack. After the	e, the 24-bit return addre	0000 32K instructio return addres ss is stacked ared Since F	s (PC + 2) is , Wn<15:1> i	PUSHed s loaded
Description:		Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits	Droutine call c CALL is made ack. After the :1> and PC<	ver the first a e, the 24-bit return addre 22:16> is cle	32K instructio return addres ess is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i	n memory PUSHed s loaded
Words:		Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits	Droutine call of CALL is made ack. After the :1> and PC<	ver the first a e, the 24-bit return addre 22:16> is cle	32K instructio return addres ess is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i	n memory PUSHed s loaded
Words:		Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits	Droutine call of CALL is made ack. After the :1> and PC<	ver the first a e, the 24-bit return addre 22:16> is cle	32K instructio return addres ess is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i	n memory PUSHed s loaded
Words:	00100 00100	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2	CALL WO	ver the first : e, the 24-bit return addre 22:16> is cle urce register ;	32K instructio return addres ess is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway	m memory PUSHed is loaded ays '0',
Words: Cycles:	00100	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT:	CALL W0	ver the first ; e, the 24-bit return addre 22:16> is cle urce register ; ;	32K instructio return addres ss is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles:	00100 00160	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2	CALL W0 CALL CALL CALL CALL CALL CALL CALL CALL	ver the first ; e, the 24-bit return addre 22:16> is cle urce register ; ;	32K instructio return addres ss is stacked ared. Since F Call BOOT s using W0 _BOOT start	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles:	00100 00160 00160	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2 Before	CALL W0 CALL CALL CALL CALL CALL CALL CALL CALL	ver the first ; e, the 24-bit return addre 22:16> is cle urce register ; , w2 ; , w6	32K instructio return addres ss is stacked ared. Since F Call BOOT s using W0 _BOOT start After	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles:	00100 00160 00160	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2	CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0	ver the first ; e, the 24-bit return addre 22:16> is cle urce register ; , w2 ; , w6	32K instructio return addres ss is stacked ared. Since F Call BOOT s using W0 _BOOT start	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles:	00100 00160 00160	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2 Before Instruction	CALL W0 CALL W0 contine call c CALL is made ack. After the call PC< gnored. select the so CALL W0  MOV #0x400 MOV #0x300 	ver the first : e, the 24-bit return addre 22:16> is cle urce register ; , w2 ; , w6	32K instructio return addres ss is stacked ared. Since F Call BOOT s using W0 _BOOT start After nstruction	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles: <u>Example 1:</u>	00100 00160 00160	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2 Before Instruction 00 1002	CALL W0 CALL W0	ver the first : e, the 24-bit return addre 22:16> is cle urce register ; , w2 ; , w6 ; PC	32K instructio return addres ss is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles: <u>Example 1:</u>	00100 00160 00160 PC W0 W0	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2 Before Instruction 00 1002 1600	CALL W0 CALL W0	ver the first : e, the 24-bit return addre 22:16> is cle urce register ; , w2 ; , w2 ; , w6 PC urce W0	32K instructio return addres ss is stacked ared. Since F	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',
Words: Cycles: <u>Example 1:</u>	00100 00160 00160 PC W0 W15 6F00	Indirect sub Before the onto the sta into PC<15 Wn<0> is ig The 's' bits 1 2 2 4 0 _BOOT: 2 Before Instruction 00 1002 1600 6F00	CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL W0 CALL CALL W0 CALL W0 CALL CALL CALL MOV #0x400 MOV #0x400 CALL	wer the first : e, the 24-bit return addre 22:16> is cle urce register ; ; , w2 ; , w6 ; , w2 ; , w6 In PC W0	After 00 1600 1600 00 1600 00 1600 00 6F04	ns of program s (PC + 2) is , Wn<15:1> i PC<0> is alway ubroutine i	m memory PUSHed is loaded ays '0',

Example 2: 00420 00420		CALL W7		all TEST su sing W7	broutine ir	ndirectly		
00550	00 _TEST: 02	W1, W DEC W1, W		TEST starts	s here			
PC W7 W15 Data 6F00 Data 6F02 SR	Before Instruction 00 4200 5500 6F00 6F00 FFFF FFFF 0000	-	In: PC W7 W15 ta 6F00 ta 6F02 SR	After struction 00 5500 5500 6F04 4202 0000 0000				
CALL		Call Indirect						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
			Х			Х		
Syntax:	{label:}	CALL	Wn					
Operands:	Wn∈ [W0	W15]						
Operation:	(PC<15:1>) (W15) + 2 - (PC<23:16) $(W15) + 2 - 0 \rightarrow SFA bit 0 \rightarrow PC<22$ (Wn<15:1>)	$ \begin{array}{l} Wn \in [W0 \dots W15] \\ (PC) + 2 \rightarrow PC \\ (PC < 15:1 >) \rightarrow TOS, SFA \ bit \rightarrow TOS < 0 > \\ (W15) + 2 \rightarrow W15 \\ (PC < 23:16 >) \rightarrow TOS \\ (W15) + 2 \rightarrow W15 \\ 0 \rightarrow SFA \ bit \\ 0 \rightarrow PC < 22:16 > \\ (Wn < 15:1 >) \rightarrow PC < 15:1 > \\ NOP \rightarrow Instruction \ Register \end{array} $						
Status Affected:	SFA							
Encoding:	0000	0001	0000	0000	0000	SSSS		
Description:	Indirect subroutine call over the first 32K instructions of program memory. Before the CALL is made, the 24-bit return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, Wn<15:1> is loaded into PC<15:1> and PC<22:16> is cleared. Since PC<0> is always '0', Wn<0> is ignored.							
	The 's' bits	select the so	urce register.					
Words:	1							
Cycles:	4							

### 16-bit MCU and DSC Programmer's Reference Manual



CALL.L		Call Indirec	t Subroutin	e Long						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E				
			Х			Х				
Syntax:	{label:}	CALL.L	Wn							
Operands:	$Wn \in [W0,$	n ∈ [W0, W2, W4, W6, W8, W10, W12]								
Operation:	$(PC<15:1>)$ $(W15)+2 \rightarrow$ $(PC<23:16:$ $(W15)+2 \rightarrow$ $0 \rightarrow SFA \text{ bit}$ $PC<23> \rightarrow$	C) +2 →PC, C<15:1>) →TOS<15:1>, SFA bit →TOS<0> /15)+2 →W15 C<23:16>) →TOS, /15)+2 →W15								
	PC<15:0> NOP →Inst	ruction Regis	ter							
Status Affected:	SFA	ruotion regio								
Encoding:	0000	0001	lwww	w000	0000	SSSS				
	address (PC+2) and the state of the Stack Frame Active bit (SFA) is pushed onto the system stack, after which the SFA bit is cleared. Then, the LS 7-bits of (Wn+1) are loaded in PC<22:16>, and the 16-bit value (Wn) is loaded into PC<15:0>. PC<23> is not modified by this instruction. The contents of (Wn+1)<15:7> are ignored. The value of Wn<0> is also ignored and PC<0> is always set to 0. The 's' bits specify the address of the Wn source register. The 'w' bits specify the address of the Wn+1 source register.									
Words:	1									
Cycles:	4									
Example 1: 0260 0260		CALL.L W4 MOV W0, 	Wl	; Cal	.l _FIR sub	routine				
0268 0268	44 _FIR: 46	MOV #0x4	400, W2	; _FI	R subrouti	ne start				
	Before Instruction		1	After nstruction						
PC	02 6000	n –	PC	02 6844						
W4	6844	_	W4	6844						
W5	0002	2	W5	0002						
W15	A268	1	W15	A26C						
Data A268	FFFF	D	ata A268	6004						
Data A26A	FFFF	D	ata A26A	0002						
SR	0000	)	SR	0000						

CLR		Clear f or W	REG			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	CLR{.B}	f			
			WREG			
Operands:	f∈ [0 81	91]				
Operation:	$0 \rightarrow destination $	ation designat	ed by D			
Status Affected:	None					
Encoding:	1110	1111	OBDf	ffff	ffff	ffff
	register 'f'	s specified, the is cleared. selects byte c			•	
	The 'D' bit	selects the de select the ade	estination ('0'	for WREG, '		
		The extension rather than a denote a wor	word operati d operation, I	on. You may but it is not re	use a .w ext equired.	
		The WREG is	s set to worki	ng register W	/0.	
Words:	1					
Cycles: Example 1: CI	1 LR.B RAM20	0 ; C	lear RAM200	) (Byte mod	e)	
RAM200 SF		RAM20 SI				
Example 2:	LR WREG	;	Clear WREG	(Word mode	)	
WRE S	Before Instruction G 0600 R 0000	WRE	After Instruction G 0000 R 0000			

CLR		Clear Wd				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	X	Х	Х	Х
Syntax:	{label:}	CLR{.B}	Wd			
			[Wd]			
			[Wd++]			
			[Wd]			
			[++Wd]			
			[Wd]			
Operands:	Wd∈ [W0	W15]				
Operation:	$0 \rightarrow Wd$					
Status Affected:	None					
Encoding:	1110	1011	0Bqq	qddd	d000	0000
	The 'B' bit s The 'q' bits The 'd' bits	select the de	word operati estination Add estination region .B in the	Iress mode. ster.		
		rather than a	word operation,	tion. You ma	yusea.we	
Words:	1		•		•	
Cycles:	1					
Example 1:	CLR.B W2	; Cl	.ear W2 (Byt	ce mode)		
	Before		After			
	Instruction W2 3333					
	W2 3333 SR 0000		V2 3300 SR 0000			
Example 2:	CLR [W0+	-	ear [W0] st-incremen	nt WO		
	Before Instruction W0 2300		After Instruction V0 2302			

CLR		Clear Acc	cumulator, P	refetch Oper	ands		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
				Х	Х	Х	
Syntax: {label:}	CLR	Acc	{,[Wx],Wxd}	{,[Wy]	,Wyd}	{,AWB}	
			$\{,[Wx] + = kx$	,Wxd} {,[Wy]	+ = ky,Wyd		
			${,[Wx] - = kx}$	,Wxd} {,[Wy]	$- = ky,Wyd\}$		
			{,[W9 + W12]	],Wxd} {,[W11	+ W12],Wyd}	}	
Operands:		W9]; kx ∈ ), W11]; ky	[-6, -4, -2, 2, ∈ [-6, -4, -2, + = 2]			]	
Operation:	$\begin{array}{l} 0 \rightarrow Acc(A \text{ or } B) \\ ([Wx]) \rightarrow Wxd; (Wx) +/- kx \rightarrow Wx \\ ([Wy]) \rightarrow Wyd; (Wy) +/- ky \rightarrow Wy \\ (Acc(B \text{ or } A)) \text{ rounded } \rightarrow AWB \end{array}$						
Status Affected:	OA, OB, SA	A, SB					
Encoding:	1100	0011	A0xx	yyii	iijj	jjaa	
Description:	operands ir the non-spe	n preparati ecified acc	e specified acc on for a MAC t umulator resund nd saturate fla	ype instruction	on and option uction clears	ally store the	
	Operands Wx, Wxd, Wy and Wyd specify optional prefetch operations, which support indirect and register offset addressing, as described in <b>Section 4.14.1 "MAC Prefetches"</b> . Operand AWB specifies the optional register direct or indirect store of the convergently rounded contents of the "other" accumulator, as described in <b>Section 4.14.4 "MAC Write Back"</b> .						
	The 'A' bit selects the other accumulator used for write back. The 'x' bits select the prefetch Wxd destination. The 'y' bits select the prefetch Wyd destination. The 'i' bits select the Wx prefetch operation. The 'j' bits select the Wy prefetch operation. The 'a' bits select the accumulator Write Back destination.						
Words:	1						
Cycles:	1						

			W4 with [W8], e ACCB to W13	post-inc W8
	Before Instruction		After Instruction	
W4	F001	W4	1221	
W8	2000	W8	2002	
W13	C623	W13	5420	
ACCA	00 0067 2345	ACCA	00 0000 0000	
ACCB	00 5420 3BDD	ACCB	00 5420 3BDD	
Data 2000	1221	Data 2000	1221	
SR	0000	SR	0000	

CLR A, [W8]+=2, W4, W13 ; Clear ACCA

Example 1:

CLR B, [W8]+=2, W6, [W10]+=2, W7, [W13]+=2 ; Clear ACCB Example 2:

; Load W6 with [W8] ; Load W7 with [W10] ; Save ACCA to [W13]

; Post-inc W8,W10,W13

	Before Instruction		After Instruction
W6	F001	W6	1221
W7	C783	W7	FF80
W8	2000	W8	2002
W10	3000	W10	3002
W13	4000	W13	4002
ACCA	00 0067 2345	ACCA	00 0067 2345
ACCB	00 5420 ABDD	ACCB	00 0000 0000
Data 2000	1221	Data 2000	1221
Data 3000	FF80	Data 3000	FF80
Data 4000	FFC3	Data 4000	0067
SR	0000	SR	0000

•
-

CLRWDT	-	Clear Watch	ndog Timer			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	CLRWDT				
Operands:	None					
Operation:	0 →WDT p	ount register rescaler A co rescaler B co				
Status Affected:	None					
Encoding:	1111	1110	0110	0000	0000	0000
Description:	prescaler c	ontents of the count registers	s. The Watcl	hdog Prescal	er A and Pre	scaler B
Words:	1					
Cycles:	1					
Example 1:	CLRWDT ;	Clear Watc	hdog Timer			
	Before Instruction SR 0000	S	After Instruction R 0000			

# **Section 5. Instruction Descriptions**

СОМ		Complemen	nt f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	COM{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	-	ation designation	ted by D			
Status Affected:	N, Z	C C	·			
Encoding:	1110	1110	1BDf	ffff	ffff	ffff
Description:	the result determines stored in W register. The 'B' bit s The 'D' bit s	ne 1's comple in the desti the destinat /REG. If WRE selects byte o selects the de select the add	nation regis ion register. EG is not spe r word opera estination ('0'	ter. The op If WREG is ecified, the re tion ('0' for w for WREG, '2	tional WRE( specified, th esult is stored	G operand ne result is d in the file vte).
	1	The extensior rather than a denote a word The WREG is	word operation, b	on. You may out it is not re	use a .w exte quired.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-moo details, s	dify-write ope	24E devices, erations on no Section 3.2.	n-CPU Spec	ial Function R le Instructio	Registers. For	
RAM20 S	Before Instruction 0 80FF R 0000	RAM20 S		(Z)		
Example 2: co	OM RAM40	00, WREG	; COM RAM ; (Word m	1400 and sto node)	ore to WREG	

	Before		After					
li li	nstructior	ו ו	nstructio	n				
WREG	1211	WREG	F7DC					
RAM400	0823	RAM400	0823					
SR	0000	SR	0008	(N = 1)				

5

COM		Compleme	nt Ws			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	COM{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws∈ [W0 Wd∈ [W0					
Operation:	(Ws) →Wd					
Status Affected:	N, Z					
Encoding:	1110	1010	1Bqq	qddd	dppp	SSSS
	The 'q' bits The 'd' bits The 'p' bits	select the de select the de select the se	estination Ad estination reg ource Addres ource register	ister. s mode.		
	Note:	The extension rather than a	on .B in the a word opera	instruction de tion. You may	yusea.we	
Words:	1	denote a wo	ra operation,	but it is not re	equirea.	
Cycles:	1 1(1)					
read-mo	odify-write op see <b>Note 3</b> ir	erations on n n Section 3.2	on-CPU Spec	cle count doe cial Function F cle Instructio	Registers. For ns".	' more
i			; Post-in	crement WO,		
W0 W1 Data 2300 Data 2400 SR	Before Instruction 2301 2400 5607 ABCD 0000	۲ Data 23 Data 24				

Example 2: COM	WO, [W1++	-	OM W0 and s ost-increme	tore to [W] ent Wl	l] (Word mc	de)
	Before		After			
In	struction		Instruction			
WO	D004	W	D004			
W1	1000	W1	l 1002			
Data 1000	ABA9	Data 1000				
SR	0000	SF	R 0000			
СР		Compare f v	vith WREG, S	Set Status F	lags	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	CP{.B}	f			
Operands:	f ∈ [0819	91]				
Operation:	(f) – (WRE0	G)				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	0011	OBOf	ffff	ffff	ffff
Description:		) – (WREG) a nt to the SUBW			-	
		selects byte o select the ado	-		ord, '1' for by	yte).
	 (	The extensior rather than a denote a word The WREG is	word operation, to the second se	on. You may out it is not re	use a .w ext quired.	
Words:	1				0.	
	-					
Cycles:	1 <sup>(1)</sup>					
read-mod	dify-write ope	24E devices, erations on no Section 3.2.	n-CPU Speci	al Function R	Registers. For	
Example 1: CP.	.B RAM4(	)0 ; Com <u>r</u>	pare RAM400	with WREG	(Byte mode	)
WREG RAM400 SR <u>Example 2:</u> CP	0823		00 0823 SR 0003	(C = 1) 0) with WRE	G (Word mo	de)
WREG Data 1200 SR	Instruction 2377 2277	WRE Data 12 S	Instruction G 2377 00 2277	(N = 1)		

СР		Compare W	b with lit5, \$	Set Status Fl	ags	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	CP{.B}	Wb,	#lit5		
Operands:	Wb ∈ [W0 lit5 ∈ [ 0 .	-				
Operation:	(Wb) – lit5	-				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1110	0001	0www	wB00	011k	kkkk
Description:	equivalent	Wb) – lit5, and to the SUB ins gister direct a	struction, but	the result of	the subtracti	
	The 'B' bit	s select the ac selects byte c provide the li	or word opera	ation ('0' for w	ord, '1' for b	
	Note:	The extension rather than a denote a wor	word opera	tion. You ma	yuse a.w.e	
Words:	1					
Cycles:	1					
Example 1:	CP.B W4, #0	x12 ;	Compare W4	with 0x12	(Byte mode	)
	Before Instruction W4 7711 SR 0000	W		N = 1)		
Example 2:	CP W4, #0	x12 ;	Compare W4	with 0x12	(Word mode	)
	Before Instruction W4 7713 SR 0000	-	After Instructior V4 7713 SR 0001	(C = 1)		

СР		Compare W	/b with lit8, \$	Set Status Fl	ags	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CP{.B}	Wb,	#lit8		
Operands:	Wb ∈ [W0 lit8 ∈ [ 0 .					
Operation:	(Wb) – lit8					
Status Affected:	DC, N, O\	′, Z, C				
Encoding:	1110	0001	0www	wBkk	k11k	kkkk
Description:	equivalent	Wb) – lit8, and to the SUB ins gister direct a	struction, but	the result of	the subtraction	
	The 'B' bit	s select the ac selects byte c provide the li	or word opera	ation ('0' for w	vord, '1' for b	
	Note:	The extension rather than a denote a wor	word opera	tion. You ma	yuse a .w e	
Words:	1					
Cycles:	1					
Example 1:	CP.B W4, #0	x12 ;	Compare W4	with 0x12	(Byte mode	)
	Before Instruction W4 7711 SR 0000	W		N, C = 1)		
Example 2:	CP W4, #0	x12 ;	Compare W4	with 0x12	(Word mode	)
	Before Instruction W4 7713 SR 0000	-	After Instructior V4 7713 SR 0001	(C = 1)		

Implemented in	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
Implemented in:	X	Х	X	X	X	X
Syntax:	{label:}	CP{.B}	Wb,	Ws		
				[Ws]		
				[Ws++]		
				[Ws]		
				[++Ws]		
				[Ws]		
Operands:	Wb∈ [W0	W15]				
	$Ws \in [W0]$	W15]				
Operation:	(Wb) – (Ws	s)				
Status Affected:	DC, N, OV,	, Z, C	-			
Encoding:	1110	0001	0www	wB00	0ppp	SSSS
	indirect add The 'w' bits The 'B' bit The 'p' bits	dressing may s select the ac selects byte c s select the sc	be used for V ddress of the or word opera ource Address	Wb source re ation ('0' for we	gister. ord, '1' for by	
	Note:	rather than a	a word opera	instruction de ition. You may but it is not re	yusea.we	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-m details	nodify-write op	operations on n in Section 3.2	on-CPU Spe 2.1 "Multi-Cy	/cle count doe cial Function F <b>cle Instructio</b> ] with W0 (1 ent W1	Registers. For ns".	
		, 1				
	D. (		A f.			
	Before		After	n		
	Before Instruction N0 ABA9	I	After Instructio W0 ABA9	-		

Data 2000

SR

D004

0000

Data 2000

SR

D004

0009 (N, C = 1)

Example 2:	CP W5, W6	; Compare W6 with W5 (Word mode)
	Before Instruction	After Instruction
	W5 2334	W5 2334
	W6 8001	W6 8001
	SR 0000	SR 000C (N, OV = 1)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·	Х	Х	Х	Х	х	Х
Syntax:	{label:}	CP0{.B}	f			
Operands:	f ∈ [0 81	91]				
Operation:	(f) – 0x0					
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	0010	OBOf	ffff	ffff	ffff
Description:		) – 0x0 and u is not stored.		ATUS registe	er. The result	of the
		selects byte o select the add			ord, '1' for by	∕te).
		The extensio rather than a denote a wor	word operat	tion. You may	yusea.we	
Words:			•			
vvolus.	1					
Cycles:	1 1(1)					
Note 1: In dsPlo read-mo details,	-	erations on no Section 3.2.	on-CPU Spec 1 "Multi-Cyc	ial Function F	Registers. For ns".	r more
Note 1: In dsPlo read-mo details,	1 <sup>(1)</sup> C33E and PIC odify-write ope see <b>Note 3</b> in	erations on no Section 3.2.	on-CPU Spec 1 "Multi-Cyc	ial Function F Ie Instructio	Registers. For ns".	r more
Cycles: Note 1: In dsPlo read-mo details, Example 1: C	1 <sup>(1)</sup> C33E and PIC odify-write op see <b>Note 3</b> in P0.в кам Before Instruction	erations on no Section 3.2. 100 ; Co	n-CPU Spec 1 "Multi-Cyc mpare RAM10 After Instruction	ial Function F le Instructio	Registers. For ns".	r more
Cycles: <b>Note 1:</b> In dsPl( read-mo details, <u>Example 1:</u> C RAM10	1 <sup>(1)</sup> C33E and PIC odify-write op see <b>Note 3</b> in P0.в кам Before Instruction	erations on no Section 3.2. 100 ; Co RAM1	mpare RAMIC After Instruction 00 44C3	ial Function F le Instructio	Registers. For ns".	r more
Cycles: <b>Note 1:</b> In dsPl0 read-mo details, <u>Example 1:</u> C RAM10	1 <sup>(1)</sup> C33E and PIC odify-write op see <b>Note 3</b> in P0.B RAM Before Instruction 00 44C3 SR 0000	erations on no Section 3.2.	mpare RAM10 After Instruction 00 44C3 SR 0009	ial Function F l <b>e Instructio</b> 00 with 0x0 n	Registers. For ns". (Byte mode	r more

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
Inpress.	X	X	X	X	X	X
Syntax:	{label:}	CP0{.B}	Ws			
	(	u	[Ws]			
			[WS] [Ws++]			
			[Ws]			
			[++Ws]			
			[Ws]			
Operands:	$Ws \in [W0]$	W15]				
Operation:	(Ws) – 0x0	-				
Status Affected:	DC, N, OV,			_		
Encoding:	1110	0000	0000	0800	0ppp	SSSS
Description:		ction is not sto	0 and update ored. Register		register. The	e result of
	The 'p' bits	select the so	or word opera ource Address ddress of the \	s mode.		yte).
		rather than a	on .B in the i a word operat ord operation,	tion. You may	yusea.we	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo	odify-write ope	erations on no	, the listed cyc on-CPU Speci 2.1 "Multi-Cyc	ial Function R	Registers. For	
	PO.B [W4] Before Instruction /4 1001	] ; Comj ; Pos	npare [W4] w. st-decrement After Instruction W4 1000	w4	e mode)	
Example 1: CP W4 Data 1000 SF	Before Instruction /4 1001 00 0034 3R 0000	] ; Comj ; Pos V Data 10	After Instruction W4 1000 000 0034 SR 0001	2 W4 (C = 1)		
Example 1: CP	Before Instruction /4 1001 00 0034 3R 0000	] ; Comj ; Pos V Data 10	After Instruction W4 1000 000 0034	2 W4 (C = 1)		

Instruction Descriptions

 $\ensuremath{\textcircled{}^{\circ}}$  2005-2011 Microchip Technology Inc.

СРВ	Compare f	with WREG	using Borro	w, Sel Statu	is riags	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	CPB{.B}	f			
Operands:	f ∈ [081	91]				
Operation:	(f) – (WRE	G) – ( <del>C</del> )				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1110	0011	1B0f	ffff	ffff	ffff
Description:	Compute (1	f) – (WREG) -	- ( <del>C</del> ), and up	date the STA	TUS register	. This
Description:	instruction subtraction The 'B' bit	f) – (WREG) - is equivalent is not stored selects byte o select the add	to the SUBB i	nstruction, but	ut the result of	of the
Description:	instruction subtraction The 'B' bit The 'f' bits <b>Note 1:</b> 2: 3:	is equivalent is not stored selects byte of select the add The extension rather than a denote a work The WREG is The Z flag is	to the SUBB i r word opera dress of the f n . B in the in word operation, l d operation, l s set to worki 'sticky" for Al	nstruction, buttion ('0' for w ile register. struction den on. You may but it is not re ng register W	ut the result of vord, '1' for b notes a byte of use a .w ext equired. /0.	of the yte). operation tension to
Description:	instruction subtraction The 'B' bit The 'f' bits <b>Note 1:</b> 2: 3:	is equivalent is not stored selects byte of select the add The extension rather than a denote a word The WREG is	to the SUBB i r word opera dress of the f n . B in the in word operation, l d operation, l s set to worki 'sticky" for Al	nstruction, buttion ('0' for w ile register. struction den on. You may but it is not re ng register W	ut the result of vord, '1' for b notes a byte of use a .w ext equired. /0.	of the yte). operation tension to

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1: CPB.B RAM400 ; Compare RAM400 with WREG using C (Byte mode)

I	Before nstructior	n I	After nstructior	ı
WREG	8823	WREG	8823	
RAM400	0823	RAM400	0823	
SR	0000	SR	0008	(N = 1)

Example 2: CPB 0x1200 ; Compare (0x1200) with WREG using C (Word mode)

	Before		After				
I	nstructior	า	Instruction				
WREG	2377	V	WREG	2377			
Data 1200	2377	Data	a 1200	2377			
SR	0001	(C = 1)	SR	0001	(C = 1)		

СРВ	Compare V	Nb with lit5 เ	using Borrov	w, Set Status	Flags	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	CPB{.B}	Wb,	#lit5		
Operands:	Wb ∈ [W0 . lit5 ∈ [ 0					
Operation:	(Wb) – lit5 -	- ( <del>C</del> )				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	0001	lwww	wB00	011k	kkkk
Description:	instruction i	is equivalent	to the SUBB i	e the STATUS nstruction, bu ect addressin	it the result o	of the
	The 'B' bit s	selects byte c	or word opera	Wb source re ition ('0' for wo d, a five bit int	ord, '1' for by	
	1 (	rather than a denote a wor	word operation, to operation and operation of the second s	struction deno on. You may u out it is not re	use a .w exte quired.	ension to
			sticky for AD an only clear	DC, CPB, S Z.	UBB and SUE	3BR. Inese
Words:	1					
Cycles:	1					
Example 1: CPE	3.B W4, #0	x12 ; Co	ompare W4 wi	ith 0x12 us	ing C (Byte	≥ mode)
	Before		After			
W4	Instruction	V	Instruction			
SR				(N = 1)		
Example 2: CPB	.B W4, #0≥	<12 ; Coi	mpare W4 wi	th 0x12 usi	ng C (Byte	mode)
	Before		After			
	Instruction	10				
W4 SR		W S		N = 1)		
Example 3: CPB			· · · · · · · · · · · · · · · · · · ·	vith Ox1F us	ing C (Mor	d mode)
<u>Example 0.</u> CFB		JAIF / COI	_	ICH UXIF US		a mode)
ſ	Before Instruction		After Instruction			
W12		W1				
SR		= 1) S		Z, C = 1)		
Example 4: CPB	W12, #0	0x1F ; Cor	mpare W12 w	ith OxlF us	ing C (Word	d mode)
	Before		After			
	Instruction		Instruction			
W12 SR	0020	W1 C = 1) SI		$\mathbf{C} = 1$		
SK	0003 (Z,	C = 1) SI	R 0001 (	C = 1)		

 $\ensuremath{\textcircled{}^{\circ}}$  2005-2011 Microchip Technology Inc.

5

CPB	-		-	w, Set Status	-	-I- DICOST
Implemented in:	PIC24F	PIC24H	PIC24E X	dsPIC30F	dsPIC33F	dsPIC33E
			^			Х
Syntax:	{label:}	CPB{.B}	Wb,	#lit8		
Operands:	Wb ∈ [W0 lit8 ∈ [ 0					
Operation:	(Wb) – lit8 ·	– ( <del>C</del> )				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	0001	lwww	wBkk	k11k	kkkk
Description:	instruction	s equivalent	to the SUBB i	te the STATUS nstruction, bu ect addressin	it the result c	of the
	The 'B' bit s	selects byte c	or word opera	Wb source re ation ('0' for w d, a five bit int	ord, '1' for by	
	2:	rather than a denote a wor	word operati d operation, l sticky" for AD	struction den on. You may but it is not re DC, CPB, S Z.	use a .w exte quired.	ension to
Words:	1					
Cycles:	1					
Example 1: CP	в.в W4, #C	x12 ; Co	ompare W4 w	ith 0x12 us	ing C (Byte	e mode)
	Before		After			
10/	Instruction	V				
W SI		-	V4 7711 SR 0008	(N = 1)		
	з.в w4, #02			.th 0x12 usi	.ng C (Byte	mode)
	Before		After			
	Instruction		Instruction			
W4		W				
SR	0000	S	R 0008 (	(N = 1)		
Example 3: CPE	3 W12, #(	)x1F ; Co	mpare W12 w	vith 0x1F us	ing C (Wor	d mode)
W12 SR		W1 = 1) S		(Z, C = 1)		
Example 4: CPE	W12, #C	x1F ; Cor	mpare W12 w	ith 0x1F us	ing C (Word	d mode)
W12 SR		W1 C = 1) SI		<b>C</b> = 1)		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	CPB{.B}	Wb,	Ws		
	(**** ,			[Ws]		
				[Ws++]		
				[Ws]		
				[++Ws]		
				[Ws]		
Operands:	Wb ∈ [W0 Ws ∈ [W0					
Operation:	(Wb) – (Ws	s) – ( <del>C</del> )				
Status Affected:	DC, N, OV,	, Z, C				
Encoding:	1110	0001	1www	wB00	0ppp	SSSS
	The 'w' bits The 'B' bit s The 'p' bits	s select the ac selects byte c select the so	ddress of the or word opera ource Address	Wb source reation ('0' for wo s mode. Ws source reg	gister. ord, '1' for byt	e).
	Note 1:	The extension rather than a denote a wor	n . B in the in word operati d operation, I	struction denc on. You may u but it is not rec	otes a byte op use a .w exte quired.	nsion to
		The Z flag is instructions c	-	DDC, CPB, S Z.	SUBB and SUB	3BR. These
Words:	1					
Cycles:	1 <sup>(1)</sup>					
	IC33E and P	IC24E dovico	a tha listad a			
read-m details	nodify-write o <sub>l</sub> s, see <b>Note 3</b>	perations on r in Section 3.3	non-CPU Spe 2.1 "Multi-Cy ompare [W1]	vith W0 usi	Registers. For ons".	
read-m details	nodify-write o , see <b>Note 3</b> PB.B W0, [	perations on r in Section 3.3	non-CPU Spe 2.1 "Multi-Cy ompare [W1] ost-increment	vith W0 usi	Registers. For ons".	r more
read-m details	nodify-write o <sub>l</sub> s, see <b>Note 3</b>	perations on r in Section 3.: [W1++] ; Cc ; Pc	non-CPU Spe 2.1 "Multi-Cy ompare [W1]	ecial Function F ycle Instruction with W0 usint W1	Registers. For ons".	r more
read-m details <u>Example 1:</u> C: V	nodify-write o s, see <b>Note 3</b> PB.B W0, [ Before Instruction V0 ABA9	perations on r in Section 3. [W1++] ; Cc ; Pc	non-CPU Spe 2.1 "Multi-Cy ompare [W1] ost-increment After Instructio W0 ABA9	vith W0 usi	Registers. For ons".	r more
read-m details <u>Example 1:</u> C: V	nodify-write o s, see <b>Note 3</b> PB.B W0, [ Before Instruction V0 ABA9 V1 1000	perations on r in Section 3. [W1++] ; Cc ; Pc	non-CPU Spe 2.1 "Multi-Cy ompare [W1] ost-increment After Instructio W0 ABA9 W1 1001	vith W0 usi nt W1	Registers. For ons".	r more

Example 2:

		; Post-increment W1	
W0 W1 Data 1000	1000 D0A9 Da	After Instruction W0 ABA9 W1 1001 ata 1000 D0A9	
SR	0001 (C = 1)	SR 0001 (C = 1)	
Example 3: CPB	W4, W5	; Compare W5 with W4 using $\overline{ extsf{C}}$ (Word mode	∋)
W4 W5 SR	Before nstruction 4000 3000 0001 (C = 1)	After Instruction W4 4000 W5 3000 SR 0001 (C = 1)	

CPB.B W0, [W1++] ; Compare [W1] with W0 using  $\overline{C}$  (Byte mode) ; Post-increment W1

CPBE	2	Compare V	Wb with Wn,	Branch if Ec	jual (Wb = V	Vn)
Implemented	in: PIC2	24F PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:	} CPBEQ{.B	} Wb,	Wn, Expr		
Operands:		[W0 W15] [W0 W15]				
Operation:	(Wb) - If (Wb	- (Wn) ) = (Wn), [(PC+2)	) + 2 * Expr] -	→PC and NO	⊃ →Instructio	on Register
Status Affecte	d: None					
Encoding:	111	0 0111	1www	wBnn	nnnn	SSSS
	next ir discar specif	ction (Wb) – (Wn astruction (fetche ded, the PC is re ed by Expr, and é (Wn), the next in	d during the c calculated ba on the next cy	current instructions sed on the 6- cle, a NOP is	tion execution bit signed of executed inst	on) is fset stead. If
	The 'E The 's	<ul> <li>i' bits select the a</li> <li>i' bit selects byte</li> <li>i' bits select the a</li> <li>i' bits select the o</li> </ul>	or word opera ddress of the	ation ('0' for v Wn source re	vord, '1' for b egister.	byte).
	Not	rather than	ion . B in the a word opera ord operation,	ation. You ma	iy use a .w	
Words:	1					
Cycles:	1 (5 if	branch taken)				
Example 1:	002002 ADE 002004 002006	E:CPBEQ.B W0, 0 W2, W3, W4; F  ASS:			-	le),
	Befo Instru			After Instruction		
		2000	PC	00 2008		
	WO	1000	WO	1000		
	W1	1000	W1	1000	( A)	
	SR	0000	SR	0002	(z = 1)	

# 5

CPBGT	Signed Co	mpare Wb wit	h Wn, Brand	ch if Greate	r Than (Wb	> Wn)		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
			Х			Х		
Syntax:	{label:}	CPBGT{.B}	Wb,	Wn, Expr				
Operands:	$Wb \in [W0]$ $Wn \in [W0]$							
Operation:	(Wb) – (Wn If (Wb) = (V	) Vn), [(PC+2) +	2 * Expr] →F	C and NOP	→Instructio	n Register		
Status Affected:	None							
Encoding:	1110	0110	0www	wBnn	nnnn	SSSS		
	next instruc discarded, specified by	(Wb) – (Wn), t tion (fetched d the PC is recal / Expr, and on i), the next inst	luring the cur culated base the next cycl	rent instruct d on the 6-b e, a NOP is e	on executio it signed off executed ins	n) is set tead. If		
	The 'B' bit s The 's' bits	The 'w' bits select the address of the Wb source register. The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 's' bits select the address of the Wn source register. The 'n' bits select the offset of the branch destination.						
		The extension rather than a v denote a word	word operation	on. You may	use a .w e			
Words:	1							
Cycles:	1 (5 if brand	ch taken)						
Example 1:	002000 HERE: 002002 002004 002006 002008 BYPASS 00200A	· · ·	, W1, BYPAS , W3, W4		> Wl (Byte m branch t			
	Before Instruction PC 00 200 W0 30F W1 26F SR 000	00 F E	PC W0 W1 SR	After Instruction 00 2008 00FF 26FE 0000	(N, C = 0)			

#### **CPBGT** Signed Compare Wb with Wn, Branch if Greater Than (Wb > Wn)

CFDLI	Signed Co	mpare wo w		ICIT II LESS I		vii)
Implemented	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
			Х			Х
Syntax:	{label:}	CPBLT{.B}	Wb,	Wn, Expr		
Operands:	$Wb \in [W0]$ $Wn \in [W0]$					
Operation:	(Wb) – (Wr If (Wb) = (V	ו) Vn), [(PC+2) -	+ 2 * Expr] →	PC and NOP	<sup>•</sup> →Instructior	n Register
Status Affecte	d: None					
Encoding:	1110	0110	lwww	wBnn	nnnn	SSSS
	next instruc discarded, by Expr, ar the next ins	(Wb) – (Wn), ction (fetched the PC is reca nd on the next struction is ex s select the ad	during the cu lculated base cycle, a NOP ecuted as no	irrent instruct ed on the 6-b is executed rmal (branch	tion execution it signed offset instead. If (W is not taken)	n) is et specifiec /b) ≠ (Wn),
	The 'B' bit : The 's' bits	selects byte o select the ad select the off	r word opera dress of the \	tion ('0' for w Nn source re	ord, '1' for by gister.	vte).
	Note:	The extensio rather than a denote a wor	word operat	ion. You may	yusea.we	
Words:	1					
Cycles:	1 (5 if bran	ch taken)				
Example 1:	002000 HERE: 002002 002004 002006 002008 BYPASS: 00200A	ADD W2, W3 • • • • • •	, W9, BYPAS , W4; Perfo		W9 (Byte m to BYPASS	node),
	Before Instruction PC 00 2000 W8 00FF W9 26FE SR 0000	-	PC W8 W9 SR	After nstruction 00 2008 00FF 26FE 0008 (N	<b>V</b> = 1)	

CPBLT	Signed Compare Wb with Wn, Branch if Less Than (Wb < Wn)
	Signed Compare wb with wh, Branch in Less than (wb < wh)

CPBN	Ē	Compare WI	b with Wn, B	ranch if Not E	Equal (Wb ≠ \	Wn)
Implemented i	n: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CPBNE{.B}	Wb,	Wn, Expr		
Operands:	$Wb \in [W0]$ $Wn \in [W0]$	-				
Operation:	(Wb) – (Wr If (Wb) = (\	n) Nn), [(PC+2) +	- 2 * Expr] →	PC and NOP	→Instruction	Register
Status Affecte	d: None					
Encoding:	1110	0111	0www	wBnn	nnnn	SSSS
	and on the instruction The 'w' bits The 'B' bit The 's' bits	ecalculated ba next cycle, a 1 is executed as select the ad selects byte or select the ado select the ado	NOP is execu s normal (bra dress of the r word opera dress of the \	ted instead. If nch is not take Wb source reg tion ('0' for wo Wn source reg	(Wb) ≠ (Wn) en). gister. rd, '1' for byt ister.	, the next
	Note:	The extension rather than a denote a word	n .в in the word opera	instruction de tion. You may	enotes a byt / use a .w e	-
Words:	1		-			
Cycles:	1 (5 if bran	ch taken)				
Example 1:	002000 HERE: 002002 002004 002006 002008 BYPAS 00200A	ADD W2, W • • •		PASS ; If W2 rform branch	-	te mode),
	Before Instruction PC 00 20 W2 000 W3 260 SR 00	00 FF =E	PC W2 W3 SR	After Instruction 00 200A 00FF 26FE 0001 (	C = 1)	

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
-	Х	Х		Х	Х	
Syntax:	{label:}	CPSEQ{.B}	Wb,	Wn		
Operands:	Wb ∈ [W0 . Wn ∈ [W0 .	-				
Operation:	(Wb) – (Wn Skip if (Wb)	י. ו)				
Status Affected:	• • • •					
Encoding:	1110	0111	lwww	wB00	0000	SSSS
	next instruc discarded a (Wb) ≠ (Wn The 'w' bits The 'B' bit s	ction (fetched and on the ne n), the next in s select the ac selects byte c	), but do not s d during the cu ext cycle, a No nstruction is e ddress of the or word opera	urrent instruc OP is execute executed as no Wb source re ation ('0' for w	ction executioned instead. If normal. register. word, '1' for b	on) is f
		•	ddress of the	•		-
	I	rather than a	on . B in the a word opera ord operation,	ation. You mag	ay use a .w	
Words:	1				•	
Cycles:	1 (2 or 3 if s	skip taken)				
0 0 0 0	002000 HERE:CP 002002GOTOBYPA 002004 002006 002008 BYPASS: 00200A	ASS; skip th  		Wl (Byte mo	ıde),	
	Before			After		
	Instruction	_		Instruction		
	PC 00 2000		PC	00 2002		
V	NO 1001		14/01	1001		
	1000	-	W0			
V	N1 1000 SR 0000	_	W1	1000		
V		_		1000		
V S <u>Example 2:</u> C C		CPSEQ W4	W1	1000 0000		
V S <u>Example 2:</u> C C	SR 0000	CPSEQ W4 CALL _F: 	W1 SR 4, W8; If W4	1000 0000		
V <u>Example 2:</u> c c c	SR 0000 018000 HERE: 018002 018006 018008 Before Instruction	CPSEQ W4 CALL _F: 	W1 SR I, W8; If W4 SIR; skip th	1000 0000 4 = W8 (Word he subroutin After nstruction		
V <u>Example 2:</u> c c c	SR 0000 018000 HERE: 018002 018006 018008 Before Instruction PC 01 8000	CPSEQ W4 CALL _F: 	W1 SR I, W8; If W4 TR; skip th PC	1000 0000 4 = W8 (Word he subroutin After nstruction 01 8006		
V <u>Example 2:</u> c c c c c c c c c c c c c c c c c c c	SR 0000 018000 HERE: 018002 018006 018008 Before Instruction	CPSEQ W4 CALL _F:  	W1 SR I, W8; If W4 SIR; skip th	1000 0000 4 = W8 (Word he subroutin After nstruction		

5

CPSEC	2	Compare W	b with Wn,	Skip if Equa	l (Wb = Wn)	
Implemented	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CPSEQ{.B}	Wb,	Wn		
Operands:	Wb∈ [W0 Wn∈ [W0					
Operation:	(Wb) – (W Skip if (Wb					
Status Affecte	d: None					
Encoding:	1110	0111	lwww	wB00	0001	SSSS
	discarded (Wb) ≠ (W	ction (fetched and on the ne n), the next in a select the ac	xt cycle, a No struction is e	DP is execute xecuted as n	ed instead. If ormal.	on) is
	The 'B' bit	s select the ac selects byte c s select the ad	or word operation	ation ('0' for w	vord, '1' for b	yte).
	Note:	The extension rather than a denote a wor	word opera	tion. You ma	y use a .w	
Words:	1		•		·	
Cycles:	1 (2 or 3 if	skip taken)				
Example 1:	002000 HERE:Cl 002002GOTOBYP2 002004 002006 002008 BYPASS 00200A	ASS; skip th  		Wl (Byte mc	bde),	
	Before           Instruction           PC         00 200           W0         100           W1         100           SR         000	0 1 0	PC   W0   W1   SR	After Instruction 00 2002 1001 1000 0000		

Example 2:	018000 HERE: 018002 018006 018008	CPSEQ CALL 	W4, W8; If _FIR; skip		
	Before Instructio			After Instruction	
	PC 01 80	000	PC	01 8006	
	W4 33	344	W4	3344	
	W8 33	344	W8	3344	
	SR 00	002 (Z = 1)	SR	0002	(Z = 1)

CPSGT	Signed C	ompare Wb wi	th Wn, Skip	if Greater T	han (Wb > \	Nn)
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х		Х	Х	
Syntax:	{label:}	CPSGT{.B}	Wb,	Wn		
Operands:	Wb ∈ [W0 Wn ∈ [W0					
Operation:	(Wb) – (V Skip if (W					
Status Affected:	None					
Encoding:	1110	0110	0www	wB00	0000	SSSS
	next instru discarded the next in The 'w' bi	n (Wb) – (Wn), uction (fetched of and on the nex nstruction is exe ts select the add	during the cu t cycle, a NO: cuted as nor dress of the V	rrent instruct ℙ is executed mal. Wb source re	ion executio d instead. O egister.	n) is therwise,
		t selects byte or s select the add				yte).
	Note:	The extension rather than a denote a word	word operati	on. You may	use a .we	•
Words:	1		•			
Cycles:	1 (2 or 3 i	f skip taken)				
Example 1:	002000 HERE 002002 002006 002008 00200A BYPA 00200C	GOTO I  	WO, W1; If BYPASS; ski			
	Before	9		After		
	Instructi			Instruction		
	PC 00 20		PC W0	00 2006 00EE		
		OFF SFE	W0 W1	00FF 26FE		
		009 (N, C = 1)	SR		(N, C = 1)	
	018000 HERE: 018002 018006 018008		4, W5; If W FIR; skip t			
	Before Instructio	'n	h	After Instruction		
	PC 01 80		PC	01 8002		
,	W4 26	00	W4	2600		
	W5 26		W5	2600		
	SR 00	04 (OV = 1)	SR	0004 (0	OV = 1)	

### CDCCT

Implemented in:	: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CPSGT{.B}	Wb,	Wn		
Operands:	$Wb \in [W0]$ $Wn \in [W0]$					
Operation:	(Wb) – (Wn Skip if (Wb)	n)				
Status Affected:	None	_	_	_		_
Encoding:	1110	0110	0www	wB00	0001	SSSS
	discarded a the next ins The 'w' bits The 'B' bit s	ction (fetched c and on the nex struction is exe s select the add selects byte or select the add	kt cycle, a NO ecuted as nor dress of the N r word operat	o⊵ is executed rmal. Wb source re tion ('0' for wo	d instead. Ot egister. ord, '1' for b <u>y</u>	therwise,
	Note:	The extension rather than a denote a word	п.в in the ir word operati	nstruction de ion. You may	enotes a byte v use a .w e	
Words:	1				-	
Cycles:	1 (2 or 3 if s	skip taken)				
Example 1:	002000 HERE: 002002 002006 002008 002008 BYPASS 00200C	GOTO H  	WO, W1; If BYPASS; ski	W0 > W1 (B; ip the GOTO	-	
	Before			• •		
				After		
	Instructio			Instruction		
	PC 00 200	00	PC	Instruction 00 2006		
	PC 00 200 W0 00F	00 FF	PC W0	Instruction 00 2006 00FF		
	PC 00 200 W0 00F W1 26F	00 FF	PC	Instruction 00 2006 00FF 26FE	(N, C = 1)	
<u>Example 2:</u>	PC 00 200 W0 00F W1 26F	00 =F =E 09 (N, C = 1) CPSGT W	PC W0 W1 SR 4, W5; If W	Instruction 00 2006 00FF 26FE	ord mode),	
Example 2:	PC 00 200 W0 00F W1 26F SR 000 018000 HERE: 018002 018006 018008 Before	00 F E 09 (N, C = 1) CPSGT W CALL _1 	PC W0 W1 SR 4, W5; If W FIR; skip t	Instruction 00 2006 00FF 26FE 0009 W4 > W5 (Wo the subrout After	ord mode),	
<u>Example 2:</u>	PC 00 200 W0 00F W1 26F SR 000 018000 HERE: 018002 018006 018008 Before Instruction	00 =F =E 09 (N, C = 1) CPSGT W CALL _1 	PC W0 W1 SR 4, W5; If W FIR; skip t	Instruction 00 2006 00FF 26FE 0009 W4 > W5 (Wo the subrout After nstruction	ord mode),	
<u>Example 2:</u>	PC 00 200 W0 00F W1 26F SR 000 018000 HERE : 018002 018006 018008 Before Instruction PC 01 8000	00 =F =E 09 (N, C = 1) CPSGT W- CALL _1 	PC W0 W1 SR 4, W5; If W FIR; skip t	Instruction 00 2006 00FF 26FE 0009 W4 > W5 (Wo the subrout After nstruction 01 8002	ord mode),	
	PC 00 200 W0 00F W1 26F SR 000 018000 HERE: 018002 018006 018008 Before Instruction	00 =F =E 09 (N, C = 1) CPSGT W CALL _1  	PC W0 W1 SR 4, W5; If W FIR; skip t	Instruction 00 2006 00FF 26FE 0009 W4 > W5 (Wo the subrout After nstruction	ord mode),	

### **CPSGT**

Signed Compare Wb with Wn, Skip if Greater Than (Wb > Wn)

CPSLT		Signed Co	mpare Wb v	vith Wn, Skip	if Less Tha	n (Wb < Wn)	
Implemented i	n:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
		Х	Х		Х	Х	
Syntax:		{label:}	CPSLT{.B}	Wb,	Wn		
Operands:		$Wb \in [W0]$ $Wn \in [W0]$					
Operation:		(Wb) – (Wr Skip if (Wb					
Status Affecte	d:	None					
Encoding:		1110	0110	lwww	wB00	0000	SSSS
		next instruct discarded a next instruct	ction (fetched and on the ne ction is exect	), but do not s I during the cu ext cycle, a NO uted as norma	urrent instruct P is executed II.	ion execution instead. Oth	n) is
		The 'B' bit s	selects byte	ddress of the or word opera ddress of the <sup>v</sup>	tion ('0' for w	ord, '1' for by	∕te).
			rather than a	on . B in the a word opera rd operation,	tion. You may	yusea.we	•
Words:		1		•			
Cycles:		1 (2 or 3 if :	skip taken)				
Example 1:	0020 0020 0020	06 08 0a bypass:	CPSLT.B GOTO  		W8 < W9 (B ip the GOTO	yte mode),	
		Before			After		
		Instruction			nstruction		
	PC W8	00 2000 00FF		PC W8	00 2002 00FF		
	W9	26FE	_	W9	26FE		
	SR	3000	B (N = 1)	SR	1) 8000	N = 1)	
Example 2:	0180 0180 0180 0180	006		W3, W6; If W _FIR; skip t			
	PC W3 W6	Before Instruction 01 800 260 300	00	PC   W3   W6	After Instruction 01 8006 2600 3000		
	SR	000		SR	0000		

#### **CPSLT** Signed Compare Wb with Wn, Skip if Less Than (Wb < Wn)

Implemented ir	in: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CPSLT{.B}	Wb,	Wn		
Operands:	$Wb \in [W0]$ $Wn \in [W0]$					
Operation:	(Wb) – (Wr Skip if (Wb	,				
Status Affected	d: None	·				_
Encoding:	1110	0110	lwww	wB00	0001	SSSS
	next instruc discarded a next instruc The 'w' bits	n (Wb) – (Wn) action (fetched and on the ne: action is execu is select the ac selects byte c	d during the co ext cycle, a NC uted as norma ddress of the	current instruct OP is executed al. Wb source re	ction execution d instead. Oth register.	on) is herwise, the
		s select the ad				/10/.
	Note:	rather than a	a word opera	e instruction de ation. You may , but it is not re	ayuse a .w e	
Words:	1		· .			
Cycles:	1 (2 or 3 if	skip taken)				
	002000 HERE: 002002 002006 002008 00200A BYPASS: 00200C	CPSLT.B GOTO   		f W8 < W9 (B cip the GOTO	-	
	Before			After		
	Instruction	I		Instruction		
	PC 00 2000		PC	00 2002		
	W8 00FF		W8	00FF		
	W9 26FE SR 0008	E 8 (N = 1)	W9 SR	26FE 0008 (1	NI 1)	
	5K	$\frac{3}{1}$ (IN = $\pm$ )		0000(.	$N = \perp j$	
Example 2:	018000 HERE: 018002 018006 018008			W3 < W6 (Wo the subrout		
	Before			After		
	Instruction	on	- 1	Instruction		
	PC 01 800	20	PC	01 8006	1	
	200				-	
	W3 260		W3	2600	Ţ	
	W3 260 W6 300 SR 000	00	W3 W6 SR	2600 3000 0000		

### **CPSLT** Signed Compare Wb with Wn, Skip if Less Than (Wb < Wn)

Implemented	n: PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
Implemented in	п. РЮ24г Х	У Х	PIC24E	X	X	USPIC33E
		X		X	~	
Syntax:	{label:}	CPSNE{.B}	Wb,	Wn		
Operands:	$Wb \in [W0]$ $Wn \in [W0]$					
Operation:	(Wb) – (Wr Skip if (Wb	,				
Status Affected	d: None					
Encoding:	1110	0111	0www	wB00	0000	SSSS
	and on the instruction The 'w' bits The 'B' bit	next cycle, a is executed a s select the ac selects byte c	NOP is execu s normal. ddress of the <sup>y</sup> or word operat	instruction ex ted instead. C Wb source reg tion ('0' for wo Vn source reg	therwise, the jister. rd, '1' for byte	e next
Words: Cycles:	<b>Note:</b> 1 1 (2 or 3 if	rather than a denote a wor	a word opera	instruction de tion. You may out it is not rec	use a .w e	
Example 1:	002000 HERE: 002002 002006 002008	CPSNE.B GOTO		f W2 != W3 ( kip the GOTC		,
	002008 BYPAS: 00200C	s: 				
	Before Instructio	n		After Instruction		
	PC 00 20		PC	00 2006		
	W2 001 W3 26F		W2 W3	00FF 26FE		
		01 (C = 1)	SR		C = 1)	
Example 2:	018000 HERE: 018002 018006 018008			W0 != W8 (W the subrout		
	Before Instructio	on	-	After Instruction		
	PC 01 80		PC	01 8002		
	W0 30 W8 30		W0 W8	3000 3000		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	CPSNE{.B}	Wb,	Wn		
Operands:	Wb ∈ [W0 . Wn ∈ [W0 .					
Operation:	(Wb) – (Wn Skip if (Wb)	י ו)				
Status Affected:	None					
Encoding:	1110	0111	0www	wB00	0001	SSSS
	and on the instruction i The 'w' bits The 'B' bit s	(fetched during next cycle, a r is executed as select the add selects byte or	NOP is execute s normal. dress of the V r word operati	ted instead. O Wb source reg ion ('0' for wo	Otherwise, the gister. ord, '1' for byte	e next
	The 's' bits	select the add	dress of the W	Vn source reg	gister.	-
	I	The extension rather than a denote a word	word operati	tion. You may	yuse a .w e	
Words:	1				1	
Cycles:	1 (2 or 3 if s	skip taken)				
	02000 HERE: 02002 02006 02008 02008 BYPASS 0200A BYPASS	GOTO I  		E W2 != W3 ( kip the GOTO	-	,
	Before			After		
r	Instruction			Instruction		
Ч	PC 00 200	പ	DO	00 20061		
١٨.			PC W2	00 2006 00FF		
	V2 00F V3 26F	FF	W2 W3	00 2008 00FF 26FE		
W	V2 00F V3 26F	FF	W2	00FF 26FE	(C = 1)	
W S <u>Example 2:</u> 01 01 01	V2 00F V3 26F	FF E 01 (C = 1) CPSNE W0	W2 W3 SR 0, W8 ; If	00FF 26FE	Word mode),	

DAW.B		Decimal Ad	just Wn			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	DAW.B	Wn			
Operands:	Wn∈ [W0	W15]				
Operation:	(Wn<3:0 Else	> > 9) or (DC )>) + 6 →Wn< )>) →Wn<3:0	:3:0>			
	(Wn<7:4 Else	> > 9) or (C = ↓>) + 6 →Wn< ↓>) →Wn<7:4	:7:4>			
Status Affected:	С				_	
Encoding:	1111	1101	0100	0000	0000	SSSS
	Carry flag addressing The 's' bits Note 1: 2:	ult. The Most is used to ind must be use select the so This instruction packed BCD This instruction extension mu	icate any dec d for Wn. urce/destinat on is used to bytes have b on operates i	tion register. correct the d been added. n Byte mode	. Register dir lata format af only and the	ect ter two
Words:	1					
Cycles:	1					
Example 1:	DAW.B WO	; Deci	mal adjust	WO		
	Before Instruction W0 771A SR 0002 (	N N	After Instruction W0 7720 SR 0002	) (DC = 1)		
Example 2:	DAW.B W3	; Deci	mal adjust	W3		
	Before Instruction W3 77AA SR 0000	V	After Instruction V3 7710 GR 0001 (1	C = 1)		

# **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	Decrement	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	х Х	х Х	X X	X	X	X
Syntax:	{label:}	DEC{.B}	f	{,WREG}		1
Operands:	f ∈ [0 81	91]				
Operation:	-	stination desig	gnated by D			
Status Affected:	DC, N, OV,					
Encoding:	1110	1101	OBDf	ffff	ffff	ffff
	destination WREG is no The 'B' bit s The 'D' bit s	register. The register. If W ot specified, the selects byte of selects the de select the add	REG is spec the result is s r word operates tination ('0'	ified, the result tored in the fi tion ('0' for we for WREG, '1	ult is stored i le register. ord, '1' for by	in WREG. I /te).
	r	The extensior rather than a v denote a word	word operation	on. You may u	use a .w exte	
		The WREG is	-		-	
	1	The WREG is	-		-	
Words: Cycles:	1 1(1)		set to workir	ng register W	0.	
Cycles: <b>Note 1:</b> In dsPlo read-mo details, <u>Example 1:</u> I Data 2	1 1 C33E and PIC odify-write ope see <b>Note 3</b> in DEC.B 0x200 Before Instruction	C24E devices, erations on no Section 3.2.	, the listed cycon-CPU Spec on-CPU Spec 1 <b>"Multi-Cyc</b> ; Decrement After Instruction	ng register W cle count doe: ial Function F cle Instructio	0. s not apply to Registers. For <b>ns</b> ".	r more
Cycles: <b>Note 1:</b> In dsPlo read-mo details, <u>Example 1:</u> I Data 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1	C24E devices, erations on no <b>Section 3.2</b> . 0 Data 2	the listed cycon-CPU Spec 1 <b>"Multi-Cyc</b> ; Decremen After Instruction 200 80FE SR 0009	cle count does ial Function F cle Instructio at $(0x200)$ ( n (N, C = 1) RAM400 and	0. s not apply to Registers. For ns". (Byte mode)	r more

PIC24F X {label:}	PIC24H X DEC{.B}	PIC24E X Ws,	dsPIC30F X	dsPIC33F X	dsPIC33 X
			Х	Х	Х
{label:}	DEC{.B}	Ws.			•
		- )	Wd		
		[Ws],	[Wd]		
		[Ws++],	[Wd++]		
		[Ws],	[Wd]		
		[++Ws],	[++Wd]		
		[Ws],	[Wd]		
-	-				
(Ws) – 1 →	Wd				
DC, N, OV,	Z, C				
1110	1001	0Bqq	qddd	dppp	SSSS
The 'B' bit s The 'q' bits The 'd' bits The 'd' bits The 'p' bits	selects byte c select the de select the de select the so	or word opera estination Add estination reg urce Address	ation ('0' for wo dress mode. ister. s mode.	ord, '1' for by	te).
Note:	rather than a	a word opera	ition. You may	/usea.we	
1					
1 <sup>(1)</sup>					
odify-write op	erations on n	on-CPU Spe		Registers. For	
	Wd $\in$ [W0 (Ws) - 1 $\rightarrow$ DC, N, OV, 1110 Subtract or result in the addressing The 'B' bits The 'q' bits The 'q' bits The 'g' bits The 's' bits Note: 1 1(1) C33E and PI0	Subtract one from the corresult in the destination addressing may be used. The 'B' bit selects byte of The 'q' bits select the det The 'q' bits select the det The 'p' bits select the so The 's' bits select the so The 's' bits select the so <b>Note:</b> The extension rather than a denote a wor 1 1(1) C33E and PIC24E devices	$[++Ws],$ $[Ws],$ $Ws \in [W0 \dots W15]$ $Wd \in [W0 \dots W15]$ $(Ws) - 1 \rightarrow Wd$ $DC, N, OV, Z, C$ $1110  1001  0Bqq$ Subtract one from the contents of the result in the destination register Wd. addressing may be used by Ws and 'The 'B' bit selects byte or word operative 'q' bits select the destination Add The 'd' bits select the destination register <b>Note:</b> The extension . B in the rather than a word operation, 1 1(1) C33E and PIC24E devices, the listed cy	$[++Ws], [++Wd]$ $[Ws], [Wd]$ $Ws \in [W0 \dots W15]$ $Wd \in [W0 \dots W15]$ $(Ws) - 1 \rightarrow Wd$ $DC, N, OV, Z, C$ $1110 1001 0Bqq qddd$ Subtract one from the contents of the source registing result in the destination register Wd. Either register addressing may be used by Ws and Wd. The 'B' bit selects byte or word operation ('0' for word The 'q' bits select the destination register. The 'B' bit select the destination register. The 'b' bits select the destination register. The 'b' bits select the source register. The 'b' bits select the source register. The 's' bits select the source register. Note: The extension .B in the instruction der rather than a word operation. You may denote a word operation, but it is not recently a state of the source a state of the state of the state of the select operation. C33E and PIC24E devices, the listed cycle count does a state of the stat	$[++Ws], [++Wd]$ $[Ws], [Wd]$ $Ws \in [W0 W15]$ $Wd \in [W0 W15]$ $(Ws) - 1 \rightarrow Wd$ $DC, N, OV, Z, C$ $\boxed{1110  1001  0Bqq  qddd  dppp}$ Subtract one from the contents of the source register Ws and pl result in the destination register Wd. Either register direct or ind addressing may be used by Ws and Wd. The 'B' bit selects byte or word operation ('0' for word, '1' for by The 'q' bits select the destination register. The 'b' bits select the destination register. The 's' bits select the source register. Note: The extension .B in the instruction denotes a byte rather than a word operation. You may use a .w e denote a word operation, but it is not required.

Data 2400

SR

AB55

0000

Data 2400 ABCD

0000

SR

Example 2: DEC W5, [W6++] ; Decrement W5 and store to [W6] (Word mode) ; Post-increment W6 Before After Instruction Instruction W5 D004 D004 W5 2000 W6 2002 W6 Data 2000 ABA9 Data 2000 D003 SR 0000 SR 0009 (N, C = 1)

5

DEC2		Decrement	f by 2			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	DEC2{.B}	f	{,WREG}		
Operands:	f∈ [0 81	91]				
Operation:	(f) – 2 →de	stination desi	gnated by D			
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1101	1BDf	ffff	ffff	ffff
	destination WREG is n The 'B' bit s The 'D' bit s	register. T register. If W ot specified, t selects byte o selects the de select the ado	REG is spec he result is s r word opera estination ('0'	ified, the rest tored in the fi tion ('0' for w for WREG, '1	ult is stored i le register. ord, '1' for by	n WREG. ⁄te).
Words:		The extensio rather than a denote a wor	word operat	tion. You may	yusea.we	
Cycles:	1 <sup>(1)</sup>					
read-m details,	odify-write op see Note 3 ir	C24E devices, erations on no n Section 3.2.	on-CPU Spec 1 "Multi-Cyc ; Decrement	ial Function F	Registers. For <b>ns</b> ".	r more
Data 20 S	Before Instruction 00 80FF 6R 0000	Data 2		(N, C = 1)		
Example 2:	DEC2 RAM	1400, WREG		z RAM400 by WREG (Word		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	DEC2{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	$(Ws) - 2 \rightarrow$	Wd				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1001	1Bqq	qddd	dppp	SSSS
Description:	result in the		register Wd.	source regis Either registe Wd.		
	The 'q' bits The 'd' bits The 'p' bits	selects byte c select the de select the de select the so select the so	stination Add stination reg urce Addres	ister. s mode.	ord, '1' for b <u>y</u>	yte).
		rather than a	word opera	instruction de tion. You may but it is not re	yusea.we	
Words:	1		•			
Cycles:	1 <sup>(1)</sup>					
read-mo	dify-write ope	erations on no	on-CPU Spec	cle count does ial Function R <b>:le Instructio</b>	legisters. For	

Example 1:	DEC2.B [W7],	[W8];	DEC	[W7]	by	2,	store t	o [W8]	(Byte mode)
			; Po	st-de	cre	men	t W7, W	3	

I	Before nstructior	ı I	After nstructior	١
W7	2301	W7	2300	
W8	2400	W8	23FF	
Data 2300	0107	Data 2300	0107	
Data 2400	ABCD	Data 2400	ABFF	
SR	0000	SR	0008	(N = 1)

### 16-bit MCU and DSC Programmer's Reference Manual

Example 2: DEC2 W5, [W6++] ; DEC W5 by 2, store to [W6] (Word mode) ; Post-increment W6 Before After Instruction Instruction D004 W5 D004 W5 W6 1002 W6 1000 Data 1000 Data 1000 D002 ABA9 SR 0000 SR 0009 (N, C = 1)

DISI		Disable Inte	errupts Tem	porarily		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	DISI	#lit14			
Operands:	lit14 ∈ [ 0	16383]				
Operation:	lit14 →DIS 1 →DISI Disable int	ICNT errupts for (lit	14 + 1) cycle	S		
Status Affected:	None					
Encoding:	1111	1100	00kk	kkkk	kkkk	kkkk
	Note 1: 2:	time critical co This instruction from running. for details. This instruction device is in S	on does not p See the spe on does not p	prevent priori cific device f	ty 7 interrupt amily referer	ice manual
Words:	1					
Cycles:	1					
002	000 HERE: 002 004	DISI #10		le interrup 00 cycles p		-
PC DISICNT INTCON2 SR	000	0 0 0	PC DISICNT INTCON2 SR	After nstruction 00 2002 0100 4000 0000	(DISI = 1)	

DIV.S		Signed Inte	ger Divide			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	DIV.S{W}	Wm, Wn			
		DIV.SD	Wm, Wn			
Operands:		0 W15] for 0, W2, W4 2 W15]			n	
Operation:	Wm →V <u>If (Wm&lt;′</u> 0xFFF <u>Else:</u> 0x0 -	1 <u>5&gt; = 1):</u> FF →W1 →W1 <sup>/</sup> Wn →W0	<u>ault):</u>			
Status Affected:	Wm + 1:					
Encoding:	1101	1000	Ottt	tvvv	vWOO	SSSS
Description:	16-bit by 16 the divisor copied to V the double quotient of is stored in This instruct (with an ite remainder. otherwise. overflow ar '0' and clea algorithm a The 't' bits operation. The 'v' bits The 'W' bit	gned integer 5-bit divide) o is stored in W V0 and sign-e operation, W the divide op W1. ction must be ration count of The N flag w The OV flag w ad cleared oth ared otherwise nd its final va select the mo These bits are select the lease selects the divide of the select the divide of the select the divide of the select of the select of the select the divide of the select the divide of the select of the select the divide of the select of the select of the select the divide of the select of the select of the select of the divide of the select of the se	r Wm + 1:Wr /n. In the defa ixtended thro m + 1:Wm is eration is sto executed 18 of 17) to gene ill be set if the will be set if the will be set if the e. The C flag lue should no st significant e clear for the ast significant ividend size (	n (for a 32-bi ault word ope ugh W1 to pe first copied to red in W0, ar times using erate the corre e remainder i ne divide ope Z flag will be is used to im to be used. word of the correct word of the '0' for 16-bit,	t by 16-bit diversion, Wm i eration, Wm i erform the op o W1:W0. The nd the 16-bit the REPEAT ect quotient a s negative all eration result set if the rem oplement the dividend for t tion.	vide) and s first peration. In ne 16-bit remainder instruction and cleared ed in an nainder is divide he double

quotient and remainder should not be used. For the word operation (DIV.S), only one type of overflow may occur (0x8000/0xFFF = + 32768 or 0x00008000), which allows the OV Status bit to interpret the result.
<ol> <li>Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.</li> </ol>
4: This instruction is interruptible on each instruction cycle
boundary. Words: 1
Cycles: 18 (plus 1 for REPEAT execution)
Example 1: DIV.S W3, W4 ; Divide W3 by W4 ; Store quotient to W0, remainder to W1
Before After
W0         5555         W0         013B           W1         1234         W1         0003
W3 3000 W3 3000
W4 0027 W4 0027
SR 0000 SR 0000
Example 2: REPEAT #17 ; Execute DIV.SD 18 times DIV.SD W0, W12 ; Divide W1:W0 by W12 ; Store quotient to W0, remainder to W1
Before After
W0         2500         W0         FA6B           W1         FF42         W1         EF00
W12 2200 W12 2200

0008 (N = 1)

SR

SR

0000

Implemented in:	PIC24F	PIC24H		Unsigned Integer Divide							
		FIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33					
	Х	Х	Х	Х	Х	Х					
Syntax:	{label:}	DIV.U{W} DIV.UD	Wm, Wn Wm, Wn								
Operands:	-	V0 W15] for V0, W2, W4 '2 W15]	•		n						
Operation:	Wm →V 0x0 →V W1:W0/		<u>ault):</u>								
	Wm + 1 W1:W0/ Remain	<u>e operation (D</u> :Wm →W1:W Wns →W0 der →W1									
Status Affected: Encoding:	N, OV, Z, 0	C	1	1	r	T					
	the divisor and W1 is Wm + 1:W operation i This instru (with an ite remainder. divide ope will be set to impleme The 't' bits operation. The 'v' bits The 'W' bits	16-bit divide) is stored in W cleared to per m is first copie is stored in W ction must be eration count of . The N flag wi ration resulted if the remaind ent the divide a select the mo These bits are s select the least t selects the divide s select the divide	In. In the wor rform the divi ed to W1:W0 0, and the 16 executed 18 of 17) to gene ill always be d in an overfile er is '0' and c algorithm and ost significant e clear for the ast significant ividend size visor register.	d operation, N ide. In the dou- b. The 16-bit of b-bit remainded times using erate the corr cleared. The bw and cleared cleared other d its final value word of the d e word opera t word of the ('0' for 16-bit,	Wm is first co uble operatio quotient of the er is stored in the REPEAT is ect quotient a OV flag will b ed otherwise. wise. The C f ie should not dividend for the tion. dividend. '1' for 32-bit	pied to WC n, e divide W1. instruction and be set if the The Z flag lag is used be used. he double ).					
	Note 1: 2: 3:	The extension (32-bit) divide .w extension Unexpected r represented i operation (DI bit will be set used. Dividing by ze first cycle of e	end rather than to denote a results will oc n 16 bits. Th V. UD). Whe and the quot ero will initiat	an a word div word operati- ccur if the quo is may only o n an overflow tient and rem	idend. You m on, but it is n titent can not ccur for the c occurs, the ainder should	ay use a ot required be double OV Status d not be					
	4:	This instruction	on is interrup	tible on each	instruction c	ycle					
Words:	<b>4</b> :	This instruction boundary.	on is interrup	tible on each	instruction c	ycle					

Example 1:	REPEAT #17 DIV.U W2, W4	; Execute DIV.U 18 times ; Divide W2 by W4 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         5555           W1         1234           W2         8000           W4         0200           SR         0000	After Instruction W0 0040 W1 0000 W2 8000 W4 0200 SR 0002 (Z = 1)
Example 2:	REPEAT #17 DIV.UD W10, W12	<pre>; Execute DIV.UD 18 times ; Divide W11:W10 by W12 ; Store quotient to W0, remainder to W1</pre>
	Before           Instruction           W0         5555           W1         1234           W10         2500           W11         0042           W12         2200           SR         0000	After Instruction W0 01F2 W1 0100 W10 2500 W11 0042 W12 2200 SR 0000

Instruction Descriptions

DIVF		Fractional	Divide			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax:	{label:}	DIVF	Wm, Wn			
Operands:	Wm ∈ [ W Wn ∈ [ W					
Operation:	0x0 →W0 Wm →W1 W1:W0/Wr Remainder					
Status Affected:	N, OV, Z, 0					
Encoding:	1101	1001	Ottt	t000	0000	SSSS
	This instruct (with an iter remainder, otherwise, overflow and '0' and clear algorithm a	f the remaind ction must be ration count of The N flag w The OV flag and cleared oth ared otherwis and its final va	executed 18 of 17) to gene ill be set if th will be set if t nerwise. The e. The C flag alue should n	times using erate the corr e remainder i he divide ope Z flag will be is used to in ot be used.	the REPEAT ect quotient a s negative a eration resulto set if the rer	instruction and nd cleared ed in an nainder is
		select the div select the div				
	2: 3:	For the fracti Wn. If Wm is will occur bea equal to 1.0. the quotient a Dividing by z first cycle of This instructi boundary.	greater than cause the frac When this oc and remainde ero will initiat execution.	or equal to V ctional result curs, the OV er should not e an arithmet	Vn, unexpect will be greate Status bit will be used. tic error trap	ed results er than or be set and during the
Words:	1					
Cycles:	18 (plus 1	for repeat e	xecution)			

Example 1:	REPEAT #17 DIVF W8, W9	; Execute DIVF 18 times ; Divide W8 by W9 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         8000           W1         1234           W8         1000           W9         4000           SR         0000	After         Instruction         W0       2000         W1       0000         W8       1000         W9       4000         SR       0002       (Z = 1)
Example 2:	REPEAT #17 DIVF W8, W9	; Execute DIVF 18 times ; Divide W8 by W9 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         8000           W1         1234           W8         1000           W9         8000           SR         0000	After         Instruction         W0 $F000$ W1 $0000$ W8 $1000$ W9 $8000$ SR $0002$ (Z = 1)
Example 3:	REPEAT #17 DIVF W0, W1	; Execute DIVF 18 times ; Divide W0 by W1 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         8002           W1         8001           SR         0000	After Instruction W0 7FFE W1 8002 SR 0008 (N = 1)

0000

DO		Initialize Hardware Loop Literal					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
				Х	Х		
Syntax:	{label:}	DO	#lit14,	Expr			
Operands:		e an absolute		•		+32767].	
Operation:	$(lit14) \rightarrow DC$ $(PC) + 4 \rightarrow$ $(PC) \rightarrow DOS$ (PC) + (2 *)	Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767]. PUSH DO shadows (DCOUNT, DOEND, DOSTART) (lit14) $\rightarrow$ DCOUNT (PC) + 4 $\rightarrow$ PC (PC) $\rightarrow$ DOSTART (PC) + (2 * Slit16) $\rightarrow$ DOEND Increment DL<2:0> (CORCON<10:8>)					
Status Affected:	DA						
Encoding:	0000	1000	00kk	kkkk	kkkk	kkkk	

nnnn

nnnn

nnnn

nnnn

Description:	tim enc val offs	es. The Do ds at the a ue (lit14) s	overhead hardware DO loop, which is executed (lit14 + 1) O loop begins at the address following the DO instruction, and address 2 * Slit16 instruction words away. The 14-bit count supports a maximum loop count value of 16384, and the 16-bit (Slit16) supports offsets of 32K instruction words in both
	Wh PU nev DL cor	en this ins SHed into w DO loop <2:0> (CC npletes ex	struction executes, DCOUNT, DOSTART and DOEND are first their respective shadow registers, and then updated with the parameters specified by the instruction. The DO level count, DRCON<8:10>), is then incremented. After the DO loop execution, the PUSHed DCOUNT, DOSTART and DOEND restored, and DL<2:0> is decremented.
	The	e 'n' bits a	pecify the loop count. re a signed literal that specifies the number of instructions that m the PC to the last instruction executed in the loop.
			tures, Restrictions:
	1 ne 1.	-	g features and restrictions apply to the DO instruction. loop count of '0' will result in the loop being executed one
	2.	Using a	loop size of -2, -1 or 0 is invalid. Unexpected results may these offsets are used.
	3.	-	y last two instructions of the DO loop cannot be:
			truction which changes program control flow
			or REPEAT instruction
	4.	-	cted results may occur if any of these instructions are used. I trap occurs in the second to last instruction or third to last
	4.	instructio	on of a $DO$ loop, the loop will not function properly. The hard ludes exceptions of priority level 13 through level 15,
	No	ha pr	ne DO instruction is interruptible and supports 1 level of ardware nesting. Nesting up to an additional 5 levels may be rovided in software by the user. See the specific device family ference manual for details.
			ne linker will convert the specified expression into the offset to e used.
Words:	2		
Cycles:	2		
Example 1:	002000 002004 002006	LOOP6:	D0 #5, END6; Initiate D0 loop (6 reps) ADD W1, W2, W3; First instruction in loop 
	002008 00200A 00200C	END6:	SUB W2, W3, W4; Last instruction in loop
		Before	After
		struction	Instruction
	PC	00 2000	PC 00 2004
DCO		0000	4
DOST		FF FFFF	
COR		FF FFFF 0000	4
CURI	SR		
	[		

Example 2:	01C00	0 LOOP12:	DO #0x160, END12;	Init DO loop (353 reps)
	01C00	)4	DEC W1, W2; First	instruction in loop
	01C00	)6		
	01C00		—	the FIR88 subroutine
	01C01		NOP	
			NOP; Last instructi	on in loop
	;	(Required	NOP filler)	
		Before		After
		Before Instruction		After Instruction
	PC		PC [	
DCO		Instruction	PC DCOUNT	Instruction
DCO DOST	PC OUNT	Instruction 01 C000		Instruction 01 C004
DOST	PC OUNT	Instruction 01 C000 0000	DCOUNT	Instruction 01 C004 0160
DOST	PC UNT ART END	Instruction 01 C000 0000 FF FFFF	DCOUNT DOSTART	Instruction 01 C004 0160 01 C004

	Initialize Hardware Loop Literal						
Implemented in:	PIC24	F PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33	
						Х	
Syntax:	{label:}	DO	#lit15,	Expr			
Operands:	lit15 $\in$ [0 32767] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767]						
Operation:	(lit15) → (PC) + 4 (PC) →E (PC) + (2	O shadows (DC DCOUNT →PC )OSTART 2 * Slit16) →DOE nt DL<2:0> (COF	END		Γ)		
Status Affected:	DA						
Encoding:	0000	1000	0kkk	kkkk	kkkk	kkkk	
	0000	0000	nnnn	nnnn	nnnn	nnnn	
	DL<2:0> complete registers The 'k' b The 'n' b	loop parameters bits (CORCON- es execution, the are restored, ar its specify the lo bits are a signed et from the PC to	<8:10>), is the PUSHed DC nd DL<2:0> is op count. literal that spe	en incremente COUNT, DOS decremented ecifies the nut	ed. After the I IART and DC d. mber of instru	DO loop DEND	
	The follo	Features, Restrowing features ar		apply to the	DO instructior		
	occ 3. The • at	ng a loop count of ng a loop size o ur if these offsets very last two ins n instruction white DO or REPEAT ir	of '0' will resul f -2, -1 or 0 s are used. structions of t ch changes p	It in the loop b is invalid. Un he ⊡0 loop ca	eing execute expected res	d one time	
	occ 3. The • a • a	ng a loop size o ur if these offsets very last two ins n instruction which	of '0' will resul f -2, -1 or 0 s are used. structions of th ch changes p nstruction	It in the loop b is invalid. Un he DO loop ca rogram contro	eing execute expected res nnot be: bl flow	d one time sults may	
	3. The • a • a Une 4. If a inst trap	ng a loop size o ur if these offsets very last two ins n instruction whic DO or REPEAT ir	of '0' will resul f -2, -1 or 0 s are used. structions of the ch changes p histruction may occur if a s in the secon pop, the loop ions of priority	It in the loop b is invalid. Un he $DO$ loop ca rogram contro any of these in nd to last inst will not funct / level 13 thro	eing execute expected res nnot be: ol flow enstructions ar fruction or thi ion properly. ugh level 15,	ed one time sults may re used. rd to last The hard inclusive.	

Instruction Descriptions

# 16-bit MCU and DSC Programmer's Reference Manual

Words:	2	
Cycles:	2	
eyelee.	-	
Example 1:	002000 LOOP6: 002004 002006 002008 00200A END6: 00200C	DO #5, END6; Initiate DO loop (6 reps) ADD W1, W2, W3; First instruction in loop  SUB W2, W3, W4; Last instruction in loop 
	Before	After
	Instruction	Instruction
	PC 00 2000	PC 00 2004
DCO	UNT 0000	DCOUNT 0005
DOST	ART FF FFF	DOSTART 00 2004
DO	END FF FFFF	DOEND 00 200A
COR	CON 0000	CORCON 0100 (DL = 1)
	SR 0001	I (C = 1) SR 0201 (DA, C = 1)
<u>Example 2:</u>	01C000 LOOP12: 01C004 01C006 01C008 01C00A 01C00C 01C00C 01C00E 01C012 01C014 END12: ; (Require	<pre>DEC W1, W2; First instruction in loop      CALL _FIR88; Call the FIR88 subroutine NOP</pre>
	Before	After
	Instruction	
	PC 01 C00	0 PC 01 C004
	OUNT 0000	
DOS		
-	DEND FF FFF	
COR	CON 0000	
	SR 0008	8 (N = 1) SR 0208 (DA, N = 1)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC3
·				Х	Х	
Syntax:	{label:}	DO	Wn,	Expr		
Operands:		W15] be an absolute solved by the lir				+327
Operation:	PUSH Sha (Wn<13:0> (PC) + 4 → (PC) →DO (PC) + (2 *	adows (DCOUI >) →DCOUNT →PC	NT, DOEND, END	DOSTART)		
Status Affected:	DA	<u> </u>	. <u>.</u>	<u> </u>		<u>.</u>
	0000	1000	1000	0000	0000	SSS
Encoding:	0000	0000	nnnn	nnnn	nnnn	nnn
	completes registers a	CORCON<8:10 execution, the re restored, ar	e PUSHed DC nd DL<2:0> is	COUNT, DOS s decremented	TART and DC d.	
	registers a The 's' bits		nd DL<2:0> is egister Wn tha	s decremented at contains the	d. e loop count.	
			merar mar sp	00		ictions t
	are offset f	from (PC + 4),	which is the I			
	are offset f <b>Special Fe</b>	irom (PC + 4), eatures, Restr	which is the l	last instruction	n executed in	the loo
	are offset f <b>Special Fe</b> The followi	from (PC + 4),	which is the l rictions: nd restrictions	last instruction s apply to the	n executed in	the loo
	are offset f <b>Special Fe</b> The followi 1. Using time. 2. Using	from (PC + 4), eatures, Restr ing features ar	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is inv	last instruction s apply to the sult in the loo	n executed in DO instructior p being exec	the loo n. cuted or
	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>Using time.</li> <li>Using if these</li> <li>The version</li> </ul>	irom (PC + 4), eatures, Restr ing features ar a loop count an offset of -2	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is inv used. structions of th	last instruction s apply to the sult in the loo valid. Unexpen the DO loop ca	n executed in DO instructior p being exec cted results n annot be:	the loo n. cuted or
	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>1. Using time.</li> <li>2. Using if thes</li> <li>3. The ve</li> <li>• an in</li> </ul>	from (PC + 4), eatures, Restr ing features ar a loop count an offset of -2 e offsets are u ery last two ins	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is inv used. structions of the ch changes p	last instruction s apply to the sult in the loo valid. Unexpen the DO loop ca	n executed in DO instructior p being exec cted results n annot be:	the loop n. cuted on
	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>Using time.</li> <li>Using if these</li> <li>The ve</li> <li>a nin</li> <li>a DO Unexp</li> </ul>	irom (PC + 4), eatures, Restr ing features ar a loop count an offset of -2 e offsets are u ery last two ins nstruction whic o or REPEAT in pected results	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is inv used. structions of the ch changes po- nstruction may occur if t	last instruction s apply to the sult in the loo valid. Unexpen- the DO loop ca program contro these last inst	n executed in DO instructior p being exec cted results n annot be: ol flow tructions are t	the loo n. cuted or nay occr used.
	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>1. Using time.</li> <li>2. Using if thes</li> <li>3. The ve</li> <li>a nin</li> <li>a Do Unexp</li> <li>Note 1:</li> </ul>	irom (PC + 4), eatures, Restr ing features ar a loop count an offset of -2 e offsets are u ery last two ins nstruction whic O or REPEAT in bected results The DO instruct Nesting up to by the user. S details.	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is invused. structions of the ch changes po- nstruction may occur if the ction is interrue an additional Gee the specific	last instruction s apply to the sult in the loo valid. Unexper the DO loop ca program contro these last inst uptible and sup I 5 levels may ic device fami	n executed in DO instruction p being exec cted results n annot be: ol flow tructions are n pports 1 level be provided ily reference n	the loo n. cuted or nay occ used. I of nest in softw manual
	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>1. Using time.</li> <li>2. Using if thes</li> <li>3. The ve</li> <li>a nin</li> <li>a Do Unexp</li> <li>Note 1:</li> </ul>	irom (PC + 4), eatures, Restr ing features ar a loop count an offset of -2 e offsets are u ery last two ins nstruction whic o or REPEAT in pected results The DO instruct Nesting up to by the user. S	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is invused. structions of the ch changes po- nstruction may occur if the ction is interrue an additional Gee the specific	last instruction s apply to the sult in the loo valid. Unexper the DO loop ca program contro these last inst uptible and sup I 5 levels may ic device fami	n executed in DO instruction p being exec cted results n annot be: ol flow tructions are n pports 1 level be provided ily reference n	the loop n. cuted on nay occu used. I of nesti in softw manual
Words:	<ul> <li>are offset f</li> <li>Special Fe</li> <li>The followi</li> <li>1. Using time.</li> <li>2. Using if thes</li> <li>3. The ve</li> <li>a nin</li> <li>a Do Unexp</li> <li>Note 1:</li> </ul>	irom (PC + 4), eatures, Restr ing features ar a loop count an offset of -2 e offsets are u ery last two ins nstruction whic o or REPEAT in bected results in The DO instruct Nesting up to by the user. S details. The linker will	which is the I rictions: nd restrictions of '0' will res 2, -1 or 0 is invused. structions of the ch changes po- nstruction may occur if the ction is interrue an additional Gee the specific	last instruction s apply to the sult in the loo valid. Unexper the DO loop ca program contro these last inst uptible and sup I 5 levels may ic device fami	n executed in DO instruction p being exec cted results n annot be: ol flow tructions are n pports 1 level be provided ily reference n	the loo n. cuted or nay occu used. I of nesti in softw manual

Instruction Descriptions

# 16-bit MCU and DSC Programmer's Reference Manual

000 LOOP6: 004 006 008 00A 00C 00C 00E 010 END6:	DO ADD   REPEAT SUB NOP	W1, W2, W3	<pre>; Initiate D0 loop (W0 reps) ; First instruction in loop ; Last instruction in loop ; (Required NOP filler)</pre>
Before Instruction 00 2000 0012 0000 FF FFFF FF FFFF 0000 0000		PC W0 DCOUNT DOSTART DOEND CORCON SR	After Instruction 00 2004 0012 00 2004 00 2010 0100 (DL = 1) 0080 (DA = 1)
000 LOOPA: 004 006 008 00A 010 ENDA:	DO SWAP   MOV	WO	<pre>; Initiate D0 loop (W7 reps) ; First instruction in loop ; Last instruction in loop</pre>
Before Instruction 00 2000 E00F 0000 FF FFFF FF FFFF 00000		PC W7 DCOUNT DOSTART DOEND CORCON	After Instruction 00 2004 E00F 200F 00 2004 00 2010 0100 (DL = 1) 0080 (DA = 1)
	004 006 008 00A 00C 00E 010 END6 : Before Instruction 00 2000 0012 0000 FF FFFF 0000 0000 000 LOOPA : 004 006 008 000 000 ENDA : Before Instruction 00 2000 000 ENDA : 00 2000 FF FFFF FF FFFF FF FFFF FF FFFF FF FFFF	004         ADD           006            008            000            000            000            000            000            0010         END6 :         NOP           Before Instruction           000         2000           0012         0000           0000         0000           FF FFFF         FF FFFF           FF FFFF         DO           000         LOOPA :         DO           000         LOOPA :         MOV           Before         Instruction         MOV           Before         Instruction         MOV           Before         FF FFFF         FF FFFF           00 2000         E00F         0000           FF FFFF         FF FFFF         FF FFFF           00 2000         FF FFFF         FF FFFF           00000         FF FFFFF	004         ADD         W1, W2, W3           006            008            000         REPEAT           000         SUB           0010         END6 :           0010         END6 :           0010         END6 :           0010         END6 :           0010         PC           0010         COUNT           FF FFFF         DOSTART           DOE         DO           0000         CORCON           0000         SR           000         LOO           000         LOO           000         DO           000         SR           000         SWAP           000         W0           0004         SWAP           0005            0004            0004            0004            0004            0005            010         ENDA:           MOV         W1, [W2++]           Before         MOV           Instruction

Syntax:       {label:}       DO       Wn,       Expr         Operands:       Wn $\in$ [W0 W15] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +327         Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) $\rightarrow$ DCOUNT (PC) + 4 $\rightarrow$ PC (PC) $\rightarrow$ DOSTART (PC) + (2 * Slit16) $\rightarrow$ DOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding: $0000$ 1000 1000 0000 sss 0000 0000 nnnn nnnn nnnn nn	Syntax:       {label.}       DO       Wn, Expr         Operands:       Wn ∈ [W0 W15] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 +3 Operation:         PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) →DOSTART (PC) + (2 * Slit16) →DOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       0000       1000       1000       0000       sexecuted (Wn + 1) The DO loop begins at the address following the DO instruction, and e the address 2 * Slit16 instruction words away. The 16 bits of Wn supp maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND at PUSHed into their respective shadow registers, and then updated wil new DO loop parameters specified by the instruction. The DO level co DL<2:0> (CORCON-8:10>), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 's' bits specify the register Wn that contains the loop count. The 's' bits apecify the register Site is struction executed in the 1 Special Features, Restrictions:         The following features and restrictions apply to the DO instruction.         0. Using a loop count of '0' will result in the loop being executed time.         0. Using a loop count of '0' will result in the loop being executed time.         1. Using a loop count of the DO loop		210045		rdware Loop	r	· 21000E						
Operands:       Wn ∈ [W0 W15] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 +327         Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) →DOSTART (PC) + (2 * Slit16) →DOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       0000 1000 1000 0000 sss 0000 0000 nnnn nnn nnn nnn nnn nnn         Description:       Initiate a no overhead hardware Do loop, which is executed (Wn + 1) tim The Do loop begins at the address following the Do instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum count value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND are ff PUSHed into their respective shadow registers, and then updated with tin new Do loop parameters specified by the instruction. The Do loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 's' bits specify the register Wn that contains the loop count. The 's' bits are a signed literal that specifies the number of instructions t are offset from (PC + 4), which is the last instruction executed on time.         Quesing an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.         3. The very last two instructions of the Do loop cannot be: a nistruction which changes program control flow a DO or REPEAT instruction Unexpected results may occur if these last instructions are used.     <	Operands:       Wn $\in$ [W0 W15] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slitt 6, where Slitt 6 $\in$ [-32768 +3         Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) ->DCOUNT (PC) + 4 ->PC (PC) ->DOSTART (PC) + (2 * Slit16) ->DOEND Increment DL-2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       Initiate a no overhead hardware DO loop, which is executed (Wn + 1) The DO loop begins at the address following the Do instruction, and e the address 2 * Slit16 instruction words away. The 16 bits of Wn supp maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction words away. The 16 bits of Wn supp maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction words away. The 16 bits of Wn supp completes execution, the PUSHed DCOUNT, DOSTART and DOEND ar PUSHed into their respective shadow registers, and then updated wi new DO loop parameters specified by the instruction. The Do level co DL<2:0> (CORCON-8:10>), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 's' bits are a signed literal that specifies the number of instruction are offset from (PC + 4), which is the last instruction executed in the I Special Features, Restrictions: The following features and restrictions apply to the Do instruction.         Using a loop count of 'o' will result in the loop being executed time.       In instruction which changes program control flow         .       a instruction which changes program control flow       a DO or REPEAT instruction Un	Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC3 X					
Expr may be an absolute address, label or expression.         Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 +327         Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) -JDCOUNT (PC) + 4 - PC (PC) - JOSTART (PC) + (2 * Slit16) -JDOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       0000 1000 1000 0000 0000 sss 0000 0000 nnnn nnn nnn nnn nnn         Description:       Initiate a no overhead hardware Do loop, which is executed (Wn + 1) tim The Do loop begins at the address following the Do instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum court value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND are find PUSHed into their respective shadow registers, and then updated with the new Do loop parameters specified by the instruction. The Do loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 'n' bits are a signed literal that specifies the number of instructions the are offset from (PC + 4), which is the last instruction executed in the loop Special Features, Restrictions: The following features and restrictions apply to the Do instruction.         1. Using a loop count of '0' will result in the loop being executed on time.       3. The very last two instructions of the Do loop cannot be: • an instruction which changes program control flow • a Do or REPEAT instruction Unexpected results may occur if these last instructions are used. <td><ul> <li>Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 +3</li> <li>Operation: PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) + 0DSTART (PC) + (2 * Slit16) →DOEND Increment DL&lt;2:0&gt; (CORCON&lt;10:8&gt;)</li> <li>Status Affected: DA</li> <li><u>0000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>1000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>1000</u> <u>1000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>s</u></li> <li><u>10000</u> <u>10000</u> <u>s</u></li> <li><u>10000</u> <u>10000</u> <u>s</u></li> <li><u>100000</u> <u>s</u></li> <li><u>100000</u> <u>10000</u> <u>s</u></li> <li><u>100000</u> <u>s</u></li> <li><u>1000000</u> <u>s</u></li> <li><u>10000000000</u> <u>s</u></li> <li><u>1000000000000000000000000000000000000</u></li></ul></td> <td>Syntax:</td> <td>{label:}</td> <td>DO</td> <td>Wn,</td> <td>Expr</td> <td></td> <td></td>	<ul> <li>Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 +3</li> <li>Operation: PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) + 0DSTART (PC) + (2 * Slit16) →DOEND Increment DL&lt;2:0&gt; (CORCON&lt;10:8&gt;)</li> <li>Status Affected: DA</li> <li><u>0000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>1000</u> <u>1000</u> <u>1000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>1000</u> <u>1000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>0000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>0000</u> <u>s</u></li> <li><u>0000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>00000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>10000</u> <u>s</u></li> <li><u>00000</u> <u>10000</u> <u>s</u></li> <li><u>10000</u> <u>10000</u> <u>s</u></li> <li><u>10000</u> <u>10000</u> <u>s</u></li> <li><u>100000</u> <u>s</u></li> <li><u>100000</u> <u>10000</u> <u>s</u></li> <li><u>100000</u> <u>s</u></li> <li><u>1000000</u> <u>s</u></li> <li><u>10000000000</u> <u>s</u></li> <li><u>1000000000000000000000000000000000000</u></li></ul>	Syntax:	{label:}	DO	Wn,	Expr							
Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) →DOSTART (PC) + (2 * Slit16) →DOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       0000 1000 1000 0000 ass 0000 0000 nnnn nnnn nnnn nnnn         Description:       Initiate a no overhead hardware DO loop, which is executed (Wn + 1) tim The DO loop begins at the address following the DO instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum count value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND are fi PUSHed into their respective shadow registers, and then updated with the new DO loop parameters specified by the instruction. The DO level count DL<2:0> (CORCON-8:10-), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 's' bits are a signed literal that specifies the number of instructions to are offset from (PC + 4), which is the last instruction executed in the loop Special Features, Restrictions: The following features and restrictions apply to the DO instruction.         1.       Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.         3.       The very last two instructions of the DO loop cannot be: • an instruction which changes program control flow • a DO or REPEAT instruction Unexpected results may occur if these last instructions are used.         4.       The first instruction of t	Operation:       PUSH Shadows (DCOUNT, DOEND, DOSTART) (Wn) →DCOUNT (PC) + 4 →PC (PC) →DOSTART (PC) + (2*Slit16) →DOEND Increment DL<2:0> (CORCON<10:8>)         Status Affected:       DA         Encoding:       0000 1000 1000 0000 0000 s 0000 0000 nnnn nnnn nnnn	Operands:	Expr may b	be an absolute				+3276					
Encoding:       0000       1000       1000       0000       sss         Description:       Initiate a no overhead hardware Do loop, which is executed (Wn + 1) tim The Do loop begins at the address following the Do instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum count value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND are fit PUSHed into their respective shadow registers, and then updated with timew Do loop parameters specified by the instruction. The Do level count DL<2:0> (CORCON-8:10>), is then incremented. After the Do loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count. The 'n' bits are a signed literal that specifies the number of instructions t are offset from (PC + 4), which is the last instruction executed in the loop Special Features, Restrictions: The following features and restrictions apply to the Do instruction.         1.       Using an offset of -2, -1 or 0 is invalid. Unexpected results may occu if these offsets are used.         3.       The very last two instructions of the Do loop cannot be: • an instruction which changes program control flow • a DO or REPEAT instruction Unexpected results may occur if these last instructions are used.         4.       The first instruction of the Do loop cannot be a PSV read or Table read Net 1: The Do instruction is interruptible and supports 1 level of nestin Nesting up to an additional 5 levels may be provided in softw by the user. See the specific device family reference manua	Encoding:       0000       1000       1000       0000       sexcuted (Wn + 1)         Description:       Initiate a no overhead hardware DO loop, which is executed (Wn + 1)         The DO loop begins at the address following the DO instruction, and e the address 2 * Slit16 instruction words away. The 16 bits of Wn suppr maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction words in both directions.         When this instruction executes, DCOUNT, DOSTART and DOEND at PUSHed into their respective shadow registers, and then updated wit new DO loop parameters specified by the instruction. The DO level coid DL<2:0> (CORCON-8:10>), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.         The 's' bits specify the register Wn that contains the loop count.       The 's' bits specify the register Wn that contains the loop count.         The 's' bits are a signed literal that specifies the number of instruction are offset from (PC + 4), which is the last instruction executed in the I         Special Features, Restrictions:       The following features and restrictions apply to the DO instruction.         1.       Using an offset of -2, -1 or 0 is invalid. Unexpected results may or if these offsets are used.         3.       The very last two instructions of the DO loop cannot be:         • an instruction which changes program control flow       • a DO or REPEAT instruction         Unexpected results may occur if these last instructions are used.       4.         3.	Operation:	PUSH Sha (Wn) →DC (PC) + 4 → (PC) →DO (PC) + (2 *	adows (DCOUI COUNT →PC START SIit16) →DOE	NT, DOEND, END	DOSTART)							
Encoding:       0000       0000       nnnn       nnn       nnn       nnn       nnn	Encoding:       0000       0000       nnnn       nnne       detter       detter       detter       nnnn	Status Affected:	DA	<u> </u>				<u> </u>					
<ul> <li>Description:</li> <li>Initiate a no overhead hardware D0 loop, which is executed (Wn + 1) tim The D0 loop begins at the address following the D0 instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum count value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.</li> <li>When this instruction executes, DCOUNT, DOSTART and DOEND are ff PUSHed into their respective shadow registers, and then updated with til new D0 loop parameters specified by the instruction. The D0 level count DL&lt;2:0&gt; (CORCON&lt;8:10&gt;), is then incremented. After the D0 loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL&lt;2:0&gt; is decremented.</li> <li>The 's' bits specify the register Wn that contains the loop count. The 's' bits are a signed literal that specifies the number of instructions t are offset from (PC + 4), which is the last instruction executed in the loop Special Features, Restrictions: The following features and restrictions apply to the D0 instruction.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may occu if these offsets are used.</li> <li>The very last two instructions of the D0 loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a D0 or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the D0 loop cannot be: Nesting up to an additional 5 levels may be provided in softw by the user. See the specific device family reference manual details.</li> <li>2: The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>Description:</li> <li>Initiate a no overhead hardware Do loop, which is executed (Wn + 1) The Do loop begins at the address following the Do instruction, and e the address 2 * Slit16 instruction words away. The 16 bits of Wn supp maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction words in both directions.</li> <li>When this instruction executes, DCOUNT, DOSTART and DOEND ar PUSHed into their respective shadow registers, and then updated wit new Do loop parameters specified by the instruction. The Do level coo DL&lt;2:0&gt; (CORCON&lt;8:10&gt;), is then incremented. After the Do loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL&lt;2:0&gt; is decremented.</li> <li>The 's' bits specify the register Wn that contains the loop count. The 'n' bits are a signed literal that specifies the number of instruction are offset from (PC + 4), which is the last instruction executed in the I Special Features, Restrictions: The following features and restrictions apply to the Do instruction.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may o if these offsets are used.</li> <li>The very last two instructions of the Do loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the Do loop cannot be a PSV read or Table r Note 1: The Do instruction is interruptible and supports 1 level of no Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> </ul>		0000	1000	1000	0000	0000	SSSS					
<ul> <li>The DO loop begins at the address following the DO instruction, and ends the address 2 * Slit16 instruction words away. The 16 bits of Wn support maximum count value of 65536, and the 16-bit offset value (Slit16) support offsets of 32K instruction words in both directions.</li> <li>When this instruction executes, DCOUNT, DOSTART and DOEND are ff PUSHed into their respective shadow registers, and then updated with til new DO loop parameters specified by the instruction. The DO level count DL&lt;2:0&gt; (CORCON&lt;8:10&gt;), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL&lt;2:0&gt; is decremented.</li> <li>The 's' bits specify the register Wn that contains the loop count. The 'n' bits are a signed literal that specifies the number of instructions.</li> <li>The following features and restrictions apply to the DO instruction.</li> <li>Using a loop count of '0' will result in the loop being executed on time.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be: <ul> <li>a n instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table read Note 1: The DO instruction is interruptible and supports 1 level of nesting weating up to an additional 5 levels may be provided in softwore by the user. See the specific device family reference manual details.</li> </ul>	<ul> <li>The DO loop begins at the address following the DO instruction, and e the address 2 * Slit16 instruction words away. The 16 bits of Wn supp maximum count value of 65536, and the 16-bit offset value (Slit16) su offsets of 32K instruction executes, DCOUNT, DOSTART and DOEND at PUSHed into their respective shadow registers, and then updated wit new DO loop parameters specified by the instruction. The DO level coo DL&lt;2:0&gt; (CORCON&lt;8:10&gt;), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL&lt;2:0&gt; is decremented.</li> <li>The 's' bits specify the register Wn that contains the loop count. The 'n' bits are a signed literal that specifies the number of instruction. The 'n' bits are a signed literal that specifies the number of instruction.</li> <li>Using a loop count of '0' will result in the loop being executed time.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may of it these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> </ul> </li> <li>The first instruction of the DO loop cannot be: <ul> <li>an instruction of the DO loop cannot be a PSV read or Table r Note 1: The DO instruction is interruptible and supports 1 level of no supports 1 level of n</li></ul></li></ul>	Encoding:	0000	0000	nnnn	nnnn	nnnn	nnnr					
<ul> <li>The 'n' bits are a signed literal that specifies the number of instructions t are offset from (PC + 4), which is the last instruction executed in the loop Special Features, Restrictions: <ul> <li>The following features and restrictions apply to the DO instruction.</li> </ul> </li> <li>Using a loop count of '0' will result in the loop being executed on time.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table read.</li> <li>The DO instruction is interruptible and supports 1 level of nestitin Nesting up to an additional 5 levels may be provided in softword by the user. See the specific device family reference manual details.</li> <li>The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>The 'n' bits are a signed literal that specifies the number of instruction are offset from (PC + 4), which is the last instruction executed in the list offset from (PC + 4), which is the last instruction executed in the list following features, Restrictions:</li> <li>The following features and restrictions apply to the DO instruction.</li> <li>Using a loop count of '0' will result in the loop being executed time.</li> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may or if these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table results instruction is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>The linker will convert the specified expression into the off be used.</li> </ul>		completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> is decremented.										
<ul> <li>The following features and restrictions apply to the D0 instruction.</li> <li>1. Using a loop count of '0' will result in the loop being executed on time.</li> <li>2. Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.</li> <li>3. The very last two instructions of the D0 loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a D0 or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>4. The first instruction of the D0 loop cannot be a PSV read or Table read.</li> <li>Note 1: The D0 instruction is interruptible and supports 1 level of nestine Nesting up to an additional 5 levels may be provided in softwork by the user. See the specific device family reference manual details.</li> <li>2: The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>The following features and restrictions apply to the Do instruction.</li> <li>1. Using a loop count of '0' will result in the loop being executed time.</li> <li>2. Using an offset of -2, -1 or 0 is invalid. Unexpected results may or if these offsets are used.</li> <li>3. The very last two instructions of the Do loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>4. The first instruction of the Do loop cannot be a PSV read or Table results may occur is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul>		The 's' bits specify the register Wh that contains the loop count. The 'n' bits are a signed literal that specifies the number of instructions that are offset from (PC + 4), which is the last instruction executed in the loop.										
<ol> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be:         <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table read.</li> <li>The first instruction of the DO loop cannot be a PSV read or Table read.</li> <li>The DO instruction is interruptible and supports 1 level of nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual details.</li> <li>The linker will convert the specified expression into the offset</li> </ol>	<ol> <li>Using an offset of -2, -1 or 0 is invalid. Unexpected results may of if these offsets are used.</li> <li>The very last two instructions of the DO loop cannot be:         <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table r Note 1: The DO instruction is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>The linker will convert the specified expression into the off be used.</li> </ol>		The following features and restrictions apply to the DO instruction.										
<ol> <li>The very last two instructions of the DO loop cannot be:         <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>The first instruction of the DO loop cannot be a PSV read or Table read Note 1: The DO instruction is interruptible and supports 1 level of nesti Nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual details.</li> <li>The linker will convert the specified expression into the offset</li> </ol>	<ul> <li>3. The very last two instructions of the DO loop cannot be: <ul> <li>an instruction which changes program control flow</li> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if these last instructions are used.</li> </ul> </li> <li>4. The first instruction of the DO loop cannot be a PSV read or Table r <ul> <li>Note 1: The DO instruction is interruptible and supports 1 level of ne Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul></li></ul>		2. Using			valid. Unexpe	cted results m	nay occu					
<ul> <li>a DO or REPEAT instruction Unexpected results may occur if these last instructions are used.</li> <li>The first instruction of the DO loop cannot be a PSV read or Table read Note 1: The DO instruction is interruptible and supports 1 level of nestin Nesting up to an additional 5 levels may be provided in softway by the user. See the specific device family reference manual details.</li> <li>The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>a DO OF REPEAT instruction Unexpected results may occur if these last instructions are used.</li> <li>The first instruction of the DO loop cannot be a PSV read or Table r Note 1: The DO instruction is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>The linker will convert the specified expression into the off be used.</li> </ul>					he DO loop ca	nnot be:						
<ul> <li>Unexpected results may occur if these last instructions are used.</li> <li>4. The first instruction of the DO loop cannot be a PSV read or Table read</li> <li>Note 1: The DO instruction is interruptible and supports 1 level of nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual details.</li> <li>2: The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>Unexpected results may occur if these last instructions are used.</li> <li>4. The first instruction of the DO loop cannot be a PSV read or Table r</li> <li>Note 1: The DO instruction is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul>												
<ul> <li>4. The first instruction of the DO loop cannot be a PSV read or Table read Note 1: The DO instruction is interruptible and supports 1 level of nestine Nesting up to an additional 5 levels may be provided in softwork by the user. See the specific device family reference manual details.</li> <li>2: The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>4. The first instruction of the DO loop cannot be a PSV read or Table r</li> <li>Note 1: The DO instruction is interruptible and supports 1 level of ne Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul>												
<ul> <li>Note 1: The DO instruction is interruptible and supports 1 level of nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual details.</li> <li>2: The linker will convert the specified expression into the offset</li> </ul>	<ul> <li>Note 1: The DO instruction is interruptible and supports 1 level of new Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference many details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul>					·	·						
<ul><li>Nesting up to an additional 5 levels may be provided in softway by the user. See the specific device family reference manual details.</li><li>2: The linker will convert the specified expression into the offset</li></ul>	<ul> <li>Nesting up to an additional 5 levels may be provided in so by the user. See the specific device family reference manu details.</li> <li>2: The linker will convert the specified expression into the off be used.</li> </ul>		Unexp	pected results i	may occur if t								
	be used.		Unexp 4. The fir	bected results instruction o	may occur if t	cannot be a P	SV read or Ta	able read					
			Unexp 4. The fir <b>Note 1:</b>	Dected results in rst instruction of The DO instruct Nesting up to by the user. So	may occur if t of the DO loop ction is interru an additional	cannot be a P uptible and su 5 levels may	SV read or Tapports 1 level be provided	able reac of nestir in softwa					

# 16-bit MCU and DSC Programmer's Reference Manual

DO	In	itialize F	lardware Loo	op Wn
Cycles: 2	2			
Example 1: 0020 0020 0020 0020 0020 0020 0020 002	06 08 0A 0C	DO ADD   REPEAT SUB	W1, W2, W3	; Initiate DO loop (WO reps) ; First instruction in loop
0020	10 END6:	NOP		; Last instruction in loop
PC W0 DCOUNT DOSTART DOEND CORCON SR	Before Instruction 00 2000 0012 0000 FF FFFF FF FFFF FF FFFF 00000 0000		PC W0 DCOUNT DOSTART DOEND CORCON SR	; (Required NOP filler) After Instruction 00 2004 0012 00 2004 00 2004 00 2010 0100 (DL = 1) 0080 (DA = 1)
0020 0020 0020 0020	006 008 00A	DO SWAP  	W7, ENDA W0	; Initiate DO loop (W7 reps) ; First instruction in loop
0020	Before	MOV	W1, [W2++]	; Last instruction in loop After Instruction
PC	00 2000		PC	
W7	E00F		W7	
DCOUNT DOSTART	0000 FF FFFF		DCOUNT DOSTART	
DOEND	FF FFFF		DOEND	
CORCON	0000		CORCON	
SR	0000		SR	0080 (DA = 1)

Implemented i	in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
•	Ì				Х	Х	Х
Syntax:	{label:}	ED	Wm * Wm,	Acc,	[Wx],	[Wy],	Wxd
-,	·					[Wy] + = ky,	
						[Wy] - = ky,	
					[W9 + W12],	[W11 +	
Operands:		$Wx \in [W8,$	∈ [W4 * W4, W9]; kx ∈ [-6 ), W11]; ky ∈	6, -4, -2, 2, 4		* W7]	
Operation:		-	m) →Acc(A o ⁄]) →Wxd →Wx	ır B)			
Status Affecte	d:	OA, OB, OA	AB, SA, SB,	SAB			
Encoding:	I	1111	0 0 mm	Alxx	00ii	iijj	jj11
		sign-extend results of [V Operands V	ded to 40 bits Nx] – [Wy] ar Nx, Wxd and	s and stored re stored in d Wyd speci	l in the specifi Wxd, which n fy the prefetc	f Wm * Wm a ied accumulat may be the sa h operations v s described in	tor. The me as Wm. which
			14.1 "MAC F				ļ
		Section 4.1 The 'm' bits The 'A' bit s The 'x' bits The 'i' bits s	14.1 "MAC F s select the o selects the ad	Prefetches" operand regi ccumulator f refetch differ x prefetch o	ster Wm for t for the result. rence Wxd de peration.	he square.	
Words:		Section 4.1 The 'm' bits The 'A' bit s The 'A' bits The 'i' bits s The 'j' bits s 1	14.1 "MAC F s select the o selects the ac select the pr select the Wa	Prefetches" operand regi ccumulator f refetch differ x prefetch o	ster Wm for t for the result. rence Wxd de peration.	he square.	
Words: Cycles: <u>Example 1:</u>	ed W	Section 4.1 The 'm' bits The 'A' bit s The 'A' bits s The 'i' bits s The 'j' bits s 1 1	14.1 "MAC F s select the o selects the ac select the pr select the Wa	Prefetches" operand regi ccumulator f refetch differ x prefetch of y prefetch of	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [W8]-[ ; Post-i	he square.	Δ.
Cycles:	ed W	Section 4.1 The 'm' bits The 'A' bit s The 'A' bits s The 'i' bits s The 'j' bits s 1 1	14.1 "MAC F         s select the obselects the acceler the preselect the preselect the Waselect the Wase	Prefetches" operand regi ccumulator f refetch differ x prefetch of y prefetch of	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [W8]-[ ; Post-i	he square. estination. e W4 to ACCA [W10] to W4 increment W8 decrement W1	<u>x</u>
Cycles:	W4 🗌	Section 4.1 The 'm' bits The 'A' bit s The 'i' bits s The 'j' bits s 1 1 N4*W4, A, [ Before Instruction 0	n	Prefetches" operand regi ccumulator f refetch differ x prefetch of y prefetch of x10]-=2, w4	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [w8]-[ ; Post-i ; Post-d After Instruct	he square. estination. w10] to W4 increment W8 decrement W1 r tion 0057	<u>x</u>
Cycles: <u>Example 1:</u>	W4 W8	Section 4.1 The 'm' bits The 'A' bits The 'i' bits The 'i' bits The 'j' bits 1 1 N4*W4, A, [] Before Instruction 0 1	14.1 "MAC F         as select the obselects the acceler select the preselect the Wasselect the Wass	Prefetches" operand regi ccumulator f refetch differ x prefetch op y prefetch op v10]-=2, w4 W4 W4	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [w8]-[ ; Post-d ; Post-d After Instruct	the square. estination. e W4 to ACCA [W10] to W4 increment W8 decrement W1 r tion 0057 1102	<u>x</u>
Cycles: <u>Example 1:</u>	W4 W8 W10	Section 4.1 The 'm' bits The 'A' bit s The 'i' bits s The 'j' bits s 1 1 N4*W4, A, [ Before Instruction 0 1 2	n 14.1 "MAC F select the o selects the ac select the pr select the Wy [W8]+=2, [W 099A 1100 2300	Prefetches"         operand reging         ccumulator for         refetch differ         x prefetch of         y prefetch of         v10]-=2, w4         W4         W4         W8         W10	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [w8]-[ ; Post-i ; Post-d After Instruct 4 3	the square. estination. estination. e W4 to ACCA [W10] to W4 increment W8 decrement W1 r tion 0057 1102 22FE	Δ.
Cycles: <u>Example 1:</u> A	W4 W8	Section 4.1 The 'm' bits The 'A' bit s The 'i' bits s The 'j' bits s 1 1 N4*W4, A, [ Before Instruction 0 1 2 00 3D0A 0	n 14.1 "MAC F select the o selects the ac select the pr select the Wy [W8]+=2, [W 099A 1100 2300	Prefetches" operand regi ccumulator f refetch differ x prefetch op y prefetch op v10]-=2, w4 W4 W4	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [W8]-[ ; Post-i ; Post-d After Instruct 4 3 4 0 0 0 000000	the square. estination. estination. e W4 to ACCA [W10] to W4 increment W8 decrement W1 r tion 0057 1102 22FE	Δ.
Cycles: <u>Example 1:</u> A Data	W4 W8 W10 ACCA	Section 4.1 The 'm' bits The 'A' bit s The 'i' bits s The 'j' bits s 1 1 N4*W4, A, [ Before Instruction 0 1 2 00 3D0A 0 0	n 14.1 "MAC F select the of selects the ac select the pr select the Wy [W8]+=2, [W 009A 1100 2300 0000	Prefetches"         operand reging         ccumulator for         refetch differ         x prefetch op         y prefetch op         v10]-=2, w4         W10]-=2, w4         W10]         W10]         ACCA	ster Wm for t for the result. rence Wxd de peration. peration. 4 ; Square ; [w8]-[ ; Post-i ; Post-d After Instruct 4 A 00 0000	he square. estination. estination. w10] to W4 increment W8 lecrement W1 r tion 0057 1102 22FE 5CA4	2

Example 2: ED W5\*W5, B, [W9]+=2, [W11+W12], W5 ; Square W5 to ACCB ; [W9]-[W11+W12] to W5 ; Post-increment W9

Before
Instruction
43C2
1200
2500
8000
00 28E3 F14C
6A7C
2B3D
0000

	After
	Instruction
W5	3F3F
W9	1202
W11	2500
W12	8000
ACCB	00 11EF 1F04
Data 1200	6A7C
Data 2508	2B3D
SR	0000

Implemented i	n:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
					Х	Х	Х
Syntax:	{label:}	EDAC	Wm * Wm,	Acc,	[Wx],	[Wy],	Wxd
					[Wx] + = kx,	[Wy] + = ky,	
					[Wx] - = kx,	[Wy] - = ky,	
					[W9 + W12],	[W11 + W12],	
Operands:		$\begin{array}{l} Acc \in [A,B] \\ Wm * Wm \in \\ Wx \in [W8, V] \\ Wy \in [W10, \\ Wxd \in [W4 \end{array}$	W9]; kx ∈ [-6 W11]; ky ∈	6, -4, -2, 2,		′ * W7]	
Operation:		(Acc(A or B) ([Wx] – [Wy] (Wx) + kx – (Wy) + ky –	]) →Wxd →Wx	(Wm) <i>→</i> Aco	c(A or B)		
Status Affected	d:	OA, OB, OA	B, SA, SB,	SAB			
Encoding:		1111	0 0mm	Alxx	00ii	iijj	jj10
		results of [W Operands W	/x] – [Wy] ar /x, Wxd and rect and reg	e stored in Wyd spec ister offset	Wxd, which i ify the prefet addressing a	ified accumula may be the sa ch operations as described in	me as Wrr which
		The 'm' bits The 'A' bit se			ister Wm for	-	
		The 'x' bits s The 'i' bits s The 'j' bits s	elect the W	efetch diffe x prefetch c	erence Wxd d operation.		
Words:		The 'i' bits s	elect the W	efetch diffe x prefetch c	erence Wxd d operation.		
		The 'i' bits s	elect the W	efetch diffe x prefetch c	erence Wxd d operation.		
Words: Cycles: <u>Example 1:</u>	EDAC	The 'i' bits s The 'j' bits s 1 1	elect the W	refetch diffe x prefetch c y prefetch c	we w		o W4 nt W8
Cycles:	EDAC	The 'i' bits s The 'j' bits s 1 1 w4*w4, A,	elect the Wy elect the Wy	refetch diffe x prefetch c y prefetch c	we w	estination. Gquare W4 an Idd to ACCA W8]-[W10] t Post-increme Post-decrement	o W4 nt W8
Cycles:	EDAC	The 'i' bits s The 'j' bits s 1 1	elect the Wy elect the Wy [W8]+=2,	refetch diffe x prefetch c y prefetch c	we w	Gquare W4 an Idd to ACCA W8]-[W10] t Post-increme Post-decremen	o W4 nt W8
Cycles:	W4 [	The 'i' bits s The 'j' bits s 1 1 w4*w4, A, Before Instructio	elect the Wy elect the Wy [w8]+=2, n 009A	efetch diffe x prefetch c y prefetch c [w10]-=2,	w4 ; s w4 ; s ; w4 ; s ; a ; [ ; F ; F After Instruct	estination. quare W4 an dd to ACCA W8]-[W10] t ost-increme ost-decrement ion 0057	o W4 nt W8
Cycles:	W4 W8	The 'i' bits s The 'j' bits s 1 1 w4*w4, A, Before Instructio	elect the Wy elect the Wy [w8]+=2, n 009A 1100	with the second	w4 ; s w4 ; s i a i a i f i f i f i f i f i f i f i f i f i f	estination. quare W4 an dd to ACCA W8]-[W10] t ost-increme ost-decrement ion 0057 1102	o W4 nt W8
Cycles: <u>Example 1:</u>	W4 W8 W10	The 'i' bits s The 'j' bits s 1 1 w4*w4, A, Before Instructio	elect the Wy elect the Wy [w8]+=2, 009A 1100 2300	w10]-=2 W4 W4 W8 W10	w4 ; s w4 ; s w4 ; s i a ; t After Instruct	estination. Equare W4 an idd to ACCA W8]-[W10] t Post-increment ion 0057 1102 22FE	o W4 nt W8
Cycles: <u>Example 1:</u>	W4 W8	The 'i' bits s The 'j' bits s 1 1 w4*w4, A, Before Instructio 0 2 00 3D0A 3	elect the Wy elect the Wy [w8]+=2, 009A 1100 2300	with the second	w4 ; s peration. peration. ; w4 ; s ; a ; [ ; F ; F After Instruct	estination. Equare W4 an idd to ACCA W8]-[W10] t Post-increment ion 0057 1102 22FE	o W4 nt W8
Cycles: <u>Example 1:</u> A Data	W4 W8 W10 ACCA	The 'i' bits s The 'j' bits s 1 1 w4*w4, A, Before Instructio 0 00 3D0A 3	elect the Wy elect the Wy [w8]+=2, 009A 1100 2300 D0A	W4 W4 W8 W10 ACCA	w4 ; s peration. peration. ; w4 ; s ; a ; [ ; F ; F After Instruct	ion 0057 1102 22FE 99AE	o W4 nt W8

Example 2: EDAC W5\*W5, B, [w9]+=2, [W11+W12], W5

; Square W5 and

; add to ACCB

; [W9]-[W11+W12] to W5

; Post-increment W9

	Before Instruction
W5	43C2
W9	1200
W11	2500
W12	0008
ACCB	00 28E3 F14C
Data 1200	6A7C
Data 2508	2B3D
SR	0000

	After
	Instruction
W5	3F3F
W9	1202
W11	2500
W12	8000
ACCB	00 3AD3 1050
Data 1200	6A7C
Data 2508	2B3D
SR	0000

# **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	EXCH	Wns,	Wnd		
Operands:	Wns $\in$ [Wo Wnd $\in$ [Wo					
Operation:	(Wns) $\leftrightarrow$ (	Wnd)				
Status Affected:	None					
Encoding:	1111	1101	0000	0ddd	d000	SSSS
Description:		the word con g must be use		vorking regist id Wnd.	ers. Register	direct
		s select the ac select the ac		first register. second regist	er.	
	Note:	This instruct	ion only exec	utes in Word	mode.	
Words:	1					
Cycles:	1					
Example 1:	XCH W1, W9	; Exch	ange the co	ntents of W	1 and W9	
Example 1:	Before		Af	ter	1 and W9	
	Before	<u>n</u>	At	iter uction	1 and W9	
v	Before	n F	Af Instru W1	ter	1 and W9	
 V	Before Instruction	n F 3	Af Instru W1	iter uction A3A3	1 and W9	
V V S	Before Instruction /1 55F /9 A3A	n F 3 0	Af Instru W1 W9 SR	iter uction A3A3 55FF		
v v s	Before Instruction /1 55F /9 A3A R 000 xCH w4, w5 Before	n F 3 0 ; Exch	Af Instru- W1 W9 SR ange the co Af	iter uction A3A3 55FF 0000 ntents of W		
V V S <u>Example 2:</u> ≖	Before Instruction /1 55F /9 A3A R 000 XCH w4, w5 Before Instruction	n F 3 0 ; Exch	Af Instru W1 W9 SR ange the cc Af Instru	iter uction A3A3 55FF 0000 ontents of W iter uction		
۷ ۷ <u>Example 2:</u> ۳	Before Instruction /1 55F /9 A3A R 000 xCH w4, w5 Before	n F 3 0 ; Exch D	Af Instru- W1 W9 SR ange the cc Af Instru- W4	iter uction A3A3 55FF 0000 ntents of W		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	FBCL	Ws, [Ws], [Ws++], [Ws], [++Ws],	Wnd		
			[Ws],			
Operands:	Ws ∈ [W0 . Wnd ∈ [W0					
Operation:		) & 0x8000 s) << 1 ift < Max_Shi Temp << 1 hift + 1	ft) && ( (Tem	p & 0x8000) :	== Sign) )	
Status Affected:	С					
Encoding:	1101	1111	0000	0ddd	dppp	SSSS
Description:	negative va	st occurrence alue), starting	from the Mos	st Significant	bit after the	sign bit of
		rking towards result is sign-				
	bit number The next M the Least S allows for tl values up. I		extended to t bit after the is allocated b use of Wd w is not found	16 bits and p sign bit is allo it number -14 vith the SFTAG , a result of -1	laced in Wn ocated bit nu I. This bit or C instruction 5 is returne	d. mber 0 and dering for scaling
	bit number The next M the Least S allows for th values up. I flag is set. V The 'd' bits The 'p' bits	result is sign- ost Significan significant bit i he immediate If a bit change	extended to t bit after the is allocated b use of Wd w is not found ange is foun stination regi urce Address	16 bits and p sign bit is allo it number -14 vith the SFTAG , a result of -1 d, the C flag i ster. s mode.	laced in Wn ocated bit nu I. This bit or C instruction 5 is returne	d. mber 0 and dering for scaling
	bit number The next M the Least S allows for th values up. I flag is set. V The 'd' bits The 'p' bits The 's' bits	result is sign- ost Significan ignificant bit i he immediate If a bit change When a bit ch select the de select the so	extended to t bit after the is allocated b use of Wd w is not found hange is foun stination regi urce Address urce register.	16 bits and p sign bit is allo it number -14 vith the SFTAG , a result of -1 d, the C flag ster.	laced in Wn ocated bit nu I. This bit or c instruction 5 is returned is cleared.	d. mber 0 and dering for scaling
Words:	bit number The next M the Least S allows for th values up. I flag is set. V The 'd' bits The 'p' bits The 's' bits	result is sign- ost Significant significant bit i he immediate If a bit change When a bit ch select the de select the so select the so	extended to t bit after the is allocated b use of Wd w is not found hange is foun stination regi urce Address urce register.	16 bits and p sign bit is allo it number -14 vith the SFTAG , a result of -1 d, the C flag ster.	laced in Wn ocated bit nu I. This bit or c instruction 5 is returned is cleared.	d. mber 0 and dering for scaling

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	FBCL W1, W9	; Find 1st bit change from left in W1 ; and store result to W9
	Before Instruction W1 55FF W9 FFFF SR 0000	After Instruction W1 55FF W9 0000 SR 0000
Example 2:	FBCL W1, W9	; Find 1st bit change from left in W1 ; and store result to W9
	Before Instruction W1 FFFF W9 BBBB SR 0000	After Instruction W1 FFFF W9 FFF1 SR 0001 (C = 1)
Example 3:	FBCL [W1++], W9	; Find 1st bit change from left in [W1] ; and store result to W9 ; Post-increment W1
Data	Before Instruction W1 2000 W9 BBBB 2000 FF0A SR 0000	After Instruction W1 2002 W9 FFF9 Data 2000 FF0A SR 0000

Implemented in	DICO4E				do DI COOL	deDICOOL
Implemented in:	PIC24F X	PIC24H X	PIC24E X	dsPIC30F X	dsPIC33F X	dsPIC33E X
Syntax:	{label:}	FF1L	Ws,	Wnd		
			[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:	Ws ∈ [W0 Wnd ∈ [W0					
Operation:	Temp =	's) iift < Max_Sh Temp << 1	ift) && !(Tem	o & 0x8000) )	1	
	Shift = S If (Shift == 0 →(Wn	Max_Shift)				
	Else Shift →(\					
Status Affected:	С					
Encoding:	1100	1111	1000	0ddd	dppp	SSSS
Description:	Ws and wo	irst occurrenc orking towards	s the Least S	ignificant bit	of the word o	perand.
	and advan of zero ind	ing begins wi ces to the Lea icates a '1' wa C flag is clea	ast Significar as not found,	nt bit (allocate	d number 16	6). A result
	The 'p' bits	select the de select the so select the so	ource Addres	s mode.		
	Note:		-	in Word mod	e only.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

0001 (C = 1)

Example 1: FI	F1L W2, W5		he 1st one from the left in W2 ore result to W5	
W W S	5 BBBB	W: W! SF	/5 000D	
Example 2: FF	1L [W2++], W!	; and s	the 1st one from the left in [W2 store the result to W5 increment W2	2]
	Before		After	
	Instruction	I	Instruction	
W	2 2000	W2	2002	
W	5 BBBB	W5	0000	
Data 2000	0000	Data 2000	0000	

SR

SR

0000

5

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	FF1R	Ws, [Ws],	Wnd		
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:	Ws ∈ [W0 Wnd ∈ [W0	-				
Operation:	Temp = Shift = S If (Shift ==	's) hift < Max_Sh Temp >> 1 Shift + 1 Max_Shift)	ift) && !(Temp	o & 0x1) )		
	0 →(Wn Else Shift →(\					
Status Affected:	С					
Encoding:	1100	1111	0000	0ddd	dppp	SSSS
Description:	Ws and wo	orking towards	s the Most Sig	rting from the gnificant bit of 16 bits and p	f the word op	erand. The
	and advand zero indica	ces to the Mo	st Significant not found, ar	Significant bit bit (allocated nd the C flag	number 16).	A result of
	The 'p' bits	select the de select the so select the so	ource Address	s mode.		
	Note:	This instructi	on operates	in Word mode	e only.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:		Find the 1st one from the right in W1 and store the result to W9
	Before Instruction W1 000A W9 BBBB SR 0000	After Instruction W1 000A W9 0002 SR 0000
Example 2:	FF1R [W1++], W9 Before	; Find the 1st one from the right in [W1] ; and store the result to W9 ; Post-increment W1 After

Before			
nstructior	n l	nstruction	
2000	W1	2002	
BBBB	W9	0010	
8000	Data 2000	8000	
0000	SR	0000	
	nstruction 2000 BBBB 8000	nstruction I 2000 W1 BBBB W9 8000 Data 2000	

Instruction Descriptions

GOTO		Uncondit	ional Jum	)						
Implemented in:	PIC24F	PIC24H	PIC2	4E	dsPIC30F	dsPIC33F	dsPIC33			
	Х	Х	Х		Х	Х	Х			
Syntax:	{label:}	GOTO	Expr							
Operands:	Expr may be label or expression (but not a literal). Expr is resolved by the linker to a lit23, where lit23 $\in$ [0 8388606].									
Operation:	lit23 →PC NOP →Instruction Register									
Status Affected:	None									
Encoding:										
1st word	0000	0100	nnn	n	nnnn	nnnn	nnn0			
2nd word	0000	0000	000	0	0000	0nnn	nnnn			
	instruction. lit23<0> is The 'n' bits	Since the F ignored. form the ta	PC must alv	vays s.	the 23-bit lite reside on an pecified expre	even address	boundary			
		used.	will resolve	110 3	pecilieu expir					
Words:	2									
Cycles:	2 (PIC24F, 4 (PIC24E,			PIC3	3F)					
Example 1:	026000 026004	GOTO MOV	_THERE W0, W1	; Jump to _THERE			RE			
	027844 _THE 027846		#0x400, 1	W2 ; Code execution ; resumes here						
	Befor Instruct PC 02 6 SR 0		After Instruction PC 02 7844 SR 0000							
Example 2:	000100 _code 026000 026004	e:  GOTO 	_code+2			cart of code				
	Before Instructi PC 02 6 SR 0	ion		PC SR	After Instruction 00 0102 0000					

GOTO Unconditional Indirect Jump									
Implemented in	:	PIC24F	PIC24	4H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
		Х	Х			Х		Х	
Syntax:	{	(label:}	GOTO		Wn				
Operands:	١	Wn ∈ [W0 .	W15]						
Operation:	(	0 →PC<22:16> (Wn<15:1>) →PC<15:1> 0 →PC<0> NOP →Instruction Register							
Status Affected	: 1	None							
Encoding:		0000	000	1	0100	0000	0000	SSSS	
	i	nto PC<15 boundary, \	:1>. Sino Wn<0> is	ce the s igno	e PC must al	ways reside	cified in (Wn) on an even a		
Words:		1							
Cycles:	2	2							
		002 MOV W0, W1 ; 					Jump unconditionally to 16-bit value in W4		
	0078- 0078-	44 _THERE 46	: MOV	#0x	400, W2		de executio sumes here	'n	
	W4 PC SR	Before Instruction 784 00 600 000	4 0		W4 PC SR	After Instruction 7844 00 7844 0000			

GOTO Unconditional Indirect Jump								
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
			Х			Х		
Syntax:	{label:}	GOTO	Wn					
Operands:	Wn∈ [W0	W15]						
Operation:	(Wn<15:1> 0 →PC<0>	0 →PC<22:16> (Wn<15:1>) →PC<15:1> 0 →PC<0> NOP →Instruction Register						
Status Affected:	None							
Encoding:	0000	0001	0000	0100	0000	SSSS		
	into PC<15 boundary, V	i:1>. Since th Wn<0> is ign	22:16> and the PC must all ored.	ways reside o				
Words:	1							
Cycles:	4							
0	006000 006002	5002 MOV W0, W1 ; to 16-k 				nconditionally bit value in W4		
	007844 _THERE 007846	: MOV #02	≤400, ₩2		de executio sumes here	n		
	Before Instruction W4 784 PC 00 600 SR 000	14 00	W4 PC SR	After Instruction 7844 00 7844 0000				

GOTO.I	-	Unconditior	nal Indirect	Jump Long		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	GOTO.L	Wn			
Operands:	Wn∈ [W0	, W2, W4, W6	, W8, W10, V	V12]		
Operation:	PC<23> PC<15:0>	₩C<23> (see r	text); (Wn+1	)<6:0> →PC	<22:16>; (Wi	n) $\rightarrow$
Status Affected:	None					
Encoding:	0000	0001	lwww	w100	0000	SSSS
Description:	Unconditio	nal indirect jui	mp to any us	er program n	nemory addre	ess.
	(Wn) is loa PC<23> is The conter The value GOTO is a The 's' bits	bits of (Wn+1) aded into PC< not modified l nts of (Wn+1)- of Wn<0> is a two-cycle ins s select the ad s specify the a	15:0>. by this instru- <15:7> are ig lso ignored a truction. dress of the <sup>1</sup>	ction. nored. and PC<0> is Wn source re	always set t egister.	
Words:	1					
Cycles:	4					
	026000 026004	GOTO.L W4 MOV W0, 	. Wl	; Cal	l _FIR sub	routine
	D26844 _FIR: D26846	MOV #0x4	400, W2	; _FI	R subroutin	ne start
		0 4 2 8 F D	PC W4 W5 W15 Oata A268 ata A26A	After nstruction 02 6844 6844 0002 A26C 6004 0002		
Data A2		F D				

		Increment f				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	INC{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f) + 1 →de	stination desi	gnated by D			
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1100	OBDf	ffff	ffff	ffff
Description:	destination destination WREG is n	o the content register. T register. If W oot specified, t	he optional REG is spec he result is s	WREG op ified, the res tored in the f	erand deter ult is stored i ile register.	mines the n WREG.
	The 'D' bit	selects byte o selects the de select the ado	stination ('0'	for WREG, '2		
		The extension rather than a denote a word	word operation, b	on. You may out it is not re	use a .w ext quired.	
Words:		The WREG is	Set to working	ng register w	0.	
Cycles:	1 1(1)					
read-mo details, s	dify-write ope	24E devices, erations on no Section 3.2.	n-CPU Speci 1 "Multi-Cyc	al Function R	egisters. For ns".	
Data 1000 SR		Data 10	After Instruction 00 8F00 SR 0101	) (DC, C = 1)		
Example 2: INC	C 0x1000,		increment 0: Word mode)	x1000 and s	tore to WRI	EG
WREG Data 1000		WRE Data 100		I		

INC		Increment \	Ns			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	INC{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	$Ws \in [W0]$ $Wd \in [W0]$					
Operation:	(Ws) + 1 –	→Wd				
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	1110	1000	0Bqq	qddd	dppp	SSSS
Description:	destination used for W	ne contents of n register Wd. /s and Wd.	Register dire	ect or indirect	addressing r	may be
	The 'q' bits The 'd' bits The 'p' bits	selects byte c s select the de s select the de s select the so s select the so	estination Add estination reg ource Address	dress mode. jister. s mode.	oru, ד וסו שי	/tej.
	Note:	rather than a	a word opera	instruction de ation. You may but it is not re	yusea.we	
Words:	1					
Cycles:	1(1)					
read-mo details, s	odify-write op	C24E devices, berations on no in Section 3.2	on-CPU Spec	cial Function F cle Instructio crement W2 ent W1 and s	Registers. Foi ons".	r more
W1 W2		,	After Instructio W1 FF7F W2 2001			

Example 2:	INC W1, W2	; Increment W1 and store to W2 ; (Word mode)
	Before Instruction W1 FF7F W2 2000 SR 0000	After Instruction W1 FF7F W2 FF80 SR 0108 (DC, N = 1)

### **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X
Syntax:	{label:}	INC2{.B}	f	{,WREG}		·
Operands:	f ∈ [0 81	91]				
Operation:	(f) + 2 →de	stination desi	gnated by D			
Status Affected:	DC, N, OV,		0			
Encoding:	1110	1100	1BDf	ffff	ffff	ffff
	destination WREG is n The 'B' bit s	register. The register. If W tot specified, t selects byte o selects the de	REG is speci he result is s r word opera	fied, the resu tored in the fi tion ('0' for w	It is stored in le register. ord, '1' for by	n WREG. If /te).
		select the add			. 101 110 103.	51617.
	Note:	The extension rather than a denote a wor	word operat	tion. You may	/usea.we	
			a oporadori,		qui cu.	
Words:	1		a operation,		quirea.	
Cycles: Note 1: In dsPIC	1 <sup>(1)</sup> 33E and PIC	C24E devices,	the listed cyc	cle count doe	s not apply to	
Cycles: Note 1: In dsPIC read-mo details, s	1 <sup>(1)</sup> C33E and PIC odify-write op see <b>Note 3</b> ir c2.B 0x100 Before Instruction	erations on no Section 3.2. 00 ; Incre ; (Byte	the listed cycon-CPU Spec 1 "Multi-Cyco ement 0x100 e mode) After Instruction	Cle count doe: ial Function F cle Instructio 0 by 2	s not apply to Registers. For	
Note 1: In dsPIC read-mo details, s	1 <sup>(1)</sup> C33E and PIC odify-write op see <b>Note 3</b> in c2.B 0x100 Before Instruction 0 8FFF 00000	erations on no Section 3.2. 00 ; Incre ; (Byte Data 10 WREG ; In	the listed cycon-CPU Spec 1 "Multi-Cyco ement 0x100 e mode) After Instruction 00 8F01 SR 0101	cle count doe: ial Function F cle Instructio	s not apply to egisters. For ns".	' more

INC2		Increment V	Vs by 2			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	INC2{.B}	Ws,	Wd		
-			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 . Wd ∈ [W0 .					
Operation:	(Ws) + 2 $\rightarrow$	Wd				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1000	1Bqq	qddd	dppp	SSSS
	for Ws and The 'B' bits The 'q' bits The 'd' bits The 'p' bits	Wd.	r word opera stination Add stination regi urce Address	ster.	-	
		rather than a	word opera	instruction de tion. You may out it is not ree	/usea.we	
Words:	1				1	
Cycles:	1(1)					
read-m details	nodify-write o <sub>l</sub> , see <b>Note 3</b>	<pre>perations on r in Section 3.2 [++W2] ; ; ;</pre>	oon-CPU Spe 2.1 "Multi-Cy Pre-increm	by 2 and sto	Registers. For ns".	
W Data 200	Before Instruction /1 FF7F /2 2000 00 ABCD 3R 0000	Data 2	After Instruction W1 FF7F W2 2001 000 81CD SR 0100		(- 1)	

Example 2:	INC2 W1, W2	; Increment W1 by 2 and store to W2 ; (word mode)
	Before Instruction	After Instruction
	W1 FF7F	W1 FF7F
	W2 2000	W2 FF81
	SR 0000	SR 0108 (DC, N = 1)

IOR		Inclusive O	R f and WRE	EG		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	IOR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f).IOR.(W	REG) <i>→</i> destin	ation design	ated by D		
Status Affected:	N, Z					
Encoding:	1011	0111	OBDf	ffff	ffff	ffff
	The 'B' bit The 'D' bit	selects byte o selects the de select the add	r word opera estination ('0'	tion ('0' for w for WREG, '	vord, '1' for b	
	WREG is r The 'B' bit The 'D' bit	selects the de	he result is s r word opera stination ('0'	stored in the f ation ('0' for w for WREG, '2	ile register. /ord, '1' for b	vte).
		The extension rather than a denote a word The WREG is	word operati d operation, l	on. You may but it is not re	use a .w ext equired.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo details, s	odify-write op		n-CPU Spec 1 "Multi-Cyc	ial Function F	Registers. For ns".	
	Before	; (В	yte mode) After			

WREG Data 1000	Before nstruction 1234 FF00	WREG 1234 Data 1000 FF34
SR	0000 0x1000	SR 0000 , WREG ; IOR (0x1000) to WREG ; (Word mode)
I	Before nstruction	After Instruction
WREG	1234	WREG 1FBF
Data 1000 SR	0FAB 0008	Data 1000         0FAB           (N = 1)         SR         0000

IOR	DIOO (F		R Literal and			
Implemented in:	PIC24F	PIC24H X	PIC24E X	dsPIC30F	dsPIC33F	dsPIC33E X
	X	^	^	X	Х	^
Syntax:	{label:}	IOR{.B}	#lit10,	Wn		
Operands:		255] for byte 1023] for wo W15]				
Operation:	lit10.IOR.(	Wn) →Wn				
Status Affected:	N, Z					
Encoding:	1011	0011	0Bkk	kkkk	kkkk	dddd
Description:	and the co	he logical incluents of the v g register Wn.	vorking regis			
	The 'k' bits	selects byte o s specify the lit s select the ad	teral operand	l		/te).
	Note 1: 2:	The extension rather than a denote a work For byte oper unsigned valu <b>Operands</b> " for Byte mode.	word operation, I d operation, I ations, the lit ue [0:255]. Se	on. You may out it is not re eral must be ee <b>Section 4.</b>	use a .w ext quired. specified as 6 "Using 10	ension to an <b>-bit Litera</b>
Words:	1					
Cycles:	1					
Example 1:	IOR.B #0xAA,		IOR 0xAA to (Byte mode)	W9		
	Before Instruction W9 1234 SR 0000		After Instruction W9 12BE SR 0008	n ] (N = 1)		
Example 2:	IOR #0x2AA,		IOR 0x2AA t (Word mode)	o W4		
	Before Instruction W4 A34D SR 0000		After Instruction N4 A3EF SR 0008	) (N = 1)		

	DIOC :T	DIGG ····				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	Х	Х	Х	Х	Х
Syntax:	{label:}	IOR{.B}	Wb,	#lit5,	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	$Wb \in [W0]$	W15]				
	lit5 ∈ [0 5	-				
On and the set	Wd ∈ [W0	-				
Operation: Status Affected:	(Wb).IOR.li					
	N, Z	0			11.11	,,,,
Encoding:	0111	0www	wBqq	qddd	d11k	kkkk
Description:	register Wt	o and the 5-bi register Wd.	t literal opera Register dire	nd and place	contents of t the result in g must be use e used for Wo	the ed for Wb.
	The 'q' bits The 'd' bits	select the de select the de provide the li The extension	estination Add estination reg iteral operand on .B in the	Iress mode. ster. d, a five-bit in instruction d	vord, '1' for by teger number enotes a byt y use a . w e	e operatior
		denote a wor	rd operation,	but it is not re	equired.	
Words:	1					
Cycles:	1					
Example 1: I	OR.B W1, #	0x5, [W9++]	; Store t		yte mode)	
	Before		After			
W	Instruction /1 AAAA		Instructio	-		
	/9 2000		W1 AAAA W9 2001			
Data 200		Data 2				
	R 0000		SR 0008	(N = 1)		
Example 2:	OR W1, #0x		IOR W1 with Store to W9		mode)	
	Before		After			
W			Instructic W1 0000	n T		
	/1 0000 /9 A34D		W1 0000 W9 0000	-		

register Ws a the destinati Either register The 'w' bits The 'B' bit so The 'q' bits so The 'd' bits so	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	ents of the ba /d. Register o direct addre ddress of the r word opera	dsPIC30F X Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws], direct address ssing may be a base register this on ('0' for w dress mode.	Vb, and place sing must be u sused for Ws er.	the result in used for Wb and Wd.
{label:} $Wb \in [W0$ $Ws \in [W0$ $Wd \in [W0$ (Wb).IOR.(V N, Z 0111 Compute the register Ws a the destination Either register The 'w' bits The 'B' bit so The 'q' bits so The 'q' bits so The 'd' bits so The 'd' bits so	IOR{.B} . W15] . W15] . W15] Vs) →Wd 0www e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	Wb, Wb, wBqq usive OR ope ents of the ba /d. Register of direct addre ddress of the r word opera	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws], direct address ssing may be base register W	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd] [Wd] contents of th Vb, and place sing must be to a used for Ws	ssss ne source the result in used for Wb and Wd.
$Wb \in [W0$ $Ws \in [W0$ $Wd \in [W0$ $(Wb).IOR.(V$ $N, Z$ $0111$ Compute the register Ws a the destination Either register Ws a the destination Either register The 'w' bits a the 'w' bits a the 'y'	. W15] . W15] . W15] Vs) →Wd on register W er direct or in select the ac elects byte o select the des	wBqq isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	[Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws], direct address ssing may be abase register W direct address ssing may be abase register	[Wd] [Wd++] [Wd] [++Wd] [Wd] [Wd] contents of th Vb, and place sing must be to a used for Ws er.	he source the result in used for Wb and Wd.
$Wb \in [W0$ $Ws \in [W0$ $Wd \in [W0$ $(Wb).IOR.(V$ $N, Z$ $0111$ Compute the register Ws a the destination Either register Ws a the destination Either register The 'w' bits a the 'w' bits a the 'y'	. W15] . W15] . W15] Vs) →Wd on register W er direct or in select the ac elects byte o select the des	wBqq isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	[Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws], direct address ssing may be abase register W direct address ssing may be abase register	[Wd++] [Wd] [++Wd] [Wd] contents of th Vb, and place sing must be to used for Ws	he source the result in used for Wb and Wd.
$Ws \in [W0Wd \in [W0(Wb).IOR.(WN, Z0111Compute theregister Ws athe destinatiEither registThe 'w' bitsThe 'w' bitsThe 'B' bit soThe 'q' bits soThe 'q' bits so$	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	[Ws++], [Ws], [++Ws], [Ws], [Ws], direct address ssing may be base register base register base register	[Wd++] [Wd] [++Wd] [Wd] contents of th Vb, and place sing must be to used for Ws	he source the result i used for Wb and Wd.
$Ws \in [W0Wd \in [W0(Wb).IOR.(WN, Z0111Compute theregister Ws athe destinatiEither registThe 'w' bitsThe 'w' bitsThe 'B' bit soThe 'q' bits soThe 'q' bits so$	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	[++Ws], [Ws], [Ws], eration of the ase register W direct address ssing may be base register tition ('0' for w	[++Wd] [Wd] contents of th Vb, and place sing must be used for Ws er.	he source the result i used for Wb and Wd.
$Ws \in [W0Wd \in [W0(Wb).IOR.(WN, Z0111Compute theregister Ws athe destinatiEither registThe 'w' bitsThe 'w' bitsThe 'B' bit soThe 'q' bits soThe 'q' bits so$	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	qddd eration of the ase register W direct address ssing may be base register tition ('0' for w	dppp contents of th Vb, and place sing must be used for Ws er.	he source the result i used for Wb and Wd.
$Ws \in [W0Wd \in [W0(Wb).IOR.(WN, Z0111Compute theregister Ws athe destinatiEither registThe 'w' bitsThe 'w' bitsThe 'B' bit soThe 'q' bits soThe 'q' bits so$	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	qddd eration of the ase register W direct address ssing may be base register tition ('0' for w	dppp contents of th Vb, and place sing must be used for Ws er.	he source the result i used for Wb and Wd.
$Ws \in [W0Wd \in [W0(Wb).IOR.(WN, Z0111Compute theregister Ws athe destinatiEither registThe 'w' bitsThe 'w' bitsThe 'B' bit sThe 'q' bits sThe 'd' bits s$	. W15] . W15] Vs) →Wd owww e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	eration of the ase register W direct address ssing may be base registe tion ('0' for w	contents of th Vb, and place sing must be to used for Ws er.	he source the result i used for Wb and Wd.
N, Z 0111 Compute the register Ws a the destinati Either register The 'w' bits The 'B' bit so The 'q' bits so The 'q' bits so	0www e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	eration of the ase register W direct address ssing may be base registe tion ('0' for w	contents of th Vb, and place sing must be to used for Ws er.	he source the result i used for Wb and Wd.
0111 Compute the register Ws a the destinati Either registe The 'w' bits The 'B' bits The 'G' bits s The 'd' bits s	e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	eration of the ase register W direct address ssing may be base registe tion ('0' for w	contents of th Vb, and place sing must be to used for Ws er.	he source the result i used for Wb and Wd.
Compute the register Ws a the destinati Either register The 'w' bits The 'B' bit so The 'q' bits so The 'q' bits so	e logical inclu and the conte on register W er direct or in select the ac elects byte o select the des	isive OR ope ents of the ba /d. Register of idirect addre ddress of the r word opera	eration of the ase register W direct address ssing may be base registe tion ('0' for w	contents of th Vb, and place sing must be to used for Ws er.	he source the result i used for Wb and Wd.
register Ws a the destinati Either register The 'w' bits The 'B' bit so The 'q' bits so The 'd' bits so	and the conte on register W er direct or in select the ac elects byte o select the des	ents of the ba /d. Register o direct addre ddress of the r word opera	ase register W direct address ssing may be base registe tion ('0' for w	Vb, and place sing must be u sused for Ws er.	the result i used for Wb and Wd.
•	select the des select the sou select the sou	stination regi urce Address	s mode.		
Note: T r	The extensio ather than a	n . B in the word opera	instruction o ation. You ma	ay use a .w	
1		,			
1(1)					
odify-write op see <b>Note 3</b> ir	erations on n n Section 3.2	+] ; IOR ; Stor	W1 and [W5]	Registers. For ons".	or more
Defere		A £1			
Before           Instruction           1         AAAA           5         2000           9         2400           0         1155           0         0000	Data 2	Instructi W1 AAA/ W5 200 W9 240 000 1155	ion A 1 1 5		
	r (1 (1) 33E and Pl( dify-write op see <b>Note 3</b> in R.B W1, [1 Before Instruction AAAA 2000 2400 0 1155	rather than a denote a word 1 1(1) 33E and PIC24E devices dify-write operations on n see <b>Note 3</b> in <b>Section 3.2</b> R.B W1, [W5++], [W9+ Before Instruction AAAA 2000 2400 0 1155 Data 2 0 0000 Data 2	rather than a word operation, denote a word operation, 1 (1) 33E and PIC24E devices, the listed c dify-write operations on non-CPU Spe see Note 3 in Section 3.2.1 "Multi-Cy R.B W1, [W5++], [W9++] ; IOR ; Stor ; Post Before After Instruction Instruction AAAA W1 AAAA 5 2000 W5 200 9 2400 W9 240 0 1155 Data 2000 1154 0 0000 Data 2400 00Fl	rather than a word operation. You madenote a word operation, but it is not readenote a word operation. The set of	rather than a word operation. You may use a .w denote a word operation, but it is not required. 1 1(1) 33E and PIC24E devices, the listed cycle count does not apply dify-write operations on non-CPU Special Function Registers. For see <b>Note 3</b> in <b>Section 3.2.1 "Multi-Cycle Instructions</b> ". R.B W1, [W5++], [W9++] ; IOR W1 and [W5] (Byte mod ; Store result to [W9] ; Post-increment W5 and W9 Before After Instruction Instruction MAAA W1 AAAA 5 2000 W5 2001 9 2400 W9 2401 0 1155 Data 2000 1155 0 0000 Data 2400 00FF

Instruction Descriptions Example 2: IOR W1, W5, W9

	Before Instruction	
W1	AAAA	
W5	5555	
W9	A34D	
SR	0000	

; IOR W1 and W5 (Word mode)

; Store the result to W9

After Instruction				
	Instruction	1		
W1	AAAA			
W5	5555			
W9	FFFF			
SR	0008	(N = 1)		

	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
Implemented in:				Х	Х	Х
Syntax:	{label:}	LAC	Ws,	{#Slit4,}	Acc	
			[Ws],			
			[Ws++],			
			[Ws],			
			[Ws],			
			[++Ws],			
			[Ws+Wb],			
Operands:	Ws ∈ [W0 Wb ∈ [W0 Slit4 ∈ [-8 Acc ∈ [A,E	W15] +7]				
Operation:	Shift <sub>Slit4</sub> (E	xtend(Ws)) →	Acc(A or B)			
Status Affected:	OA, OB, C	AB, SA, SB, S	SAB			
Encoding:	1100	1010	Awww	wrrr	rggg	SSSS
Description:		contents of the	source regist	er, optionally p	perform a sigr	ned 4-hit
	where a ne operand in register is	ore the result egative operan dicates an arit assumed to be ded (through b	n the specifie d indicates ar hmetic right s 1.15 fraction	hift. The data al data and is	ft shift and a p stored in the automatically	nge is -8:7, positive source
	where a ne operand in register is sign-exten shifting. The 'A' bit The 'A' bit The 'r' bits The 'g' bits	egative operan dicates an arit assumed to be	n the specifie d indicates ar hmetic right s e 1.15 fraction it 39) and zer lestination acc ffset register ccumulator pr urce Address	n arithmetic lei hift. The data al data and is o-backfilled (b cumulator. Wb. e-shift. mode.	ft shift and a p stored in the automatically	nge is -8:7, positive source
	where a ne operand in register is sign-exten shifting. The 'A' bit The 'A' bit The 'r' bits The 'g' bits	egative operan dicates an arit assumed to be ded (through b specifies the c s specify the o encode the ac s select the so	n the specifie d indicates ar hmetic right s 1.15 fraction it 39) and zer lestination ac ffset register v ccumulator pr urce Address bource register on moves moulator registe	n arithmetic lei hift. The data al data and is o-backfilled (b cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or	ft shift and a p stored in the automatically bits [15:0]), pri -extension da causes a sat	nge is -8:7, positive source or to
Words:	where a ne operand in register is sign-exten shifting. The 'A' bit The 'A' bit The 'r' bits The 'g' bits The 's' bits	egative operan dicates an arit assumed to be ded (through b specifies the c specify the o encode the are select the so specify the so of the operation upper Accum	n the specifie d indicates ar hmetic right s 1.15 fraction it 39) and zer lestination ac ffset register v ccumulator pr urce Address bource register on moves moulator registe	n arithmetic lei hift. The data al data and is o-backfilled (b cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or	ft shift and a p stored in the automatically bits [15:0]), pri -extension da causes a sat	nge is -8:7, positive source or to

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1: LAC	[W4++], #-3, B	<pre>; Load ACCB with [W4] &lt;&lt; 3 ; Contents of [W4] do not change ; Post increment W4 ; Assume saturation disabled ; (SATB = 0)</pre>
	Before	After
	Instruction	Instruction
W4	2000	W4 2002
ACCB	00 5125 ABCD	ACCB FF 9108 0000
Data 2000	1221	Data 2000 1221
SR	0000	SR 4800 (OB, OAB = 1)
Example 2: LAC	[W2], #7, A	; Pre-decrement W2 ; Load ACCA with [W2] >> 7
		; Contents of [W2] do not change ; Assume saturation disabled ; (SATA = 0)
	Before	; Assume saturation disabled
	Before Instruction	<pre>; Assume saturation disabled ; (SATA = 0)</pre>
W2 [		<pre>; Assume saturation disabled ; (SATA = 0) After</pre>
W2 ACCA	Instruction	<pre>; Assume saturation disabled ; (SATA = 0) After Instruction</pre>
	Instruction 4002	; Assume saturation disabled ; (SATA = 0) After Instruction W2 4000
ACCA	Instruction 4002 00 5125 ABCD	; Assume saturation disabled ; (SATA = 0) After Instruction W2 4000 ACCA FF FF22 1000

### **Section 5. Instruction Descriptions**

LNK		Allocate St	ack Frame			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	LNK	#lit14			
Operands:	lit14 ∈ [0	. 16382]				
Operation:	(W14) →(T (W15) + 2 (W15) →W (W15) + lit	→W15 /14				
Status Affected:	None					
Encoding:	1111	1010	00kk	kkkk	kkkk	kkk0
	the conten updated St the Stack F	ts of the Fran ack Pointer ( Pointer by the	ence. The Sta ne Pointer (W W15) to the F e unsigned 14 ack Frame of	/14) onto the Frame Pointer I-bit literal op	stack, storin and then inc erand. This in	g the crementing
	The 'k' bits	specify the s	size of the Sta	ack Frame.		
	Note:	Since the St lit14 must be	ack Pointer o even.	can only resid	de on a word	d boundary
Words:	1					
Cycles:	1					
Example 1: LNK	#0xA0	; Allocate	a stack fra	ame of 160	bytes	
Example 1: LNK	#0xA0 Befor Instruct	е	a stack fra	ame of 160 After Instructic	_	
Example 1: LNK	Befor	е	a stack fra W14	After Instructio	_	

0000

0000

Data 2000

SR

2000

0000

Data 2000

SR

LNK		Allocate Sta	ack Frame			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	LNK	#lit14			
Operands:	lit14 ∈ [0	. 16382]				
Operation:	(W14) →(T (W15) + 2 (W15) →W 1 →SFA bi (W15) + lit	→Ŵ15 /14 t				
Status Affected:	SFA					
Encoding:	1111	1010	00kk	kkkk	kkkk	kkk0
Description:	subroutine the conten updated St the Stack F	ction allocate calling seque ts of the Fran ack Pointer (\ Pointer by the maximum Sta	ence. The Stance. The Stance. The Pointer (W W15) to the F unsigned 14	ack Frame is /14) onto the frame Pointer -bit literal op	allocated by stack, storin r and then in erand. This i	PUSHing g the crementing
	The 'k' bits	specify the s	ize of the Sta	ack Frame.		
	Note:	Since the St lit14 must be		can only resid	de on a word	d boundary
Words:	1					
Cycles:	1					
Example 1: LNK	#0xA0	; Allocate	a stack fra	ame of 160	bytes	
	Befor	е		After		
	Instruct	ion		Instructio	on	
W14		2000	W14		2002	
W15		2000	W15		20A2	

0000

0000

0000

Data 2000

CORCON

SR

Data 2000

CORCON

SR

2000

0000

LSR		Logical Shif	ft Right f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	LSR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f<0>) → For word o 0 →Dest	<7> →Dest<6:0> ¢C <u>peration:</u> <15> ) →Dest<14:0	>			
	0-	►C				
Status Affected:	N, Z, C					
Encoding:	1101	0101	0BDf	ffff	ffff	ffff
Description:	in the desti shifted into Most Signit The optiona is specified	ontents of the f nation register the Carry bit of ficant bit of the al WREG oper l, the result is ored in the file	r. The Least of the STATU e destination rand determi stored in WF	Significant bit JS register. Ze register. nes the destir	of the file reg ro is shifted i nation register	ister is nto the :. If WREG
	The 'D' bit	selects byte of selects the de select the add	stination ('0'	for WREG, '1		
		The extension rather than a v denote a word The WREG is	word operation, b	on. You may u out it is not rec	se a .w exter juired.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
<ul> <li>Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".</li> <li>Example 1: LSR.B 0x600 ; Logically shift right (0x600) by one ; (Byte mode)</li> </ul>						
	Before		After			
Data 60 S	Instructior	n Data	Instructi	on =		

0000

0000

WREG

SR

Example 2:	LSR 0x600, WRI	GG ; Logically shift right ; Store to WREG ; (Word mode)	(0x600) by one
Da	Before Instruction ta 600 55FF	After Instruction Data 600 55FF	

WREG

SR

2AFF

0001 (C = 1)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
in presses and a	X	X	X	X	X	X
Syntax:	{label:}	LSR{.B}	Ws,	Wd		
Oymax.	נוטא בייין	LΟι,	ws, [Ws],	[Wd]		
			[₩S], [WS++],	[Wd] [Wd++]		
			[WS++], [Ws],	[Wd++] [Wd]		
			[++Ws],	[++Wd]		
			[++vvs], [Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	(Ws<0>) <u>For word (</u> 0 →Wd< (Ws<15) (Ws<0>)	<7> 1>) →Wd<6:0: ) →C <u>operation:</u> <15> :1>) →Wd<14 ) →C				
Status Affected:	0 <b>-≻</b> N, Z, C	<u>→</u> C				
Encoding:	1101	0001	0Bqq	qddd	dppp	SSSS
Description:	the result in shifted into Most Signi may be use	n the destinat o the Carry bit ificant bit of W red for Ws and	tion register V t of the STAT Vd. Either reg d Wd.	ster Ws one b Wd. The Leas FUS register. Z gister direct or ration ('0' for w	it to the right, st Significant Zero is shifted r indirect add	bit of Ws is d into the Iressing
	The 'q' bits The 'd' bits The 'p' bits	selects byte c s select the de s select the de s select the so s select the so	estination Ad estination recount	ddress mode. gister. ss mode.	WIU, ⊥ ιοι ο	yte <i>j</i> .
	Note:	rather than a	a word opera	e instruction de ation. You may	nyusea.we	
Words:	1					

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:

LSR.B W0, W1 ; LSR W0 (Byte mode) ; Store result to W1

Before		
Instruction		
W0	FF03	
W1	2378	
SR	0000	

After					
Instruction					
W0	FF03				
W1	2301				
SR	0001	(C = 1)			

Example 2:

LSR W0, W1 ; LSR W0 (Word mode) ; Store the result to W1

Before Instruction			
W0	8000		
W1	2378		
SR	0000		

	After
In	struction
W0	8000
W1	4000
SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X
Syntax:	{label:}	LSR	Wb,	#lit4,	Wnd	
Operands:	Wb ∈ [W0 lit4 ∈ [0 Wnd ∈ [W	15]				
Operation:		→Shift_Val 15:15-Shift_Va nift_Val> →Wn		/al:0>		
Status Affected:	N, Z					
Encoding:	1101	1110	0www	wddd	d100	kkkk
Description:	unsigned l addressing	ift right the co iteral and stor g must be use	e the result ir d for Wb and	h the destinat Wnd.	ion register V	
	The 'd' bit	s select the ac s select the de s provide the l	estination reg	ister.	er.	
	Note:	This instructi	on operates	in Word mod	le only.	
Words:	1					
Cycles:	1					
Example 1:	LSR W4, #1		LSR W4 by 1 Store resul			
	Before Instruction W4 C800 W5 1200 SR 0000	V	After Instruction V4 C800 V5 0003 GR 0000	1		
Example 2:	LSR W4, #1		LSR W4 by 1 Store resu			
	Before Instruction W4 0505 W5 F000 SR 0000		After Instruction V4 0505 V5 0282			

LSR			Logical Shi	ft Right by V	Vns		
Implemented in:		PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
		Х	Х	Х	Х	Х	Х
Syntax:		{label:}	LSR	Wb,	Wns,	Wnd	
Operands:		Wb ∈ [W0 Wns ∈ [W0 Wnd ∈ [W0	0W15]				
Operation:		0 →Wnd<1	→Shift_Val 15:15-Shift_V ift_Val> →Wr		_Val:0>		
Status Affected:		N, Z					
Encoding:		1101	1110	0www	wddd	d000	SSSS
Description:		Significant	ift right the co bits of Wns ( register Wno	only up to 15	positions) ar	nd store the r	esult in the
		The 'd' bits	s select the a s select the de s select the so	estination reg	ister.	er.	
			This instructi	-		-	
\\/ordo:			If Wns is grea	ater than 15,	Wnd will be	loaded with (	)x0.
Words: Cycles:		1 1					
Cycles.		I					
Example 1:	LSR	W0, W1		LSR W0 by W Store resul			
		Before		After			
	F	nstruction		Instruction	ı		
	W0	nstruction C00C		Instruction	1		
	W0 W1	nstruction C00C 0001	V	Instruction V0 C00C V1 0001	1		
	W0	nstruction C00C	V V	Instruction	1		
Example 2:	W0 W1 W2	nstruction C00C 0001 2390	V V S , W3 ;	Instruction V0 C00C V1 0001 V2 6006	14		
Example 2:	W0 W1 W2 SR	nstruction <u>C00C</u> 0001 2390 0000 w5, w4 Before	V V S , W3 ;	Instruction V0 C00C V1 0001 V2 6006 SR 0000 LSR W5 by V Store resul After	14		
Example 2:	W0 W1 W2 SR LSR	nstruction C00C 0001 2390 0000 w5, w4 Before struction	V V S , W3 ; ;	Instruction V0 C00C V1 0001 V2 6006 SR 0000 LSR W5 by V Store resul After Instruction	14		
Example 2:	W0   W1   W2   SR   LSR   N3	nstruction C00C 0001 2390 0000 W5, W4 Before struction DD43	V V S , W3 ; ; V	Instruction V0 C00C V1 0001 V2 6006 GR 0000 LSR W5 by V Store result After Instruction V3 0000	14		
<u>Example 2:</u>	W0 W1 W2 SR LSR	nstruction C00C 0001 2390 0000 w5, w4 Before struction	V V S , W3 ; ; V V V	Instruction V0 C00C V1 0001 V2 6006 SR 0000 LSR W5 by V Store resul After Instruction	14		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
				Х	Х	Х
Syntax: {label:}	MAC	Wm*Wn, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd}	{,AWB}
			${,[Wx] + = kx}$	k, Wxd}	${,[Wy] + = ky}$	, Wyd}
			${,[Wx] - = k}$	x, Wxd}	$\{,[Wy] - = ky$	, Wyd}
			{,[W9 + W12	2], Wxd}	{,[W11 + W12	2], Wyd}
Operands:	$\begin{array}{l} Acc \in \ [A,B] \\ Wx \in \ [W8, \\ Wy \in \ [W10 \end{array} \end{array}$	[W4 * W5, W4 W9]; kx ∈ [-6, , W11]; ky ∈ [- 3, [W13] + = 2	-4, -2, 2, 4, 6 6, -4, -2, 2, 4	6]; Wxd ∈ [W	/4 W7]	W6 * W7]
Operation:	([Wx]) →Wx ([Wy]) →Wy	)) +(Wm) * (Wi xd; (Wx) + kx – rd; (Wy) + ky – )) rounded <i>—</i> A	→Wx →Wy	r B)		
Status Affected:	OA, OB, OA	AB, SA, SB, SA	٨B			
Encoding:	1100					
0	1100	Ommm	A0xx	yyii	iijj	jjaa
Description:	Multiply the in preparation unspecified	Ommm contents of tw on for another accumulator r ed to 40 bits a	o working re MAC type ins esults. The 3	gisters, optio struction and 32-bit result o	onally prefetch optionally sto of the signed r	n operands ore the multiply is
0	Multiply the in preparation unspecified sign-extend Operands V which support Section 4.1 store of the	contents of tw on for another accumulator r	o working re MAC type ins esults. The 3 nd added to nd Wyd spee I register offs ofetches". O ulator, as de	gisters, optic truction and 32-bit result of the specified cify optional set addressir perand AWE	onally prefetch optionally sto of the signed i d accumulator prefetch oper ng, as describ	n operands ore the multiply is : ations, ed in
0	Multiply the in preparation unspecified sign-extend Operands V which suppor Section 4.1 store of the Section 4.1 The 'm' bits The 'A' bits The 'A' bits The 'A' bits The 'a' bits The 'a' bits The 'a' bits Section 4.1 The 'a' bits The 'a' bits The 'a' bits The 'a' bits The 'a' bits The 'a' bits	contents of two on for another accumulator r ed to 40 bits a Vx, Wxd, Wy a ort indirect and <b>4.1 "MAC Pre</b> "other" accum <b>4.4 "MAC Wr</b> select the ope elects the accuse select the prefe select the prefe select the Wx p select the Wx p select the accu fhe IF bit (COF ractional or an The US<1:0> b CON<12> in ds unsigned, signed	o working re MAC type ins esults. The 3 nd added to nd Wyd spee register offs ofetches". O ulator, as de ite Back". erand registe unulator for etch Wyd de orefetch oper unulator Wri RCON<0>), o integer. bits (CORCO SPIC30F/dsF ed, or mixed-	gisters, optio gisters, optio struction and 32-bit result of the specified cify optional set addressir perand AWE scribed in rs Wm and V the result. stination. stination. te Back dest determines if N<13:12> in PIC33F) dete	onally prefetch optionally sto of the signed i d accumulator prefetch oper ng, as describ 3 specifies the Wn for the mu ination. i the multiply i dsPIC33E, C rmine if the m	n operands re the multiply is ations, ed in optional ltiply. s COR- pultiply is
0	Multiply the in preparation unspecified sign-extend Operands V which suppor Section 4.1 store of the Section 4.1 The 'm' bits The 'A' bits The 'A' bits The 'A' bits The 'a' bits The 'a' bits The 'a' bits Section 4.1 The 'a' bits The 'a' bits The 'a' bits The 'a' bits The 'a' bits The 'a' bits	contents of two on for another accumulator r ed to 40 bits a Vx, Wxd, Wy a ort indirect and <b>4.1 "MAC Pre</b> "other" accum <b>4.4 "MAC Wr</b> select the ope elects the accus select the prefe select the prefe select the Wy p select the Wy p select the accus for IF bit (COF ractional or an The US<1:0> b CON<12> in ds	o working re MAC type ins esults. The 3 nd added to nd Wyd spee register offs ofetches". O ulator, as de ite Back". erand registe unulator for etch Wyd de orefetch oper unulator Wri RCON<0>), o integer. bits (CORCO SPIC30F/dsF ed, or mixed-	gisters, optio gisters, optio struction and 32-bit result of the specified cify optional set addressir perand AWE scribed in rs Wm and V the result. stination. stination. te Back dest determines if N<13:12> in PIC33F) dete	onally prefetch optionally sto of the signed i d accumulator prefetch oper ng, as describ 3 specifies the Wn for the mu ination. i the multiply i dsPIC33E, C rmine if the m	n operands re the multiply is ations, ed in optional ltiply. s COR- pultiply is

....

- Example 1: MAC W4\*W5, A, [W8]+=6, W4, [W10]+=2, W5 ; Multiply W4\*W5 and add to ACCA
  - ; Fetch [W8] to W4, Post-increment W8 by 6
  - ; Fetch [W10] to W5, Post-increment W10 by 2  $\,$
  - ; CORCON = 0x00C0 (fractional multiply, normal saturation)

	Before Instruction		After Instruction
W4	A022	W4	2567
W5	B900	W5	909C
W8	0A00	W8	0A06
W10	1800	W10	1802
ACCA	00 1200 0000	ACCA	00 472D 2400
Data 0A00	2567	Data 0A00	2567
Data 1800	909C	Data 1800	909C
CORCON	00C0	CORCON	00C0
SR	0000	SR	0000

Example 2:

- ; Multiply W4\*W5 and add to ACCA
- ; Fetch [W8] to W4, Post-decrement W8 by 2
- ; Fetch [W10] to W5, Post-increment W10 by 2

MAC W4\*W5, A, [W8]-=2, W4, [W10]+=2, W5, W13

- ; Write Back ACCB to W13
- ; CORCON = 0x00D0 (fractional multiply, super saturation)

	Before Instruction
W4	1000
W5	3000
W8	0A00
W10	1800
W13	2000
ACCA	23 5000 2000
ACCB	00 0000 8F4C
Data 0A00	5BBE
Data 1800	C967
CORCON	00D0
SR	0000

	After Instruction	
W4	5BBE	
W5	C967	
W8	09FE	
W10	1802	
W13	0001	
ACCA	23 5600 2000	
ACCB	00 0000 1F4C	
Data 0A00	5BBE	
Data 1800	C967	
CORCON	00D0	
SR	8800	(OA, OAB = 1)

MAC		Square and	d Accumulat	te		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
				Х	Х	Х
Syntax: {label:}	MAC W	/m*Wm, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd	}
			$\{,[Wx] + = k\}$	k, Wxd}	${,[Wy] + = k}$	y, Wyd}
			${,[Wx] - = k}$	x, Wxd}	${,[Wy] - = k}$	y, Wyd}
			{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}
Operands:	$Acc \in [A,B]$ $Wx \in [W8,$	∈ [W4 * W4, ] W9]; kx ∈ [-6 ), W11]; ky ∈	6, -4, -2, 2, 4	, 6]; Wxd ∈ [	- W4 W7]	I
Operation:	([Wx]) →W	8)) +(Wm) * (\ xd; (Wx) + kx yd; (Wy) + ky	→Wx	or B)		
Status Affected:	OA, OB, O	AB, SA, SB, S	SAB			
Encoding:	1111	0 0 mm	A0xx	yyii	iijj	jj00
Description:	preparation	contents of a for another r tiply is sign-e or.	MAC type inst	ruction. The	32-bit result	of the
	which supp	Wx, Wxd, Wy oort indirect ar 14.1 "MAC P	nd register of			
	The 'A' bits The 'x' bits The 'y' bits The 'i' bits The 'j' bits Note 1:	s select the op selects the ac select the pro- select the pro- select the Wx select the Wy The IF bit (CC tional or an in	cumulator fo efetch Wxd c efetch Wyd c prefetch op prefetch op DRCON<0>) iteger.	r the result. lestination. lestination. eration. eration. , determines	if the multip	-
		The US<1:0> CORCON<12 multiply is un devices supp	2> in dsPIC3 signed, signe	0F/dsPIC33I ed, or mixed-	F) determine sign. Only d	if the
Words:	1					

Example 1:

- MAC W4\*W4, B, [W9+W12], W4, [W10]-=2, W5
  - ; Square W4 and add to ACCB
  - ; Fetch [W9+W12] to W4
  - ; Fetch [W10] to W5, Post-decrement W10 by 2  $\,$
  - ; CORCON = 0x00C0 (fractional multiply, normal saturation)

	Before Instruction		After Instruction
W4	A022	W4	A230
W5	B200	W5	650B
W9	0C00	W9	0C00
W10	1900	W10	18FE
W12	0020	W12	0020
ACCB	00 2000 0000	ACCB	00 67CD 0908
Data 0C20	A230	Data 0C20	A230
Data 1900	650B	Data 1900	650B
CORCON	00C0	CORCON	00C0
SR	0000	SR	0000

Example 2:

MAC W7\*W7, A, [W11]-=2, W7 ; Square W7 and add to ACCA

; Fetch [W11] to W7, Post-decrement W11 by 2

; CORCON = 0x00D0 (fractional multiply, super saturation)

	Before Instruction	
W7	76AE	
W11	2000	
ACCA	FE 9834 4500	
Data 2000	23FF	C
CORCON	00D0	(
SR	0000	

	After	
	Instruction	
W7	23FF	
W11	1FFE	
ACCA	FF 063E 0188	
Data 2000	23FF	
CORCON	00D0	
SR	8800	(OA, OAB = 1)
		•

MOV		Move f to D	estination			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f) →destina	ation designa	ted by D			
Status Affected:	N, Z					
Encoding:	1011	1111	1BDf	ffff	ffff	ffff
Description:	The option WREG is	ontents of the al WREG of specified, th	perand dete e result is	ermines the stored in W	destination /REG. If WI	register. I REG is no
Description:	The option WREG is specified, th to modify th The 'B' bit s The 'D' bit s	al WREG d	perand dete e result is ored back to egister. r word opera estination ('0	ermines the stored in W the file regis ation ('0' for w ' for WREG, '	destination /REG. If Wi ster and the o word, '1' for t	register. I REG is no only effect is oyte).
Description:	The option WREG is specified, th to modify th The 'B' bit s The 'D' bit s The 'f' bits s Note 1:	al WREG c specified, th he result is st selects byte o selects the de select the add The extension rather than a denote a word The WREG is When moving to Wnd" (pag	perand dete e result is ored back to egister. or word operation ('0' dress of the f n .B in the in word operation, s set to worki g word data finge 281) instru	ermines the stored in W the file regis ation ('0' for W r for WREG, file register. astruction der on. You may but it is not r ng register V rom file regis uction allows	destination /REG. If Wi ster and the o word, '1' for k '1' for file reg notes a byte y use a . w ex equired. V0. ter memory, ' any working	register. I REG is no only effect is oyte). jister). operation ttension to
Description:	The option WREG is specified, th to modify th The 'B' bit s The 'D' bit s The 'f' bits s Note 1:	al WREG c specified, th he result is st selects byte o selects the de select the add The extension rather than a denote a word The WREG is When moving	perand dete e result is ored back to egister. or word operation ('0' dress of the f n .B in the in word operation, s set to worki g word data finge 281) instru	ermines the stored in W the file regis ation ('0' for W r for WREG, file register. astruction der on. You may but it is not r ng register V rom file regis uction allows	destination /REG. If Wi ster and the o word, '1' for k '1' for file reg notes a byte y use a . w ex equired. V0. ter memory, ' any working	register. I REG is no only effect is oyte). jister). operation ttension to

- **Note 1:** In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1** "**Multi-Cycle Instructions**".
- Example 1: MOV.B TMR0, WREG ; move (TMR0) to WREG (Byte mode)

Before Instruction	After Instruction
WREG (W0) 9080	WREG (W0) 9055
TMR0 2355	TMR0 2355
SR 0000	SR 0000
Example 2: MOV 0x800	; update SR based on (0x800) (Word mode)
Before	After
Instruction	Instruction
Data 0800 B29F	Data 0800 B29F
SR 0000	SR 0008 (N = 1)

MOV		Move WRE	G to f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV{.B}	WREG,	f		
Operands:	f ∈ [0 8 <sup>·</sup>	191]				
Operation:	(WREG) –	→f				
Status Affected:	None					
Encoding:	1011	0111	1B1f	ffff	ffff	ffff
Description:		contents of the	e default wor	king register	WREG into t	he
		selects byte of select the ad			vord, '1' for b	yte).
	2: 3:	The extensio than a word i word move, I The WREG i When moving register merr allows any w ter.	move. You m out it is not re s set to worki g word data f nory, the "MOV	ay use a .W equired. ing register V rom the work 7 Wns to f	extension to V0. ting register a " (page 282)	denote a array to file instruction
Words:	1					
Cycles:	1					
Example 1: MO	V.B WREG,	0x801 ;	move WREG	to 0x801 (	Byte mode)	
WREG (W0 Data 0800 SF	0 4509	WREG ( Data 0		3		
Example 2: MO	W WREG,	DISICNT	; move WREG	G to DISICN	Т	
WREG (W0 DISICN <sup>-</sup> SF	T 0000	WREG ( DISI		) )		

### **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X
	L	<u>I</u>	<u> </u>	<u> </u>	<u> </u>	
Syntax:	{label:}	MOV	f,	Wnd		
Operands:	f ∈ [0 65 Wnd ∈ [W0					
Operation:	(f) $\rightarrow$ Wnd					
Status Affected:	None					
Encoding:	1000	Offf	ffff	ffff	ffff	dddd
Description:	register ma be word-ali The 'f' bits	vord contents ay reside anyv igned. Registe select the add	where in the 3 er direct addi dress of the f	32K words of ressing must file register.	data memor	y, but must
	The 'd' bits	select the de	stination reg	ister.		
	2:	This instruction Since the file upper 15 bits assumed to b To move a by to Destina	register add of the file reg be '0'). /te of data fro	ress must be gister addres om file registe	word-aligned s are encode er memory, th	ed (bit 0 is ne "MOV f
Words:	1					0000.
Cycles:	1(1)					
		erations on not Section 3.2.				' more
Example 1: MOX	Before Instruction 78FA	W				
	Before Instruction 78FA 00F0 0000	W <sup>,</sup> CORCC S	After Instructior 12 00F0	n     		

MOV		Move Wns f	o f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV	Wns,	f		
Operands:	f ∈ [0 65 Wns ∈ [W					
Operation:	(Wns) →f					
Status Affected:	None					
Encoding:	1000	lfff	ffff	ffff	ffff	SSSS
Description:	register. Th	vord contents ne file register ut must be wo 'n.	may reside a	anywhere in t	he 32K word	s of data
		select the add select the so				
	Note 1: 2:	This instruction Since the file upper 15 bits assumed to b	register addr of the file reg	ess must be	word-aligned	
	3:	To move a by to f" instruc	te of data to f			10V WREG
Words:	1					
Cycles:	1					
Example 1: MO	V W4, XM	DOSRT ;	move W4 to	XMODSRT		
WA XMODSR <sup>-</sup> SF	T 1340	XMODS	After Instructio W4 1200 RT 1200 SR 0000	n     		
Example 2: MO	V W8, 0	x1222 ;	move W8 to	data addre	ss 0x1222	
Wa Data 122 SF	2 FD88	Data 12	After Instructio W8 F200 222 F200 SR 0000	n   		

MOV.B		Move 8-bit	Literal to Wr	d		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV.B	#lit8,	Wnd		
Operands:	lit8 ∈ [0 Wnd ∈ [Wi					
Operation:	lit8 →Wnd					
Status Affected:	None					
Encoding:	1011	0011	1100	kkkk	kkkk	dddd
		specify the vaselect the ad This instruction	dress of the	working regis		3 extensio
		must be prov		)		
Words:	1					
Cycles:	1					
Example 1:	MOV.B #0x1	7,W5 ;	load W5 wi	th #0x17 (E	Byte mode)	
	Before Instruction W5 7899 SR 0000		After Instructio W5 7817 SR 0000	n ] ]		
Example 2:	MOV.B #0xF	E, W9 ;	load W9 wi	th #0xFE (E	Byte mode)	
	Before Instruction W9 AB23 SR 0000		After Instructio W9 ABFE SR 0000	n ] ]		

MOV		Move 16-bit	t Literal to W	/nd		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV	#lit16,	Wnd		
Operands:	lit16 ∈ [-32 Wnd ∈ [W	2768 65535 0 W15]	5]			
Operation:	lit16 →Wn	d				
Status Affected:	None					
Encoding:	0010	kkkk	kkkk	kkkk	kkkk	dddd
Description:	The 16-bit be used fo	literal 'k' is loa r Wnd.	aded into Wr	nd. Register d	lirect address	sing must
		s specify the v s select the ac			ster.	
	Note 1: 2:	This instruction The literal material material	ay be specifie	ed as a signe		68:32767]
Words:	1	5		-1		
Cycles:	1					
Example 1: MC	DV #0x423	l, W13 ;	load W13	with #0x423	1	
W1 Si			After Instructio 113 4231 SR 0000	n ] ]		
Example 2: MC	)V #0x4, N	N2 ;	; load W2 w	ith #0x4		
W			After Instructio W2 0004 SR 0000	-		
Example 3: MC	DV #-1000	, W8 ;	; load W8 w	ith #-1000		
W			After Instructio W8 FC18 SR 0000	n ] ]		

Implemented in:	PIC24E	PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33I						
Implemented III.	X	X	X	X	X	X		
Syntax:	{label:}	MOV{.B}	[Ws + Slit10],	Wnd				
Operands:	Slit10 ∈ [-5 Slit10 ∈ [-1	Ws ∈ [W0 W15] Slit10 ∈ [-512 511] for byte operation Slit10 ∈ [-1024 1022] (even only) for word operation Wnd ∈ [W0 W15]						
Operation:	[Ws + Slit1	0] →Wnd						
Status Affected:	None							
Encoding:	1001	0kkk	kBkk	kddd	dkkk	SSSS		
	maintain w used for th The 'k' bits The 'B' bit The 'd' bits The 's' bits	ord address e source, and specify the v selects byte of select the do select the so	sed to [-1024 . alignment. Reg d direct addres value of the lite or word operati estination regis ource register.	gister indirect sing must be ral. ion ('0' for we ster.	addressing used for W ord, '1' for b	nust be nd. yte).		
	2:	than a word move. You may use a . w extension to denote a word move, but it is not required.						
Words:	1	represents a		et nom ws.				
Cycles:	1(1)							
read-m	odify-write op	erations on ne	, the listed cyclon-CPU Specia .1 " <b>Multi-Cycl</b>	al Function R	egisters. For			

Example 1:	MOV	.B [W8	+0x13],	W10 ; ;	load W10 (Byte mod		[W8+0x13]	
	Before Instruction							
	W8	1008		W8	1008			
	W10	4009		W10	4033			
Data	101A	3312	D	ata 101A	3312			

SR

0000

SR

Example 2: MOV	7 [W4+0x		; load W2 w ; (Word mod	ith [W4+0x3] e)	E8]	
W2 W4 Data 0BE8 SR	0800 5634	Data 0B	After Instructio W2 5634 W4 0800 BE8 5634 SR 0000	n ] - -		
MOV		Move Wns	to [Wd with	offset]		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MOV{.B}	Wns,	[Wd + Slit10]		
Operands:	-	12 511] in 024 1022]	•	in Word mode		
Operation:	(Wns) →[W	/d + Slit10]				
Status Affected:	None					
Encoding:	1001	1kkk	kBkk	kddd	dkkk	SSSS
Description:	of Slit10 is maintain wo	increased to ord address a	[-1024 102 alignment. Re	/d + Slit10]. In 22] and Slit10 egister direct a g must be use	must be evended addressing n	en to nust be
	The 'B' bit s The 'd' bits	selects byte of select the de	alue of the lit or word opera estination reg ource register	ation ('0' for wo	ord, '1' for b	yte).
	t	than a word r		struction denc ay use a .w e quired.	•	
	:	· · · · · · · · · · · · · · · · · · ·				
Words:	1					
Cycles:	1					
Example 1: MOV	и.в WO, [W		; store WO ; (Byte mod	to [W1+0x7] e)		
W0 W1 Data 1806 SR	1800 2345	Data 18	After Instructio W0 9015 W1 1800 306 1545 SR 0000	n       		

Example 2: MOV	7 W11, [W1-		store W11 (Word mode	to [W1-0x4 e)	00]	
ا W1 W11 Data 0C00 SR		W1 Data 0C0				
MOV		Move Ws to				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}		[Ws++], [Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [Wd] [++Wd] [Wd + Wb]		
Operands:	Ws ∈ [W0 . Wb ∈ [W0 . Wd ∈ [W0 .	W15]				
Operation:	(Ws) →Wd	-				
Status Affected:	None					
Encoding:	0111	lwww	wBhh	hddd	qaaa	SSSS
Description:		contents of the ster direct or ir				
The 'w' bits define the offset reg The 'B' bit selects byte or word The 'h' bits select the destinatio The 'd' bits select the destinatio The 'g' bits select the source Ac The 's' bits select the source reg				ation ('0' for w dress mode. gister. ss mode.	vord, '1' for b	yte).
	Note 1:	The extension than a word m word move, b	n .	nstruction den ay use a . We	-	
	<b>2:</b> \	When Register source and de the 'w' encodi	er Offset Add estination, the	dressing modene offset must	t be the same	
	3:	The instruction	on"PUSH Ws"	" translates to	o MOV Ws,[V	
	4:	The instruction	n"POP Wd"f	translates to	MOV [W1	5], Wd.

Words:1Cycles:1(1)Note 1:In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read an read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".Example 1:MOV.B $[W0]$ , W4; Move $[W0]$ to W4 (Byte mode) ; Post-decrement W0BeforeAfter InstructionInstruction W42989 2989 Data 0A00Data 0A008988 8 SRData 0A00 00008988 SROddoExample 2:MOV $[W6++]$ , $[W2+W3]$ ; Move $[W6]$ to $[W2+W3]$ (Word mode) ; Post-increment W6Example 2:MOV $[W6++]$ , $[W2+W3]$ ; Move $[W6]$ to $[W2+W3]$ (Word mode) ; Post-increment W6BeforeAfter Instruction W3Oddo W3W20800 W3W2 0040 W30800 040 W3W20800 W3W2 0690 Data 12280800 0690 SRData 08409870 0890 SRData 0840 0690 SR0000	MOV			Move Ws to Wd
Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read an read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".         Example 1:       MOV.B       [W0], W4 ; Move [W0] to W4 (Byte mode) ; Post-decrement W0         Before       After         Instruction       Mov         W0       OA01         W0       OA01         W4       2976         Data 0A00       8988         SR       0000         SR       0000         Example 2:       MOV [W6++], [W2+W3] ; Move [W6] to [W2+W3] (Word mode) ; Post-increment W6         Before       After         Instruction       Instruction         W2       0800         W2       0800         W3       0040         W3       0040         W6       1228         Data 0840       9870         Data 0840       0690	Words:		1	
read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions". Example 1: MOV.B [W0], W4 ; Move [W0] to W4 (Byte mode) ; Post-decrement W0  Before After Instruction  W0 0A01 W0 0A00  W4 2976 W4 2989  Data 0A00 8988 Data 0A00 8988  SR 0000 SR 0000 $Example 2: Mov [W6++], [W2+W3] ; Move [W6] to [W2+W3] (Word mode) ; Post-increment W6  Before After Instruction  W2 0800 W2 0800  W3 0040 W3 0040  W3 0040 W3 0040  W3 0040 W3 0040  W6 1228 W6 122A  Data 0840 9870 Data 0840 0690  Data 1228 0690 Data 1228 0690$	Cycles:		1 <sup>(1)</sup>	
; Post-decrement W0 Before After Instruction Instruction W0 0A01 W0 0A00 W4 2976 W4 2989 Data 0A00 8988 Data 0A00 8988 SR 0000 Example 2: MOV [W6++], [W2+W3] ; Move [W6] to [W2+W3] (Word mode) ; Post-increment W6 Before After Instruction Instruction W2 0800 W2 0800 W3 0040 W3 0040 W6 1228 W6 122A Data 0840 9870 Data 0840 0690 Data 1228 0690 Data 1228 0690	Note 1:	read-mod	lify-write op	erations on non-CPU Special Function Registers. For more
Instruction       Instruction         W0       0A01       W0       0A00         W4       2976       W4       2989         Data 0A00       8988       Data 0A00       8988         SR       0000       SR       0000         Example 2:       Mov       [W6++], [W2+W3]       ; Move [W6] to [W2+W3] (Word mode)         ; Post-increment W6       SR       SR       0000         W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data 0840       9870       Data 0840       0690         Data 1228       0690       Data 1228       0690	Example 1	<u>1:</u> MOV	.B [W0	_
W0       0A01       W0       0A00         W4       2976       W4       2989         Data 0A00       8988       Data 0A00       8988         SR       0000       SR       0000         Example 2:       MOV       [W6++], [W2+W3]       ; Move [W6] to [W2+W3] (Word mode)         ; Post-increment W6       SR       SR         Before       After         Instruction       Instruction         W2       0800       W2         W3       0040       W3         W6       1228       W6         Data 0840       9870       Data 0840         Data 1228       0690       Data 1228				
W4       2976       W4       2989         Data 0A00       8988       Data 0A00       8988         SR       0000       SR       0000         Example 2:       MOV       [W6++], [W2+W3]       ; Move [W6] to [W2+W3] (Word mode)         ; Post-increment W6       Before       After         Instruction       Instruction       Number of the struction         W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data 0840       9870       Data 0840       0690         Data 1228       0690       Data 1228       0690				
Data 0A00       8988       Data 0A00       8988         SR       0000       SR       0000         Example 2:       MOV       [W6++], [W2+W3]       ; Move [W6] to [W2+W3] (Word mode)         ; Post-increment W6       Before       After         Instruction       Instruction       Instruction         W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data 0840       9870       Data 0840       0690         Data 1228       0690       Data 1228       0690				
SR       0000       SR       0000         Example 2:       MOV       [W6++], [W2+W3]       ; Move [W6] to [W2+W3] (Word mode)         Before       After         Instruction       Instruction         W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data       0840       9870       Data       0840       0690         Data       1228       0690       Data       1228       0690				
Example 2:       MOV [W6++], [W2+W3] ; Move [W6] to [W2+W3] (Word mode)         Before       After         Instruction       Instruction         W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data       0840       9870       Data       0890         Data       1228       0690       0690	D			
; Post-increment W6 Before After Instruction Instruction W2 0800 W2 0800 W3 0040 W3 0040 W6 1228 W6 122A Data 0840 9870 Data 0840 0690 Data 1228 0690 Data 1228 0690		SR	0000	SR 0000
Instruction         Instruction           W2         0800         W2         0800           W3         0040         W3         0040           W6         1228         W6         122A           Data         0840         9870         Data         0840           Data         1228         0690         Data         1228	Example 2	<u>2:</u> mov	[₩6++],	
W2       0800       W2       0800         W3       0040       W3       0040         W6       1228       W6       122A         Data 0840       9870       Data 0840       0690         Data 1228       0690       Data 1228       0690				
W3     0040     W3     0040       W6     1228     W6     122A       Data 0840     9870     Data 0840     0690       Data 1228     0690     Data 1228     0690				
W6         1228         W6         122A           Data 0840         9870         Data 0840         0690           Data 1228         0690         Data 1228         0690				
Data 0840         9870         Data 0840         0690           Data 1228         0690         Data 1228         0690		-		
Data 1228 0690 Data 1228 0690	-			
	L			
			0000	

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	MOV.D	Wns, [Ws],	Wnd				
			[Ws++],					
			[Ws],					
			[++Ws],					
			[Ws],					
Operands:	Ws∈ [W0	Wns ∈ [W0, W2, W4 W14] Ws ∈ [W0 W15] Wnd ∈ [W0, W2, W4 W14]						
Operation:	For direct a Wns →V Wns + 1 For indirect	For direct addressing of source: Wns $\rightarrow$ Wnd Wns + 1 $\rightarrow$ Wnd + 1 For indirect addressing of source: See Description						
Status Affected:	None							
Encoding:	1011	1110	0000	0ddd	0ppp	SSSS		
Description:	register pai source, the are moved Ws specifie double wor	r (Wnd:Wnd contents of to Wnd:Wnd es the effectiv d. Any pre/po	specified by th + 1). If register two successiv + 1. If indirect /e address for pst-increment date for the do	er direct addre ve working reg t addressing r the least sign or pre/post-d	essing is use gisters (Wns:' is used for th nificant word	d for the Wns + 1) le source, of the		
	The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the address of the first source register.							
		for information on how double words are aligned in memory.						
		The instructio	on"POP.D Wi	nd" translates	to MOV.D [	W15],		
Words:	1							
Cycles:	2(1)							

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

MOV.D W2, W6

Example 1:

W2 W3 W6 W7 SR	Before nstruction 12FB 9877 9833 FCC6 0000	W2 W3 W6 W7 SR ], W4 ; Move [	After Instruction 12FB 9877 12FB 9877 0000	4 (Double mode)
	Before		After	
I	nstruction	I I	Instruction	1
W4	B012	W4	A319	
W5	FD89	W5	9927	
W7	0900	W7	08FC	
Data 0900	A319	Data 0900	A319	
Data 0902	9927	Data 0902	9927	
SR	0000	SR	0000	

; Move W2 to W6 (Double mode)

MOVPAG	Move Literal to Page Register						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
			Х			Х	
<b>a</b>							
Syntax:	{label:}	MOVPAG	#lit10,	DSRPAG			
			#lit9,	DSWPAG			
			#lit8,	TBLPAG			
Operands:	lit10 ∈ [0 .	1023], lit9 ∈	[0 511], li	it8 ∈ [0 25	5]		
Operation:	lit10 →DSI	lit10 $\rightarrow$ DSRPAG or lit9 $\rightarrow$ DSWPAG or lit8 $\rightarrow$ TBLPAG					
Status Affected:	None						
Encoding:	1111	1110	1100	PPkk	kkkk	kkkk	
Description:	into the DS restricts th	priate numbe SRPAG, DSW e literal to a 9 and an 8-bit	PAG, or TBL bit unsigned	PAG register	. The assem the destinat	bler ion is	
		s select the design of the version o		-			
	Note:	This instruct	ion operates	in word mod	e only.		
Words:	1						
Cycles:	1						
Example 1: MOV	PAG #0x02,	DSRPAG					
	Before		After				
I	Instruction		Instructior	n			
DSRPAG	0000	DSRPA	G 0002				

### MOVDAC

MOVPAG		Move Ws to	o Page Regi	ster		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MOVPAG	Wn,	DSRPAG		
				DSWPAG		
				TBLPAG		
Operands:	Wn∈ [W0	W15]				
Operation:	Wn<9:0> -	Wn<9:0> →DSRPAG or Wn<8:0> →DSWPAG or Wn<7:0> →TBLPAG				
Status Affected:	None					
Encoding:	1111	1110	1101	PP00	0000	SSSS
Description:	DSRPAG, literal to a	oriate numbe DSWPAG, or 9-bit unsigned ned value wh	TBLPAG reg d value when	ister. The as the destinati	sembler rest	ricts the
		s select the description of the select the s				
	Note:	This instruct	ion operates	in word mod	e only.	
Words:	1					
Cycles:	1					
Example 1: MOVPAG W2, DSRPAG						
DSRPAG W2	Before nstruction 0000 0002	DSRPA	After Instruction G 0002 V2 0002	1		

MOVSAC	Prefetch Operands and Store Accumulator						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33	
				Х	Х	Х	
Syntax: {label:}	MOVSAC A	Acc {,[Wx],	Wxd}	{,[Wy], Wyd	}	{,AWB}	
		${,[Wx] + = k}$	x, Wxd}	${,[Wy] + = k}$	y, Wyd}		
		${,[Wx] - = k}$	x, Wxd}	${,[Wy] - = k}$	y, Wyd}		
		{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}		
Operands:	Wy ∈ [W10	W9]; kx ∈  [-6 , W11]; ky ∈  3, [W13] + =	[-6, -4, -2, 2,			]	
Operation:	([Wy]) →Wy	xd; (Wx) + kx vd; (Wy) + ky )) rounded –	→Wy				
Status Affected:	None						
Encoding:	1100	0111	A0xx	yyii	iijj	jjaa	
Description:	instruction a though an a	orefetch oper and optionally accumulator o r must be sp	y store the up operation is r	nspecified ac not performe	cumulator re d in this instr	esults. Eve ruction, an	
	which support Section 4.1 store of the	Vx, Wxd, Wy ort indirect ar 4.1 "MAC P "other" accu 4.4 "MAC W	nd register of <b>refetches"</b> . mulator, as c	ffset address Operand AW	ing, as desc	ribed in	
	The 'x' bits s The 'y' bits s The 'i' bits s The 'j' bits s	elects the oth select the pro- select the pro- select the Wx select the Wy select the ac	efetch Wxd c efetch Wyd c prefetch op prefetch op	lestination. lestination. eration. eration.			
Words:	1						
Cycles:	1						

#### MOVEAC

```
Example 1:
```

MOVSAC B, [W9], W6, [W11]+=4, W7, W13 ; Fetch [W9] to W6 ; Fetch [W11] to W7, Post-increment W11 by 4

; Store ACCA to W13

	Before Instruction		After Instruction
W6	A022	W6	7811
W7	B200	W7	B2AF
W9	0800	W9	0800
W11	1900	W11	1904
W13	0020	W13	3290
ACCA	00 3290 5968	ACCA	00 3290 5968
Data 0800	7811	Data 0800	7811
Data 1900	B2AF	Data 1900	B2AF
SR	0000	SR	0000

Example 2:

MOVSAC A, [W9]-=2, W4, [W11+W12], W6, [W13]+=2
; Fetch [W9] to W4, Post-decrement W9 by 2
; Fetch [W11+W12] to W6

; Store ACCB to [W13], Post-increment W13 by 2

	Before Instruction
W4	76AE
W6	2000
W9	1200
W11	2000
W12	0024
W13	2300
ACCB	00 9834 4500
Data 1200	BB00
Data 2024	52CE
Data 2300	23FF
SR	0000

	After Instruction
W4	BB00
W6	52CE
W9	11FE
W11	2000
W12	0024
W13	2302
ACCB	00 9834 4500
Data 1200	BB00
Data 2024	52CE
Data 2300	9834
SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax: {label:}	MPY Wm	n * Wn, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd	}
			$\{,[Wx] + = kx$	x, Wxd}	${,[Wy] + = k}$	y, Wyd}
			${,[Wx] - = k}$	x, Wxd}	${,[Wy] - = k}$	y, Wyd}
			{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}
Operands: Operation:	$\begin{array}{l} Acc \in [A,B] \\ Wx \in [W8, \\ Wy \in [W10] \\ AWB \in [W10] \\ (Wm) * (Wr) \\ ([Wx]) \rightarrow W. \end{array}$	] W9]; kx ∈ [-	r B) k →Wx	, 6]; Wxd ∈ [	W4 W7]	
Status Affected:	OA, OB, OAB, SA, SB, SAB					
Encoding:	1100	0 mmm	A0xx	yyii	iijj	jj11
Description:	Multiply the contents of two working registers, optionally prefetch operands in preparation for another MAC type instruction. The 32-bit result of the signed multiply is sign-extended to 40 bits and stored to the specified accumulator.					
	Operands Wx, Wxd, Wy and Wyd specify optional prefetch operations which support indirect and register offset addressing, as described in <b>Section 4.14.1 "MAC Prefetches</b> ".					
	<ul> <li>The 'm' bits select the operand registers Wm and Wn for the multiply:</li> <li>The 'A' bit selects the accumulator for the result.</li> <li>The 'x' bits select the prefetch Wxd destination.</li> <li>The 'y' bits select the prefetch Wyd destination.</li> <li>The 'i' bits select the Wx prefetch operation.</li> <li>The 'j' bits select the Wy prefetch operation.</li> <li>Note 1: The IF bit, CORCON&lt;0&gt;, determines if the multiply is fractional or an integer.</li> </ul>					
	2: The US<1:0> bits (CORCON<13:12> in dsPIC33E, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed, or mixed-sign. Only dsPIC33E devices support mixed-sign multiplication.					
						31 1000L
Words:						

```
Example 1:
```

MPY W4\*W5, A, [W8]+=2, W6, [W10]-=2, W7 ; Multiply W4\*W5 and store to ACCA

; Fetch [W8] to W6, Post-increment W8 by 2

- ; Fetch [W10] to W7, Post-decrement W10 by 2
- ; CORCON = 0x0000 (fractional multiply, no saturation)

	Before Instruction	After Instruction	
W4	C000	W4	C000
W5	9000	W5	9000
W6	0800	W6	671F
W7	B200	W7	E3DC
W8	1780	W8	1782
W10	2400	W10	23FE
ACCA	FF F780 2087	ACCA	00 3800 0000
Data 1780	671F	Data 1780	671F
Data 2400	E3DC	Data 2400	E3DC
CORCON	0000	CORCON	0000
SR	0000	SR	0000

Example 2:

MPY W6\*W7, B, [W8]+=2, W4, [W10]-=2, W5

; Multiply W6\*W7 and store to ACCB

; Fetch [W8] to W4, Post-increment W8 by 2

; Fetch [W10] to W5, Post-decrement W10 by 2  $\,$ 

; CORCON = 0x0000 (fractional multiply, no saturation)

	Before Instruction		After Instruction
W4	C000	W4	8FDC
W5	9000	W5	0078
W6	671F	W6	671F
W7	E3DC	W7	E3DC
W8	1782	W8	1784
W10	23FE	W10	23FC
ACCB	00 9834 4500	ACCB	FF E954 3748
Data 1782	8FDC	Data 1782	8FDC
Data 23FE	0078	Data 23FE	0078
CORCON	0000	CORCON	0000
SR	0000	SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	1102 11	1102111	1.102.12	X	X	X	
Syntax: {labe	el:} MPY Wm *	Wm, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd	}	
			$\{,[Wx] + = kx$	k, Wxd}	${,[Wy] + = k}$	y, Wyd}	
			${,[Wx] - = k}$	x, Wxd}	${,[Wy] - = k_1}$	y, Wyd}	
			{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}	
Operands:	$Acc \in [A,B]$ $Wx \in [W8,$	- W9]; kx ∈ [-	W5 * W5, W 6, -4, -2, 2, 4 [-6, -4, -2, 2	, 6]; Wxd ∈ [	-		
Operation:	([Wx]) →Wx	n) →Acc(A o xd; (Wx) + kx ⁄d; (Wy) + ky	x→Wx				
Status Affected:	OA, OB, OA	AB, SA, SB,	SAB				
Encoding:	1111	0 0 mm	A0xx	yyii	iijj	jj01	
Description:	preparation signed mult	Square the contents of a working register, optionally prefetch operands in preparation for another MAC type instruction. The 32-bit result of the signed multiply is sign-extended to 40 bits and stored in the specified accumulator.					
	which supp	Operands Wx, Wxd, Wy and Wyd specify optional prefetch operations which support indirect and register offset addressing, as described in <b>Section 4.14.1 "MAC Prefetches</b> ".					
	The 'A' bit s The 'x' bits s The 'y' bits s The 'i' bits s The 'j' bits s Note 1: 1 f 2: 1	elects the ac select the pr select the pr select the W select the W fhe IF bit (C ractional or fhe US<1:0 CORCON<1 nultiply is un	an integer. > bits (CORC 2> in dsPIC3 signed, sign	or the result. destination. destination. eration. , determines ON<13:12> 0F/dsPIC33 ed, or mixed	if the multipl in dsPIC33E F) determine sign. Only d	, if the	
Words:	1	ievices supp	ort mixed-sig	gn muitiplicat			
vvoras: Cycles:	1						
0,000.	1						

Example 1:

MPY W6\*W6, A, [W9]+=2, W6

; Square W6 and store to ACCA

; Fetch [W9] to W6, Post-increment W9 by 2

; CORCON = 0x0000 (fractional multiply, no saturation)

	Before Instruction		After Instruction
W6	6500	W6	B865
W9	0900	W9	0902
ACCA	00 7C80 0908	ACCA	00 4FB2 0000
Data 0900	B865	Data 0900	B865
CORCON	0000	CORCON	0000
SR	0000	SR	0000

Example 2: MPY W4\*W4, B, [W9+W12], W4, [W10]+=2, W5

; Square W4 and store to ACCB

; Fetch [W9+W12] to W4  $\,$ 

; Fetch [W10] to W5, Post-increment W10 by 2

; CORCON =  $0 \times 00000$  (fractional multiply, no saturation)

	Before Instruction
W4	E228
W5	9000
W9	1700
W10	1B00
W12	FF00
ACCB	00 9834 4500
Data 1600	8911
Data 1B00	F678
CORCON	0000
SR	0000

	After
	Instruction
W4	8911
W5	F678
W9	1700
W10	1B02
W12	FF00
ACCB	00 06F5 4C80
Data 1600	8911
Data 1B00	F678
CORCON	0000
SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax: {label:}	MPY.N Wm	* Wn, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd	}
			$\{,[Wx] + = kx$	k, Wxd}	${,[Wy] + = k_{1}}$	y, Wyd}
			$\{,[Wx] - = kx$	k, Wxd}	${,[Wy] - = k_{y}}$	y, Wyd}
			{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}
Operands:	$Acc \in [A,B]$ $Wx \in [W8, T]$	W9]; kx ∈ [-	W4 * W6; W4 6, -4, -2, 2, 4 [-6, -4, -2, 2,	, 6]; Wxd ∈ [	W4 W7]	
Operation:	([Wx]) →Wx	n) →Acc(A o d; (Wx) + kx rd; (Wy) + ky	x→Wx			
Status Affected:	OA, OB, OA	B				
Encoding:	1100	Ommm	Alxx	yyii	iijj	jj11
Description:			a working red	gister by the	negative of t	he contents
	another MAC	type instruer r results. The	ter, optionally ction and opt e 32-bit resul and stored t	v prefetch op ionally store t of the signe	the unspecified multiply is	paration fo ied
	another MAC accumulato sign-extend The 'm' bits The 'A' bit s The 'y' bits s The 'j' bits s The 'j' bits s Note 1:	c type instruct r results. The ed to 40 bits select the of select the pr select the pr elect the Wy elect the Wy	ter, optionally ction and opt e 32-bit resul and stored t perand regis ccumulator for efetch Wxd o efetch Wyd o v prefetch op v prefetch op ORCON<0>)	y prefetch op ionally store t of the signe o the specifi ters Wm and or the result. destination. lestination. eration. eration.	the unspecified multiply is ed accumula I Wn for the r	paration fo ied tor. nultiply.
	another MAC accumulato sign-extend The 'm' bits The 'A' bits The 'y' bits The 'y' bits The 'j' bits <b>Note 1:</b>	type instruct results. The ed to 40 bits select the o elects the ac select the pr select the pr select the Wy elect the Wy che IF bit (Cu ractional or a che US<1:0= CORCON<1: nultiply is un	ter, optionally ction and opt e 32-bit resul and stored t perand regis ccumulator for efetch Wxd o efetch Wyd o v prefetch op v prefetch op ORCON<0>)	v prefetch op ionally store t of the signe o the specifi ters Wm and or the result. destination. eration. eration. , determines ON<13:12> 0F/dsPIC33 ed, or mixed	the unspecified multiply is ed accumula I Wn for the r if the multipl in dsPIC33E F) determine sign. Only ds	paration fo ied tor. nultiply. y is , if the
Words:	another MAC accumulato sign-extend The 'm' bits The 'A' bits The 'y' bits The 'y' bits The 'j' bits <b>Note 1:</b>	type instruct results. The ed to 40 bits select the o elects the ac select the pr select the pr select the Wy elect the Wy che IF bit (Cu ractional or a che US<1:0= CORCON<1: nultiply is un	ter, optionally ction and opt e 32-bit resul and stored t perand regis ccumulator for efetch Wxd of efetch Wyd of orefetch op prefetch op ORCON<0>) an integer. bits (CORC 2> in dsPIC3 signed, signed	v prefetch op ionally store t of the signe o the specifi ters Wm and or the result. destination. eration. eration. , determines ON<13:12> 0F/dsPIC33 ed, or mixed	the unspecified multiply is ed accumula I Wn for the r if the multipl in dsPIC33E F) determine sign. Only ds	paration fo ied tor. nultiply. y is , if the

#### 5

```
Example 1:
```

MPY.N W4\*W5, A, [W8]+=2, W4, [W10]+=2, W5
; Multiply W4\*W5, negate the result and store to ACCA
; Fetch [W8] to W4, Post-increment W8 by 2

; Fetch [W10] to W5, Post-increment W10 by 2

; CORCON = 0x0001 (integer multiply, no saturation)

	Before Instruction		After Instruction
W4	3023	W4	0054
W5	1290	W5	660A
W8	0B00	W8	0B02
W10	2000	W10	2002
ACCA	00 0000 2387	ACCA	FF FC82 7650
Data 0B00	0054	Data 0B00	0054
Data 2000	660A	Data 2000	660A
CORCON	0001	CORCON	0001
SR	0000	SR	0000

Example 2:

MPY.N W4\*W5, A, [W8]+=2, W4, [W10]+=2, W5
; Multiply W4\*W5, negate the result and store to ACCA

; Fetch [W8] to W4, Post-increment W8 by 2

; Fetch [W10] to W5, Post-increment W10 by 2

; CORCON = 0x0000 (fractional multiply, no saturation)

	Before	
	Instruction	
W4	3023	
W5	1290	
W8	0B00	
W10	2000	V
ACCA	00 0000 2387	AC
Data 0B00	0054	Data 0E
Data 2000	660A	Data 20
CORCON	0000	CORC
SR	0000	

	After
	Instruction
W4	0054
W5	660A
W8	0B02
W10	2002
ACCA	FF F904 ECA0
Data 0B00	0054
Data 2000	660A
CORCON	0000
SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax: {label:}	MSC Wm	* Wn, Acc	{,[Wx], Wxd	}	{,[Wy], Wyd	} {,AWB
			${,[Wx] + = kx}$	k, Wxd}	$\{,[Wy] + = k\}$	y, Wyd}
			${,[Wx] - = k}$	x, Wxd}	$\{, [Wy] - = k\}$	y, Wyd}
			{,[W9 + W12	2], Wxd}	{,[W11 + W1	2], Wyd}
Operands:	$\begin{array}{l} Acc \in \ [A,B \\ Wx \in \ [W8, \\ Wy \in \ [W10 \end{array} \end{array}$	] W9]; kx ∈ [-	W4 * W6, W4 6, -4, -2, 2, 4 [-6, -4, -2, 2 = 2]	, 6]; Wxd ∈ [	W4 W7]	
Operation:	([Wx]) →W ([Wy]) →W	8)) –(Wm) * (' xd; (Wx) + kx yd; (Wy) + ky ()) rounded -	/ →Wy	or B)		
Status Affected:	OA, OB, O	AB, SA, SB,	SAB			
Encoding:	1100	Ommm	Alxx	yyii	iijj	jjaa
Description:	operands in store the up	n preparation nspecified ac sign-extende	two working for another ccumulator re d to 40 bits a	MAC type inst sults. The 32	ruction and or bit result of	optionally the signed
	accumulate					
	Operands Which supp Section 4.1 store of the	Wx, Wxd, Wy ort indirect a 14.1 "MAC F e "other" accu	v and Wyd sp nd register o Prefetches". Imulator as d Vrite Back".	ffset address Operand AW	ing as descr	ibed in
	Operands V which supp Section 4. store of the Section 4. The 'm' bits The 'A' bits The 'A' bits The 'y' bits The 'j' bits The 'j' bits	Wx, Wxd, Wy ort indirect a 14.1 "MAC F "other" accu 14.4 "MAC V s select the o select sthe a select the pr select the pr select the Wy select the Wy	nd register o <b>refetches</b> ". Imulator as d	ffset address Operand AW escribed in ters Wm and or the result. destination. destination. eration. eration.	ing as descr /B specifies t Wn for the r	ibed in the optiona
	Operands V which supp Section 4. store of the Section 4. The 'm' bits The 'A' bits The 'A' bits The 'y' bits The 'j' bits The 'j' bits The 'a' bits Note:	Wx, Wxd, Wy port indirect a 14.1 "MAC F "other" accu 14.4 "MAC V s select the of selects the ac select the pr select the pr select the Wy select the Wy select the Wy	nd register o <b>Prefetches</b> ". Imulator as d <b>Vrite Back</b> ". perand regis ccumulator for efetch Wxd of efetch Wyd of x prefetch op y prefetch op ccumulator W CORCON<0>	ffset address Operand AW escribed in ters Wm and or the result. destination. destination. eration. rite Back de	ing as descr /B specifies t Wn for the r stination.	ibed in the optiona nultiply.
Words:	Operands V which supp Section 4. store of the Section 4. The 'm' bits The 'A' bits The 'A' bits The 'y' bits The 'j' bits The 'j' bits The 'a' bits Note:	Wx, Wxd, Wy port indirect a 14.1 "MAC F "other" accu 14.4 "MAC V s select the of selects the ac select the pr select the pr select the Wy select the Wy select the Wy select the Wy select the ac The IF bit (C	nd register o <b>Prefetches</b> ". Imulator as d <b>Vrite Back</b> ". perand regis ccumulator for efetch Wxd of efetch Wyd of x prefetch op y prefetch op ccumulator W CORCON<0>	ffset address Operand AW escribed in ters Wm and or the result. destination. destination. eration. rite Back de	ing as descr /B specifies t Wn for the r stination.	ibed in the optiona nultiply.

```
Example 1:
```

MSC W6\*W7, A, [W8]-=4, W6, [W10]-=4, W7
; Multiply W6\*W7 and subtract the result from ACCA
; Fetch [W8] to W6, Post-decrement W8 by 4
; Fetch [W10] to W7, Post-decrement W10 by 4

; CORCON = 0x0001 (integer multiply, no saturation)

	Before Instruction		After Instruction
W6	9051	W6	D309
W7	7230	W7	100B
W8	0C00	W8	0BFC
W10	1C00	W10	1BFC
ACCA	00 0567 8000	ACCA	00 3738 5ED0
Data 0C00	D309	Data 0C00	D309
Data 1C00	100B	Data 1C00	100B
CORCON	0001	CORCON	0001
SR	0000	SR	0000

Example 2:

MSC W4\*W5, B, [W11+W12], W5, W13

; Multiply W4\*W5 and subtract the result from ACCB

; Fetch [W11+W12] to W5

; Write Back ACCA to W13

; CORCON = 0x0000 (fractional multiply, no saturation)

	Before
	Instruction
W4	0500
W5	2000
W11	1800
W12	0800
W13	6233
ACCA	00 3738 5ED0
ACCB	00 1000 0000
Data 2000	3579
CORCON	0000
SR	0000

	After Instruction
W4	0500
W5	3579
W11	1800
W12	0800
W13	3738
ACCA	00 3738 5ED0
ACCB	00 0EC0 0000
Data 2000	3579
CORCON	0000
SR	0000

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MUL{.B}	f			
Operands:	f ∈ [0 8	191]				
Operation:	For word o	)<7:0> * (f)<7				
Status Affected:	None					
Encoding:	1011	1100	OBOf	ffff	ffff	ffff
Description:	register ar and the re	e default worl d place the re sult are interp	esult in the V preted as uns	V2:W3 registe signed intege	er pair. Both rs. If this ins	operands truction is
Description:	register an and the re executed i the most s least signi The 'B' bit	Id place the re sult are interp n Byte mode, ignificant wor icant word of selects byte o	esult in the V preted as uns the 16-bit re d of the 32-bit the 32-bit re pr word oper	V2:W3 registe signed intege esult is stored bit result is stored esult is stored ation ('0' for	er pair. Both rs. If this insi I in W2. In W ored in W3, a I in W2.	operands truction is ford mode and the
Description:	register an and the re executed i the most s least signi The 'B' bit	d place the re sult are interp n Byte mode, ignificant word cant word of selects byte of select the ad The extension rather than a denote a word The WREG in The IF bit (C This is the or	esult in the V preted as uns the 16-bit re d of the 32-bit the 32-bit re or word oper dress of the m . B in the in word operation, s set to work ORCON<0>	V2:W3 register signed intege esult is stored bit result is stored ation ('0' for file register. Instruction de tion. You may but it is not r king register N ), has no effe	er pair. Both rs. If this insi I in W2. In W ored in W3, a I in W2. word, '1' for notes a byte / use a . w ex required. W0. ect on this op	operands truction is 'ord mode and the byte). operation ctension to eration.
Description: Words:	register ar and the re executed i the most s least signi The 'B' bit The 'f' bits <b>Note 1:</b> 2: 3:	d place the re sult are interp n Byte mode, ignificant word cant word of selects byte of select the ad The extension rather than a denote a word The WREG in The IF bit (C	esult in the V preted as uns the 16-bit re d of the 32-bit the 32-bit re or word oper dress of the m . B in the in word operation, s set to work ORCON<0>	V2:W3 register signed intege esult is stored bit result is stored ation ('0' for file register. Instruction de tion. You may but it is not r king register N ), has no effe	er pair. Both rs. If this insi I in W2. In W ored in W3, a I in W2. word, '1' for notes a byte / use a . w ex required. W0. ect on this op	operands truction is 'ord mode and the byte). operation ctension to eration.

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1: MUI	L.B 0x800	; Multiply (	(0x800)*WREG	(Byte mode)
	Before Instruction		After Instruction	
WREG (W0)	9823	WREG (W0)	9823	
W2	FFFF	W2	13B0	
Wa	FFFF	W3	FFFF	
Data 0800	2690	Data 0800	2690	
SR	0000	SR	0000	

## 16-bit MCU and DSC Programmer's Reference Manual

Example 2: MUL	TMR1	; Multiply (	TMR1)*WREG	(Word mode)
I	Before nstructior	1	After Instruction	
WREG (W0)	F001	WREG (W0)	F001	
W2	0000	W2	C287	
W3	0000	W3	2F5E	
TMR1	3287	TMR1	3287	
SR	0000	SR	0000	

Implemente	d in:	PIC24	F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х		Х	Х	Х	Х	Х
Syntax:		{label:}		MUL.SS	Wb,	Ws,	Wnd	
						[Ws],		
						[Ws++],		
						[Ws],		
						[++Ws],		
						[Ws],		
Operands:		Wb ∈ [\ Ws ∈ [\ Wnd ∈	V0		W12]			
Operation:		signed (	Wb	) * signed (V	Vs) →Wnd:W	Vnd + 1		
	ted <sup>.</sup>	None						
Status Affec	.00.							
Status Affec Encoding: Description:		1011 Multiply result in the resu register	the two ilt is ), ai	o successive s stored in W nd the most :	working reg nd (which m significant w	wddd contents of V gisters. The le oust be an ev ord of the res	east significa en numberec sult is stored	nt word of d working in
Encoding:		1011 Multiply result in the resu register Wnd + two's co used for	the two ult is ), au 1. B omp r W	contents of o successive s stored in W nd the most soth source c lement signe	Wb with the working reg nd (which m significant w perands and ed integers. I	contents of V gisters. The le	Ws, and store east significa en numbered sult is stored /nd are inter ct addressing	e the 32-bit nt word of d working in preted as g must be
Encoding:		1011 Multiply result in the result register Wnd + 2 two's co used for may be The 'w' The 'd' I The 'd' I	the two lilt is ), ai 1. B omp r W use bits bits	e contents of o successive s stored in W nd the most s oth source c lement signe b and Wnd. I ed for Ws.	Wb with the working reg nd (which m significant w perands and ad integers. I Register dire ddress of the ddress of the burce Addres	contents of N gisters. The le just be an ev ord of the result W Register dire- ect or register e base regist e lower destir ss mode.	Ws, and store east significa en numbered sult is stored /nd are inter ct addressing indirect add	e the 32-bit nt word of d working in preted as g must be ressing
Encoding:		1011 Multiply result in the result register Wnd + 2 two's co used for may be The 'w' The 'd' I The 'd' I	the two it is it is it. B omp r W use bits bits bits bits	e contents of o successive s stored in W and the most s oth source of lement signe b and Wnd. I ed for Ws. s select the a select the a select the so	Wb with the working reg nd (which m significant w perands and ed integers. I Register dire ddress of the ddress of the burce Addres	contents of N gisters. The le just be an ev ord of the result W Register dire- ect or register e base regist e lower destir ss mode.	Ws, and store east significa en numbered sult is stored /nd are inter ct addressing indirect add er. er. nation registe	e the 32-bit nt word of d working in preted as g must be ressing
Encoding:		1011 Multiply result in the resu register Wnd + two's co used for may be The 'w' The 'd' I The 'p' I The 's' I	the two llt is ), an 1. B omp r W use bits bits bits bits	e contents of o successive s stored in W nd the most s oth source of lement signe b and Wnd. I ed for Ws. s select the a select the a select the so select the so This instructi Since the pro an even work	Wb with the working reg nd (which m significant w perands and ad integers. I Register dire ddress of the ddress of the burce Addres ource register on operates oduct of the r king register.	contents of N gisters. The le just be an ev ord of the result W Register dire- ect or register e base regist e lower destir ss mode. er. in Word moo multiplication . See Figure	Ws, and store east significa en numbered sult is stored /nd are interp ct addressing indirect add er. hation registe de only. is 32 bits, Wi 4-2 for inform	e the 32-bit nt word of d working in preted as g must be ressing er.
Encoding:		1011 Multiply result in the resu register Wnd + 3 two's co used for may be The 'w' The 'd' 1 The 'd' 1 The 's' 1 Note 1	the two llt is ), and 1. B omp r W use bits bits bits bits bits	e contents of o successive s stored in W nd the most s oth source of lement signed b and Wnd. I ed for Ws. s select the a select the a select the a select the so select the so This instructi Since the pro- an even work how double	Wb with the working reg nd (which m significant w perands and ad integers. I Register dire ddress of the ddress of the burce Addres ource register on operates burce of the r king register words are ali	contents of N gisters. The le just be an ev ord of the result W Register dire- ect or register e base regist e lower destir ss mode. er. in Word moo nultiplication	Ws, and store east significa en numbered sult is stored /nd are inter ct addressing indirect add er. hation registe de only. is 32 bits, Wi 4-2 for inform nory.	e the 32-bit nt word of d working in poreted as g must be ressing er.
Encoding:		1011 Multiply result in the resu register Wnd + 2 two's co used for may be The 'w' The 'd' I The 'c' I The 's' I Note 1 2	the two llt is ), and 1. B omp r W bits bits bits bits bits bits	e contents of o successive s stored in W and the most s oth source of lement signed b and Wnd. I ed for Ws. s select the a select the a select the so select the so This instructi Since the pro an even work how double w Wnd may no The IF bit and	Wb with the working reg nd (which m significant w perands and ed integers. I Register dire ddress of the ddress of the ddress of the burce Addres burce registe on operates bduct of the r king register words are ali t be W14, si d the US<1:0	contents of N pisters. The le nust be an ev ord of the result V Register dire- ect or register e base regist e lower destir ss mode. er. in Word moo nultiplication . See Figure igned in men nce W15<0> D> bits in the	Ws, and store east significa en numbered sult is stored /nd are interp ct addressing r indirect add er. hation register de only. is 32 bits, Wi 4-2 for inform hory. is fixed to ze	e the 32-bit nt word of d working in oreted as g must be ressing er.
Encoding:		1011 Multiply result in the resu register Wnd + 2 two's co used for may be The 'w' The 'd' I The 'd' I The 'c' I Note 1 2 3	the two llt is ), and 1. B omp r W bits bits bits bits bits bits	e contents of o successive s stored in W nd the most s oth source of lement signe b and Wnd. I ed for Ws. s select the a select the a select the so select the so This instructi Since the pro an even work how double w	Wb with the working reg nd (which m significant w perands and ed integers. I Register dire ddress of the ddress of the ddress of the burce Addres burce registe on operates bduct of the r king register words are ali t be W14, si d the US<1:0	contents of N pisters. The le nust be an ev ord of the result V Register dire- ect or register e base regist e lower destir ss mode. er. in Word moo nultiplication . See Figure igned in men nce W15<0> D> bits in the	Ws, and store east significa en numbered sult is stored /nd are interp ct addressing r indirect add er. hation register de only. is 32 bits, Wi 4-2 for inform hory. is fixed to ze	e the 32-bit nt word of d working in oreted as g must be ressing er.

#### 

Example 1:

	; Store the result to W12:W13
Before Instruction W0 9823 W1 67DC W12 FFFF W13 FFFF SR 0000	After           Instruction           W0         9823           W1         67DC           W12         D314           W13         D5DC           SR         0000
Example 2: MUL.SS W2, [W	N4], W0 ; Pre-decrement W4 ; Multiply W2*[W4] ; Store the result to W0:W1
Before	After
Instruction	Instruction
W0 FFFF	W0 28F8
W1 FFFF	W1 0000
W2 0045	W2 0045
W4 27FE	W4 27FC
Data 27FC 0098	Data 27FC 0098
SR 0000	SR 0000

MUL.SS W0, W1, W12 ; Multiply W0\*W1

MUL.SS		Integer 16 Destination		ned Multiply v	with Accum	ulator
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
						Х
Syntax:	{label:}	MUL.SS	Wb,	Ws,		
-				[Ws],	А	
				[Ws++],	В	
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 ACC ∈ [A,	) W15]				
Operation:	_	/b) * signed (\	Ws) →ACC(/	A or B)		
Status Affected:	None	·				
Encoding:	1011	1001	lwww	w111	Appp	SSSS
Description:	stored in or 32-bit resul target accu	one of the DS ult is sign exte umulator.	SP engine ac tended to bit 3	nultiply with a ccumulators, <i>F</i> 39 prior to be as integer or fr	ACCA or ACC eing loaded in	CB. The nto the
	depending	g upon the op	perating mod	as integer or fr de of the DSP ource operanc	engine (as d	defined by
	The 'd' bits The 'p' bits	s select the a select sourc	address of the ce Address r	ne base regist ne source regis mode 2. nccumulator fo	ister.	t.
				es in Word mo	-	
		The state o	of the multipli	lier mode bits e operation of	(US<1:0> in	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo	C33E and PIC2 odify-write oper see <b>Note 3</b> in S	rations on no	on-CPU Spec	cial Function R	Registers. For	
Example 1: MUI	JL.SS W0, W1	., A				
	Before			After		
Wo	Instructio	on 9823	WO	Instruction 982	ادر	
W1		9823 67DC	W1	982 67D0		
Acc A	_			F D5DC D31		
SR			SR		1	

 $\ensuremath{\textcircled{}^\circ}$  2005-2011 Microchip Technology Inc.

Instruction Descriptions

MUL.SU	J	Integer 16x	16-bit Signe	ed-Unsigned	I Short Liter	al Multiply
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MUL.SU	Wb,	#lit5,	Wnd	
Operands:	lit5 ∈ [0.	0 W15] 31] V0, W2, W4	W12]			
Operation:	signed (V	Vb) * unsigned	lit5 →Wnd:\	Vnd + 1		
Status Affected:	None					
Encoding:	1011	1001	0www	wddd	d11k	kkkk
	The Wb c complem integer. R The 'w' b The 'd' bi	and the most s operand and th ent signed inte degister direct its select the a ts select the a ts define a 5-b	e result Who eger. The lite addressing r ddress of the ddress of the	d are interpre ral is interpre nust be used e base regist e lower destir	eted as a two eted as an un for Wb and er. nation registe	's Isigned Wnd.
	Note 1: 2: 3: 4:	This instruction Since the pro- an even work how double work Wnd may no The IF bit an no effect on	on operates oduct of the r king register. words are ali t be W14, si d the US<1:0	in Word moo nultiplication See Figure gned in merr nce W15<0> )> bits in the	le only. is 32 bits, Wi 4-3 for inforn hory. is fixed to ze	nation on ero.
Words:	1					
Cycles:	1					
Example 1:	MUL.SU W0, #	0x1F, W2 ; ;		0 by liter result to 1		
	Before Instruction W0 C000 W2 1234 W3 C9BA SR 0000	W W S	/2 4000			

Example 2:	MUL.	SU W2,	#0x10, W0			W2 by literal 0x10 result to W0:W1
		Before			After	
	I	nstructior	า		Instructio	n
	W0	ABCD		W0	2400	
	W1	89B3		W1	000F	
	W2	F240		W2	F240	
	SR	0000		SR	0000	

5

Syntax: Operands:	{lab	X pel:}	X MUL.SU	X Wb,	X	Х	Х		
	{lat	el:}	MUL.SU	Wb,					
Operands:					Ws,	Wnd			
Operands:					[Ws],				
Operands:					[Ws++],				
Operands:					[Ws],				
Operands:					[++Ws],				
Operands:					[Ws],				
	Ws	∈ [W0	W15] W15] 0, W2, W4	W12]					
Operation:	sigi	ned (W	b) * unsigned	(Ws) →Wno	d:Wnd + 1				
Status Affecte	ed: Noi	ne							
Encoding:	1	.011	1001	0www	wddd	dppp	SSSS		
	reg The con uns	ister), a Wb op npleme igned i	nd the most s berand and th nt signed inte nteger. Regis	significant wa he result Wh eger. The We ster direct ad	nust be an even ord of the res d are interpre s operand is i dressing mus rect addressi	ult is stored i ited as a two nterpreted as st be used fo	n Wnd + 1. 's s an r Wb and		
	The The	The 'w' bits select the address of the base register. The 'd' bits select the address of the lower destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.							
	No	te 1: 2:	Since the pro an even wor	oduct of the i king register	in Word moo multiplication . See Figure igned in mem	is 32 bits, Wi 4-3 for inforn			
		3: 4:		d the US<1:	nce W15<0> 0> bits in the n.				
Words:	1								
Cycles:	1 <sup>(1)</sup>								

DS70157F-page 310

		; Store the result to WU:WI
W0 W1 W8 W9 Data 178C SR	Before nstruction 68DC AA40 F000 178C F000 0000	After           Instruction           W0         0000           W1         F100           W8         F000           W9         178C           Data 178C         F000           SR         0000
Example 2: MUL	.SU W2,	<pre>[++W3], W4 ; Pre-Increment W3 ; Multiply W2*[W3] ; Store the result to W4:W5</pre>
W2 W3 W4 W5 Data 0282 SR	Before nstruction 0040 0280 1819 2021 0068 0000	After Instruction W2 0040 W3 0282 W4 1A00 W5 0000 Data 0282 0068 SR 0000

Example 1: MUL.SU W8, [W9], W0 ; Multiply W8\*[W9] ; Store the result to W0:W1

Instruction Descriptions

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
						Х			
Syntax:	{label:}	MUL.SU	Wb,	Ws,					
				[Ws],	А				
				[Ws++],	В				
				[Ws],					
				[++Ws],					
				[Ws],					
Operands:	Wb ∈ [W0 Ws ∈ [W0 ACC ∈ [A;	W15]							
Operation:	signed (W	b) * unsigned	I (Ws) →ACC	C(A or B)					
Status Affected:	None								
Encoding:	1011	1001	0www	w111	Appp	SSSS			
Description:	Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators, ACCA or ACCB. The 32-bit result is sign extended to bit 39 prior to being loaded into the target accumulator.								
	depending the IF bit in two's com	e operands a J upon the op n CORCON< plement sign d as an unsig	erating mode 0>). The first ed value and	e of the DSP t source oper	engine (as d and is interp	efined by reted as a			
	The 'd' bits The 'p' bits	s select the a s select the a s select sourc selects the d	ddress of the ce Address n	e source regis node 2.	ster.				
	Note 1: 2:	The state of	the multiplie	in Word moo r mode bits ( operation of t	US<1:0> in C				
Words:	1								
Cycles:	1 <sup>(1)</sup>								

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

- - -

Example 1: MUL.SU W8, W9, A

	Before Instruction		After Instruction
W8	F000	W8	F000
W9	F000	W9	F000
Acc A	00 0000 0000	Acc A	FF F100 0000
SR	0000	SR	0000

5

Instruction Descriptions

MUL.SU			16-bit Signe nulator Dest		Short Liter	al Multiply			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33			
						Х			
Syntax:	{label:}	MUL.SU	Wb,	#lit5,	A B				
Operands:	Wb ∈ [Wb lit5 ∈ [0 ACC ∈ [A	-							
Operation:	signed (W	/b) * unsigned	l (lit5) →ACC	(A or B)					
Status Affected:	None								
Encoding:	1011	1001	0www	w111	Allk	kkkk			
	32-bit res accumula The sourd dependin the IF bit two's com	<ul> <li>stored in one of the DSP engine accumulators, ACCA or ACCB. The 32-bit result is sign extended to bit 39 prior to being loaded into the target accumulator.</li> <li>The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON&lt;0&gt;). The first source operand is interpreted as a two's complement signed value and the second source operand is interpreted as an unsigned value.</li> </ul>							
	The 'k' bit	The 'w' bits select the address of the base register. The 'k' bits select the 5-bit literal value. The 'A' bit selects the destination accumulator for the product.							
	Note 1: 2:	The state of	the multiplie	r mode bits (	-				
Words:	1								
Cycles:	1								
Example 1:	MUL.SU W8, #	0x02, A							
	Befor Instruct W8		V8	After nstruction 0042	2				

Acc A

SR

00 0000 0084

0000

00 0000 0000

0000

Acc A SR

#### Integer 16x16-bit Signed-Unsigned Short Literal Multiply

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MUL.US	Wb,	Ws,	Wnd	
				[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb∈[W0					
	$Ws \in [W0]$	0 W15] /0, W2, W4	\\/10]			
Operation:	-	(Wb) * signed	-	d·Wnd + 1		
Status Affected:	None	() e.ge	(110) / //			
Encoding:	1011	1000	1www	wddd	dqqb	SSSS
	The Wb o and the re integer. Re	and the most s perand is inte sult Wnd are egister direct lirect or regist	rpreted as a interpreted a a a	n unsigned ir as a two's coi must be used	iteger. The W mplement sig for Wb and	/s operand jned Wnd.
	The 'w' bit The 'd' bit The 'p' bit	s select the a s select the a s select the s s select the so	ddress of th ddress of the ource Addre	e base regist e lower destir ss mode.	er.	
	Note 1:	This instruct	ion operates	in Word mod	de only.	
	2:	an even wor	king register	multiplication See Figure igned in men	4-3 for inforn	
	3:			nce W15<0>	-	ero.
	4:		d the US<1:	0> bits in the		
Words:	1					
Cycles:	1(1)					
read-mod	dify-write ope	24E devices, 1 rations on nor <b>Section 3.2.1</b>	n-CPU Speci	al Function R	egisters. For	

#### ....

Instruction Descriptions

### 16-bit MCU and DSC Programmer's Reference Manual

Example 1:

<u> </u>		; Sto	ore the result to W2:W3	
	Before		After	
I	nstructior	n l	Instruction	
W0	C000	W0	C000	
W1	2300	W1	2300	
W2	00DA	W2	0000	
W3	CC25	W3	F400	
Data 2300	F000	Data 2300	F000	
SR	0000	SR	0000	
Example 2: MUL	.US W6,	; 5	Mult. W6*[W5] (unsigned-signe Store the result to W10:W11 Post-Increment W5	:d)
	Before		After	
I	nstructior	n l	Instruction	
W5	0C00	W5	0C02	
W6	FFFF	W6	FFFF	
W10	0908	W10	8001	
W11	6EEB	W11	7FFE	
Data 0C00	7FFF	Data 0C00	7FFF	
SR	0000	00	0000	
	0000	SR	0000	

MUL.US W0, [W1], W2 ; Multiply W0\*[W1] (unsigned-signed)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
						Х
Syntax:	{label:}	MUL.US	Wb,	Ws,	A	
				[Ws],	В	
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 ACC ∈ [A,	W15]				
Operation:	unsigned (	Wb) * signed	(Ws) →ACC	C(A or B)		
Status Affected:	None					
Encoding:	1011	1000	0www	w111	Appp	SSSS
Description:	stored in o	a 16-bit x 16-b ne of the DSI It is sign exte or.	P engine acc	cumulators, A	CCA or ACC	B. The
	depending the IF bit ir unsigned v	e operands an upon the open CORCON< value and the nt signed value	erating mode 0>). The first second sour	e of the DSP source oper	engine (as d and is interp	efined by reted as a
	The 'd' bits The 'p' bits	s select the a s select the ac s select sourc selects the de	ddress of the e Address m	e source regis node 2.	ster.	
	Note 1: 2:	This instructi The state of have no effe	the multiplie	r mode bits (	US<1:0> in C	
NA7 1	1					
Words:						

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

# Integer 16x16-bit Unsigned-Signed Multiply with

Example 1: אַטא	.US W0, W1, B		
	Before Instruction		After Instruction
WC	C000	W0	0000
W1	F000	W1	F000
Acc B	00 0000 0000	Acc B	FF F400 0000
SR	0000	SR	0000

Implemented in	i: PIC2	24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	(	Х	Х	Х	Х	Х
Syntax:	{label:	:}	MUL.UU	Wb,	#lit5,	Wnd	
Operands:	lit5 ∈	[0 3	W15] 31] ), W2, W4	W12]			
Operation:	unsig	ned (V	Nb) * unsign	ed lit5 →Wno	<b>d:Wnd +</b> 1		
Status Affected	: None						
Encoding:	101	11	1000	0www	wddd	d11k	kkkk
	regista Both c Regis The 'v The 'c	ter), an operar ster dire w' bits d' bits	nd the most s nds and the r rect addressi select the ac select the ac	significant wo result are into ing must be u ddress of the ddress of the	ord of the rest terpreted as u used for Wb a base registe lower destin	er. nation registe	n Wnd + 1. egers.
				C C	nteger literal.		
	Note			-	in Word mod	de only. is 32 bits, Wr	lanuat ba
		a h 3: V 4: T	an even work how double v Wnd may not The IF bit and	king register. words are alig ot be W14, sir	See Figure 4 gned in mem nce W15<0> )> bits in the 0	4-3 for inform	nation on ero.
Words:	1		10 011011	110 0			
Cycles:	1						
Example 1:	Before Instructio	e	;	Store the After Instruction	70 by litera result to W		
	W0         2323           W12         4512           W13         7821           SR         0000	2	W W1 W1 SI	2 0F0D 3 0002			
Example 2:	MUL.UU W7	', #0x			17 by litera result to W		
	Before Instruction W0 780E W1 3805 W7 F240	ion B 5	W W W	/1 001D			

#### ----. . . .

Instruction Descriptions

. . . . . . . . . . . .

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	MUL.UU	Wb,	Ws,	Wnd	
				[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W		W12]			
Operation:	unsigned (	(Wb) * unsign	ed (Ws) →W	Vnd:Wnd + 1		
Status Affected:	None					
Encoding:	1011	1000	0www	wddd	dppp	SSSS
Description:	result in tw the result i register), a Wnd + 1. I unsigned i	e contents of vo successive s stored in W and the most Both source on ntegers. Reg ister direct or	working reg nd (which m significant w perands and ister direct a	gisters. The le nust be an ev ord of the res d the result a ddressing mu	east significa en numbered sult is stored re interpreted ust be used f	nt word of d working in d as or Wb and
	The 'd' bits The 'p' bits	s select the a s select the ac s select the so s select the so	ddress of the ource Addre	e lower destir ss mode.		er.
	Note 1: 2:	This instructi Since the pro	•		-	nd must be
	£.		king register	See Figure	4-3 for inform	
	3:	Wnd may no		•	•	ero.
	4:	The IF bit and no effect on			CORCON re	gister have
Words:	1		•			

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

I	Before nstructior	1	After Instruction
W0	FFFF	W0	FFFF
W2	2300	W2	0001
W3	00DA	W3	FFFE
W4	FFFF	W4	FFFF
SR	0000	SR	0000

Example 1: MUL.UU W4, W0, W2 ; Multiply W4\*W0 (unsigned-unsigned)
; Store the result to W2:W3

Example 2:

MUL.UU W0, [W1++], W4 ; Mult. W0\*[W1] (unsigned-unsigned) ; Store the result to W4:W5

; Post-Increment W1

	Before		After
I	nstructior	1	Instruction
W0	1024	W0	1024
W1	2300	W1	2302
W4	9654	W4	6D34
W5	BDBC	W5	0D80
Data 2300	D625	Data 2300	D625
SR	0000	SR	0000

5

MUL.UU		Destination			y with Accu	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
						Х
Syntax:	{label:}	MUL.UU	Wb,	Ws,	A	
				[Ws],	В	
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 ACC ∈ [A,	W15]				
Operation:	unsigned (	Wb) * unsign	ed (Ws) →A	CC(A or B)		
Status Affected:	None					
Encoding:	1011	1000	0www	w111	Appp	SSSS
Description:	stored in o	a 16-bit x 16-l ne of the DS Ilt is zero exte umulator.	P engine aco	cumulators, A	ACCA or ACC	B. The
	depending	e operands a upon the open CORCON </td <td>erating mode</td> <td>e of the DSP</td> <td>engine (as d</td> <td>efined by</td>	erating mode	e of the DSP	engine (as d	efined by
	The 'd' bits The 'p' bits	s select the a s select the ad s select sourc selects the do	ddress of the e Address m	e source regis node 2.	ster.	
	Note 1: 2:	This instructi The state of have no effe	the multiplie	r mode bits (	US<1:0> in C	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mod	ify-write oper	24E devices, t ations on nor Section 3.2.1	-CPU Specia	al Function R	egisters. For	

Example 1: MUL.UU W4, W0, B

	Before Instruction		After Instruction
W0	FFFFF	W0	FFFFF
W4	FFFFF	W4	FFFFF
Acc B	00 0000 0000	Acc B	FF FFFE 0001
SR	0000	SR	0000

MUL.UU			x16-bit Unsig tor Destinati		Literal Multij	oly with
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
						Х
Syntax:	{label:}	MUL.UU	Wb,	#lit5,	A B	
Operands:	Wb ∈ [W0 lit5 ∈ [0 ACC ∈ [A	. 31]				
Operation:	-	,	ned (lit5) →A	CC(A or B)		
Status Affected:	None		· · ·			
Encoding:	1011	1000	0www	w111	Allk	kkkk
Description:	stored in c	a 16-bit x 16- one of the DS ult is zero exte cumulator.	SP engine aco	cumulators, A	ACCA or ACC	CB. The
	depending	ce operands a g upon the op in CORCON< values.	erating mode	e of the DSP	engine (as d	lefined by
	The 'w' bit The 'k' bit	ts select the a s select the 5 t selects the d	-bit literal.	-		i.
	Note 1: 2:	The state of	tion operates the multiplie ect upon the o	r mode bits (	US<1:0> in C	
Words:	1					
Cycles:	1					
Example 1: MUL		0x02, A				
	Before Instructi	ion		After Instruction	2	
W8 Acc A SR	00 0000	0042 00000 0000	W8 Acc A SR	004 00 0000 008 000	4	

. . . . . . . . . . .

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·			Х			Х
Syntax:	{label:}	MULW.SS	Wb,	Ws,	Wnd	
				[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb∈ [W0					
	Ws∈ [W0 Wnd∈ [W	W15] 0, W2, W4	W12]			
Operation:	signed (W	b) * signed (V	Vs) →Wnd			
Status Affected:	None					
Encoding:	1011	1001	lwww	wddd	dppp	SSSS
	used for W The 'w' bit	d Wnd. Regis /s. s select the a	ddress of th	e base regist	er.	ig may be
					lation registe	er.
	The 'p' bits	s select the so s select the so	ource Addre	ss mode.	lation registe	er.
	The 'p' bits	s select the so s select the so This instructi	ource Addre ource registe on operates	ss mode. er. in Word mod	le only.	ır.
	The 'p' bits The 's' bits Note 1: 2:	s select the so select the so This instructi Wnd must be	ource Addre ource registe on operates e an even we	ss mode. er. in Word mod orking registe	de only. er.	
	The 'p' bits The 's' bits Note 1:	s select the so select the so This instructi Wnd must bo Wnd may no The IF bit an	ource Addre ource registe on operates e an even we t be W14, si d the US<1:0	ss mode. er. in Word mod orking registe nce W15<0> D> bits in the	le only. er. is fixed to ze	ero.
Words:	The 'p' bits The 's' bits <b>Note 1:</b> 2: 3:	s select the so select the so This instructi Wnd must bo Wnd may no	ource Addre ource registe on operates e an even we t be W14, si d the US<1:0	ss mode. er. in Word mod orking registe nce W15<0> D> bits in the	le only. er. is fixed to ze	ero.
Words: Cycles:	The 'p' bits The 's' bits Note 1: 2: 3: 4:	s select the so select the so This instructi Wnd must bo Wnd may no The IF bit an	ource Addre ource registe on operates e an even we t be W14, si d the US<1:0	ss mode. er. in Word mod orking registe nce W15<0> D> bits in the	le only. er. is fixed to ze	ero.
Cycles: Note 1: In dsPIC33 read-modif	The 'p' bits The 's' bits Note 1: 2: 3: 4: 1 1(1) BE and PIC2 fy-write oper	s select the so select the so This instructi Wnd must bo Wnd may no The IF bit an	burce Addre burce registe on operates e an even we t be W14, si d the US<1:0 this operatio he listed cyc n-CPU Speci	ss mode. er. orking registe nce W15<0> 0> bits in the n. ele count does al Function R	de only. er. is fixed to ze CORCON rea s not apply to egisters. For	ero. gister have read and
Cycles: Note 1: In dsPIC33 read-modif	The 'p' bits The 's' bits Note 1: 2: 3: 4: 1 1(1) 3E and PIC2 fy-write oper e Note 3 in 5	s select the so select the so This instruction Wnd must be Wnd may no The IF bit an no effect on 24E devices, to section 3.2.1	burce Addre burce registe on operates e an even we t be W14, si d the US<1:0 this operatio he listed cyc h-CPU Speci "Multi-Cyc	ss mode. er. in Word mod orking registe nce W15<0> D> bits in the n. le count does al Function R le Instruction	de only. er. is fixed to ze CORCON reg on t apply to egisters. For ns".	ero. gister have read and
Cycles: Note 1: In dsPIC33 read-modif details, see <u>Example 1:</u> MULW	The 'p' bits The 's' bits Note 1: 2: 3: 4: 1 1(1) 3E and PIC2 fy-write oper e Note 3 in 5	s select the so select the so This instruction Wnd must be Wnd may no The IF bit an no effect on 24E devices, to section 3.2.1	burce Addre burce registe on operates e an even we t be W14, si d the US<1:0 this operatio he listed cyc h-CPU Speci "Multi-Cyc	ss mode. er. in Word mod orking registe nce W15<0> 0> bits in the n. le count does al Function R le Instruction 0*W1 result to W	de only. er. is fixed to ze CORCON reg on t apply to egisters. For ns".	ero. gister have read and

W1

W12

SR

67DC

D314

0000

W1

W12

SR

67DC

FFFF

0000

Example 2: MUL	W.SS W2,	. [W4], WO	;	Multip	crement W4 ly W2*[W4] the result t	:o W0
	Before			After		
	Instruction		l	nstructior	า	
W0	FFFF		W0	28F8		
W2	0045		W2	0045		
W4	27FE		W4	27FC		
Data 27FC	0098	Data 27	7FC	0098		
SR	0000		SR	0000		

MULW.SU		Integer 16x Result	16-bit Signe	ea-Unsigned	l Multiply wi	tn 16-bit
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MUL.SU	Wb,	Ws,	Wnd	
				[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W		W12]			
Operation:	signed (Wi	o) * unsigned	(Ws) →Wnd			
Status Affected:	None					
Encoding:	1011	1001	0www	wddd	dppp	SSSS
	unsigned in Wnd. Regi The 'w' bits The 'd' bits The 'p' bits	nt signed inte nteger. Regis ster direct or s select the a s select the ac s select the so select the so	ter direct add register indir ddress of the ddress of the burce Addres	dressing mus ect addressi base regist lower destir ss mode.	st be used for ng may be us er.	r Wb and sed for Ws
		This instructi	-		le only.	
		Wnd must be	•		•	
		Wnd may no The IF bit and				
		no effect on t				<u></u>
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-modif	y-write oper	4E devices, the ations on non <b>Section 3.2.1</b>	-CPU Specia	al Function R	egisters. For	
Example 1: MULW	.SU W8, [	W9], WO	-	y W8*[W9] the result	to WO	
	Before		After			
-	struction					
W0 W8	68DC F000	W W				
W9	178C	W				
Data 178C	F000	Data 178				

Example 2: MUL	w.su w2,		Multiply	
	Before		After	
	Instruction		Instruction	n
W2	0040	W2	0040	
W3	0280	W3	0282	
W4	1819	W4	1A00	
Data 0282	0068	Data 0282	0068	
SR	0000	SR	0000	

MULW.SU		Integer 16x with 16-bit		ed-Unsigned	Short Liter	al Multiply
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MULW.SU	Wb,	#lit5,	Wnd	
Operands:	Wb ∈ [W0 lit5 ∈ [0					
	-	/0, W2, W4	-			
Operation:		b) * unsigned	(lit5) →Wnd			
Status Affected:	None	-	1	1	1	1
Encoding:	1011	1001 le contents of	0www	wddd	d11k	kkkk
	The Wb o compleme Wb and W	ng register, wh perand and th ent signed inte /nd. cs select the a	ne result Wno eger. Registe	d are interpre r direct addre	eted as a two essing must l	's
	The 'd' bit	s select the a s select the 5	ddress of the	e lower destir		er.
	Note 1:	This instruct	on operates	in Word mod	de only.	
	2:	Wnd must be				
	3:	•			is fixed to ze	
	4:	no effect on			CORCON re	gister have
Words:	1		•			
Cycles:	1					
Example 1: MULW	V.SU W8, ‡	#0x04, W0	-	ly W8 * #0x0 the result		
 W0 W8 SR	Before nstruction 68DC 1000 0000	W W S		I		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MULW.US	Wb,	Ws,	Wnd	
,			·	[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W		. W12]			
Operation:	unsigned (	(Wb) * signed	l (Ws) →Wnc	ł		
Status Affected:	None					
Encoding:	1011	1000	lwww	wddd	dppp	SSSS
	The 'd' bits The 'p' bits The 's' bits	s select the a s select the ac s select the so s select the so This instructi	ddress of the ource Addres ource registe	e lower destir ss mode. er.	nation registe	ır.
		Wnd must be	=		-	
		Wnd may no				ero.
	4:	The IF bit and no effect on t			CORCON re	gister have
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-modif	fy-write oper	24E devices, t rations on nor <b>Section 3.2.1</b>	n-CPU Specia	al Function R	egisters. For	
Example 1: MULW	.US W0, [	[W1], W2 ; M ; S		)*[W1] (uns result to W		èd )
	Before Instruction C000 2300	W		1		

 $\ensuremath{\textcircled{}^{\circ}}$  2005-2011 Microchip Technology Inc.

Instruction Descriptions

Example 2: MULT	W.US W6,	[W5++], W10	; Store	W6*[W5] (unsigned-signed) the result to W10 ncrement W5
	Before		After	
	Instruction		Instruction	n
W5	0C00	W5	0C02	
W6	FFFF	W6	FFFF	
W10	0908	W10	8001	
Data 0C00	7FFF	Data 0C00	7FFF	
SR	0000	SR	0000	

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MULW.UU	Wb,	Ws,	Wnd	
				[Ws],		
				[Ws++],		
				[Ws],		
				[++Ws],		
				[Ws],		
Operands:	$Wb \in [W0]$					
	Ws∈ [W0		14/4 01			
Operation:	-	0, W2, W4 (Wb) * unsign	-	Vnd		
Status Affected:	None	VVDj J	50 (m.c.)	/nu		
Encoding:	1011	1000	0www	wddd	qqqb	SSSS
Description:		e contents of				
	Wnd. Regin The 'w' bits The 'd' bits The 'p' bits	ntegers. Regi ister direct or s select the ac s select the ac s select the so s select the so	indirect add address of the ddress of the ource Addres	Iressing may e base registe e lower destir ss mode.	be used for V ter.	Ws.
	Note 1:	This instructi	ion operates	in Word mod	de only.	
			-	orking registe	-	
		-		ince W15<0>		
		The IF bit and no effect on t		0> bits in the 0 on.	CORCON reg	gister have
	1		-			
Words:						
Words: Cycles:	1 <sup>(1)</sup>					
Cycles: Note 1: In dsPIC33 read-modif	3E and PIC2 fy-write oper	24E devices, t rations on nor Section 3.2.1	n-CPU Speci	ial Function R	Registers. For	
Cycles: Note 1: In dsPIC33 read-modif	3E and PIC2 fy-write oper e <b>Note 3</b> in \$	rations on nor Section 3.2.1	n-CPU Speci I " <mark>Multi-Cyc</mark> l	ial Function R le Instruction	Registers. For ns".	· more
Cycles: Note 1: In dsPIC33 read-modif details, see <u>Example 1:</u> MULW	3E and PIC2 fy-write oper e <b>Note 3</b> in § ערטט א4, א Before	rations on nor Section 3.2.1	n-CPU Speci I "Multi-Cycl ltiply W4*W ore the res After	ial Function R le Instruction W0 (unsigned sult to W2	Registers. For ns".	· more
Cycles: <b>Note 1:</b> In dsPIC33 read-modif details, see <u>Example 1:</u> MULW	3E and PIC2 fy-write oper e <b>Note 3</b> in § 7.00 w4, w Before nstruction	rations on nor Section 3.2.1 VO, W2 ; Mul ; Sto	n-CPU Speci I "Multi-Cycl Itiply W4*W ore the res After Instructior	ial Function R le Instruction W0 (unsigned sult to W2	Registers. For ns".	· more
Cycles: <b>Note 1:</b> In dsPIC33 read-modif details, see <u>Example 1:</u> MULW Ir W0	3E and PIC2 fy-write oper e <b>Note 3</b> in \$ 7.00 w4, w Before nstruction FFFF	rations on nor Section 3.2.1 10, W2 ; Mul ; Sto	n-CPU Speci I "Multi-Cycl Itiply W4*V ore the res After Instructior /0 FFFF	ial Function R le Instruction W0 (unsigned sult to W2	Registers. For ns".	· more
Cycles: <b>Note 1:</b> In dsPIC33 read-modif details, see <u>Example 1:</u> MULW	3E and PIC2 fy-write oper e <b>Note 3</b> in § 7.00 w4, w Before nstruction	rations on nor Section 3.2.1 10, W2 ; Mul ; Sto W W	n-CPU Speci I "Multi-Cycl Itiply W4*W ore the res After Instructior	ial Function R le Instruction W0 (unsigned sult to W2	Registers. For ns".	· more

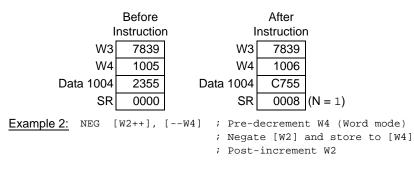
		16-bit Resu			Literal Multi	, with
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	MULW.UU	Wb,	#lit5,	Wnd	
Operands:	lit5 ∈ [0	/0 W15] 31] W0, W2, W4	W12]			
Operation:	unsigned	I (Wb) * unsign	ed →Wnd			
Status Affected:	None					
Encoding:	1011	1000	0www	wddd	d11k	kkkk
	The 'd' b	its select the a its select the a its select the 5-	ddress of the	e lower destin		er.
	Note 1:	This instruct	on operates	in Word mod	de only.	
	2:	Wnd must be	e an even wo	orking registe	er.	
	3:	Wnd may no				
	4:	The IF bit an no effect on			CORCON re	gister have
Words:	1					
Cycles:	1					
	w.uu w4,	#0x04, W2	-	-	insigned-un	signed)
Example 1: MUL			; Store t	the result	to W2	
Example 1: MUL	Before		After	the result	to W2	

	Before		After
I	nstructior	n l	Instructior
W2	2300	W2	4000
W4	1000	W4	1000
SR	0000	SR	0000

NEG		Negate f				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	NEG{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	191]				
Operation:		estination des	signated by [	)		
Status Affected:	DC, N, OV		-			
Encoding:	1110	1110	OBDf	ffff	ffff	ffff
Description:	place the r determines stored in W register. The 'B' bit	he two's com result in the d s the destinat VREG. If WR selects byte selects the d	lestination re tion register. EG is not spe or word oper	gister. The op If WREG is s ecified, the re ration ('0' for	ptional WRE pecified, the soult is stored word, '1' for	G operanc result is d in the file byte).
	The 'f' bits	select the ac	ddress of the	file register.		
		rather than a denote a wo	a word opera rd operation,	tion. You may but it is not r	y use a .w ex required.	-
	<b>.</b>	The WDEC		ving register )	MO	
		THE WREG	is set to work	ang register		
Cycles:	1 1 <b>(1)</b>					road and
Cycles: Note 1: In dsPIC3 read-mod	1 1 <sup>(1)</sup> 33E and PIC2 dify-write oper ee <b>Note 3</b> in \$	24E devices, t rations on nor Section 3.2.1 WREG ; Neg	the listed cyc n-CPU Specia I <b>"Multi-Cycl</b>	le count does al Function R le Instruction	s not apply to egisters. For 1s".	
read-mod details, se <u>Example 1:</u> NEG	1 1(1) 33E and PIC2 dify-write oper ee <b>Note 3</b> in \$ .B 0x880, Before Instruction 9080 2355 0000	24E devices, t rations on nor Section 3.2.1 WREG ; Neg ; Sto WREG (W0 Data 0880 SF	the listed cyc n-CPU Specia ("Multi-Cycl gate (0x880 pre result After Instruction () 90AB 0 2355 R 0008 (N	le count does al Function R le Instruction	e not apply to egisters. For ns". <sup>de )</sup>	
Cycles: Note 1: In dsPIC3 read-mod details, se Example 1: NEG WREG (W0) Data 0880 SR Example 2: NEG	1 1(1) 33E and PIC2 dify-write oper ee <b>Note 3</b> in \$ . B 0x880, Before Instruction 9080 2355 0000 0x1200 Before Instruction 8923	24E devices, t rations on nor Section 3.2.1 WREG ; Neg ; Sto WREG (W0 Data 0880 SF ; Neg Data 1200 SF	the listed cyc n-CPU Specia ("Multi-Cycl gate (0x880 ore result After Instruction 0 2355 R 0008 (N gate (0x120 After Instruction 0 76DD R 0000	le count does al Function R le Instruction )) (Byte mod to WREG V = 1)	e not apply to egisters. For ns". <sup>de )</sup>	
Cycles: Note 1: In dsPIC3 read-mod details, se Example 1: NEG WREG (W0) Data 0880 SR Example 2: NEG I Data 1200 SR NEG	1 1(1) 33E and PIC2 dify-write oper ee Note 3 in \$ .B 0x880, Before Instruction 9080 2355 0000 0x1200 Before Instruction 8923 0000	24E devices, t rations on nor Section 3.2.1 WREG ; Neg ; Sto WREG (W0 Data 0880 SF ; Neg Data 1200 SF Negate Ws	the listed cyc n-CPU Specia ("Multi-Cycl gate (0x880 ore result After Instruction ) 90AB 0 2355 R 0008 (N gate (0x120 After Instruction 0 76DD R 0000	le count does al Function R le Instruction )) (Byte mod to WREG N = 1) )) (Word mod	not apply to egisters. For ns". de )	more
Cycles: Note 1: In dsPIC3 read-mod details, se Example 1: NEG WREG (W0) Data 0880 SR Example 2: NEG	1 1(1) 33E and PIC2 dify-write oper ee Note 3 in \$ . B 0x880, Before Instruction 9080 2355 0000 0x1200 Before Instruction 8923 0000 PIC24F	24E devices, t rations on nor Section 3.2.1 WREG ; Neg ; Sto WREG (W0 Data 0880 SF ; Neg Data 1200 SF Negate Ws PIC24H	the listed cyc n-CPU Specia ("Multi-Cycl gate (0x880 ore result After Instruction 0 2355 R 0008 (N gate (0x120 After Instruction 0 76DD R 0000	le count does al Function R le Instruction )) (Byte mod to WREG N = 1) )0) (Word mod dsPIC30F	ode)	more dsPIC33
Cycles: Note 1: In dsPIC3 read-mod details, se Example 1: NEG WREG (W0) Data 0880 SR Example 2: NEG I Data 1200 SR NEG	1 1(1) 33E and PIC2 dify-write oper ee Note 3 in \$ .B 0x880, Before Instruction 9080 2355 0000 0x1200 Before Instruction 8923 0000	24E devices, t rations on nor Section 3.2.1 WREG ; Neg ; Sto WREG (W0 Data 0880 SF ; Neg Data 1200 SF Negate Ws	the listed cyc n-CPU Specia ("Multi-Cycl gate (0x880 ore result After Instruction ) 90AB 0 2355 R 0008 (N gate (0x120 After Instruction 0 76DD R 0000	le count does al Function R le Instruction )) (Byte mod to WREG N = 1) )) (Word mod	not apply to egisters. For ns". de )	more

NEG		Negate Ws				
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	(Ws) + 1 -	→Wd				
Status Affected:	DC, N, O	/, Z, C				
Encoding:	1110	1010	0Bqq	qddd	dppp the source r	ssss oaistor Wa
Encoding: Description:	Compute and place or indirect The 'B' bit	1010 the two's com the result in t addressing n selects byte s select the de	plement of the destination of th	ne contents of on register We for both Ws a ration ('0' for	the source r d. Either regi and Wd. word, '1' for l	egister Wa ster direct
U U	Compute and place or indirect The 'B' bit The 'q' bit The 'd' bit The 'p' bit	the two's com the result in t addressing n selects byte	plement of the he destination nay be used or word ope estination A estination re pource Addre	ne contents of on register We for both Ws a ration ('0' for ddress mode. gister. ss mode.	the source r d. Either regi and Wd. word, '1' for l	egister Waster direct
U U	Compute and place or indirect The 'B' bit The 'q' bit The 'q' bit The 'p' bit The 's' bit Note: 1	the two's com the result in t addressing n selects byte s select the d s select the d s select the se	plement of the he destination hay be used or word ope estination A estination re burce Addre burce registe . B in the ir word operation	ne contents of on register We for both Ws a ration ('0' for ' ddress mode. gister. ss mode. er. hstruction den on. You may u	the source r d. Either regi and Wd. word, '1' for l otes a byte o use a .w ext	egister Ws ster direct byte).
U U	Compute and place or indirect The 'B' bit The 'q' bit The 'q' bit The 'p' bit The 's' bit Note: 1	the two's com the result in t addressing n s selects byte s select the d s select the d s select the so s select the so s select the so The extension ather than a v	plement of the he destination hay be used or word ope estination A estination re burce Addre burce registe . B in the ir word operation	ne contents of on register We for both Ws a ration ('0' for ' ddress mode. gister. ss mode. er. hstruction den on. You may u	the source r d. Either regi and Wd. word, '1' for l otes a byte o use a .w ext	egister Waster direct

- Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".
- Example 1: NEG.B W3, [W4++] ; Negate W3 and store to [W4] (Byte mode)
  ; Post-increment W4



	Before		After
Instruction		n I	nstruction
W2	0900	W2	0902
W4	1002	W4	1000
Data 0900	870F	Data 0900	870F
Data 1000	5105	Data 1000	78F1
SR	0000	SR	0000

NEG		Negate Acc	cumulator			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax:	{label:}	NEG	Acc			
Operands:	Acc∈ [A,E	3]				
Operation:	<u>If (Acc = A</u> -ACCA - <u>Else:</u> -ACCB -	→ACCA				
Status Affected:	OA, OB, O	AB, SA, SB,	SAB			
Encoding:	1100	1011	A001	0000	0000	0000
Description:	accumulate		s of the Sat	he contents ouration mode ulator.		
	The 'A' bit	specifies the	selected acc	cumulator.		
Words:	1					
Cycles:	1					
Example 1: NEG	; S	egate ACCA tore result ORCON = 0x0		uturation)		
	Before			After		
F	Instructio		-	Instructio		
ACCA	00 3290 9		ACCA	FF CD6F A		
CORCON SR		0000	CORCON SR		0000	
Example 2: NEG	B ; N ; S	egate ACCB tore result	to ACCB	l saturatio		
	Before Instructio	on		After Instructio		
	FF F230 1			00 0DCF E		
CORCON SR		0000	CORCON SR		0000	
OR		0000				

NOP		No Operation				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	NOP				
Operands:	None					
Operation:	No Operati	on				
Status Affected:	None					
Encoding:	0000	0000	xxxx	xxxx	xxxx	xxxx
Description:	No Operati	on is perform	ied.	•		
	The 'x' bits	can take any	value.			
Words:	1					
Cycles:	1					
Example 1: NO	P ; exec	cute no ope	ration			
	Before			After		
	Instruction		In	struction		
_						
PC	00 1092		PC	00 1094		
PC SR				00 1094 0000		
	00 1092 0000	cute no ope	PC SR			
SR	00 1092 0000 P ; exec	cute no ope	PC SR	0000		
SR	00 1092 0000	cute no ope	PC SR			
SR [ <u>Example 2:</u> NO PC [	00 1092 0000 P ; exec Before	cute no ope	PC SR ration	0000 After		
SR [ <u>Example 2:</u> NO	00 1092 0000 P ; exec Before Instruction	cute no ope	PC SR ration	0000 After Istruction		
SR [ <u>Example 2:</u> NO PC [	00 1092 0000 P ; exec Before Instruction 00 08AE	cute no ope	PC SR ration PC SR	0000 After Istruction 00 08B0		
SR [ <u>Example 2:</u> № PC [ SR [ <b>NOPR</b>	00 1092 0000 P ; exec Before Instruction 00 08AE	_	PC SR ration PC SR	0000 After Istruction 00 08B0	dsPIC33F	dsPIC33I
SR <u>Example 2:</u> NO PC SR	00 1092 0000 P ; exec Before Instruction 00 08AE 0000	No Operatio	PC SR ration PC SR on	0000 After Istruction 00 08B0 0000	dsPIC33F X	dsPIC33
SR [ <u>Example 2:</u> № PC [ SR [ <b>NOPR</b>	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F	No Operation	PC	0000 After astruction 00 08B0 0000 dsPIC30F		
SR [ <u>Example 2:</u> NO PC [ SR [ NOPR Implemented in:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X	No Operatio PIC24H X	PC	0000 After astruction 00 08B0 0000 dsPIC30F		
SR [ Example 2: NO PC [ SR [ NOPR Implemented in: Syntax:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:}	No Operation PIC24H X NOPR	PC	0000 After astruction 00 08B0 0000 dsPIC30F		dsPIC33I X
SR [ <u>Example 2:</u> NO PC [ SR [ NOPR Implemented in: Syntax: Operands:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:}	No Operation PIC24H X NOPR	PC	0000 After astruction 00 08B0 0000 dsPIC30F		
SR [ Example 2: NO PC SR [ NOPR Implemented in: Syntax: Operands: Operation:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:} None No Operati	No Operation PIC24H X NOPR	PC	0000 After astruction 00 08B0 0000 dsPIC30F		
SR [ <u>Example 2:</u> NO PC [ SR [ NOPR Implemented in: Syntax: Operands: Operands: Operation: Status Affected:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:} None No Operati None 1111	No Operation PIC24H X NOPR on	PC SR ration PC SR DN PIC24E X	0000 After Istruction 00 08B0 0000 dsPIC30F X	X	X
SR [ Example 2: NO PC SR [ NOPR Implemented in: Syntax: Operands: Operation: Status Affected: Encoding:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:} None No Operati None 1111 No Operati	No Operation PIC24H X NOPR on	PC SR ration	0000 After Istruction 00 08B0 0000 dsPIC30F X	X	X
SR [ Example 2: NO PC SR [ NOPR Implemented in: Syntax: Operands: Operation: Status Affected: Encoding:	00 1092 0000 P ; exec Before Instruction 00 08AE 0000 PIC24F X {label:} None No Operati None 1111 No Operati	No Operation PIC24H X NOPR on 1111 on is perform	PC SR ration	0000 After Istruction 00 08B0 0000 dsPIC30F X	X	X

Example 1:	NOPR ; ex	ecute no ope	ration			
	Before Instruction PC 00 2430 SR 0000		In PC SR	After struction 00 2432 0000		
Example 2:	NOPR ; e>	ecute no ope	ration			
	Before Instruction PC 00 1466 R 0000		In PC SR	After struction 00 1468 0000		
POP		Pop TOS to	f			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	POP	f			
Operands:	f ∈ [0	-				
Operation:	(W15) – ∶ (TOS) →					
Status Affected:	None					. <u> </u>
Encoding:	1111	1001	ffff	ffff	ffff	fff0
Description:	(TOS) wo	k Pointer (W15 ord is written to e in the lower 3	the specified	d file register	, which may	
	The 'f' bi	s select the ad	dress of the	file register.		
	Note 1:	This instruction	on operates	in Word mod	le only.	
	2:	The file regis	ter address i	must be wore	d-aligned.	
Words:	1					
Cycles:	1					
Example 1:	POP 0x1230	; Pop TOS	to 0x1230			
V Data 10 Data 12		W15 Data 1004 Data 1230 SR	A401 A401			

Example 2: POP	0x880	; Pop TOS	to 0x880			
	Before struction 2000 E3E1 A090 0000	W15 Data 0880 Data 1FFE SR	A090 A090			
POP		Pop TOS to	Wd			
Implemented in:	PIC24F X	PIC24H X	PIC24E X	dsPIC30F X	dsPIC33F X	dsPIC33E X
Syntax:	{label:}	POP	Wd [Wd] [Wd++] [Wd] [Wd] [++Wd] [Wd+Wb]			
Operands:	$Wd \in [W0]$ $Wb \in [W0]$					
Operation:	(W15) – 2 (TOS) →W	→W15				
Status Affected: Encoding: Description:	None 0111 The Stack (TOS) word may be use The 'w' bits The 'h' bits The 'd' bits	1www Pointer (W15 d is written to ed for Wd. s define the o select the de select the de	Wd. Either r ffset register estination Ad estination reg	egister direc Wb. dress mode. gister.	t or indirect a	
	2:	This instruction This instruction instruction (M MOV.	on is a speci	fic version of	the "MOV W	
Words:	1					
Cycles:	1					
	W4 Before struction EDA8 1008 C45A 0000	; Pop TOS W4 W15 Data 1006 SR	After Instruction C45A 1006 C45A			

Example 2: POP	[++W10]	; Pre-incre ; Pop TOS t				
	Before Instruction 0E02 1766 E3E1 C7B5 0000	l W10 W15 Data 0E04 Data 1764 SR	1764 C7B5 C7B5			
POP.D		Double Pop		d:Wnd+1		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	POP.D	Wnd			<u> </u>
Operands:	Wnd $\in$ [W	/0, W2, W4,	W14]			
Operation:	$(W15) - 2 \rightarrow W15$ $(TOS) \rightarrow Wnd + 1$ $(W15) - 2 \rightarrow W15$ $(TOS) \rightarrow Wnd$					
Status Affected:	None					
Encoding:	1011	1110	0000	0ddd	0100	1111
Description:	Wnd:Wnd least signif	word is POPpe + 1. The most ficant word is s Pointer (W15)	t significant v stored to Wr	word is store nd. Since a d	d to Wnd + 1	, and the
	The 'd' bits	s select the ad	Idress of the	destination r	register pair.	
	2:	This instruction information of Wnd must be	n how doubl an even wo	e words are a orking registe	aligned in me r.	emory.
		This instruction (Mo MOV.D.				
Words:	1					
Cycles:	2					
Example 1: POP.I	D W6	; Double	pop TOS to	o W6		
	Before 1struction 07BB 89AE 0850 3210 7654 0000	l W6 W7 W15 Data 084C Data 084E SR	7654 084C 3210 7654			

Instruction Descriptions

Example 2: POP.I	0 W 0	; Double	pop TOS to	0 W 0		
	Before struction 673E DD23 0BBC 791C D400 0000	W0 W1 W15 Data 0BB8 Data 0BBA SR	D400 0BB8 791C			
POP.S		Pop Shadov	v Registers			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	POP.S				
Operands:	None					
Operation:	POP shade	w registers				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1111	1110	1000	0000	0000	0000
Description:	primary reg	in the shado jisters. The fo I and DC STA	llowing regis	sters are affe		
		The shadow i only be acces				hey may
	2:	The shadow i	registers are	only one-lev	el deep.	
Words:	1					
Cycles:	1					
Example 1: POP.S ; Pop the shadow registers ; (See PUSH.S Example 1 for contents of shadows)						
	Before struction 07BB 03FD 9610 7249 00E0 (IPL		/1 1000 /2 2000 /3 3000	(IPL = 7, C =	1)	



After instruction execution, contents of shadow registers are NOT modified.

PUSH		Push f to T	os			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	PUSH	f			
Operands:	f ∈ [0 65	534]				
Operation:	(f) →(TOS (W15) + 2					
Status Affected:	None					
Encoding:	1111	1000	ffff	ffff	ffff	fff0
Description:	(TOS) loca	its of the spe tion and then jister may res	the Stack P	ointer (W15)	is increment	ed by 2.
	The 'f' bits select the address of the file register.					
		This instructi The file regis	-		-	
Words:	1					
Cycles:	1 <sup>(1)</sup>					

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	PUSH 0x2004	; Push (	0x2004) to TOS		
	Before Instruction	In	After struction		
	W15 0B00	W15	0B02		
Data	a 0B00 791C	Data 0B00	D400		
Data	a 2004 D400	Data 2004	D400		
	SR 0000	SR	0000		
Example 2:	PUSH 0xC0E	; Push (O	xCOE) to TOS		
	Before	After			
	Instruction	Ins	struction		
	W15 0920	W15	0922		
Data	a 0920 0000	Data 0920	67AA		
Data	0C0E 67AA	Data 2004	67AA		
	SR 0000	SR	0000		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	X	Х	х	Х	Х
Syntax:	{label:}	PUSH	Ws			
Cymax.	[labol.]	1 0011	[Ws]			
			[Ws++]			
			[Ws]			
			[Ws]			
			[++Ws]			
			[Ws+Wb]			
Operands:	Ws ∈ [W0 Wb ∈ [W0					
Operation:	(Ws) →(T0 (W15) + 2 ·	,				
Status Affected:	None					
Encoding:	0111	lwww	w001	1111	1ggg	SSSS
Description:			written to th (W15) is incr	•	. ,	ation and
	The 'g' bits	select the s	offset registe ource Addre ource registe	ss mode.		
	Note 1:	This instruct	ion operates	in Word mo	de only.	
			ion is a spec MOV Wຣ, [ທ			
Words:	1					
Cycles:	<sub>1</sub> (1)					

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

; Push W2 to TOS

W2

SR

W15

Data 1566

After

Instruction

6889

1568

6889

0000

DS70157F-page 342

Example 1:

PUSH W2

W2

SR

W15

Data 1566

Before

Instruction

6889

1566

0000

I	Before nstructior	ו ו	After nstruction
W5	1200	W5	1200
W10	0044	W10	0044
W15	0806	W15	0808
Data 0806	216F	Data 0806	B20A
Data 1244	B20A	Data 1244	B20A
SR	0000	SR	0000

PUSH [W5+W10]

## PUSH.D

Example 2:

Double Push Wns:Wns+1 to TOS

; Push [W5+W10] to TOS

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	PUSH.D	Wns				
Operands:	Wns∈ [W0	0, W2, W4	W14]				
Operation:	(Wns) →(T (W15) + 2 (Wns + 1) → (W15) + 2	→W15 →(TOS)					
Status Affected:	None						
Encoding:	1011	1110	1001	1111	1000	sss0	
Description:	The least s most signif	A double word (Wns:Wns + 1) is PUSHed to the Top-of-Stack (TOS). The least significant word (Wns) is PUSHed to the TOS first, and the most significant word (Wns + 1) is PUSHed to the TOS last. Since a double word is PUSHed, the Stack Pointer (W15) gets incremented by 4.					
	The 's' bits	select the ac	dress of the	source regis	ster pair.		
		This instruction of the second					
		Wns must be					
		This instruction (M as MOV.D.					
Words:	1						
Cycles:	2						
Example 1: PUSH	.D W6	; Push	1 W6:W7 to '	TOS			
	Before Instruction C451 3380 1240 B004 0891 0000	W6 W7 W15 Data 1240 Data 1242 SR	7 3380 5 1244 0 C451 2 3380				

#### 5

Instruction Descriptions

Example 2:	PUSH.D	W10	;	Push	W10:W11	l to I	OS
	Be Inst	ı	After Instruction				
	W10	30D3	V	V10	80D3		
	W11	4550	١	N11	4550		
	W15 0	0C08	V	V15	0C0C		
Data	0C08	79B5	Data 00	C08	80D3		
Data	0C0A	008E	Data 00	COA	4550		
	SR	0000		SR	0000		

PUSH.S	Push Shadow Registers					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	PUSH.S				
Operands:	None					
Operation:	PUSH sha	dow registers	3			
Status Affected:	None					
Encoding:	1111	1110	1010	0000	0000	0000
		The shadow only be acce The shadow	ssed with PU	JSH.S and P	OP.S.	They may
Words:	∠: 1	The shadow	registers are	e only one-le	vei deep.	
Cycles:	1					
Example 1: PUS	H.S ; Pu	sh primary	registers	into shadov	w registers	3
W0 W1 W2	1000 2000	v v	After Instruction /0 0000 /1 1000 /2 2000			
W3 SR			V3 3000 R 0001	(C = 1)		

**Note:** After an instruction execution, contents of the shadow registers are updated.

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	Х	X	X	X	X
Syntax:	{label:}	PWRSAV	#lit1		1	
Cyntax.	(label.)		mitt			
Operands:	lit1 ∈ [0,1]					
Operation:	0 →WDT p 0 →WDT p 0 →WDTO 0 →SLEEF 0 →IDLE ( <u>If (lit1 = 0</u> ):	eep mode	ount			
Status Affected:	None					
Encoding:	1111	1110	0100	0000	0000	000k
	peripherals shutdown. CPU shuts peripherals This instrue Prescaler ( the Reset 3 <b>Note 1:</b>	le is entered. s are shutdow If lit1 = '1', ldl s down, but th s continue to ction resets th Count register System and C The processor processor Re device data s	m. If an on-o le mode is e e clock sour operate. ne Watchdog rs. In additio Control regis or will exit fro eset or Watc	thip oscillator ntered. In Idle ce remains a g Timer Cour n, the WDTO ter (RCON) a om Idle or Sle hdog Time-o	is being use e mode, the o active and the nt register an o, Sleep and I are reset. rep through a	ed, it is also clock to the e d the dle flags o n interrupt
	2:	If awakened and the clock	from Idle mo	ode, Idle bit (		s set to '1'
	3:	If awakened '1' and the cl	•		bit (RCON<3	3>) is set to
		If awakened (RCON<4>)		hdog Time-o	ut, WDTO bi	t
Words:	1	· · · ·				
Cycles:	1					
Example 1: PWR	SAV #0	; Enter SL	EEP mode			
SR	Before Instruction 0040 (IP		After Instruction 0040 (I	PL = 2)		
Example 2: PWR	SAV #1	; Enter ID	LE mode			
SR	Before Instruction 0020 (IP		After Instruction 0020 (I	PL = 1)		

RCALL		Relative (	Call			
Implemented in:	PIC24F	PIC24H	I PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	RCALL	Expr			
Operands:	• •		ute address, lat e linker to a Slit			3 32767]
Operation:	(W15) + 2 (PC) + (2 *	<ul> <li>→(TOS)</li> <li>→W15</li> <li>⇒) →(TOS)</li> </ul>				
Status Affected:	None					
Encoding:	0000	0111	nnnn	nnnn	nnnn	nnnn
	sign-extend and the res The 'n' bits	ded 17-bit v sult is stored	ed literal that sp	) is added to t	the contents of	of the PC
		When poss	sible, this instructly consumes on			
Words:	1					
Cycles:	2					
	2004 2006	RCALL ADD 	_Task1 WO, W1, W2	; Cal	l _Taskl	
	2458 _Task1 245A	 SUB	WO, W2, W3	; _Ta;	skl subrout	ine
PC W15 Data 0810 Data 0812	5 081 0 FFF	04 10 F	li PC W15 Data 0810 Data 0812	After nstruction 01 2458 0814 2006 0001		

	00620 0062		RCALL MOV	_Init W0, [W4++	; Call _Init
	•				
			· · ·		. Tarih andreashing
	00700	00 _Init:	CLR	W2	; _Init subroutine
	00700	02	• • •		
		Before			After
		Instruction			Instruction
	PC	00 620E		PC	00 7000
V	V15	0C50		W15	0C54
Data 00	C50	FFFF		Data 0C50	6210
Data 00	C52	FFFF		Data 0C52	0000
	SR	0000		SR	0000

RCALL		Relative Ca	all			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	RCALL	Expr			
Operands:			te address, lat linker to a Slit			3 32767].
Operation:	(W15) + 2 (PC<22:16 (W15) + 2 0 →SFA bit (PC) + (2 *	) →TOS<15: →W15 >) →(TOS) →W15		TOS<0>		
Status Affected:	SFA		-	1	1	1
Encoding:	0000	0111	nnnn	nnnn	nnnn	nnnn
	PUSHed or sign-extend and the res The 'n' bits	nto the stack ded 17-bit va sult is stored	l literal that sp	irn address is is added to t	stacked, the	of the PC
		When possil	ole, this instruc			
Words:	1	-			-	
Cycles:	4					
	2004 2006 -		_Taskl WO, W1, W2	; Cal	l _Taskl	
	458 _Task1 45A	: SUB	WO, W2, W3	; _Ta:	skl subrout	ine
PC W15 Data 0810 Data 0812 SR	081 FFF FFF	4 0 F F	II PC W15 Data 0810 Data 0812 SR	After nstruction 01 2458 0814 2006 0001 0000		

	00620 0062		RCALL MOV	_Init W0, [W4++	; Call _Init
	•				
			· · ·		. Tarih andreashing
	00700	00 _Init:	CLR	W2	; _Init subroutine
	00700	02	• • •		
		Before			After
		Instruction			Instruction
	PC	00 620E		PC	00 7000
V	V15	0C50		W15	0C54
Data 00	C50	FFFF		Data 0C50	6210
Data 00	C52	FFFF		Data 0C52	0000
	SR	0000		SR	0000

RCALL		Comput	ed Relative C	Call		
Implemented in:	PIC24F	PIC24	H PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	RCALL	Wn			
Operands:	Wn∈ [W0	W15]				
Operation:	(PC) + 2 - (PC<15:0> (W15) + 2 (PC<22:16 (W15) + 2 (PC) + (2 <sup>-</sup> ) NOP →Inst	>) →(TOS) →W15 6>) →(TOS →W15 * (Wn)) →F	S) PC			
Status Affected:	None					
Encoding:	0000	0001	0010	0000	0000	SSSS
	stack. After (Wn)) is ac	r the return Ided to the	address is sta	address (PC + 2 icked, the sign-e e PC and the re used for Wn.	extended 17-b	oit value (2 *
		s select the	e source regis	ster.		
Words:	1					
Cycles:	2					
Example 1: 00F		INC	W2, W3	; Dest	ination of	RCALL
010 010 010	AOC		W6 W4, [W10]	; RCAL	L with W6	
PC W6 W15 Data 1004	Before Instruction 01 000A FFC0 1004 98FF		PC W6 W15 Data 1004	After Instruction 00 FF8C FFC0 1008 000C		
Data 1004 Data 1006 SR	2310 0000	- - -	Data 1004 SR	0001 0000		

Example 2: 000		RCALL FF1L	W2 W0, W1	; RC	CALL with W2
000		 CLR 	W2	; De	estination of RCALL
	Before Instruction			After Instruction	
PC	00 0302		PC	00 0450	
W2	00A6		W2	00A6	
W15	1004		W15	1008	
Data 1004	32BB		Data 1004	0304	
Data 1006	901A		Data 1006	0000	
SR	0000		SR	0000	

	Computed	Relative Ca	II		
PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
		Х			Х
{label:}	RCALL	Wn			
Wn∈ [W0	W15]				
$(PC) + 2 \rightarrow PC$ $(PC<15:1>) \rightarrow TOS<15:1>, SFA bit \rightarrow TOS<0>$ $(W15) + 2 \rightarrow W15$ $(PC<22:16>) \rightarrow (TOS)$ $(W15) + 2 \rightarrow W15$ $0 \rightarrow SFA bit$ $(PC) + (2 * (Wn)) \rightarrow PC$					
SFA		-			
0000	0001	0000	0010	0000	SSSS
range of the call is 32K program words forward or back from the current PC. Before the call is made, the return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, the sign-extended 17-bit value (2 * (Wn)) is added to the contents of the PC and the result is stored in the PC. Register direct addressing must be used for Wn.					current PC. onto the bit value (2 *
The 's' bits	select the s	ource registe	r.		
1					
4					
C EX1: E	INC W2	, W3	; Dest	ination of	RCALL
A	RCALL W6		; RCAL	L with W6	
2	MOVE W4	, [W10]			
Before nstruction 01 000A FFC0 1004 98FF 2310		PC W6 W15 ata 1004 ata 1006	00 FF8C FFC0 1008 000C 0001		
	{label:} $Vn \in [W0$ (PC) + 2 - (PC < 15:1> (W15) + 2 (PC < 22:16 (W15) + 2 $(PC) + (2 * NOP \rightarrow Instr SFA 0000 Computed, range of the stack. After (Vn)) is adRegister diaThe 's' bits142Ex1:34588989814589898114589898114589811458989811458981111111111111$	PIC24FPIC24H $  $ <	PIC24FPIC24HPIC24EImage: Non-Section 2010X{label:}RCALLWnWn $\in$ [W0 W15](PC) + 2 $\rightarrow$ PC(PC + 2 $\rightarrow$ PC(PC < 15:1>) $\rightarrow$ TOS < 15:1>, SFA bit - (W15) + 2 $\rightarrow$ W15(PC < 22:16>) $\rightarrow$ (TOS)(W15) + 2 $\rightarrow$ W150 $\rightarrow$ SFA bit(PC) + (2 * (Wn)) $\rightarrow$ PCNOP $\rightarrow$ Instruction RegisterSFA00000001000000010000Computed, relative subroutine call spectra and of the call is 32K program wordsBefore the call is made, the return address is stack(Wn)) is added to the contents of the IRegister direct addressing must be used the 's' bits select the source register142EX1:INCW2, W323RCALLMOVEW4, [W10]BeforeNOVEnstructionInstruction01 000APCFFC0W61004W1598FFData 10042310Data 1006	Image: space of the second state of the second st	PIC24FPIC24HPIC24EdsPIC30FdsPIC33FImage: space state st

Example 2: 000		RCALL FF1L	W2 W0, W1	; RC	CALL with W2
000		 CLR 	W2	; De	estination of RCALL
	Before Instruction			After Instruction	
PC	00 0302		PC	00 0450	
W2	00A6		W2	00A6	
W15	1004		W15	1008	
Data 1004	32BB		Data 1004	0304	
Data 1006	901A		Data 1006	0000	
SR	0000		SR	0000	

REPEAT	Repeat Next Instruction 'lit14 + 1' Times								
Implemented in:	PIC24F	PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33							
	Х	Х		Х	Х				
		1	1		1	- <b>I</b> ,			
Syntax:	{label:}	REPEAT	#lit14						
Operands:	lit14 ∈ [0	16383]							
Operation:	(lit14) →R0 (PC) + 2 → Enable Co								
Status Affected:	RA								
Encoding:	0000	1001	00kk	kkkk	kkkk	kkkk			
	<ul> <li>When this repeat cou with each e zero, the ta instruction.</li> <li>The 'k' bits <b>Special Fe</b></li> <li>1. When the RA</li> <li>2. The ta <ul> <li>an in</li> <li>a DO instruction.</li> </ul> </li> </ul>	s are an unsig eatures, Res on the repeat lit A bit is not se arget REPEAT instruction that O, DISI, L truction -word instruct pected results	xecutes, the licified in the irr the target ins ion is execute ontinues with gned literal the <b>strictions</b> : teral is '0', RI et. F instruction c at changes pl LNK, MOV.D tion s may occur i	RCOUNT reg nstruction. RC struction. Whe ted one more the instruction hat specifies the EPEAT has the cannot be: program flow p, PWRSAV, if these targe	gister is load COUNT is de len RCOUNT e time, and th on following t the loop cour he effect of a REPEAT or et instructions	led with the ecremented equals uen normal the target nt. NOP and UNLK			
	Note:	The REPEAT	and target	instruction a	re interruptib	ole.			
Words:	1								
Cycles:	1								
Example 1: 00045 00045			1, [W2++]		e ADD 10 tin update	nes			
	Instruction	_		struction					
PC	00 0452	_	PC	00 0454					
RCOUNT	0000	K		0009	·				
SR	0000	]	SR	0010 (R	$A = \perp$ )				

Example 2: 0008			rute CLR 1024 times ar the scratch space
	Before Instruction		After Instruction
PC	00 089E	PC	00 08A0
RCOUNT	0000	RCOUNT	03FF
SR	0000	SR	0010 (RA = 1)

REPEAT	Repeat Next Instruction 'lit15 + 1' Times						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
			Х			Х	
Syntax:	{label:}	REPEAT	#lit15				
Operands:	lit15 ∈ [0	. 32767]					
Operation:	(lit15) →RCOUNT (PC) + 2 →PC Enable Code Looping						
Status Affected:	RA						
Encoding:	0000	1001	0kkk	kkkk	kkkk	kkkk	
Description:	<ul> <li>Repeat the instruction immediately following the REPEAT instruction (lit15 + 1) times. The repeated instruction (or target instruction) is held in the instruction register for all iterations and is only fetched once. When this instruction executes, the RCOUNT register is loaded with the repeat count value specified in the instruction. RCOUNT is decremented with each execution of the target instruction. When RCOUNT equals zero, the target instruction is executed one more time, and then normal instruction execution continues with the instruction following the target instruction. The 'k' bits are an unsigned literal that specifies the loop count.</li> <li>Special Features, Restrictions:</li> <li>1. When the repeat literal is '0', REPEAT has the effect of a NOP and the RA bit is not set.</li> <li>2. The target REPEAT instruction cannot be: <ul> <li>an instruction that changes program flow</li> <li>a DISI, LNK, MOV.D, PWRSAV, REPEAT or UNLK instruction</li> <li>a 2-word instruction</li> <li>Unexpected results may occur if these target instructions are used.</li> </ul> </li> </ul>						
Words:	1						
Cycles:	1						
Example 1:         000452         REPEAT #9         ; Execute ADD 10 times           000454         ADD         [W0++], W1, [W2++]         ; Vector update           Before         After							
	Instruction		Ins	struction			
PC	00 0452	_	PC	00 0454			
RCOUNT SR	0000	R	COUNT SR	0009 0010 (R	A = 1)		

Example 2: 0008			rute CLR 1024 times ar the scratch space		
	Before Instruction	After Instruction			
PC	00 089E	PC	00 08A0		
RCOUNT	0000	RCOUNT	03FF		
SR	0000	SR	0010 (RA = 1)		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33	
·	Х	Х		Х	Х		
Syntax:	{label:}	REPEAT	Wn				
Operands:	Wn∈ [W0	-					
Operation:	(Wn<13:0>) →RCOUNT (PC) + 2 →PC Enable Code Looping						
Status Affected:	RA						
Encoding:	0000	1001	1000	0000	0000	SSSS	
	(Wn<13:0>	<ul> <li>instruction ir</li> <li>) times. The he instruction</li> </ul>	instruction to	be repeated	d (or target ir	nstruction)	
	lower 14 bi the target i is executed	instruction ex its of Wn. RC nstruction. W d one more til with the instru	OUNT is de hen RCOUN me, and the	cremented w IT equals zer n normal inst	ith each exe o, the target ruction exect	cution of instruction	
	The 's' bits specify the Wn register that contains the repeat count.						
	1. When not se 2. The ta	arget REPEAT	PEAT has th	cannot be:	NOP and the	RA bit is	
	• a D0	nstruction tha D, DISI, Li ruction		-	REPEAT Or	ULNK	
	a 2-word instruction						
	Unexpected results may occur if these target instructions are used.						
	Note:	The REPEAT	and target	instruction a	re interruptib	le.	
Words:	1						
Cycles:	1						
Example 1: 000A		W4 [WO++], [W		ecute COM ector comple		25	
	Before			After			
F	Instruction			struction			
PC	Instruction 00 0A26		PC	struction 00 0A28			
PC W4 RCOUNT	Instruction			struction			

Instruction Descriptions

	00089E 0008A0	REPEAT TBLRDL	W10 [W2++],		Execute TBLRD (V Decrement (0x840		
Before				After			
Instruction				Instruction			
	PC	00 089E		PC	00 08A0		
V	/10	00FF		W10	00FF		
RCOU	NT	0000		RCOUNT	00FF		
	SR	0000		SR	0010 (RA =	= 1)	

REPEAT		Repeat Ne:	xt Instructio	on Wn+1 Tim	.es	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	REPEAT	Wn			
Operands:	Wn∈ [W0	/ W15]				
Operation:	(Wn) →RC (PC) + 2 → Enable Co					
Status Affected:	RA					
Encoding:	0000	1001	1000	0000	0000	SSSS
Description:	(Wn) times	e instruction in s. The instruc ruction registe	ction to be rep	peated (or tar	rget instruction	on) is held
	RCOUNT i When RCC time, and t	instruction ex is decrement OUNT equals then normal in following the	ted with each s zero, the tar instruction ex	n execution of rget instructio kecution conti	of the target in	nstruction. d one more
	The 's' bits	s specify the \	Wn register t	that contains	the repeat cr	ount.
	<ul> <li>Special Fe</li> <li>1. When not se</li> <li>2. The ta</li> <li>a nii</li> <li>a DO instr</li> </ul>	eatures, Res n (Wn) = 0, RE	strictions: EPEAT has th I instruction at changes p LNK, MOV.D	he effect of a cannot be: program flow	NOP and the	e RA bit is
	Unexŗ	pected results	s may occur	if these targe	et instructions	s are used.
	Note:	The REPEAT	r and target	t instruction a	re interruptib	ole.
Words:	1					
Cycles:	1					
Example 1: 000A28		W4 [WO++], [W		xecute COM ector comple		es
PC W4 RCOUNT SR	Before Instruction 00 0A26 0023 0000 0000	       	PC W4 RCOUNT SR	After Instruction 00 0A28 0023 0023 0010 (R.	(A = 1)	

	00089E 0008A0	REPEAT TBLRDL	W10 [W2++],		Execute TBLRD (V Decrement (0x840	
	E	Before			After	
	Ins	struction			Instruction	
	PC	00 089E		PC	00 08A0	
V	/10	00FF		W10	00FF	
RCOU	NT	0000		RCOUNT	00FF	
	SR	0000		SR	0010 (RA =	= 1)

# **Section 5. Instruction Descriptions**

RESET		Reset				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	RESET				
Operands:	None					
Operation:	condition.	egisters that a	are affected b	by a MCLR F	eset to their	Reset
Status Affected:	OA, OB, O	AB, SA, SB,	SAB, DA, DO	C, IPL<2:0>,	RA, N, OV, 2	Z, C, SFA
Encoding:	1111	1110	0000	0000	0000	0000
Description:	peripheral ( '0', the loca	ction provides registers will ation of the R .), will be set	take their po	wer-on value	e. The PC wi The SWR bit	ll be set to
	Note:	Refer to the power-on va			eference mai	nual for the
Words:	1					
Cycles:	1					

#### 5

Instruction Descriptions

Example 1: 00	202A I	RESET	; Execu	te soft.	ware RESET	on dsPIC33F
	Bef				After	
	Instru	iction		-	Instruction	-
P	00	202A		PC	00 0000	
W	)	8901		W0	0000	
W	1	08BB		W1	0000	
W	2	B87A		W2	0000	
W	3	872F		W3	0000	
W	1	C98A		W4	0000	
W	5	AAD4		W5	0000	
W	6	981E		W6	0000	
W	7	1809		W7	0000	
W	3	C341		W8	0000	
W	Э	90F4		W9	0000	
W1	)	F409		W10	0000	
W1	1	1700		W11	0000	
W1	2	1008		W12	0000	
W1	3	6556		W13	0000	
W1	1	231D		W14	0000	
W1	5	1704		W15	0800	
SPLIN	1	1800		SPLIM	0000	
TBLPAC	6	007F	Т	BLPAG	0000	
PSVPAG	6	0001	Р	SVPAG	0000	
CORCO	1	00F0	CC	ORCON	0020	(SATDW = 1)
RCO	1	0000		RCON	0040	(SWR = 1)
SI	2	0021	(IPL, C = 1)	SR	0000	]

RETFIE		Return from	n Interrupt			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	RETFIE				
Operands:	None					
Operation:	(TOS<7>) (TOS<6:0> (W15) - 2 - (TOS<15:0	8>) →(SR<7:0 →(IPL3, COR >) →(PC<22:1	RCON<3>) 16>) 5:0>)			
Status Affected:	IPL<3:0>,	RA, N, OV, Z,	, C			
Encoding:	0000	0110	0100	0000	0000	0000
Description:	loads the lo the Most S loads the lo	m Interrupt Se low byte of the Significant Byte lower 16 bits o Restoring IPL	e STATUS re te of the PC. of the PC.	egister, IPL<3 . The stack is	3> (CORCON POPped aga	N<3>) and ain, which
			Interrupt Price	ority Level to		
	2:	Before RETE:	IE is execut	ted, the appro		
<u>.</u>		must be clear	red in softwa	are to avoid r	ecursive inte	rrupts.
Words:	1 3 (2 if exce	" - nondin				
Cycles:	3 (2 II елсе	eption pending	g)			
Example 1: 000A2	26 RETFIE	; Return	n from ISR			
	Before		L.	After		
		1		nstruction		
PC W15	00 0A26	4	PC	01 0230		
W15 Data 0830	0834	l Da	W15	0830		
Data 0830 Data 0832	0230 8101	_	ata 0830 ata 0832	0230 8101		
CORCON	0001		ORCON	0001		
SR	0000	-	SR		PL = 4, C = 1	.)
Example 2: 00805	50 RETFIE	: ; Returr	n from ISR			
	Before			After		
	Instruction			struction		
PC	00 8050	1	PC	00 7008		
W15	0926	]	W15	0922		
Data 0922	7008	_	ata 0922	7008		
Data 0924	0300		ata 0924	0300		
	0000			0000	1	
SR	0000	j	SR	0003 (Z,	, C = 1)	

# 5

Instruction Descriptions

RETFIE		Return from	Interrupt			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
			Х			Х
Syntax:	{label:}	RETFIE				
Operands:	None					
Operation:	(TOS<7>) (TOS<6:0> (W15) - 2 - (TOS<15:1 TOS<0> -	8>) →(SR<7:0: →(IPL3, COR →) →(PC<22:10 →W15 I>) →(PC<15:'	CON<3>) δ>) 1>)			
Status Affected:	IPL<3:0>,	RA, N, OV, Z,	C, SFA		-	
Encoding:	0000	0110	0100	0000	0000	0000
Description:	loads the le the Most S loads the le	n Interrupt Se ow byte of the significant Byte ower 16 bits of	STATUS rest of the PC. f the PC.	egister, IPL<3 . The stack is	3> (CORCON POPped ag	N<3>) and ain, which
		Restoring IPL restores the In execution was	nterrupt Pri	ority Level to		
		Before RETFI				
Words:	1	must be clear	ed in softw	are to avoid r	ecursive inte	errupts.
Cycles:	-	eption pending	)			
	- (	· · · · · · · · · · · · · · · · · · ·	,			
Example 1: 000A	26 RETFIE	; Return	from ISR			
	Before			After		
<b>P</b> O [	Instruction	I		struction		
PC	00 0A26		PC	01 0230		
W15	0834		W15	0830		
Data 0830	0230		a 0830	0230		
Data 0832	8101		a 0832	8101		
CORCON	0001			0001		、
SR	0000		SR	0081 (IF	PL = 4, C = 1	)
Example 2: 0080	50 RETFIE	; Return	from ISR			
	Before			After		
<b>~</b> ~ [	Instruction	I		struction		
PC	00 8050		PC	00 7008		
W15	0926	_	W15	0922		
Data 0922	7008		a 0922	7008		
Data 0924	0300		a 0924	0300		
CORCON	0000	CO	RCON	0000		
SR	0000		SR		, C = 1)	

RETLW		Return with	Literal in V	Vn		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х		Х	Х	
Syntax:	{label:}	RETLW{.B}	#lit10,	Wn		
Operands:	-	255] for byt 1023] for w W15]	•	ſ		
Operation:	TOS<7:0> (W15) – 2 (TOS) →(F lit10 →Wn	> →SR<7:0> →IPL<3> : P →W15 PC<15:0>)				
Status Affected:	None					
Encoding:	0000	0101	0Bkk	kkkk	kkkk	dddd
	Pointer (W The 'B' bit The 'k' bits	ral is stored ir (15) is decren selects byte of s specify the v s select the de	nented by 4. or word oper value of the li	ation ('0' for iteral.		
	Note 1: 2:	The extension rather than a denote a wood For byte ope unsigned val eral Operant operands in	word operation, d operation, rations, the l ue [0:255]. S ds" for infor	tion. You may but it is not i iteral must b See <mark>Section</mark>	y use a . w e equired. e specified a 4.6 "Using 1	xtension to s an I <mark>0-bit Lit-</mark>
Words:	1		<b>,</b>			
Cycles:		eption pendin	g)			
Example 1: 0004	40 RETLW.	.B #0xA, W0	; Return	with OxA	in WO	
	Before			After		
	Instruction	-	In	struction		
PC	00 0440		PC	00 7006		
WO	9846		W0	980A		
W15	1988	-	W15	1984		
Data 1984	7006		ta 1984	7006		
Data 1986	0000	Da	ta 1986	0000		
SR	0000		SR	0000		

mpie z.	0005	OA REILW	#UX230, W2 , Ret	urn with UX2	30 11
		Before Instruction		After Instruction	
	PC	00 050A	PC	01 7008	
	W2	0993	W2	0230	
	W15	1200	W15	11FC	
Data 1	1FC	7008	Data 11FC	7008	
Data 1	1FE	0001	Data 11FE	0001	
	SR	0000	SR	0000	

Example 2: 00050A RETLW #0x230, W2 ; Return with 0x230 in W2

RETLW		Return with	h Literal in V	Nn		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
			Х			Х
Syntax:	{label:}	RETLW{.B}	#lit10,	Wn		
Operands:	-	255] for byt 1023] for wo ) W15]	-	n		
Operation:	(W15) - 2 TOS<15:8: TOS<7:0> (W15) - 2 (TOS<15:1 TOS<0> lit10 -→Wn NOP -→Ins	→W15 →SR<7:0> →IPL<3>: P( →W15 1>) →(PC<15 →SFA bit	C<22:16> 5:1>)			
Status Affected:	SFA					
Encoding: Description:	0000	0101 m subroutine	0Bkk	kkkk	kkkk	dddd
	Pointer (W The 'B' bit The 'k' bits	ral is stored in /15) is decrem selects byte o s specify the v s select the de	mented by 4. or word oper value of the I	ration ('0' for literal.		
		rather than a	a word opera	instruction de ation. You may , but it is not i	yusea.we	
		For byte ope unsigned val	erations, the l lue [0:255]. S ids" for infor	literal must be See <mark>Section</mark> rmation on us	e specified a 4.6 "Using 1	10-bit Lit-
Words:	1					
Cycles:	6 (5 if exce	eption pending	g)			
Example 1: 0004	440 RETLW.	.B #0xA, W0	; Returr	n with OxA	in WO	
	Before			After		
	Instruction	_		struction		
PC	00 0440	-	PC	00 7006		
WO	9846	-	W0	980A		
W15	1988		W15	1984		
	7000	٦ D-				
Data 1984	7006		ata 1984	7006		
	7006 0000 0000	Da		7006 0000 0000		

Before After	) 1r
Instruction Instruction	
PC 00 050A PC 01 7008	
W2 0993 W2 0230	
W15 1200 W15 11FC	
Data 11FC 7008 Data 11FC 7008	
Data 11FE 0001 Data 11FE 0001	
SR 0000 SR 0000	

Example 2: 00050A RETLW #0x230, W2 ; Return with 0x230 in W2

RETURN		Return				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х		X	Х	
Syntax:	{label:}	RETURN				
Operands:	None					
Operation:	(W15) – 2 (TOS) →(P	PC<22:16>) →W15	er			
Status Affected:	None					
Encoding:	0000	0110	0000	0000	0000	0000
Description:		n subroutine. nce two POPs ed by 4.				
Words:	1					
Cycles:	3 (2 if exce	ption pending	g)			
Example 1: 001A	06 RETURN Before Instruction 00 1A06	; Retu	rn from su In PC	After struction 01 0004		
W15	1248		W15	1244		
Data 1244	0004	Da	ta 1244	0004		
Data 1246	0001	Da	ta 1246	0001		
SR	0000		SR	0000		
L						
Example 2: 0054		; Retu	rn from su			
L		; Retu	L			
Example 2: 0054	04 RETURN Before Instruction	; Retu	rn from su	After struction		
Example 2: 0054	04 RETURN Before Instruction 00 5404	; Retu	rn from su	After Struction 00 0966		
Example 2: 0054 PC [ W15 ]	04 RETURN Before Instruction 00 5404 090A		rn from su In PC W15	After Struction 00 0966 0906		
Example 2: 0054 PC [ W15 ] Data 0906 ]	04 RETURN Before Instruction 00 5404 090A 0966	Da	rn from su In PC W15 ta 0906	After Struction 00 0966 0906 0966		
Example 2: 0054 PC [ W15 ]	04 RETURN Before Instruction 00 5404 090A	Da	rn from su In PC W15	After Struction 00 0966 0906		

# 5

RETURN		Return				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
			Х			Х
Syntax:	{label:}	RETURN				
Operands:	None					
Operation:	(W15) – 2 (TOS<15:1 TOS<0> –	PC<22:16>) →W15 ) →(PC<15:1				
Status Affected:	SFA	C				
Encoding:	0000	0110	0000	0000	0000	0000
Description:		n subroutine. nce two POPs ed by 4.				
Words:	1					
Cycles:	6 (5 if exce	eption pending	g)			
Example 1: 001AC	)6 RETURN	; Retu	rn from si	ubroutine		
	Before			After		
	Before Instruction		In	After struction		
PC			In PC			
PC W15	Instruction			struction		
W15 Data 1244	Instruction 00 1A06		PC W15 ta 1244	struction 01 0004		
W15 Data 1244 Data 1246	Instruction 00 1A06 1248 0004 0001		PC W15 ta 1244 ta 1246	struction 01 0004 1244 0004 0001		
W15 Data 1244	Instruction 00 1A06 1248 0004		PC W15 ta 1244	01 0004 1244 0004		
W15 Data 1244 Data 1246	Instruction 00 1A06 1248 0004 0001 0000	Da	PC W15 ta 1244 ta 1246	struction 01 0004 1244 0004 0001 0000		
W15 Data 1244 Data 1246 SR	Instruction 00 1A06 1248 0004 0001 0000	Da	PC	struction 01 0004 1244 0004 0001 0000		
W15 Data 1244 Data 1246 SR	Instruction 00 1A06 1248 0004 0001 0000 04 RETURN	Da	PC	struction 01 0004 1244 0004 0001 0000 ubroutine		
W15 Data 1244 Data 1246 SR	Instruction 00 1A06 1248 0004 0001 0000 04 RETURN Before	Da	PC	struction 01 0004 1244 0004 0001 0000 ubroutine After		
W15 Data 1244 Data 1246 SR <u>Example 2:</u> 00540 PC W15	Instruction 00 1A06 1248 0004 0001 0000 04 RETURN Before Instruction 00 5404 090A	Da ; Retu	PC	struction 01 0004 1244 0004 0001 0000 abroutine After struction 00 0966 0906		
W15 Data 1244 Data 1246 SR <u>Example 2:</u> 00540 PC W15 Data 0906	Instruction 00 1A06 1248 0004 0001 0000 04 RETURN Before Instruction 00 5404	Da ; Retu Da	PC	struction 01 0004 1244 0004 0001 0000 ubroutine After struction 00 0966		
W15 Data 1244 Data 1246 SR <u>Example 2:</u> 00540 PC W15	Instruction 00 1A06 1248 0004 0001 0000 04 RETURN Before Instruction 00 5404 090A	Da ; Retu Da	PC	struction 01 0004 1244 0004 0001 0000 abroutine After struction 00 0966 0906		

# **Section 5. Instruction Descriptions**

RLC		Rotate Left	f through C	arry				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	RLC{.B}	f	{,WREG}				
Operands:	f∈ [0 8	191]						
Operation:	For byte operation: (C) $\rightarrow$ Dest<0> (f<6:0>) $\rightarrow$ Dest<7:1> (f<7>) $\rightarrow$ C For word operation: (C) $\rightarrow$ Dest<0> (f<14:0>) $\rightarrow$ Dest<15:1> (f<15>) $\rightarrow$ C							
Status Affected:	N, Z, C	1		1				
Encoding:	1101	0110	1BDf	ffff	ffff	ffff		
Description:	Rotate the contents of the file register f one bit to the left through the Carry flag and place the result in the destination register. The Carry flag of the STATUS Register is shifted into the Least Significant bit of the destination, and it is then overwritten with the Most Significant bit of Ws. The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.							
	The 'D' bit	selects byte of selects the d select the ad	estination ('0	)' for f, '1' for		byte).		
	<ul> <li>Note 1: The extension .B in the instruction denotes a byte operation rather than a word operation. You may use a .w extension to denote a word operation, but it is not required.</li> <li>2: The WREG is set to working register W0.</li> </ul>							
Words:	1							
Cycles:	1							
Example 1: RLC.	Before	; Rotat	After	C (0x1233)	(Byte mode	)		
lr Data 1232 SR	E807 0000	Data 1232 SR		I, C = 1)				

Example 2: RLC 0x820, WREG ; Rotate Left w/ C (0x820) (Word mode) ; Store result in WREG Before After Instruction Instruction WREG (W0) WREG (W0) 5601 42DD Data 0820 Data 0820 216E 216E SR SR 0000 (C = 0) 0001 (C = 1)

PIC24F X	Х	Х	Х	dsPIC33F X	dsPIC33E X
{label:}	RLC{.B}	Ws,	Wd		
(**** ,		[Ws],	[Wd]		
		[WS], [Ws++],	[Wd++]		
		[Ws],	[Wd]		
		[++Ws],	[++Wd]		
		[Ws],	[Wd]		
<u>For byte op</u> (C) →Wo	peration: /d<0>				
(Ws<6:0	0>) →Wd<7:1	1>			
(C) →Wo	/d<0>				
		5:1>			
(***	>)~ ]				
-C-					
N, Z, C		_,			
1101 Detete the	0010	1Bqq	qddd	dppp	SSSS
the Carry fl Carry flag o of Wd, and	flag and place of the STATL d it is then ove	ce the result in US register is verwritten wit	in the destinat s shifted into t th the Most Sig	tion register \ the Least Sig ignificant bit c	Wd. The gnificant bit of Ws.
The 'q' bits The 'd' bits The 'p' bits	s select the d s select the d s select the s	destination Ad destination re source Addre	ddress mode. egister. ess mode.		byte).
		-			
Note:	rather than a	a word opera	ation. You may	ayusea.we	
1	<b>uu</b>		1, 00.	104.	
1					
LC.B W0, W3				te mode)	
		A ft o r			
Before Instruction		After Instruction			
Before Instruction /0 9976	W	Instruction			
<u> </u>	$Ws \in [W0$ $Wd \in [W0$ $For byte op$ $(C) \rightarrow Wc$ $(Ws < 6:0$ $(Ws < 7>)$ $For word o$ $(C) \rightarrow Wc$ $(Ws < 14:$ $(Ws < 15:$ ) Intermediate the the Carry flag of Wd, and Either regists The 'a' bits The 's' bits The	$Ws \in [W0 \dots W15]$ $Wd \in [W0 \dots W15]$ $For byte operation:$ $(C) \rightarrow Wd<0>$ $(Ws<6:0>) \rightarrow Wd<7:1$ $(Ws<7>) \rightarrow C$ For word operation: $(C) \rightarrow Wd<0>$ $(Ws<14:0>) \rightarrow Wd<1$ $(Ws<15>) \rightarrow C$ $C \rightarrow C$ $M, Z, C$ $1101  0010$ Rotate the contents of the Carry flag and place Carry flag of the STATU of Wd, and it is then ov Either register direct or The 'B' bit selects byte The 'q' bits select the d The 'd' bits select the d The 'd' bits select the s The 's' bits select the s Note: The extension rather than a denote a wo 1 1 C.B W0, W3 ; Rotat	[Ws], [Ws++], [Ws], [++Ws], [Ws], Ws $\in$ [W0 W15] For byte operation: (C) $\rightarrow$ Wd<0> (Ws<6:0>) $\rightarrow$ Wd<7:1> (Ws<7>) $\rightarrow$ C For word operation: (C) $\rightarrow$ Wd<0> (Ws<14:0>) $\rightarrow$ Wd<15:1> (Ws<15>) $\rightarrow$ C II 01 0010 1Bqq N, Z, C 1101 0010 1Bqq Rotate the contents of the source re the Carry flag and place the result in Carry flag of the STATUS register is of Wd, and it is then overwritten witt Either register direct or indirect add The 'B' bit selects byte or word oper The 'q' bits select the destination Add The 'B' bit selects byte or word oper The 'q' bits select the destination re The 'g' bits select the source register Note: The extension .B in the rather than a word operation 1 1 1 AC.B W0, W3 ; Rotate Left w/	$\begin{bmatrix} [Ws], & [Wd]\\ [Ws++], & [Wd++]\\ [Ws], & [Wd]\\ [++Ws], & [++Wd]\\ [Ws], & [Wd] \end{bmatrix}$ $\begin{cases} Ws \in [W0 W15]\\ For byte operation:\\ (C) \rightarrow Wd<0>\\ (Ws<6:0>) \rightarrow Wd<7:1>\\ (Ws<7>) \rightarrow C\\ For word operation:\\ (C) \rightarrow Wd<0>\\ (Ws<14:0>) \rightarrow Wd<15:1>\\ (Ws<15>) \rightarrow C\\ \hline \hline \Box - \Box - \Box \\ \hline \Box - \Box \\ \hline \Box \hline \Box$	$\begin{bmatrix} Ws \end{bmatrix}, & [Wd ]\\ [Ws++], & [Wd++] \\ [Ws], & [Wd] \\ [++Ws], & [++Wd] \\ [-Ws], & [-Wd] \end{bmatrix}$ $Ws \in [W0 W15] \\ \hline Wd \in [W0 W15] \\ \hline For byte operation: \\ (C) & \rightarrow Wd < 0.5 \\ (Ws < 6:0.5) & \rightarrow Wd < 7:1.5 \\ (Ws < 7.5) & \rightarrow C \\ \hline For word operation: \\ (C) & \rightarrow Wd < 0.5 \\ (Ws < 14:0.5) & \rightarrow Wd < 15:1.5 \\ (Ws < 15.5) & \rightarrow C \\ \hline \hline \Box & \bullet \\ $

Instruction Descriptions

 $\ensuremath{\textcircled{}^\circ}$  2005-2011 Microchip Technology Inc.

Example 2: RLC	[W2++]	;	Post-	increme		(Word mode)
	Before			After		
I	nstructior	n	I	nstructior	า	
W2	2008		W2	200A		
W8	094E		W8	094E		
Data 094E	3689	Data	a 094E	8082		
Data 2008	C041	Data	a 2008	C041		
SR	0001	(C = 1)	SR	0009	(N, C = 1)	

#### **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E			
	X	X	X	X	X	X			
•	<u> </u>								
Syntax:	{label:}	RLNC{.B}	f	{,WREG}					
Operands:	f ∈ [0 81	f ∈ [0 8191]							
Operation:	For byte operation: $(f<6:0>) \rightarrow Dest<7:1>$ $(f<7>) \rightarrow Dest<0>$ For word operation: $(f<14:0>) \rightarrow Dest<15:1>$ $(f<15>) \rightarrow Dest<0>$								
Status Affected:	N, Z								
Encoding:	1101	0110	0BDf	ffff	ffff	ffff			
Description:	result in the	contents of tl e destination Significant bit	register. The	Most Signifi	icant bit of f i	s stored in			
	WREG is s	al WREG ope specified, the he result is s	result is stor	ed in WREG					
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'D' bit selects the destination ('0' for WREG, '1' for file register). The 'f' bits select the address of the file register.								
	<ul> <li>Note 1: The extension .B in the instruction denotes a byte operation. You may use a .W extendenote a word operation, but it is not required.</li> <li>2: The WREG is set to working register W0.</li> </ul>								
Words:	1			3 - 3	-				
	1 <sup>(1)</sup>								

read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	RLNC.B	0x1233	;	Rotat	e Left	(0x1233)	(Byte )	mode)
BeforeAfterInstructionInstructionData 1232E807Data 1233D107SR0000SR0008								
Example 2: RLNC 0x820, WREG ; Rotate Left (0x820) (Word mode) ; Store result in WREG								ode)
Before After								
WREG	(W0) 5	5601	WREG	G (W0)	42DC	]		
Data	0820 2	16E	Data	ata 0820 216E				
	SR (	0001 (C	= 1)	SR	0000	(C = 0)		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E				
	X	X	X	X	X	X				
Syntax:	{label:}	RLNC{.B}	Ws,	Wd						
			[Ws],	[Wd]						
			[Ws++],	[Wd++]						
			[Ws],	[Wd]						
			[++Ws],	[++Wd]						
			[Ws],	[Wd]						
Operands:	-	Ws ∈ [W0 W15] Wd ∈ [W0 W15]								
Operation:	<u>For byte operation:</u> (Ws<6:0>) →Wd<7:1> (Ws<7>) →Wd<0> <u>For word operation:</u> (Ws<14:0>) →Wd<15:1> (Ws<15>) →Wd<0>									
Status Affected:	N, Z		<u>.</u>			<u>.</u>				
Encoding:	1101	0010	0Bqq	qddd	dppp	SSSS				
Description:	the result i stored in tl	n the destina he Least Sigr	ition register nificant bit of	egister Ws one Wd. The Mos f Wd, and the direct address	t Significant Carry flag is	bit of Ws is not				
	The 'B' bit selects byte or word operation ('0' for byte, '1' for word). The 'q' bits select the destination Address mode. The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.									
	Note:	-								
Words:	1									

 In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

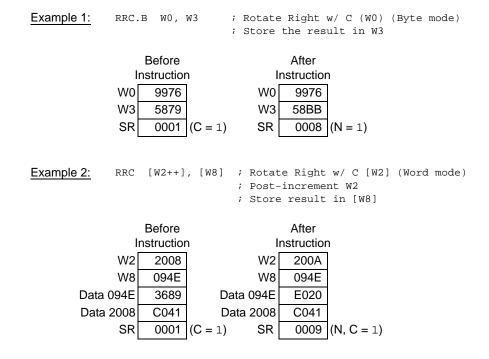
Example 1:	RLNC.B W0, W3	; Rotate Left (W0) (Byte mode) ; Store the result in W3
	Before Instruction W0 9976 W3 5879 SR 0001 (C = 1	After Instruction W0 9976 W3 58EC ) SR 0009 (N, C = 1)
Example 2:	RLNC [W2++], [W8	]; Rotate Left [W2] (Word mode) ; Post-increment W2 ; Store result in [W8]
	Before	After
	Instruction	Instruction
	W2 2008	W2 200A
	W8 094E	W8 094E
Data	094E 3689	Data 094E 8083
Data	2008 C041	Data 2008 C041
	SR 0001 (C = 1	) SR 0009 (N, C = 1)

RRC		-	ht f through	-				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	X	Х	Х	Х	Х		
Syntax:	{label:}	RRC{.B}	f	{,WREG}				
Operands:	-	-						
Operation:	$f \in [0 \dots 8191]$ $For byte operation:$ $(C) \rightarrow Dest<7>$ $(f<7:1>) \rightarrow Dest<6:0>$ $(f<0>) \rightarrow C$ $For word operation:$ $(C) \rightarrow Dest<15>$ $(f<15:1>) \rightarrow Dest<14:0>$ $(f<0>) \rightarrow C$							
Status Affected:	N, Z, C				_	_		
Encoding:	1101	0111	1BDf	ffff	ffff	ffff		
	The option WREG is s specified, The 'B' bit The 'D' bit	n, and it is the nal WREG op specified, the the result is s t selects byte o t selects the d s select the ad	perand deterr e result is sto stored in the or word oper destination ('o	mines the des ored in WREG file register. eration ('0' for '0' for WREG,	stination regis 6. If WREG is byte, '1' for v	ster. If not word).		
	Note 1:	rather than a	a word opera	instruction de ation. You may	y use a .we	•		
	2:		-	rking register \	-			
Words:	1			-				
Cycles:	1 <sup>(1)</sup>							
read-modif	ify-write oper	24E devices, t erations on nor Section 3.2.1	n-CPU Speci	ial Function R	Registers. For			
Example 1: RRC. F	B 0x1233	; Rotat	te Right w,	/ C (0x1233)	) (Byte moc	le)		
	Before		After					
In Data 1232	nstruction E807	Data 1232	Instruction					

Example 2:	RRC	0x820,	WREG		te Righ e resul		C (0x820) WREG	(Word	mode)
		Af	ter						
	Ir	nstructior	า		Instru	ictioi	า		
WREG	(W0)	5601	V	VREG (\	N0) 90	)B7			
Data	0820	216E		Data 0	820 2	16E			
	SR	0001	(C = 1)		SR 0	800	(N = 1)		

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	Х	Х	Х	Х	Х	Х		
Syntax:	{label:}	RRC{.B}	Ws,	Wd				
-			[Ws],	[Wd]				
			[Ws++],	[Wd++]				
			[Ws],	[Wd]				
			[++Ws],	[++Wd]				
			[Ws],	[Wd]				
Operands:	Ws ∈ [W0 W15] Wd ∈ [W0 W15]							
Operation:	(Ws<0> <u>For word c</u> (C) →W	d<7> l>) →Wd<6:0 ) →C <u>operation:</u> d<15> :1>) →Wd<1						
	▶	_ <b>→</b> C						
Status Affected:	N, Z, C							
Encoding:	1101	0011	1Bqq	qddd	dppp	SSSS		
Description:	the Carry f Carry flag of Wd, and	lag and plac of the STATL tit is then ov	e the result i JS Register i verwritten wit	egister Ws one n the destinat is shifted into h the Least Si ressing may b	ion register the Most Sig ignificant bit	Wd. The nificant bit of Ws.		
	The 'q' bits The 'd' bits The 'p' bits	-	lestination A lestination re source Addre	ss mode.		byte).		
	Note:	rather than	a word operation	instruction de ation. You ma n, but it is not	yusea.we			
Words:	1							
Cycles:	1 <sup>(1)</sup>							

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".



Implemented in:	PIC24F	Rotate Righ	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	X	X	X	X	X	X		
Syntax:	{label:}	RRNC{.B}	f	{,WREG}				
Operands:	f ∈ [0 8191]							
Operation:	For byte operation: $(f<7:1>) \rightarrow Dest<6:0>$ $(f<0>) \rightarrow Dest<7>$ For word operation: $(f<15:1>) \rightarrow Dest<14:0>$ $(f<0>) \rightarrow Dest<15>$							
	<b>&gt;</b>							
Status Affected:	N, Z		-					
Encoding:	1101	0111	OBDf	ffff	ffff	ffff		
Description:	Rotate the contents of the file register f one bit to the right and place the result in the destination register. The Least Significant bit of f is stored in the Most Significant bit of the destination, and the Carry flag is not affected.							
	The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.							
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'D' bit selects the destination ('0' for WREG, '1' for file register). The 'f' bits select the address of the file register.							
<b>Note 1:</b> The extension .B in the instruction denotes a byte rather than a word operation. You may use a .w ex denote a word operation, but it is not required.								
	ي.		•		•			
Words:	<b>2</b> :		•	king register	•			

In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1:	RRNC.B	0x1233 ;	Rotate Right	: (0x1233) (Byte mode)					
Data	Befo Instruc 1232 E8	ction	After Instruction Data 1232 7407						
	SR 00	000	SR 0000						
Example 2:	RRNC 0x	•	Rotate Right Store result	: (0x820) (Word mode) : in WREG					
	Befo	ore	After						
	Instruc	ction	Instruction						
WREG	(W0) 56	01 WR	EG (W0) 10E	37					
Data	0820 21	6E D	ata 0820 216	)E					
	SR 00	001 (C = 1)	SR 000	D1 (C = 1)					

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
	X	Х	X	X	X	X		
	L				1	<u>.</u>		
Syntax:	{label:}	RRNC{.B}	Ws,	Wd				
			[Ws],	[Wd]				
			[Ws++],	[Wd++]				
			[Ws],	[Wd]				
			[++Ws],	[++Wd]				
			[Ws],	[Wd]				
Operands:	Ws ∈ [W0 Wd ∈ [W0							
Operation:	(Ws<0> <u>For word c</u> (Ws<15	1>) →Wd<6:0 >) →Wd<7>						
Status Affected:	N, Z							
Encoding:	1101	0011	0Bqq	qddd	dppp	SSSS		
Description:	Rotate the place the r of Ws is st affected. E and Wd.	e contents of t result in the d tored in the M Either register	the source re destination re lost Significa r direct or ind	register Ws on egister Wd. Th ant bit of Wd, a direct address	he bit to the ri he Least Sigi and the Carry sing may be u	ight and nificant bit y flag is not ised for Ws		
	The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'q' bits select the destination Address mode. The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.							
	Note:	rather than	a word operation	e instruction d ration. You ma n, but it is not	ayusea.wie			
Words:	1		·		·			

details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

SR

0000

Example 1:	RRNC	.B W0,			te Right (WO) (Byte mode) e the result in W3
	lr W0 W3 SR	Before nstruction 9976 5879 0001	) (C = 1)	l W0 W3 SR	After nstruction 9976 583B 0001 (C = 1)
Example 2:	RRNC	[W2++	], [W8]	; Post	ate Right [W2] (Word mode) increment W2 re result in [W8]
		Before			After
	Ir	nstructio	า	I	nstruction
	W2	2008		W2	200A
	W8	094E		W8	094E
Data	094E	3689	Dat	a 094E	E020
Data	2008	C041	Dat	ta 2008	C041

SR

0008 (N = 1)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				X	X	X
			<u> </u>		· .	<u> </u>
Syntax:	{label:}	SAC	Acc,	{#Slit4,}	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[Wd]	
					[++Wd]	
					[Wd + Wb]	
Operands:	Acc ∈ [A,B Slit4 ∈ [-8 Wb. Wd ∈		1			
Operation:		cc) (optional)				
	(Acc[31:16					
Status Affected:	None	0.0	1 <u> </u>	1	I	
Encoding: Description:	1100	1100	Awww	wrrr	rhhh cified accumu	dddd
		- ~ otiv	d in	···· ·	- Inft	
	positive op or indirect The 'A' bit The 'w' bits The 'r' bits The 'h' bits	perand indica addressing n specifies the s specify the encode the o s select the d	tes an arithn nay be used source accu offset registe optional acce estination Ad	for Wd. umulator. er Wb. umulator pre- ddress mode	ift. Either regi -shift.	
	positive op or indirect The 'A' bit The 'w' bits The 'r' bits The 'h' bits The 'd' bits	perand indica addressing n specifies the s specify the encode the d s select the d s specify the d	tes an arithn nay be used source accu offset registe optional acce estination A destination r	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd.	ift. Either regi -shift.	ister direct
	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'h' bits The 'd' bits <b>Note 1:</b> 2:	berand indica addressing n specifies the s specify the encode the d s select the d s select the d This instruct This instruct instruction s	tes an arithm nay be used source accu offset registe optional acce estination A destination r ion does not ion stores th AC.R may b	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode register Wd. a modify the c e truncated c	ift. Either regi -shift.	ister direct cc. cc. The
	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'h' bits The 'd' bits <b>Note 1:</b> 2:	berand indica addressing n specifies the s specify the encode the d s select the d s specify the This instruct This instruct instruction s accumulator	tes an arithm nay be used source accu offset registe optional acce estination Ac destination r ion does not ion stores th AC.R may b contents.	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode register Wd. modify the c re truncated c re used to sto	ift. Either regi -shift. - contents of Ac contents of Ac core the rounde	ister direct cc. cc. The ed
	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3:	berand indica addressing n specifies the s specify the encode the d s select the d s specify the This instruction This instruction instruction s accumulator If Data Write	tes an arithm nay be used source accu offset registe optional acce estination Ac destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjec	-shift. -shift. - contents of Accontents of Ac	cc. cc. cc. The ed CON<5>,
Words:	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3:	perand indica addressing n specifies the s specify the encode the d s select the d s specify the This instruction s accumulator If Data Write = 1), the value	tes an arithm nay be used source accu offset registe optional acce estination Ac destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjec	ift. Either regi -shift. contents of Ac contents of Ac ore the rounde	cc. cc. cc. The ed CON<5>,
Words: Cycles:	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3:	perand indica addressing n specifies the s specify the encode the d s select the d s specify the This instruction s accumulator If Data Write = 1), the value	tes an arithm nay be used source accu offset registe optional acce estination Ac destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjec	ift. Either regi -shift. contents of Ac contents of Ac ore the rounde	cc. cc. The ed CON<5>,
Cycles: <u>Example 1:</u> SAC ; R. ; S	positive op or indirect : The 'A' bit The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3: 1	addressing n specifies the specifies the select the d select the d select the d specify the This instruction S accumulator If Data Write = 1), the value optional shift	tes an arithm nay be used source accu- offset registe optional acce estination A destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to t is performe	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjec	ift. Either regi -shift. contents of Ac contents of Ac ore the rounde	cc. cc. The ed CON<5>,
Cycles: <u>Example 1:</u> SAC ; R. ; S	positive op or indirect : The 'A' bit: The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3: 1 1 1 2: 4. #4, W5 5: 5: 5: 5: 5: 5: 5: 5: 5: 5: 5: 5: 5:	addressing n specifies the s specify the encode the d s select the d s select the d s specify the This instruction S accumulator If Data Write = 1), the value optional shift	tes an arithm nay be used source accu- offset registe optional acce estination A destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to t is performe	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode register Wd. modify the c re truncated c re used to sto s enabled (S/ Wd is subjected.	ift. Either regi -shift. contents of Ac contents of Ac ore the rounde	cc. cc. The ed CON<5>,
Cycles: <u>Example 1:</u> SAC ; R. ; S	positive op or indirect a The 'A' bits The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3: 1 1 1 2: 4. #4, W5 5: 5: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	addressing n specifies the s specify the encode the d s select the d s select the d s specify the This instruction S accumulator If Data Write = 1), the value optional shift	tes an arithm nay be used source accu- offset registe optional acce estination A destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to t is performe	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjec	-shift. -shift. - contents of According to According the rounded ATDW, COR( at to saturation	cc. cc. The ed CON<5>,
Cycles: <u>Example 1:</u> SAC ; R ; S ; C W5	positive op or indirect a The 'A' bits The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3: 1 1 2 3: 1 2 3: 2 3: 2 3: 2 3:	addressing n specifies the specifies the specifies the select the d select the d specify the case of the select the d specify the select the d select the value specify the select the select the d specify the select the select the select the select the select the select the select the selec	tes an arithm nay be used source accu- offset registe optional acce estination A destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to t is performe = 1)	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode egister Wd. modify the c e truncated c e used to sto s enabled (Sa Wd is subjected. Md is subjected.	on 0120	cc. cc. The ed CON<5>,
Cycles: <u>Example 1:</u> SAC ; R ; S ; C	positive op or indirect a The 'A' bits The 'w' bits The 'r' bits The 'd' bits Note 1: 2: 3: 1 1 2 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	addressing n specifies the specifies the specifies the select the d select the d specify the case of the select the d specify the select the d select the value specify the select the select the d specify the select the select the select the select the select the select the select the selec	tes an arithm nay be used source accur offset registe optional accer estination A destination r ion does not ion stores th AC.R may b contents. e saturation is ue stored to t is performe	netic right shi for Wd. umulator. er Wb. umulator pre- ddress mode register Wd. a modify the c re truncated c re used to sto s enabled (SA Wd is subject ad. After Instructic 00 120F 1	on 0120	cc. cc. The ed CON<5>,

Instruction Descriptions

© 2005-2011 Microchip Technology Inc.

<pre>Example 2: SAC B, #-4, [W5++] ; Left shift ACCB by 4 ; Store result to [W5], Post-increment W5 ; CORCON = 0x0010 (SATDW = 1)</pre>							
Before After							
	Instruction		Instruction				
W5	2000	W5	2002				
ACCB	FF C891 1F4C						
Data 2000	Data 2000	8000					
CORCON	0010	CORCON	0010				
SR	0000	SR	0000				

		Store Roun	ded Accum	ulator		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax:	{label:}	SAC.R	Acc,	{#Slit4,}	Wd [Wd] [Wd++] [Wd] [Wd]	
					[++Wd] [Wd + Wb]	
Operands:	Acc ∈ [A,E Slit4 ∈ [-8 Wb ∈ [W0 Wd ∈ [W0	+7] W15]				
Operation:	Shift <sub>Slit4</sub> (A Round(Ac (Acc[31:16					
Status Affected:	None					
Encoding:	1100	1101	Awww	wrrr	rhhh	dddd
	<b>D</b> (					
Description:	store the r range is -8 and a posi mode (Col	n optional, sig ounded conte I:7, where a r tive operand nventional or <1>. Either re	ents of ACCx egative ope indicates an Convergent)	H (Acc[31:16 rand indicate arithmetic rig is set by the	s an arithme ght shift. The RND bit,	e shift tic left shift Rounding
Description:	store the r range is -8 and a posi mode (Con CORCON for Wd. The 'A' bit The 'A' bit The 'r' bits The 'h' bits	ounded conte 27, where a r tive operand nventional or	ents of ACCx egative oper indicates an Convergent) gister direct source accu offset register optional accu estination Ac	H (Acc[31:16 rand indicate arithmetic rig is set by the or indirect ac mulator. er Wb. umulator pre- ddress mode.	S)) to Wd. The s an arithme ght shift. The RND bit, ddressing ma shift.	e shift tic left shift Rounding
Description:	store the r range is -8 and a posi mode (Con CORCON for Wd. The 'A' bit The 'A' bit The 'r' bits The 'h' bits	specifies the specifies the specifies the specify the encode the de specify the this instruction	ents of ACCx egative oper indicates an Convergent) gister direct source accu offset register optional accu estination Ac destination re on does not on stores the AC may be u	H (Acc[31:16 rand indicate arithmetic rig is set by the or indirect ac mulator. er Wb. umulator pre- ddress mode. egister Wd. modify the c e rounded co	S)) to Wd. The s an arithmer ght shift. The RND bit, ddressing ma shift. ontents of the intents of Acc	e shift tic left shift Rounding by be used by be used c. C. The
Description:	store the r range is -8 and a posi mode (Con CORCON for Wd. The 'A' bit The 'A' bit The 'r' bits The 'h' bits The 'd' bits	specifies the specifies the specifies the specifies the specify the encode the de specify the this instruction this instruction s.	ents of ACCx egative oper indicates an Convergent) gister direct source accu offset register optional accu estination Ac destination Ac on does not on stores the AC may be u contents. saturation is ue stored to 1	H (Acc[31:16 rand indicate arithmetic rig is set by the or indirect ac mulator. er Wb. umulator pre- ddress mode. egister Wd. modify the c e rounded co sed to store s enabled (S/ Wd is subjec	<ul> <li>b) to Wd. The s an arithmer ght shift. The RND bit, ddressing ma shift.</li> <li>ontents of the intents of Acc the truncated ATDW, CORC</li> </ul>	e shift tic left shift Rounding ty be used ty be used to con<5>,
Description: Words:	store the r range is -8 and a posi mode (Con CORCON for Wd. The 'A' bit The 'A' bits The 'A' bits The 'h' bits The 'h' bits The 'd' bits <b>Note 1:</b> 2:	specifies the specifies the specifies the specifies the specify the encode the de specify the this instruction this instruction accumulator If Data Write = 1), the value	ents of ACCx egative oper indicates an Convergent) gister direct source accu offset register optional accu estination Ac destination Ac on does not on stores the AC may be u contents. saturation is ue stored to 1	H (Acc[31:16 rand indicate arithmetic rig is set by the or indirect ac mulator. er Wb. umulator pre- ddress mode. egister Wd. modify the c e rounded co sed to store s enabled (S/ Wd is subjec	<ul> <li>b) to Wd. The s an arithmer ght shift. The RND bit, ddressing ma shift.</li> <li>ontents of the intents of Acc the truncated ATDW, CORC</li> </ul>	e shift tic left shift Rounding ty be used ty be used to con<5>,

# 5

<pre>Example 1: SAC.R A, #4, W5 ; Right shift ACCA by 4 ; Store rounded result to W5 ; CORCON = 0x0010 (SATDW = 1)</pre>								
Before After								
	Instruction		Instruction					
W5	B900	W5	0121					
ACCA	00 120F FF00	ACCA	00 120F FF00					
CORCON	0010	CORCON	0010					
SR	0000	SR	0000					
Example 2: SAC.R B, #-4, [W5++] ; Left shift ACCB by 4 ; Store rounded result to [W5], Post-increment W5 ; CORCON = 0x0010 (SATDW = 1)								
	Before		After					
	Instruction		Instruction					
W5	2000	W5	2002					
ACCB	FF F891 8F4C	ACCB	FF F891 8F4C					
Data 2000	5BBE	Data 2000	8919					

0010

0000

CORCON

SR

0010

0000

CORCON

SR

-	Sign-Exten	d ws					
PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E		
Х	Х	Х	Х	Х	Х		
{label:}	SE	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd				
-	-						
<u>If (Ws&lt;7&gt; =</u> 0xFF →V <u>Else:</u>	<u>= 1):</u> Wnd<15:8>						
N, Z, C							
1111	1011	0000	0ddd	dppp	SSSS		
Sign-extend the byte in Ws and store the 16-bit result in Wnd. Either register direct or indirect addressing may be used for Ws, and register direct addressing must be used for Wnd. The C flag is set to the complement of the N flag.							
The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.							
Note 1: This operation converts a byte to a word, and it uses no . B or . w extension.							
2:	The source V	Ws is address	•	e operand, so	o any		
1							
1 <sup>(1)</sup>							
	Ws ∈ [W0 Wnd ∈ [W0 Ws<7:0> – If (Ws<7:0> – 0 xFF → 0 →Wnd- N, Z, C 1111 Sign-extend register direct addre complemen The 'd' bits The 'p' bits The 's' bits Note 1: 2:	Ws ∈ [W0 W15] Wnd ∈ [W0 W15] Ws<7:0> →Wnd<7:0> If (Ws<7> = 1): 0xFF →Wnd<15:8> Else: 0 →Wnd<15:8> N, Z, C 1111 1011 Sign-extend the byte in register direct or indirect direct addressing must be complement of the N flather The 'd' bits select the define 'p' bits select the define The 's' bits select the define 's' bits select the second The 's' bits select the second Note 1: This operation . w extension 2: The source V address mod 1	$[Ws], \\ [Ws++], \\ [Ws], \\ [++Ws], \\ [Ws], \\ \\ Ws \in [W0 W15] \\ Wnd \in [W0 W15] \\ Ws<7:0> \rightarrow Wnd<7:0> \\ If (Ws<7> = 1): \\ 0xFF \rightarrow Wnd<15:8> \\ \hline Else: \\ 0 \rightarrow Wnd<15:8> \\ \hline N, Z, C \\ \hline 1111 1011 0000 \\ \hline Sign-extend the byte in Ws and store register direct or indirect addressing direct addressing must be used for V complement of the N flag. \\ \hline The 'd' bits select the destination reg The 'p' bits select the source Address The 's' bits select the source register Note 1: This operation converts a W extension. \\ 2: The source Ws is address address modification is by 1 \\ \hline$	$[Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws], [Ws], [Ws], [Ws], [Ws], [Ws], [Ws], [Ms<7:0> \rightarrowWnd<7:0> \frac{If (Ws<7> = 1):}{0xFF} \rightarrowWnd<15:8> \frac{Else:}{0} \rightarrowWnd<15:8> N, Z, C 1111 1011 0000 0ddd Sign-extend the byte in Ws and store the 16-bit reregister direct or indirect addressing may be used direct addressing must be used for Wnd. The C f complement of the N flag. The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register. Note 1: This operation converts a byte to a work wextension. 2: The source Ws is addressed as a byte address modification is by '1'. 1$	$[Ws], [Ws++], [Ws], [++Ws], [Ws], [++Ws], [Ws], [Ws], Ws \in [W0 W15] Ws <7:0> \rightarrowWnd <7:0> If (Ws <7> = 1): 0xFF \rightarrowWnd <7:0> If (Ws <7> = 1): 0xFF \rightarrowWnd <15:8> Else: 0 \rightarrowWnd <15:8> N, Z, C 1111 1011 0000 0ddd dppp Sign-extend the byte in Ws and store the 16-bit result in Wnd. register direct or indirect addressing may be used for Ws, and direct addressing must be used for Wnd. The C flag is set to t complement of the N flag. The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register. Note 1: This operation converts a byte to a word, and it user w extension. 2: The source Ws is addressed as a byte operand, so address modification is by '1'. 1$		

0039

0001 (C = 1)

W4

SR

1005

0000

W4

SR

Example 2: SE [7	W2++], W12	; Sign-exte ; Post-incr		and store to W12 2
	Before		After	
I	nstruction	li li	nstructior	า
W2	0900	W2	0901	
W12	1002	W12	FF8F	
Data 0900	008F	Data 0900	008F	
SR	0000	SR	8000	(N = 1)

SETM		Set f or WR	EG			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SETM{.B}	f WREG			
Operands:	f ∈ [0 81	91]				
Operation:	For word o	destination d				
Status Affected:	None					
Encoding:	1110	1111	1BDf	ffff	ffff	ffff
Description:		of the specifi NREG are se				
	The 'D' bit The 'f' bits <b>Note 1:</b>	selects byte of selects the di select the ad The extension rather than a denote a wore The WREG in	estination ('0 dress of the n . B in the in word operat d operation,	' for WREG, file register. nstruction de ion. You may but it is not i	'1' for file re enotes a byte / use a .w e required.	gister). operation
Words:	1					
Cycles:	1					
Example 1: SETM.B	0x891	; Set 0x89	1 (Byte mo	de)		
	Before struction 2739 0000 WREG	Data 0890 SR ; Set WREG		e )		
	Before struction 0900 0000	WREG (W0) SR				

SETM		Set Ws				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SETM{.B}	Wd			
			[Wd]			
			[Wd++]			
			[Wd]			
			[++Wd]			
			[Wd]			
Operands:	$Wd \in [W0]$	W15]				
Operation:	<u>For byte o</u> 0xFF →	<u>peration:</u> Wd for byte c	peration			
	For word of 0xFFFF	peration: →Wd for wo	rd operation			
Status Affected:	None					
Encoding:	1110	1011	1Bqq	qddd	d000	0000
Description:	indirect ad The 'B' bits The 'q' bits	of the specif dressing may s selects byte s select the d	<ul> <li>be used for</li> <li>or word ope</li> <li>estination Ad</li> </ul>	Wd. ration ('0' for dress mode.	r word, '1' foi	
	Note:	rather than a	estination rec on .B in the a word opera ord operation,	instruction d tion. You ma	iy use a .w e	
Words:	1					
Cycles:	1					
Example 1: SETM.B	W13	; Set W13	(Byte mode	)		
	Before struction 2739 0000 [W6]	W13 SF ; Pre-decr ; Set [W6]	<b>0000</b>	Word mode)		
	Before Instruction 1250 3CD9 0000	We Data 124E SF	After Instruction 124E FFFF			

SFTAC		Anthinetic	Shint Accun	nulator by SI	110	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
				X	Х	Х
Syntax:	{label:}	SFTAC	Acc,	#Slit6		
Operands:	Acc ∈ [A,B Slit6 ∈ [-16	-				
Operation:	Shift <sub>k</sub> (Acc)	→Acc				
Status Affected:	OA, OB, O	AB, SA, SB,	SAB			
Encoding:	1100	1000	A000	0000	01kk	kkkk
	shift range positive op the accum	is -16:16, wh erand indica ulator are los	nere a negativ tes a right sh st.	sult back into the operand in we operand in hift. Any bits w	dicates a left	shift and a
			ccumulator for	or the result. f bits to be sh	ifted	
	2:	CORCON<7 the accumul If the shift ar modification	v> or SATB, ( ator is subject mount is greated	or the target a CORCON<6> ct to saturatio ater than 16 o to the accun	), the value s n. r less than -1	stored to
Words:	1					
Cycles:	1					
; St	AC A, #12 rithmetic r: tore result ORCON = 0x00	to ACCA	_			
	Before			After		
	Instructio		r	Instruction	n	
ACCA	00 120F F		ACCA	00 0001 2		
CORCON		0080	CORCON		080	
SR		0000	SR	0	000	
Example 2: SFTA	rithmetic le tore result		-			
	RCON = 0x00					
	Before			After		
; cc		<u>n</u>	_	Instruction		
; cc ACCB	Before Instructio FF FFF1 8	F4C	ACCB	Instruction FF C63D 30		
; cc	Before Instructio FF FFF1 8	F4C	ACCB CORCON SR	Instruction FF C63D 30	000 040 000	

 $\ensuremath{\textcircled{}^{\circ}}$  2005-2011 Microchip Technology Inc.

SFTAC		Arithmetic	Shift Accun	nulator by W	/b	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
				Х	Х	Х
Syntax:	{label:}	SFTAC	Acc,	Wb		
Operands:	Acc∈ [A,E Wb∈ [W0					
Operation:	Shift <sub>(Wb)</sub> (A	.cc) →Acc				
Status Affected:	OA, OB, C	AB, SA, SB,	SAB			
Encoding:	1100	1000	A000	0000	0000	SSSS
Description:	the result the result the result the result the are used to negative version of the shift. Any the shift.	shift the 40-b back into the b specify the alue indicates bits which are selects the a	accumulator shift amount s a left shift a shifted out o	The Least S The shift rai and a positive of the accum	Significant 6 I nge is -16:16 value indica ulator are los	bits of Wb 6, where a ates a right st.
	Note 1:	the accumula If the shift an	is enabled fo > or SATB, ( ator is subject nount is great will be made	or the target a CORCON<6> ct to saturatio ater than 16 c to the accur	<ul> <li>b), the value</li> <li>c), the value</li> <li>c)</li> <lic)< li=""> <li>c)</li> <li>c)</li> <li>c)</li></lic)<></ul>	stored to 16, no
Words:	1					
Cycles:	1					
; Sto	ithmetic s ore result	hift ACCA k to ACCA 000 (satura		led)		
	Before	1		After		
_	Instructio	on	-	Instructio	n	
WO		FFC	W0		FFC	
ACCA CORCON	00 320F		ACCA CORCON	03 20FA E		
SR		0000	SR		0000 3800 (OA, C	DAB = 1)
; Ari ; Sto	ore result	hift ACCB b to ACCB 040 (SATB =				
	Before			After		
	Deloie					
٣	Instructio		r	Instruction		
W12	Instructio	000F	W12	00	00F	
W12 ACCB CORCON	Instructio ( FF FFF1 8	000F 8F4C	W12 ACCB CORCON	00 FF FFFF FF	00F	

## **Section 5. Instruction Descriptions**

SL		Shift Left f				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SL{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	191]				
Operation:	0 →Des <u>For word c</u> (f<15>)	$\rightarrow$ (C) $\rightarrow$ Dest<7:1> st<0> <u>operation:</u> $\rightarrow$ (C) $\rightarrow$ Dest<15:				
Status Affected:	 N, Z, C					
Encoding:	1101	0100	OBDf	ffff	ffff	ffff
	shifted into the Least \$ The option WREG is \$	tination registe to the Carry bit Significant bit nal WREG ope specified, the the result is si	it of the STAT t of the destin perand determ result is stor	TUS register, nation registe mines the des red in WREG	and zero is s er. stination regis	shifted into ster. If
	The 'D' bit	t selects byte o t selects the do s select the ad	destination ('0	0' for WREG,		
	Note 1:	The extensio rather than a denote a wor	on . B in the in a word operat ord operation,	instruction de ation. You may , but it is not r	y use a .w ex required.	•
	2:	The WREG I	is set to work	king register \	W0.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-modi	lify-write oper	24E devices, therations on non <b>Section 3.2.1</b>	n-CPU Specia	ial Function R	Registers. For	
Example 1: SL.B	0x909 ;	; Shift left	: (0x909) (	Byte mode)		
I	Before Instruction		After Instruction			

#### 16-bit MCU and DSC Programmer's Reference Manual

0x1650, WREG ; Shift left (0x1650) (Word mode) Example 2: SL ; Store result in WREG Before After Instruction Instruction WREG (W0) 0900 WREG (W0) 80CA Data 1650 Data 1650 4065 4065 SR SR 0008 (N = 1) 0000

SL		Shift Left V	Vs				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	SL{.B}	Ws,	Wd			
			[Ws],	[Wd]			
			[Ws++],	[Wd++]			
			[Ws],	[Wd]			
			[++Ws],	[++Wd]			
			[Ws],	[Wd]			
Operands:		Ws ∈ [W0 W15] Wd ∈ [W0 W15]					
Operation:	0 →Wd< <u>For word c</u> (Ws<15	) →C >>) →Wd<7:1 <0> p <u>peration:</u> >) →C :0>) →Wd<1					
		-0					
Status Affected:	N, Z, C		-	-			
Encoding:	1101	0000	0Bqq	qddd	dppp	SSSS	
Description:	the result i shifted into Least Sign	n the destina the Carry bi	tion register t of the STAT Wd. Either re	ister Ws one I Wd. The Mos US register, a egister direct o	t Significant and '0' is shif	bit of Ws is ted into the	
	The 'q' bits The 'd' bits The 'p' bits		estination A estination re ource Addre	ss mode.		byte).	
	Note:	rather than	a word oper	instruction de ation. You ma n, but it is not	yusea.we		
Words:	1						

Note 1: In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and read-modify-write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Data 0900

Data 1002

SR

800F

6722

0000

Example 1: SL.B W3, W4 ; Shift left W3 (Byte mode)

	; s	tore result to W4
Example 2:	Before Instruction W3 78A9 W4 1005 SR 0000 SL [W2++], [W12]	After Instruction W3 78A9 W4 1052 SR 0001 (C = 1) ; Shift left [W2] (Word mode) ; Store result to [W12]
	Before Instruction W2 0900 W12 1002	; Post-increment W2 After Instruction W2 0902 W12 1002

Data 0900

Data 1002

SR

800F

001E

0001 (C = 1)

SL		Shift Left b	y Short Lite	ral		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SL	Wb,	#lit4,	Wnd	
Operands:	Wb ∈ [W0 W15] lit4 ∈ [015] Wnd ∈ [W0 W15]					
Operation:	lit4<3:0> $\rightarrow$ Shift_Val Wnd<15:Shift_Val> = Wb<15-Shift_Val:0> Wd <shift_val -="" 1:0=""> = 0</shift_val>					
Status Affected:	N, Z					
Encoding:	1101	1101	0www	wddd	d100	kkkk
	shifted out used for W The 'w' bits The 'd' bits	store the resu of the source b and Wnd. s select the a select the de provide the l	e register are ddress of the estination reg	e lost. Direct a e base regist gister.	addressing n	nust be
	Note:	This instruct			•	
Words:	1					
Cycles:	1					
Example 1: SL W	12, #4, W2		eft W2 by esult to W			
W2 SR	Before nstruction 78A9 0000 73, #12, W8	W2 SR ; shift l		12		
I W3 W8 SR	Before nstruction 0912 1002 0000	W3 W8 SR	2000			

SL		Shift Left b	y Wns				
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33	
	Х	Х	Х	Х	Х	Х	
Syntax:	{label:}	SL	Wb,	Wns,	Wnd		
Operands:	$Wns \in [W]$	Wb ∈ [W0 W15] Wns ∈ [W0W15] Wnd ∈ [W0 W15]					
Operation:	Wnd<15:S	Wns<4:0> $\rightarrow$ Shift_Val Wnd<15:Shift_Val> = Wb<15 - Shift_Val:0> Wd <shift_val -="" 1:0=""> = 0</shift_val>					
Status Affected:	N, Z						
Encoding:	1101	1101	0www	wddd	d000	SSSS	
bits of Wns (only up to 15 positions) and store the result in the destination register Wnd. Any bits shifted out of the source register and lost. Register direct addressing must be used for Wb, Wns and Wnd. The 'w' bits select the address of the base register. The 'd' bits select the destination register. The 's' bits select the source register. <b>Note 1:</b> This instruction operates in Word mode only.							
	2:	If Wns is gre	ater than 15	, Wnd will be	loaded with	0x0.	
Words:	1						
Cycles:	1						
Example 1: SL	WO, W1, W2		left W0 by result to W				
	Before Instruction		After Instruction				
W		W					
W		W					
W. SI		W2 SF					
Example 2: SL	W4, W5, W6		left W4 by result to W				
	Before		After				
W	Instruction 4 A409	W	Instruction				
W		W					
W		We					
SI	R 0000	SF	R 0000				

## **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUB{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	-	G) →destina	tion designa	ted by D		
Status Affected:	DC, N, OV,	,	Ũ	,		
Encoding:	1011	0101	0BDf	ffff	ffff	ffff
Description:	contents of destination destination	e contents of the specified register. The register. If V not specified	d file registe e optional W /REG is spe	r, and place REG operan cified, the re	the result in d determine sult is stored	the s the d in WREG
	The 'D' bit The 'f' bits	selects byte selects the d select the ac	estination (' Idress of the	o' for WREG file register.	, '1' for file re	egister).
		The extensic rather than a denote a wo	word opera rd operation	tion. You mag , but it is not	y use a .w e required.	
	2:	The WREG i	a act to wor	dia a na alatan		
			S SEL LO WOI	king register	WO.	
	1		S SEL LO WOIT	king register	VVO.	
Cycles:	1 1 <sup>(1)</sup>					o read and
read-modi details, se <u>Example 1:</u> SUB.B Ir WREG (W0) Data 1FFE SR	1 1 <sup>(1)</sup> 3E and PIC2- fy-write opera e <b>Note 3</b> in S 0x1FFF ; Before nstruction 7804 9439 0000	4E devices, t ations on nor Section 3.2.1 Sub. WREG Store resu WREG (W Data 1FF S	he listed cyc -CPU Speci "Multi-Cyc from (0x1F lit to 0x1F After Instruction 0) 7804 E 9039 SR 0001	le count doe al Function F le Instructio FFF) (Byte FFF (C = 1)	s not apply t Registers. Fo ns". <sup>mode</sup> )	
Cycles: Note 1: In dsPIC3: read-modi details, se Example 1: SUB.B Ir WREG (W0) Data 1FFE SR Example 2: SUB	1 1(1) 3E and PIC2- fy-write opera e <b>Note 3</b> in S 0x1FFF ; Before ostruction 7804 9439	4E devices, t ations on nor Section 3.2.1 Sub. WREG Store resu WREG (W Data 1FF S ; sub.	he listed cyc h-CPU Speci "Multi-Cyc from (0x1F alt to 0x1F After Instruction 0) 7804 E 9039 SR 0001 WREG from e result to After Instruction	the count doe al Function F le Instructio FFF) (Byte FFF (C = 1) $(0 \ge 0 \ge 0 \le 1)$	s not apply t Registers. Fo ns". <sup>mode</sup> )	

SUB		Subtract Li	teral from W	/n		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUB{.B}	#lit10,	Wn		
Operands:	lit10 $\in$ [0 255] for byte operation lit10 $\in$ [0 1023] for word operation Wn $\in$ [W0 W15]					
Operation:	(Wn) – lit10 →Wn					
Status Affected:	DC, N, OV, Z, C					
Encoding:	1011	0001	0Bkk	kkkk	kkkk	dddd
Description:	Subtract the 10-bit unsigned literal operand from the contents of the working register Wn, and store the result back in the working register Wn. Register direct addressing must be used for Wn.					
	The 'k' bits The 'd' bits Note 1: 2:	selects byte of specify the li select the ad The extension rather than a denote a work For byte ope unsigned val eral Operant operands in l	teral operan ddress of the n . B in the in word operat d operation, rations, the I ue [0:255]. S ds" for infor	d. working reg nstruction de ion. You may but it is not r iteral must be see <b>Section</b>	notes a byte / use a . w e required. e specified a: 4.6 "Using 1	xtension to s an <mark>0-bit Lit-</mark>
Words:	1					
Cycles:	1					
Example 1: SUB.B	#0x23, W0		0x23 from result to	WO (Byte mo WO	ode)	
	Before nstruction 7804 0000 #0x108, W	<b>SR</b> 4 ; Sub.		<b>  = 1)</b> W4 (Word m W4	node)	
lr W4 SR	Before Instruction 6234 0000	W4 SR		C = 1)		

SUB		Subtract S	nort Literal			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUB{.B}	Wb,	#lit5,	Wd	
	•	•			[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	(Wb) – lit5	-				
Status Affected:	DC, N, OV					
Encoding:	0101	0www	wBqq	qddd	d11k	kkkk
Description:				perand from t		
·	register Wi direct addr	b, and place	the result in be used for	the destination Wb. Register	on register W	d. Registe
		The extensi rather than	literal opera ion . B in the a word opera	nd, a five-bit i e instruction d ation. You ma	denotes a byt ay use a .w e	te operatio
		denote a wo	ord operation	n, but it is not	required.	
Words:	1					
Cycles:	1					
Example 1: SUB.B	W4, #0x10		ub. 0x10 fr tore result	rom W4 (Byt) t to W5	e mode)	
	Before		After			
In W4	nstruction 1782	W	Instruction			
W5	7804	Wi Wi	_			
SR	0000	SF		OV, C = 1)		
Example 2: SUB	W0, #0x8,			from W0 (W	d mode)	
Example 2. 505	WU, #UAU,			sult to [W2		
	Before nstruction		After Instruction			
In	-SHERCHUND					
In W0	F230	W				
-		W0 W2	0 F230			
W0	F230		0 F230 2 2006 4 F228	N, C = 1)		

PIC24F X abel:}	PIC24H X	PIC24E			
	Х	110212	dsPIC30F	dsPIC33F	dsPIC33E
abel:}	~	Х	Х	Х	Х
	SUB{.B}	Wb,	Ws,	Wd	
			[Ws],	[Wd]	
			[Ws++],	[Wd++]	
			[Ws],	[Wd]	
			[++Ws],	[++Wd]	
			[Ws],	[Wd]	
/s ∈ [W0	W15]				
Vb) – (Ws	s) →Wd				
C, N, OV,	Z, C				
0101	0www	wBqq	qddd	dppp	SSSS
he 'w' bits he 'B' bit : he 'q' bits he 'd' bits he 'p' bits	select the a selects byte select the de select the de select the de	ddress of the or word oper estination Ac estination regource Addres	e base regist ration ('0' for Idress mode gister. ss mode.	er. word, '1' for l	byte).
Note:	rather than a	a word opera	ation. You ma	ny use a .w e	
		•	,	•	
	$Vs \in [W0]$ $Vd \in [W0]$ Vb) - (Ws) C, N, OV, 0101 ubtract the ase register dial direct add he 'w' bits he 'g' bits he 'g' bits he 's' bits Note:	ubtract the contents of ase register Wb and p egister direct addressi direct addressing may he 'w' bits select the a he 'B' bit selects byte he 'q' bits select the de he 'd' bits select the de he 's' bits select the so he 's' bits select the so <b>Note:</b> The extension rather than a denote a wo	$Vs \in [W0 W15]$ $Vd \in [W0 W15]$ $Vb) - (Ws) → Wd$ $C, N, OV, Z, C$ $0101 0www wBqq$ ubtract the contents of the source ase register Wb and place the resu egister direct addressing must be direct addressing may be used for he 'w' bits select the address of the he 'B' bit selects byte or word oper he 'q' bits select the destination re- he 'd' bits select the source Addres he 's' bits select the source register he 's' bits select the source register <b>Note:</b> The extension . B in the rather than a word oper	$[++Ws],$ $[Ws],$ $[b \in [W0 W15]$ $[b \in [W0 W15]$ $[d \in [W0 W15]$ $[d \in [W0 W15]$ $[Vb) - (Ws) \rightarrow Wd$ $C, N, OV, Z, C$ $\boxed{0101}  0_{WWW}  wBqq  qddd$ $wbtract the contents of the source register Ws for ase register Wb and place the result in the dest egister direct addressing must be used for Wb. direct addressing may be used for Ws and Wd the 'w' bits select the address of the base register bit selects byte or word operation ('0' for he 'q' bits select the destination Address mode. The 'p' bits select the source register. The 's' bits select the source register. The extension .B in the instruction direct addression and the 's' bits select the source register. The extension .B in the instruction direct addression address mode. The instruction direct addression address mode. The 's' bits select the source register. The extension .B in the instruction direct addression addression address mode. The 's' bits select the source register. The extension .B in the instruction direct addression ad$	$[++Ws], [++Wd]$ $[Ws], [Wd]$ $/b \in [W0 W15]$ $/s \in [W0 W15]$ $/d \in [W0 W15]$ $/b) - (Ws) \rightarrow Wd$ $C, N, OV, Z, C$ $0101  0www  wBqq  qddd  dppp$ ubtract the contents of the source register Ws from the contrase register Wb and place the result in the destination registe egister direct addressing must be used for Wb. Either registed direct addressing may be used for Ws and Wd. $he 'w' bits select the address of the base register.$ $he 'B' bit selects byte or word operation ('0' for word, '1' for I)$ $he 'q' bits select the destination register.$ $he 'g' bits select the source Address mode.$ $he 's' bits select the source register.$ $Note: The extension .B in the instruction denotes a byte rather than a word operation. You may use a .w e$

W1

SR

7844

0108 (DC, N = 1)

W1

SR

7844

0000

Example 2: SUB	W7, [W8++]	;	; Sub. [W8] from W7 (Word mode) ; Store result to [W9] ; Post-increment W8 ; Post-increment W9
	Before		After
I	nstruction	li	nstruction
W7	2450	W7	2450
W8	1808	W8	180A
W9	2020	W9	2022
Data 1808	92E4	Data 1808	92E4
Data 2020	A557	Data 2020	916C
SR	0000	SR	010C (DC, N, OV = 1)

SUB		Subtract A	ccumulator	S		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
				Х	Х	Х
Syntax:	{label:}	SUB	Acc			
Operands:	Acc ∈ [A,B]	]				
Operation:	$\frac{\text{If (Acc = A)}}{\text{Acc A}}$		<b>C</b> A			
	Else:	АССВ →АС АССА →АС	-			
Status Affected:	OA, OB, OA	AB, SA, SB,	SAB			
Encoding:	1100	1011	A011	0000	0000	0000
Description:		store the re		ified accumul o Acc. This ir		
	The 'A' bit s	pecifies the	destination	accumulator.		
Words:	1					
Cycles:	1					
Example 1: SUB	; St	ore the re	CB from ACC esult to AC 0000 (no sa	CCA		
	Before			After		
	Instructio	n		Instructio	n	
ACCA	76 120F 0		ACCA	52 1EFC 4		
ACCB	23 F312 B		ACCB	23 F312 E		
CORCON		0000	CORCON		0000	
SR	(	0000	SR		1100 (OA, C	DB = 1)
Example 2: SUB	B ; Su	btract ACC	A from ACC	B		
<b>i</b>			sult to AC			
	iCC	RCON = 0XU	040 (SATB	= 1)		
	Before			After		
F	Instruction			Instruction		
ACCA	FF 9022 2I		ACCA	FF 9022 2E		
ACCB	00 2456 8		ACCB	00 7FFF FF		
CORCON					040	
SR	0	000	SR	14	400   (SB, S	AB = 1)

SUBB	<b>DIA</b> - 1-			Carry bit from		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBB{.B}	f	{,WREG}		
Operands:	f ∈ [0 8′	191]				
Operation:	(f) – (WRE	$(\overline{C}) \to de$	estination de	signated by	D	
Status Affected:	DC, N, OV	', Z, C				
Encoding:	1011	0101	1BDf	ffff	ffff	ffff
	register an WREG op specified, t	g (Carry flag i Id place the re erand determ the result is s ored in the file	esult in the c ines the des tored in WR	destination re stination regi	egister. The oster. If WRE	optional G is
	The 'D' bit	selects byte selects the d select the ac	estination (	0' for WREG	, '1' for file re	. ,
	Note 1:	The extension rather than a denote a wo	word opera	tion. You ma , but it is not	y use a . w e required.	
	2: 3:	The WREG i The Z flag is These instru	"sticky" for 2	ADDC, CPB		SUBBR.
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mod details, se	lify-write oper ee Note 3 in 9 B 0x1FFF ;	24E devices, t rations on nor Section 3.2.1 Sub. WREG Store resu	-CPU Speci "Multi-Cyc	al Function F le Instructio	Registers. Fo ns".	r more
	Before		After			
	Instruction		Instruction			
WREG (W0) Data 1FFE SR	9439	WREG (W0) Data 1FFE SF	8F39	DC, C = 1)		
	DxA04, WREG	; Sub. WF		from (0xA04	) (Word mo	de)
l WREG (W0) Data 0A04 SR	6235	WREG (W0 Data 0A04 SR	6235	C = 1)		

SUBB		Subtract W	n from Lite	al with Borr	ow	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBB{.B}	#lit10,	Wn		
Operands:	-	. 255] for byt . 1023] for we W15]	•	n		
Operation:	(Wn) – lit1	$O - (\overline{C}) \rightarrow Wn$				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1011	0001	1Bkk	kkkk	kkkk	dddd
Description:	flag inverse	the unsigned 1 e, $\overline{C}$ ) from the back in the wo sed for Wn.	e contents of	the working	register Wn,	and store
	The 'k' bits The 'd' bits	selects byte of specify the list select the ac The extensio	teral operan ddress of the	d. working reg	ister.	
		rather than a denote a wor	word operat	tion. You may	usea.we	
		For byte ope unsigned val eral Operan operands in	ue [0:255]. S ds" for infor	See Section	4.6 "Using 1	0-bit Lit-
	3:	The Z flag is	"sticky" for 2		SUBB and s	SUBBR.
Words:	1					
Cycles:	1					
Example 1: SUBB.B	#0x23, W		0x23 and e result t	⊂ from W0 o W0	(Byte mode)	
	Before struction 7804 0000 #0x108, W		2 0108 (E	DC, N = 1) C from W4 W4	(Word mode)	
	Before struction 6234 0001 (C	W4 = 1) SR		C = 1)		

SUBB		Subtract SI	hort Literal	from Wb wit	h Borrow	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBB{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	(Wb) – lit5	$-(\overline{C}) \rightarrow Wd$				
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	0101	lwww	wBqq	qddd	d11k	kkkk
	used for W Wd. The 'w' bits The 'B' bit The 'q' bits The 'd' bits The 'k' bits	e destination /b. Either regi s select the a selects byte s select the de s select the de s provide the	ster direct or ddress of the or word oper estination Ac estination re- literal operar	e base regist ration ('0' for ddress mode, gister, nd, a five-bit i	ressing may b er. word, '1' for l nteger numb	be used for byte). er.
	Note 1:	The extension rather than a denote a wo	word opera	tion. You mag	yusea.we	
	2:	The Z flag is These instru	-		, SUBB and	SUBBR.
Words:	1					
Cycles:	1					
Example 1: SUBB.B	W4, #0x1		ub. 0x10 an core result		4 (Byte mod	.e)
lr W4 W5 SR	Before nstruction 1782 7804 0000	W2 W5 SF	5 7871	DV, C = 1)		

**Example 2:** SUBB W0, #0x8, [W2++]; Sub. 0x8 and  $\overline{C}$  from W0 (Word mode) ; Store result to [W2] ; Post-increment W2 Before After Instruction Instruction 0009 0009 W0 W0 W2 2004 W2 2006 Data 2004 Data 2004 A557 0000 SR 0002 (Z = 1) SR 0103 (DC, Z, C = 1)

SUBB		Subtract W	/s from Wb	with Borrow	,	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBB{.B}	Wb,	Ws,	Wd	
				[Ws],	[Wd]	
				[Ws++],	[Wd++]	
				[Ws],	[Wd]	
				[++Ws],	[++Wd]	
				[Ws],	[Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]				
Operation:	(Wb) – (W	$s) - (\overline{C}) \rightarrow Wo$	d			
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	0101	lwww	wBqq	qddd	dppp	SSSS
Description:	(Carry flag place the r must be us for Ws and		from the con estination re legister direc	tents of the b gister Wd. Re t or indirect a	ase register egister direct addressing m	Wb, and addressing
	The 'B' bit The 'q' bits The 'd' bits The 'p' bits	s select the a selects byte s select the d s select the d s select the s s select the s	or word oper estination Ac estination re ource Addre	ration ('0' for ddress mode gister. ss mode.	word, '1' for	byte).
	Note 1:	The extension rather than a	on . B in the i	instruction de tion. You may , but it is not	/usea.we	
	2:		s "sticky" for a ctions can o	ADDC,CPB, nly clear Z.	SUBB and S	SUBBR.
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-modi	fy-write oper e <b>Note 3</b> in 9		n-CPU Speci I "Multi-Cyc . W1 and C	al Function R le Instruction	egisters. For ns".	
		; Stoi	re result t	.o WU		
Ir	Before struction		After Instruction			
wo	1732	W				
W1	7844	W				
SR	0000	SF	R 0108 (I	DC, N = 1)		

Example 2: SUBB	W7,[W8++],[W9	; Sto ; Pos			(Word mode)
	Before		After		
I	nstruction	li	nstructior	ו	
W7	2450	W7	2450		
W8	1808	W8	180A		
W9	2022	W9	2024		
Data 1808	92E4	Data 1808	92E4		
Data 2022	A557	Data 2022	916B		
SR	0000	SR	010C	(DC, N, OV = 1)	)

SUBBR		Subtract f f	rom WREG	with Borro	w	
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBBR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(WREG) –	$(f) - (\overline{C}) \rightarrow de$	stination de	signated by	D	
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1011	1101	1BDf	ffff	ffff	ffff
Description:	(Carry flag in the desti destination	e contents of inverse, C) fr nation registe register. If W not specified	om the cont er. The optio REG is spe	ents of WRE nal WREG o cified, the re	G, and place operand dete sult is stored	e the result rmines the I in WREG.
	The 'D' bit The 'f' bits	selects byte c selects the de select the ade	estination ('0 dress of the	)' for WREG, file register.	'1' for file re	egister).
		The extension rather than a denote a wor	word operat	tion. You ma	yusea.we	
		The WREG is				
		The Z flag is ' These instruc	-		SUBB and	SUBBR.
Words:	1					
Cycles:	1(1)					
read-modi details, se	fy-write opera e Note 3 in S B 0x803 ;	4E devices, th ations on non Section 3.2.1 Sub. (0x80 Store resu	-CPU Specia "Multi-Cycl	al Function R e Instruction	Registers. Fo ns".	
Ir WREG (W0) Data 0802 SR	Before istruction 7804 9439 0002 (Z =	WREG (W Data 080 = 1) S		1		
L		G; Sub. (0:			G (Word mo	de)
lr WREG (W0) Data 0A04 SR	Before Instruction 6234 6235 0000	WREG (W Data 0A S				

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X
		X				~
Syntax:	{label:}	SUBBR{.B}	Wb,	#lit5,	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb∈[W0					
	lit5 ∈ [0 Wd ∈ [W0					
Operation:	-	$-(\overline{C}) \rightarrow Wd$				
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	0001	lwww	wBqq	qddd	d11k	kkkk
Description:	flag invers destinatio	the contents of $(e, \overline{C})$ from the negister Wd.	e 5-bit unsig Register dir	ned literal an ect addressir	d place the rend for the rend for the rendered to the second second second second second second second second s	esult in the
	The 'B' bit The 'q' bit The 'd' bit	s select the a selects byte o s select the de s select the de s provide the l	or word open estination Action Action re	ration ('0' for ddress mode gister.	word, '1' for	
		The extensio rather than a denote a wor	n .в in the word opera	instruction d tion. You ma	enotes a byte y use a .w e	e operation
	2:	The Z flag is These instrue	"sticky" for 2	ADDC, CPB,	-	SUBBR.
Words:	1					
Cycles:	1					
Example 1: SUBBE	R.В W0, #0₂	10, W1 ; Su ; St	b. W0 and ore result		0 (Byte mod	le)

I	Before nstructior	n	I	After nstructior	ı
W0	F310		W0		
W1	786A		W1	7800	
SR	0003	(Z, C = 1)	SR	0103	(DC, Z, C = 1)

		After				
I	nstructior	า	I	nstructior	า	
W0	0009		W0	0009		
W2	2004		W2	2006		
Data 2004	A557	Data 2	004	FFFE		
SR	0020	(Z = 1)	SR	0108	(DC, N = 1)	

 $\label{eq:subbrack} \underline{\text{Example 2:}} \quad \text{SUBBR W0, \#0x8, [W2++]; Sub. W0 and } \overline{\text{C}} \text{ from 0x8 (Word mode)} \\ & ; \text{ Store result to [W2]} \\ & ; \text{ Post-increment W2} \\ \end{aligned}$ 

5

SUBBR		Subtract W	b from Ws	with Borrow		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBBR{.B}	Wb,	Ws,	Wd	
				[Ws],	[Wd]	
				[Ws++],	[Wd++]	
				[Ws],	[Wd]	
				[++Ws],	[++Wd]	
				[Ws],	[Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]				
Operation:	(Ws) – (W	b) – $(\overline{C}) \rightarrow Wc$	l			
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	0001	lwww	wBqq	qddd	dppp	SSSS
	The 'B' bit The 'q' bits The 'd' bits The 'p' bits	d. s select the a selects byte of s select the do s select the do s select the so s select the so	or word oper estination Ac estination re ource Addre	ration ('0' for ddress mode gister. ss mode.	word, '1' for	byte).
	Note 1:	The extension rather than a denote a work	word opera	tion. You mag	yusea.we	
	2:	The Z flag is These instrue	"sticky" for a	ADDC, CPB,	-	SUBBR.
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo	33E and PIC2 dify-write oper see <b>Note 3</b> in 9	ations on non	-CPU Speci	al Function R	egisters. For	
Example 1: SUBBR	R.B W0, W1,		. W0 and C re result	from W1 () to W0	Byte mode)	
	Before		After			

	Before		After				
I	nstructior	n I	Instruction				
W0	1732	WO	1711				
W1	7844	W1	7844				
SR	0000	SR	0001	(C =			

1)

Example 2: SUBBR	W7,[W8++],[W	; Sto ; Pos			(Word mode)
	Before		After		
I	nstruction	I	nstructior	ו	
W7	2450	W7	2450		
W8	1808	W8	180A		
W9	2022	W9	2024		
Data 1808	92E4	Data 1808	92E4		
Data 2022	A557	Data 2022	6E93		
SR	0000	SR	0005	(OV, C = 1)	

# 16-bit MCU and DSC Programmer's Reference Manual

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
··· F ·····	X	X	X	X	X	X
Syntax:	{label:}	SUBR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(WREG) –	(f) →destina	tion designa	ited by D		
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1011	1101	OBDf	ffff	ffff	ffff
	destination destination If WREG is	working regi register. The register. If V not specifie	e optional W VREG is spe d, the result	REG operan ecified, the re is stored in t	d determine sult is stored he file regist	s the d in WREC er
	The 'D' bit The 'f' bits	selects byte selects the d select the ac The extensio	estination (' Idress of the	0' for WREG file register.	, '1' for file re	egister).
		rather than a denote a wo				extension t
		The WREG i	-		-	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mo	C33E and PIC2 odify-write oper see <b>Note 3</b> in § .B 0x1FFF	ations on nor Section 3.2.1 ; Sub. (0x	n-CPU Speci "Multi-Cyc	ial Function F le Instructio	Registers. Fo ns".	
WREG (WC Data 1FFI SI	E 9439 R 0000	WREG (W0) Data 1FFE SF	7039 0000			
Example 2: SUBR	0xA04, WRE		result to	om WREG (Wo WREG	rd mode)	
	Before Instruction		After Instruction			
WREG (WC	) 6234	WREG (WO	) FFFF			
Data 0A0	4 6235	Data 0A04	6235			

SR 0008 (N = 1)

SR 0000

SUBR						
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	X
Syntax:	{label:}	SUBR{.B}	Wb,	#lit5	Wd	
					[Wd]	
					[Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	lit5 – (Wb)	→Wd				
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	0001	0www	wBqq	qddd	d11k	kkkk
Description:	Subtract th	ne contents o	f the base re	gister Wb fro	om the unsigi	ned 5-bit
	Register d	and, and placi irect address dressing may	ing must be	used for Wb.		
	The 'q' bits The 'd' bits	selects byte s select the d s select the d s provide the	estination A estination re	ddress mode gister.		
	Note:	rather than	a word oper	instruction d ation. You ma , but it is not	ay use a .w e	
Words:	1					
Cycles:	1					
Example 1: SUBR.B	W0, #0x1	.0, W1 ; Sı ; St	ub. WO from tore result		e mode)	
	Before		After			
ir Wo	struction F310	W	Instruction F310			
W0 W1	786A	W <sup>2</sup>				
SR	0000	SF		DC, Z, C = 1)		
Example 2: SUBR		[W2++] ;				
<u>Example 2.</u> 502.	, , , , , , , , , , , , , , , , , , ,	;		ult to [W2]	14	
le.	Before		After			
u Wo	struction 0009	W	Instruction 0 0009			
W2	2004	VV2	2 2006			
002 Data 2004	2004 A557	W2 Data 2004				

Instruction Descriptions

 $\ensuremath{\textcircled{}^\circ}$  2005-2011 Microchip Technology Inc.

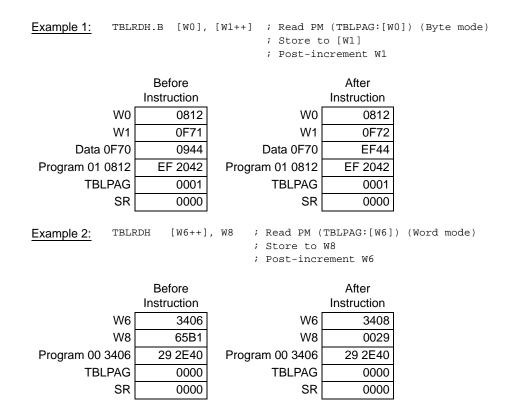
SUBR		Subtract W	/b from Ws			
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SUBR{.B}	Wb,	Ws,	Wd	
				[Ws],	[Wd]	
				[Ws++],	[Wd++]	
				[Ws],	[Wd]	
				[++Ws],	[++Wd]	
				[Ws],	[Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]				
Operation:	(Ws) – (W	b) →Wd				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	0001	0www	wBqq	qddd	dppp	SSSS
	The 'w' bits The 'B' bit The 'q' bits The 'd' bits The 'p' bits	s select the a selects byte	address of th or word ope estination Ad estination re ource Addre	ss mode.	er. word, '1' for	byte).
	Note:	rather than	a word opera	instruction d ation. You ma , but it is not	ay use a .w e	
Words:	1		•		·	
Cycles:	1 <sup>(1)</sup>					
read-modi details, se <u>Example 1:</u> SUBR.в	fy-write oper e <b>Note 3</b> in \$ w0, w1, Before nstruction	ations on no Section 3.2.1 W0 ; Sub ; Sto:	n-CPU Speci Multi-Cyc W0 from W re result t After Instruction	ele count does al Function R le Instruction M1 (Byte mod to W0	egisters. For ns".	
W0 W1 SR	1732 7844 0000	W W SF	1 7844	C = 1)		

Example 2: SUBR	W7, [W8++]	;	Store r Post-in	from [W8] (Word mode) esult to [W9] crement W8 crement W9
	Before		After	
I	Instruction	l.	nstructior	า
W7	2450	W7	2450	
W8	1808	W8	180A	
W9	2022	W9	2024	
Data 1808	92E4	Data 1808	92E4	
Data 2022	A557	Data 2022	6E94	
SR	0000	SR	0005	(OV, C = 1)

SWAP		Byte or Nit	oble Swap V	Vn		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	SWAP{.B}	Wn			
Operands:	Wn∈ [W0	W15]				
Operation:	For word o	$4 \rightarrow (Wn) < 3$				
Status Affected:	None					
Encoding:	1111	1101	1800	0000	0000	SSSS
	The 's' bits	select the ac	ddress of the	e working reg	jister.	• •
	The 'B' bit s	anged. Regi	or word oper	ration ('0' for	word, '1' for	
	Note:	The extension	on . B in the	instruction of	lenotes a by	te operatio
		rather than a denote a wo				extension to
Words:	1				·	
Cycles:	1					
Example 1: SWAP.B	WO ; 1	Nibble swap	(W0)			
	Before		After			
F	struction		Instruction			
W0 SR	AB87 0000	WC SR				
34			L]			
Example 2: SWAP		Byte swap (	w0)			
Example 2: SWAP	w0 ; E Before		After			
Example 2: SWAP	W0 ; E		After Instruction			

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	ГІС24F Х	Х	X X	X	X	X
		<u> </u>	L	<u> </u>	<u> </u>	<u> </u>
Syntax:	{label:}	TBLRDH{.B}		Wd		
			[Ws++],	[Wd]		
			[Ws],	[Wd++]		
			[++Ws],	[Wd]		
			[Ws],	[++Wd]		
				[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	For byte op If (LSB(V 0 →W	<i>N</i> s) = 1)				
	Else					
		am Mem [(TBL	.PAG),(Ws)] ·	<23:16> →W	d	
	<u>For word o</u> r Program 0 →Wd <	Mem [(TBLPA	∖G),(Ws)] <2:	3:16> →Wd <	:7:0>	
Status Affected:	None					
Encoding:	1011	1010	1Bqq	qddd	dppp	SSSS
Description:	store it to th memory is TBLPAG<7 addressing	contents of the he destination of formed by con 7:0>, with the e g must be used g may be used	register Wd. neatenating the effective addro I for Ws, and	The target wo ne 8-bit Table ress specified	ord address o Pointer regis by Ws. Indir	of program ster, rect
	In Word mo register (du memory by	ode, zero is sto ue to non-existe /te (PM<23:16> ne Least Signifi	ored to the Mo ent program >) at the spec	memory) and cified program	the third pro	ogram
	not word-al non-exister memory by	de, the source ligned, zero is nt program me /te (PM<23:16 ne destination r	stored to the mory). If Ws >) at the spec	e destination r	egister (due ed, the third	to program
	The 'q' bits The 'd' bits The 'p' bits	selects byte or select the des select the des select the sou select the sou	stination Addr stination regis urce Address	ress mode. ster.	ord, '1' for by	'te).
		The extension than a word r word move, bu	move. You m	nay use a .w		
Words:	1					
Words: Cycles:		PIC24H, dsPl	C30F, dsPIC	33F)		

#### 16-bit MCU and DSC Programmer's Reference Manual



TBLRDL	-					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	TBLRDL{.B}	[Ws],	Wd		
			[Ws++],	[Wd]		
			[Ws],	[Wd++]		
			[++Ws],	[Wd]		
			[Ws],	[++Wd]		
				[Wd]		
Operands:	$Ws \in [W0]$ $Wd \in [W0]$	-				
Operation:		<u>peration:</u> <u>Ws) = 1</u> ) ram Mem [(TBI	LPAG),(Ws)]	<15:8> →Wd	l	
	Progra For word o	ram Mem [(TBL <u>operation:</u> n Mem [(TBLP/				
Status Affected:	None		· ·			
Encoding:	1011	1010	0Bqq	qddd	dppp	SSSS
Description:	store it to the store it to the memory is TBLPAG<7 addressing	contents of the the destination formed by cor 7:0>, with the e g must be used g may be used	register Wd. ncatenating the effective add d for Ws, and	. The target we the 8-bit Table Iress specified	ord address Pointer regis by Ws. Indir	of program ster, rect
	In Word me destination contents of	ode, the lower n register. In By f Ws. If Ws is r	r 2 bytes of pr yte mode, the	e source addronged, the seco	ress depends and byte of th	s on the ne program
	word-align	ord (PM<15:7> ed, the first by he destination	vte of the prog	gram memory		
	word-aligned stored to the The 'B' bit The 'q' bits The 'd' bits The 'p' bits	ed, the first by	register. register. r word opera stination Add stination regi urce Address	gram memory ation ('0' for wo dress mode. ister. s mode.	/ word (PM<7	7:0>) is
	word-aligned stored to the The 'B' bit The 'q' bits The 'd' bits The 'p' bits	ed, the first by he destination selects byte or s select the des s select the des s select the sou s select the sou The extension	rte of the prog register. In word opera stination Add stination regi urce Address urce register. n . B in the in move. You n	gram memory ation ('0' for wo dress mode. ister. s mode. nstruction den may use a . w	v word (PM<7 ord mode, '1' notes a byte r	7:0>) is ' for byte). move rathe
Words:	word-aligne stored to th The 'B' bit The 'q' bits The 'q' bits The 'p' bits The 's' bits	ed, the first by he destination selects byte or s select the des s select the des s select the sou s select the sou The extension than a word	rte of the prog register. In word opera stination Add stination regi urce Address urce register. n . B in the in move. You n	gram memory ation ('0' for wo dress mode. ister. s mode. nstruction den may use a . w	v word (PM<7 ord mode, '1' notes a byte r	7:0>) is ' for byte). move rathe
Words: Cycles:	word-aligne stored to th The 'B' bit The 'q' bits The 'q' bits The 'p' bits The 's' bits <b>Note:</b>	ed, the first by he destination selects byte or s select the des s select the des s select the sou s select the sou The extension than a word	rte of the prog register. In word opera stination Add stination regi urce Address urce register. n .B in the in move. You n but it is not re	gram memory ation ('0' for we dress mode. ister. s mode. nstruction den may use a . we equired.	v word (PM<7 ord mode, '1' notes a byte r	7:0>) is ' for byte). move rathe

# 16-bit MCU and DSC Programmer's Reference Manual

Example 1: TBL	RDL.B [W0++	], W1 ; Read PM ( ; Store to ; Post-incr	Wl	(Byte mode)
	Instruction		Instruction	
W0	0813	W0	0814	
W1	0F71	W1	0F20	
Data 0F70	0944	Data 0F70	EF44	
Program 01 0812	EF 2042	Program 01 0812	EF 2042	
TBLPAG	0001	TBLPAG	0001	
SR	0000	SR	0000	
Example 2: TBLE	RDL [W6],	[W8++] ; Read PM	I (TBLPAG:[W6	(Nord mode)
		; Store t		(word mode)
	Before	; Store t	.o W8	)) (Wold mode)
	Before Instruction	; Store t	o W8 Acrement W8	j) (word mode)
W6		; Store t	o W8 Corement W8 After	j) (word mode)
W6 W8	Instruction	; Store t ; Post-in	o W8 crement W8 After Instruction	j) (word mode)
	Instruction 3406	; Store t ; Post-in W6	o W8 ccrement W8 After Instruction 3406	j) (word mode)
W8	Instruction 3406 1202	; Store t ; Post-in W6 W8	After Instruction 3406 1204	j) (word mode)
W8 Data 1202	Instruction 3406 1202 658B	; Store t ; Post-in W6 W8 Data 1202	After Instruction 3406 1204 2E40	j) (word mode)

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	TBLWTH{.B}	Ws,	[Wd]		
-			[Ws],	[Wd++]		
			[Ws++],	[Wd]		
			[Ws],	[++Wd]		
			[++Ws],	[Wd]		
			[Ws],	-		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	For byte op	-				
	(Ws) - For word o	→Program Mer operation: :0> →Program N				
Status Affected:	None					
Encoding:	1011	1011	1Bqq	qddd	dppp	SSSS
Description:	word of pro memory is TBLPAG<7	contents of the v ogram memory. formed by con 7:0>, with the e ldressing may b Vd.	v. The destinancatenating the	ation word add he 8-bit Table ress specified	dress of prog Pointer regis by Wd. Eithe	gram ster, er direct or
	upper byte a Wd that i	gram memory is of program me is word-aligned a Wd that is not	emory (PM<2 d in Byte mod	23:16>). This r de or Word mo	may be perfor ode. If Byte m	ormed using mode is
	The 'q' bits The 'd' bits The 'p' bits	selects byte or s select the des s select the des s select the sou s select the sou	stination Addr stination regis urce Address	lress mode. ster. s mode.	ərd, '1' for byt	te).
	Note:	The extension than a word move, but it is	nove. You may	ay use a .wext		
Words:	1					
Cycles:	2 <sup>(1)</sup>					
read-m	modify-write op	IC24E devices, perations on no in Section 3.2.	on-CPU Spec	cial Function R	Registers. For	

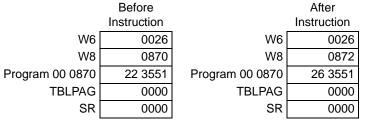
#### © 2005-2011 Microchip Technology Inc.

#### DS70157F-page 431

Example 1: TBL	WTH.B [W0++			e mode) TBLPAG:[W1])
	Before Instruction		After Instruction	
W0	0812	WO	0814	
W1	0F70	W1	0F70	
Data 0812	0944	Data 0812	EF44	
Program 01 0F70	EF 2042	Program 01 0F70	44 2042	
TBLPAG	0001	TBLPAG	0001	
SR	0000	SR	0000	

**Note:** Only the Program Latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

Example 2:	TBLWTH	W6, [W8++]	; Write W6 (Word mode)
			; to PM Latch High (TBLPAG:[W8])
			; Post-increment W8



**Note:** Only the Program Latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
•	X	X	X	X	X	X
Syntax:	{label:}	TBLWTL{.B}	Ws.	[Wd]		
Jymax.	נומההייייייייייייייייייייייייייייייייייי		ws, [Ws],	[Wd++]		
			[vvs], [Ws++],			
			[VVS++], [Ws],	[++Wd]		
			[vvs], [++Ws],			
			[Ws],	[- ** ~]		
Operands:	Ws ∈ [W0 . Wd ∈ [W0 .					
Operation:	For byte op If (LSB(V	peration:				
	(Ws) -	<u>→</u> Program Mer	m [(TBLPAG`	),(Wd)] <15:8:	>	
	<u>Else</u> (Ws) -	→Program Mer		·\ (\\/d\] ~7·0>		
	For word or	-				
Status Affected:	None		X	- / .		
Encoding:	1011	1011	0Bqq	qddd	dppp	SSSS
	memory is TBLPAG<7	ogram memory formed by con 7:0>, with the e dressing may b /d.	ncatenating the	he 8-bit Table ress specified	Pointer regis	ster, er direct or
	In Word mo Byte mode, If Wd is not memory (P	ode, Ws is stor e, the Least Sig ot word-aligned, PM<15:8>). If W nemory (PM<7:	gnificant bit of I, Ws is storec Vd is word-ali	f Wd determin d to the secon	nes the destin nd byte of pro	nation byte. ogram
	The 'B' bit s The 'q' bits The 'd' bits The 'p' bits	selects byte or s select the des s select the des s select the sou s select the sou	r word operati stination Addr stination regis urce Address	ress mode. ster. s mode.	rd, '1' for byt	e).
		The extension than a word m move, but it is	nove. You may	ny use a . w ext	•	
Words:	1					
Cycles:	2 <sup>(1)</sup>					
Note 1: In dsPI0	C33E and P	IC24E devices	, the listed cy	cle count doe	s not apply to	read and

© 2005-2011 Microchip Technology Inc.

5

Instruction Descriptions

# 16-bit MCU and DSC Programmer's Reference Manual

Example 1: TBL	VTL.B WO, [W	Vl++] ; Write W0. ; to PM Lat ; Post-incr	ch Low (TBL			
	Before	After				
	Instruction		Instruction			
W0	6628	W0	6628			
W1	1225	W1	1226			
Program 00 1224	78 0080	Program 01 1224	78 2880			
TBLPAG	0000	TBLPAG	0000			
SR	0000	SR	0000			
Note	: Only the P	rogram Latch is writt	en to. The co	ntents of pr		

**Note:** Only the Program Latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

Example 2: TBLV	WTL [W6],		] (Word r ch Low (TBLE ement W8			
	Before	After				
	Instruction		Instruction			
W6	1600	W6	1600			
W8	7208	W8	7208			
Data 1600	0130	Data 1600	0130			
Program 01 7208	09 0002	Program 01 7208	09 0130			
TBLPAG	0001	TBLPAG	0001			
SR	0000	SR	0000			

**Note:** Only the Program Latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

# **Section 5. Instruction Descriptions**

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х		Х	Х	
Syntax:	{label:}	ULNK				
Operands:	None					
Operation:	W14 →W1 (W15) – 2 (TOS) →W	→W15				
Status Affected:	None					
Encoding:	1111	1010	1000	0000	0000	0000
	(W15) equ	The Stack Fra al to the Fram e Frame Point	ne Pointer (			
Words:	1					
Words: Cycles:	1 1					
Cycles: Example 1: ULNK	1 ; Unlink	the stack				
Cycles: Example 1: ULNK	1	the stack	frame After Instruction			
Cycles: Example 1: ULNK	1 ; Unlink Before	the stack W14	After Instruction			
Cycles: <u>Example 1:</u> ULNK In: W14 W15	1 ; Unlint Before struction 2002 20A2	W14 W15	After Instruction 2000 2000			
Cycles: <u>Example 1:</u> ULNK In: W14 W15 Data 2000	1 ; Unlink Before struction 2002 20A2 2000	W14 W15 Data 2000	After Instruction 2000 2000 2000			
Cycles: <u>Example 1:</u> ULNK In: W14 W15	1 ; Unlint Before struction 2002 20A2	W14 W15	After Instruction 2000 2000 2000			
Cycles: <u>Example 1:</u> ULNK In: W14 W15 Data 2000	1 ; Unlink Before struction 2002 20A2 2000 0000	W14 W15 Data 2000	After Instruction 2000 2000 2000 0000			
Cycles: <u>Example 1:</u> ULNK Ins W14 W15 Data 2000 SR <u>Example 2:</u> ULNK	1 ; Unlink Before struction 2002 20A2 2000 0000 ; Unlink Before	W14 W15 Data 2000 SR	After Instruction 2000 2000 2000 2000 frame After			
Cycles: <u>Example 1:</u> ULNK In: W14 W15 Data 2000 SR <u>Example 2:</u> ULNK	1 ; Unlink Before struction 2002 2002 2000 0000 ; Unlink Before struction	W14 W15 Data 2000 SR	After Instruction 2000 2000 2000 2000 frame After Instruction			
Cycles: <u>Example 1:</u> ULNK In: W14 W15 Data 2000 SR <u>Example 2:</u> ULNK	1 ; Unlink Before struction 2002 20A2 2000 0000 ; Unlink Before struction 0802	W14 W15 Data 2000 SR the stack W14	After Instruction 2000 2000 2000 2000 frame After Instruction 0800			
Cycles: <u>Example 1:</u> ULNK In: W14 W15 Data 2000 SR <u>Example 2:</u> ULNK	1 ; Unlink Before struction 2002 2002 2000 0000 ; Unlink Before struction	W14 W15 Data 2000 SR	After Instruction 2000 2000 2000 2000 frame After Instruction 0800 0800			

5

ULNK		De-allocate	Stack Fran	ne		
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
			Х			Х
Syntax:	{label:}	ULNK				
Operands:	None					
Operation:	W14 →W1 (W15) – 2 (TOS) →W 0 →SFA bi	→W15 /14				
Status Affected:	SFA					
Encoding:	1111	1010	1000	0000	0000	0000
Mordo	(W15) equ to reset the	The Stack Fr al to the Fran e Frame Poin	ne Pointer (			
Words:	1					
Cycles:	1					
Example 1: ULNK	; Unlink	the stack	frame			
	Before struction		After Instruction			
W14	2002	W14				
W15	20A2	W15				
Data 2000	2000	Data 2000	2000			
SR	0000	SR	0000			
Example 2: ULNK	; Unlink	the stack	frame			
E	Before	the stack	After			
E Ins	Before		After Instruction			
E Ins W14	Before struction 0802	W14	After Instruction			
E Ins W14 	Before struction 0802 0812	W14 W15	After Instruction 0800 0800			
E Ins W14	Before struction 0802	W14	After Instruction 0800 00000			

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	Х	X	X	X	X
Syntax:	{label:}	XOR{.B}	f	{,WREG}	<u>I</u>	<u> </u>
				·		
Operands:	f ∈ [0 8′	-				
Operation:	.,	WREG) →des	stination desig	gnated by D		
Status Affected:	N, Z					
Encoding:	1011	0110	1BDf	ffff	ffff	ffff
	specified, t result is ste	the result is s tored in the fil	stored in WRI ile register.	stination regis REG. If WREG	G is not speci	ified, the
	result is sto The 'B' bit The 'D' bit The 'f' bits	tored in the fil t selects byte t selects the c s select the ac	ile register. or word oper destination ('0 ddress of the	eration ('0' for 0' for WREG,	word, '1' for , '1' for file re	byte). gister).
		rather than a denote a wo	a word operat ord operation,	ation. You may n, but it is not r	y use a .we required.	
		The WREG	is set to work	king register	W0.	
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mod	33E and PIC2 dify-write oper see <b>Note 3</b> in \$ 3 0x1FFF	Frations on nor Section 3.2.1	on-CPU Specia 1 "Multi-Cycl	ial Function R cle Instruction	Registers. For ns".	
	Before Instruction		After Instruction			
WREG (W0) Data 1FFE SR	9439	WREG (W0 Data 1FFE SF	E 9039	N = 1)		
Example 2: XOR	0xA04, WR		(0xA04) and e result to	d WREG (Word o WREG	d mode)	
	Before		After			

	Before		After				
I	nstructior	Instruction					
WREG (W0)	6234	WREG (W0)	C267				
Data 0A04	A053	Data 0A04	A053				
SR	0000	SR	0008	(N = 1)			

5

XOR	Exclusive OR Literal and Wn					
Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	XOR{.B}	#lit10,	Wn		
Operands:	lit10 $\in$ [0 255] for byte operation lit10 $\in$ [0 1023] for word operation Wn $\in$ [W0 W15]					
Operation:	lit10.XOR.	(Wn) →Wn				
Status Affected:	N, Z					
Encoding:	1011	0010	1Bkk	kkkk	kkkk	dddd
Description:	operand a	he logical exc nd the conten working regi /n.	ts of the wor	king register	Wn and store	e the result
	<ul> <li>The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'k' bits specify the literal operand.</li> <li>The 'd' bits select the address of the working register.</li> <li>Note 1: The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . w extension denote a word operation, but it is not required.</li> <li>2: For byte operations, the literal must be specified as an unsigned value [0:255]. See Section 4.6 "Using 10-bit Li eral Operands" for information on using 10-bit literal operands in Byte mode.</li> </ul>					operation ktension to s an <b>0-bit Lit-</b>
Words:	1					
Cycles:	1					
Example 1: XOR.B	#0x23, W0		0x23 and W e result t	0 (Byte mod o W0	le)	
W0 SR	Before nstruction 7804 0000 #0x108, W4	W0 SR ; xor	0000	W4 (Word mc o W4	ode)	
lr W4 SR	Before nstruction 6134 0000	W4 SR				

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	XOR{.B}	Wb,	#lit5,	Wd [Wd]	
					[Wd] [Wd++]	
					[Wd]	
					[++Wd]	
					[Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	. 31]				
Operation:	(Wb).XOR	R.lit5 →Wd				
Status Affected:	N, Z					
Encoding:	0110	lwww	wBqq	qddd	d11k	kkkk
Description:	register W the destine	the logical ex /b and the un ation register r register dire	signed 5-bit li Wd. Registe	iteral operant or direct addre	d and place the sing must b	he result in be used for
	The 'B' bit The 'q' bit The 'd' bit	ts select the a selects byte s select the d s select the d s provide the	or word oper lestination Ac lestination reg	ration ('0' for ddress mode. gister.	word, '1' for l	
	Note:	rather than		ation. You ma	lenotes a byt y use a .w e required.	
Words:	1				·	
Cycles:	1					
Example 1: XOF	R.B W4, #02	ĸ14, W5		and 0x14 (B esult to W5	yte mode)	
	Before		After			
W4	Instruction C822	W	Instruction	i		
W5			V5 1234			
SR	0000	S	SR 0000			
Example 2: XOF	र ₩2, #0₂	x1F, [W8++]				
				result to [] ncrement W8	W8]	
	Before		After			
			Instruction			
l W2	Instruction	v	Instruction V2 8505			
	Instruction 8505					

Instruction Descriptions

 $\ensuremath{\textcircled{}^\circ}$  2005-2011 Microchip Technology Inc.

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
·	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	XOR{.B}	Wb,	Ws, [Ws],	Wd [Wd]	
				[Ws++],	[Wd++]	
				[Ws],	[Wd]	
				[++Ws],	[++Wd]	
				[Ws],	[Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]				
Operation:	(Wb).XOR	.(Ws) →Wd				
Status Affected:	N, Z					
Encoding:	0110	lwww	wBqq	qddd	dppp	SSSS
Description:	register W result in th	the logical exe s and the cor de destination /b. Either reg /d.	ntents of the register Wd	base registe . Register dir	r Wb, and pla ect addressir	ice the ng must be
	The 'B' bit The 'q' bit The 'd' bit The 'p' bit	s select the a selects byte s select the d s select the d s select the s s select the s	or word oper estination Ac estination re ource Addre	ration ('0' for ddress mode gister. ss mode.	word, '1' for	byte).
	Note:	rather than	a word opera		denotes a byt ay use a .w e required.	
Words:	1					
Cycles:	1(1)					

Note 1:	In dsPIC33E and PIC24E devices, the listed cycle count does not apply to read and
	read-modify-write operations on non-CPU Special Function Registers. For more
	details, see Note 3 in Section 3.2.1 "Multi-Cycle Instructions".

Example 1: XOF	2.B Wl, [W5+	+], [W9++]	; XOR W1 and [W5] (Byte mode) ; Store result to [W9] ; Post-increment W5 and W9
W1 W5 W9 Data 2000 Data 2600	2600 115A	W1 W5 W9 Data 2000 Data 2600	After Instruction AAAA 2001 2601 115A 00F0
Example 2: XOF	0000	<b>SR</b> ; xc	0008 (N = 1) OR W1 and W5 (Word mode) core the result to W9
ו 1000 1000 1000 1000 1000 1000 1000 10	1234 A34D		After struction FEDC 1234 ECE8 0008 (N = 1)

5

# 16-bit MCU and DSC Programmer's Reference Manual

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	Х	Х	Х	Х	Х	Х
Syntax:	{label:}	ZE	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd		
Operands:	Ws∈ [W0 Wnd∈ [W	-				
Operation:	Ws<7:0> - 0 <i>→</i> Wnd<	→Wnd<7:0> 15:8>				
Status Affected:	N, Z, C					
Encoding:	1111	1011	1000	0ddd	dppp	SSSS
Description:	Zero-extend the Least Significant Byte in source working register Ws t a 16-bit value and store the result in the destination working register Wnd. Either register direct or indirect addressing may be used for Ws, and register direct addressing must be used for Wnd. The N flag is cleared and the C flag is set, because the zero-extended word is alway positive.				egister d for Ws, lag is	
	The 'd' bits select the destination register. The 'p' bits select the source Address mode. The 's' bits select the source register.					
	Note 1:	This operation		a byte to a wo	rd, and it use	esno.Bor
	2:	The source \ address mod		sed as a byte y '1'.	e operand, se	o any
Words:	1					
Cycles:	1 <sup>(1)</sup>					
read-mod	ify-write oper	24E devices, t ations on nor Section 3.2.1	n-CPU Specia	al Function R	egisters. For	

Example 1:	ZE	W3,	W4		zero-extend Store result		
			efore ructior	n	I	After nstructior	า
	W:	3	7839		W3	7839	
	W	4	1005		W4	0039	
	SF	२	0000		SR	0001	(C = 1)

Example 2:	ZE [W	2++], Wi	;	Zero-exte Store to Post-incr	W12	2
	Ir	Before Instruction	n	I	After nstructior	า
	W2	0900		W2	0901	
	W12	1002		W12	008F	
Data	a 0900	268F		Data 0900	268F	
	SR	0000		SR	0001	(C = 1)

5

NOTES:



# **Section 6. Built-in Functions**

# HIGHLIGHTS

This section of the manual contains the following major topics:

6.1	Introduction	. 446
6.2	Built-in Function List	. 447

#### 6.1 INTRODUCTION

This section describes the built-in functions that are specific to the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30).

Built-in functions give the C programmer access to assembler operators or machine instructions that are currently only accessible using in-line assembly, but are sufficiently useful that they are applicable to a broad range of applications. Built-in functions are coded in C source files syntactically like function calls, but they are compiled to assembly code that directly implements the function, and do not involve function calls or library routines.

There are a number of reasons why providing built-in functions is preferable to requiring programmers to use in-line assembly. They include the following:

- Providing built-in functions for specific purposes simplifies coding. 1.
- Certain optimizations are disabled when in-line assembly is used. This is not the case for 2. built-in functions.
- 3. For machine instructions that use dedicated registers, coding in-line assembly while avoiding register allocation errors can require considerable care. The built-in functions make this process simpler as you do not need to be concerned with the particular register requirements for each individual machine instruction.

The built-in functions are listed below followed by their individual detailed descriptions.

- \_\_builtin\_addab
- \_\_builtin\_add
- \_\_builtin\_btg
- \_\_\_builtin\_clr
- \_\_builtin\_clr\_prefetch
- \_\_builtin\_divf
- \_\_builtin\_divmodsd
- \_\_builtin\_divmodud
- \_\_builtin\_divsd
- builtin divud
- \_\_builtin\_dmaoffset
- builtin ed
- builtin edac
- \_\_builtin\_edsoffset
- builtin edspage
- \_\_builtin\_fbcl
- \_\_builtin\_lac
- builtin\_mac
- \_\_builtin\_modsd
- \_\_builtin\_modud
- builtin movsac
- \_\_builtin\_mpy

- \_\_builtin\_mpyn
- builtin msc
- builtin\_mulss
- \_\_\_builtin\_mulsu
- \_\_builtin\_mulus
- \_\_builtin\_muluu
- \_\_builtin\_nop
- \_\_builtin\_psvpage
- \_\_builtin\_psvoffset
- builtin readsfr
- \_\_builtin\_return\_address
- builtin\_sac
- builtin sacr
- \_\_builtin\_sftac
- builtin\_subab
- \_\_builtin\_tbladdress
- \_\_builtin\_tblpage
- builtin tbloffset
- \_\_builtin\_tblrdh
- \_\_builtin\_tblrdl
- builtin tblwth
- \_\_builtin\_tblwtl

This section describes only the built-in functions related to the CPU operations. The compiler provides additional built-in functions for operations such as writing to Flash program memory and changing the oscillator settings. Refer to the "MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide" (DS51284) for a complete list of compiler built-in functions.

# 6.2 BUILT-IN FUNCTION LIST

This section describes the programmer interface to the compiler built-in functions. Since the functions are "built-in", there are no header files associated with them. Similarly, there are no command-line switches associated with the built-in functions – they are always available. The built-in function names are chosen such that they belong to the compiler's namespace (they all have the prefix \_\_builtin\_), so they will not conflict with function or variable names in the programmer's namespace.

# \_builtin\_addab

# **Description:**

Add accumulators A and B with the result written back to the specified accumulator. For example:

```
register int result asm("A");
register int B asm("A");
```

result = \_\_builtin\_addab(result,B);

will generate:

add A

Prototype:

int \_\_builtin\_addab(int Accum\_a, int Accum\_b);

# Argument:

Accum\_a First accumulator to add. Accum\_b Second accumulator to add.

**Return Value:** 

Returns the addition result to an accumulator.

## Assembler Operator / Machine Instruction:

add

**Error Messages:** 

An error message appears if the result is not an accumulator register.

# \_builtin\_add

## **Description:**

Add value to the accumulator specified by result with a shift specified by literal shift. For example:

register int result asm("A"); int value; result = \_\_builtin\_add(result,value,0);

If value is held in w0, the following will be generated:

add w0, #0, A

# Prototype:

```
int __builtin_add(int Accum,int value,
const int shift);
```

## Argument:

Accum Accumulator to add.

value Integer number to add to accumulator value.

shift Amount to shift resultant accumulator value.

# Return Value:

Returns the shifted addition result to an accumulator.

# Assembler Operator / Machine Instruction:

add

# Error Messages:

- the result is not an accumulator register
- argument 0 is not an accumulator
- the shift value is not a literal within range

# \_builtin\_btg

## **Description:**

```
This function will generate a btg machine instruction. Some examples include:
int i; /* near by default */
int l __attribute__((far));
struct foo {
  int bit1:1;
} barbits;
int bar;
void some_bittoggles() {
  register int j asm("w9");
  int k;
  k = i;
  __builtin_btg(&i,1);
  __builtin_btg(&j,3);
  __builtin_btg(&k,4);
  __builtin_btg(&l,11);
  return j+k;
```

```
}
```

Note that taking the address of a variable in a register will produce warning by the compiler and cause the register to be saved onto the stack (so that its address may be taken); this form is not recommended. This caution only applies to variables explicitly placed in registers by the programmer.

# Prototype:

```
void __builtin_btg(unsigned int *, unsigned int 0xn);
```

## Argument:

- \* A pointer to the data item for which a bit should be toggled.
- $0x_n$  A literal value in the range of 0 to 15.

# **Return Value:**

Returns a btg machine instruction.

Assembler Operator / Machine Instruction:

btg

## **Error Messages:**

An error message appears if the parameter values are not within range.

# \_builtin\_clr

# **Description:**

Clear the specified accumulator. For example:

```
register int result asm("A");
result = __builtin_clr();
```

will generate:

clr A

Prototype:

int \_\_builtin\_clr(void);

Argument:

None

Return Value:

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

clr

**Error Messages:** 

An error message appears if the result is not an accumulator register.

# Section 6. Built-in Functions

# \_builtin\_clr\_prefetch

### **Description:**

Clear an accumulator and prefetch data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

*yptr* may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If AWB is non null, the other accumulator will be written back into the referenced variable.

## For example:

```
register int result asm("A");
register int B asm("B");
int x_memory_buffer[256]
__attribute__((space(xmemory)));
int y_memory_buffer[256]
__attribute__((space(ymemory)));
int *xmemory;
int *ymemory;
int *ymemory;
int awb;
int xVal, yVal;
xmemory = x_memory_buffer;
ymemory = y_memory_buffer;
```

### May generate:

clr A, [w8]+=2, w4, [w10]+=2, w5, w13

The compiler may need to spill w13 to ensure that it is available for the write-back. It may be recommended to users that the register be claimed for this purpose.

After this instruction:

- · result will be cleared
- xVal will contain x\_memory\_buffer[0]
- yVal will contain y\_memory\_buffer[0]
- xmemory and ymemory will be incremented by 2, ready for the next mac operation

# Prototype:

```
int __builtin_clr_prefetch(
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr, int *AWB,
int AWB_accum);
```

# \_builtin\_clr\_prefetch (Continued)

## Argument:

xptr	Integer pointer to x prefetch.
xval	Integer value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to y prefetch.
yval	Integer value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Accumulator write back location.
AWB_accum	Accumulator to write back.

**Note:** The arguments *xptr* and *yptr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

clr

# **Error Messages:**

- the result is not an accumulator register
- xval is a null value but xptr is not null
- yval is a null value but yptr is not null
- AWB\_accum is not an accumulator and AWB is not null

# **Section 6. Built-in Functions**

# \_builtin\_divf

## **Description:**

Computes the quotient *num* / *den*. A math error exception occurs if *den* is zero. Function arguments are unsigned, as is the function result.

## Prototype:

unsigned int \_\_builtin\_divf(unsigned int num, unsigned int den);

## Argument:

num numerator den denominator

## **Return Value:**

Returns the unsigned integer value of the quotient num / den.

Assembler Operator / Machine Instruction:

div.f

# \_builtin\_divmodsd

## **Description:**

Issues the 16-bit architecture's native signed divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture both the quotient and remainder.

## Prototype:

signed int \_\_builtin\_divmodsd(
signed long dividend, signed int divisor,
signed int \*remainder);

## Argument:

dividend number to be divided divisor number to divide by remainder pointer to remainder

# **Return Value:**

Quotient and remainder.

## Assembler Operator / Machine Instruction:

divmodsd

## Error Messages:

None.

# \_builtin\_divmodud

## **Description:**

Issues the 16-bit architecture's native unsigned divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture both the quotient and remainder.

# Prototype:

unsigned int \_\_builtin\_divmodud( unsigned long dividend, unsigned int divisor, unsigned int \*remainder);

#### Argument:

dividend	number to be divided
divisor	number to divide by
remainder	pointer to remainder

### **Return Value:**

Quotient and remainder.

Assembler Operator / Machine Instruction:

divmodud

**Error Messages:** 

None.

## \_builtin\_divsd

## **Description:**

Computes the quotient *num* / *den*. A math error exception occurs if *den* is zero. Function arguments are signed, as is the function result. The command-line option -Wconversions can be used to detect unexpected sign conversions.

## Prototype:

int \_\_builtin\_divsd(const long num, const int den);

## Argument:

num numerator den denominator

#### **Return Value:**

Returns the signed integer value of the quotient num / den.

## Assembler Operator / Machine Instruction:

div.sd

# \_\_builtin\_divud

# **Description:**

Computes the quotient *num* / *den*. A math error exception occurs if *den* is zero. Function arguments are unsigned, as is the function result. The command-line option -Wconversions can be used to detect unexpected sign conversions.

# Prototype:

unsigned int \_\_builtin\_divud(const unsigned long num, const unsigned int den);

# Argument:

num numerator den denominator

# Return Value:

Returns the unsigned integer value of the quotient num / den.

Assembler Operator / Machine Instruction:

div.ud

# \_builtin\_dmaoffset

## **Description:**

Obtains the offset of a symbol within DMA memory.

For example:

unsigned int result; char buffer[256] \_\_attribute\_\_((space(dma)));

result = \_\_builtin\_dmaoffset(&buffer);

May generate:

mov #dmaoffset(buffer), w0

## Prototype:

unsigned int \_\_builtin\_dmaoffset(const void \*p);

## Argument:

\*p pointer to DMA address value

## **Return Value:**

Returns the offset to a variable located in DMA memory.

Assembler Operator / Machine Instruction:

dmaoffset

## Error Messages:

An error message appears if the parameter is not the address of a global symbol.

# \_builtin\_ed

## **Description:**

Squares sqr, returning it as the result. Also prefetches data for future square operation by computing \*\*xptr - \*\*yptr and storing the result in \*distance.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```
register int result asm("A");
int *xmemory, *ymemory;
int distance;
result = __builtin_ed(distance,
```

&xmemory, 2, &ymemory, 2, &distance);

May generate:

ed w4\*w4, A, [w8]+=2, [W10]+=2, w4

# Prototype:

int \_\_builtin\_ed(int sqr, int \*\*xptr, int xincr, int \*\*yptr, int yincr, int \*distance);

## Argument:

sqr	Integer squared value.
xptr	Integer pointer to pointer to x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yincr	Integer increment value of y prefetch.
distance	Integer pointer to distance.

**Note:** The arguments *xptr* and *yptr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the squared result to an accumulator.

Assembler Operator / Machine Instruction:

ed

## **Error Messages:**

- · the result is not an accumulator register
- xptr is null
- yptr is null
- distance is null

# \_\_builtin\_edac

## **Description:**

Squares sqr and sums with the nominated accumulator register, returning it as the result. Also prefetches data for future square operation by computing \*\*xptr - \*\*yptr and storing the result in \*distance.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

## For example:

```
register int result asm("A");
int *xmemory, *ymemory;
int distance;
```

&distance);

## May generate:

edac w4\*w4, A, [w8]+=2, [W10]+=2, w4

## Prototype:

int \_\_builtin\_edac(int Accum, int sqr, int \*\*xptr, int xincr, int \*\*yptr, int yincr, int \*distance);

## Argument:

Accum	Accumulator to sum.
sqr	Integer squared value.
xptr	Integer pointer to pointer to x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yincr	Integer increment value of y prefetch.
distance	Integer pointer to distance.

**Note:** The arguments  $x_{ptr}$  and  $y_{ptr}$  must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the squared result to specified accumulator.

# Assembler Operator / Machine Instruction:

edac

## Error Messages:

- the result is not an accumulator register
- Accum is not an accumulator register
- xptr is null
- yptr is null
- distance is null

# \_builtin\_edsoffset

## **Description:**

Returns the eds page offset of the object whose address is given as a parameter. The argument p must be the address of an object in extended data space; otherwise an error message is produced and the compilation fails. See the space attribute in Section 2.3.1 "Specifying Attributes of Variables" of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

# Prototype:

unsigned int \_\_builtin\_edsoffset(int \*p);

# Argument:

p object address

# Return Value:

Returns the eds page number of the object whose address is given as a parameter

Assembler Operator / Machine Instruction:

edsoffset

# \_builtin\_edspage

# Description:

Returns the eds page number of the object whose address is given as a parameter. The argument p must be the address of an object in extended data space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

## Prototype:

unsigned int \_\_builtin\_edspage(int \*p);

## Argument:

p object address

## **Return Value:**

Returns the eds page number of the object whose address is given as a parameter.

Assembler Operator / Machine Instruction:

edspage

# **Section 6. Built-in Functions**

# \_builtin\_fbcl

## **Description:**

Finds the first bit change from left in value. This is useful for dynamic scaling of fixed-point data. For example:

int result, value; result = \_\_builtin\_fbcl(value);

May generate:

# fbcl w4, w5

Prototype:

int \_\_builtin\_fbcl(int value);

## Argument:

value Integer number of first bit change.

## **Return Value:**

Returns the shifted addition result to an accumulator.

Assembler Operator / Machine Instruction:

fbcl

## **Error Messages:**

An error message appears if the result is not an accumulator register.

# \_builtin\_lac

## **Description:**

Shifts value by shift (a literal between -8 and 7) and returns the value to be stored into the accumulator register. For example:

register int result asm("A"); int value; result = \_\_builtin\_lac(value,3);

May generate:

lac w4, #3, A

# Prototype:

int \_\_builtin\_lac(int value, int shift);

## Argument:

value Integer number to be shifted. shift Literal amount to shift.

## Return Value:

Returns the shifted addition result to an accumulator.

# Assembler Operator / Machine Instruction:

lac

## Error Messages:

- · the result is not an accumulator register
- · the shift value is not a literal within range

# \_builtin\_mac

## **Description:**

Computes  $a \ge b$  and sums with accumulator; also prefetches data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

*yptr* may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If AWB is non null, the other accumulator will be written back into the referenced variable.

For example:

```
register int result asm("A");
register int B asm("B");
int *xmemory;
int *ymemory;
int xVal, yVal;
result = __builtin_mac(result, xVal, yVal,
&xmemory, &xVal, 2,
&ymemory, &yVal, 2, 0, B);
```

May generate:

mac w4\*w5, A, [w8]+=2, w4, [w10]+=2, w5

Prototype:

int \_\_builtin\_mac(int Accum, int a, int b, int \*\*xptr, int \*xval, int xincr, int \*\*yptr, int \*yval, int yincr, int \*AWB, int AWB\_accum);

## Argument:

Accum	Accumulator to sum.
а	Integer multiplicand.
b	Integer multiplier.
xptr	Integer pointer to pointer to x prefetch.
xval	Integer pointer to value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yval	Integer pointer to value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Accumulator write-back location.
AWB_accum	Accumulator to write-back.

**Note:** The arguments *xptr* and *yptr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

mac

# \_\_builtin\_mac (Continued)

# Error Messages:

- the result is not an accumulator register
- Accum is not an accumulator register
- xval is a null value but xptr is not null
- ${\it yval}$  is a null value but  ${\it yptr}$  is not null
- AWB\_accum is not an accumulator register and AWB is not null

# \_builtin\_modsd

## **Description:**

Issues the 16-bit architecture's native signed divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture only the remainder.

# Prototype:

signed int \_\_builtin\_modsd(signed long dividend, signed int divisor);

## Argument:

dividend number to be divided divisor number to divide by

**Return Value:** 

Remainder.

Assembler Operator / Machine Instruction:

modsd

**Error Messages:** 

None.

# \_builtin\_modud

## **Description:**

Issues the 16-bit architecture's native unsigned divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture only the remainder.

## Prototype:

unsigned int \_\_builtin\_modud(unsigned long dividend, unsigned int divisor);

## Argument:

dividend number to be divided divisor number to divide by

## **Return Value:**

Remainder.

Assembler Operator / Machine Instruction:

modud

Error Messages:

None.

# Section 6. Built-in Functions

# \_builtin\_movsac

### **Description:**

Computes nothing, but prefetches data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

*yptr* may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If AWB is not null, the other accumulator will be written back into the referenced variable.

### For example:

register int result asm("A"); int \*xmemory; int \*ymemory; int xVal, yVal;

## May generate:

movsac A, [w8]+=2, w4, [w10]+=2, w5

## Prototype:

int \_\_builtin\_movsac(
int \*\*xptr, int \*xval, int xincr,
int \*\*yptr, int \*yval, int yincr, int \*AWB
int AWB\_accum);

## Argument:

xptr	Integer pointer to pointer to x prefetch.
xval	Integer pointer to value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yval	Integer pointer to value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Accumulator write back location.
AWB_accum	Accumulator to write back.

**Note:** The arguments *x*<sub>*p*</sub>*tr* and *y*<sub>*p*</sub>*tr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns prefetch data.

Assembler Operator / Machine Instruction:

movsac

## Error Messages:

- · the result is not an accumulator register
- xval is a null value but xptr is not null
- yval is a null value but yptr is not null
- AWB\_accum is not an accumulator register and AWB is not null

# \_builtin\_mpy

## **Description:**

Computes *a* x *b* ; also prefetches data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

yptr may be null to signify no Y prefetch to be performed, in which case the values of yincr and yval are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

#### For example:

```
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;
result = __builtin_mpy(xVal, yVal,
&xmemory, &xVal, 2,
&ymemory, &yVal, 2);
```

## May generate:

```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

## Prototype:

```
int __builtin_mpy(int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr);
```

## Argument:

а	Integer multiplicand.
b	Integer multiplier.
xptr	Integer pointer to pointer to x prefetch.
xval	Integer pointer to value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yval	Integer pointer to value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Integer pointer to accumulator selection.

**Note:** The arguments *x*<sub>*p*</sub>*tr* and *y*<sub>*p*</sub>*tr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

mpy

## **Error Messages:**

- the result is not an accumulator register
- xval is a null value but xptr is not null
- yval is a null value but yptr is not null

# Section 6. Built-in Functions

# \_builtin\_mpyn

### **Description:**

Computes -a x b; also prefetches data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

*yptr* may be null to signify no Y prefetch to be performed, in which case the values of *yincr* and *yval* are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

register int result asm("A"); int \*xmemory; int \*ymemory; int xVal, yVal;

## May generate:

mac w4\*w5, A, [w8]+=2, w4, [w10]+=2, w5

### Prototype:

int \_\_builtin\_mpyn(int a, int b, int \*\*xptr, int \*xval, int xincr, int \*\*yptr, int \*yval, int yincr);

## Argument:

а	Integer multiplicand.
b	Integer multiplier.
xptr	Integer pointer to pointer to x prefetch.
xval	Integer pointer to value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yval	Integer pointer to value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Integer pointer to accumulator selection.

**Note:** The arguments *xptr* and *yptr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

mpyn

#### Error Messages:

- · the result is not an accumulator register
- xval is a null value but xptr is not null
- yval is a null value but yptr is not null

# \_builtin\_msc

## **Description:**

Computes  $a \ge b$  and subtracts from accumulator; also prefetches data ready for a future MAC operation.

*xptr* may be null to signify no X prefetch to be performed, in which case the values of *xincr* and *xval* are ignored, but required.

yptr may be null to signify no Y prefetch to be performed, in which case the values of yincr and yval are ignored, but required.

xval and yval nominate the address of a C variable where the prefetched value will be stored.

xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If AWB is non null, the other accumulator will be written back into the referenced variable.

For example:

### Argument:

Accum	IAccumulator to sum.
а	Integer multiplicand.
b	Integer multiplier.
xptr	Integer pointer to pointer to x prefetch.
xval	Integer pointer to value of x prefetch.
xincr	Integer increment value of x prefetch.
yptr	Integer pointer to pointer to y prefetch.
yval	Integer pointer to value of y prefetch.
yincr	Integer increment value of y prefetch.
AWB	Accumulator write back location.
AWB_accum	Accumulator to write back.

**Note:** The arguments *xptr* and *yptr* must point to the arrays located in the x data memory and y data memory, respectively.

## **Return Value:**

Returns the cleared value result to an accumulator.

Assembler Operator / Machine Instruction:

msc

# \_builtin\_msc (Continued)

# Error Messages:

- the result is not an accumulator register
- Accum is not an accumulator register
- xval is a null value but xptr is not null
- yval is a null value but yptr is not null
- AWB\_accum is not an accumulator register and AWB is not null

# \_builtin\_mulss

## **Description:**

Computes the product  $p0 \ x \ p1$ . Function arguments are signed integers, and the function result is a signed long integer. The command-line option -wconversions can be used to detect unexpected sign conversions.

# Prototype:

signed long \_\_builtin\_mulss(const signed int p0, const signed int p1);

## Argument:

- p0 multiplicand
- p1 multiplier

# **Return Value:**

Returns the signed long integer value of the product p0 x p1.

Assembler Operator / Machine Instruction:

mul.ss

# \_builtin\_mulsu

## **Description:**

Computes the product  $p0 \ x \ p1$ . Function arguments are integers with mixed signs, and the function result is a signed long integer. The command-line option -Wconversions can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand p1.

## Prototype:

signed long \_\_builtin\_mulsu(const signed int p0, const unsigned int p1);

# Argument:

p0 multiplicand p1 multiplier

# **Return Value:**

Returns the signed long integer value of the product p0 x p1.

Assembler Operator / Machine Instruction:

mul.su

### \_\_builtin\_mulus

#### **Description:**

Computes the product  $p0 \ x \ p1$ . Function arguments are integers with mixed signs, and the function result is a signed long integer. The command-line option <u>-Wconversions</u> can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction.

#### Prototype:

signed long \_\_builtin\_mulus(const unsigned int p0, const signed int p1);

#### Argument:

- p0 multiplicand
- p1 multiplier

#### Return Value:

Returns the signed long integer value of the product p0 x p1.

#### Assembler Operator / Machine Instruction:

mul.us

#### \_builtin\_muluu

#### **Description:**

Computes the product  $p0 \times p1$ . Function arguments are unsigned integers, and the function result is an unsigned long integer. The command-line option <u>-Wconversions</u> can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including immediate mode for operand p1.

#### Prototype:

unsigned long \_\_builtin\_muluu(const unsigned int p0, const unsigned int p1);

#### Argument:

- p0 multiplicand
- p1 multiplier

#### **Return Value:**

Returns the signed long integer value of the product p0 x p1.

Assembler Operator / Machine Instruction:

mul.uu

#### \_builtin\_nop

Description:

Generates a nop instruction. Prototype: void \_\_builtin\_nop(void); Argument: None. Return Value: Returns a no operation (nop). Assembler Operator / Machine Instruction: nop

## \_builtin\_psvoffset

#### Description:

Returns the psv page offset of the object whose address is given as a parameter. The argument p must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

Prototype:

```
unsigned int __builtin_psvoffset(const void *p);
```

Argument:

p object address

Return Value:

Returns the psv page number offset of the object whose address is given as a parameter.

Assembler Operator / Machine Instruction:

psvoffset

#### Error Messages:

The following error message is produced when this function is used incorrectly:

"Argument to \_\_builtin\_psvoffset() is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if obj is object in an executable or read-only section, the following syntax is valid:

unsigned page = \_\_builtin\_psvoffset(&obj);

## \_builtin\_psvpage

## **Description:**

Returns the psv page number of the object whose address is given as a parameter. The argument p must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

## Prototype:

unsigned int \_\_builtin\_psvpage(const void \*p);

## Argument:

p object address

## Return Value:

Returns the psv page number of the object whose address is given as a parameter.

## Assembler Operator / Machine Instruction:

## psvpage

## Error Messages:

The following error message is produced when this function is used incorrectly:

"Argument to  $\_builtin\_psvpage()$  is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if obj is object in an executable or read-only section, the following syntax is valid:

unsigned page = \_\_builtin\_psvpage(&obj);

## \_builtin\_readsfr

## **Description:**

Reads the SFR. Prototype:

unsigned int \_\_builtin\_readsfr(const void \*p);

## Argument:

p object address

## Return Value:

Returns the SFR.

Assembler Operator / Machine Instruction:

readsfr

## Error Messages:

The following error message is produced when this function is used incorrectly:

#### \_builtin\_return\_address

#### **Description:**

Returns the return address of the current function, or of one of its callers. For the *level* argument, a value of 0 yields the return address of the current function, a value of 1 yields the return address of the caller of the current function, and so forth. When level exceeds the current stack depth, 0 will be returned. This function should only be used with a non-zero argument for debugging purposes.

#### Prototype:

int \_\_builtin\_return\_address (const int level);

### Argument:

*level* Number of frames to scan up the call stack.

#### Return Value:

Returns the return address of the current function, or of one of its callers.

#### Assembler Operator / Machine Instruction:

return\_address

## \_builtin\_sac

#### Description:

Shifts value by *shift* (a literal between -8 and 7) and returns the value.

#### For example:

```
register int value asm("A");
int result;
```

result = \_\_builtin\_sac(value,3);

May generate:

sac A, #3, w0

#### Prototype:

int \_\_builtin\_sac(int value, int shift);

#### Argument:

value Integer number to be shifted. shift Literal amount to shift.

#### Return Value:

Returns the shifted result to an accumulator.

#### Assembler Operator / Machine Instruction:

sac

#### **Error Messages:**

An error message appears if:

- the result is not an accumulator register
- the shift value is not a literal within range

## \_builtin\_sacr

#### **Description:**

Shifts value by *shift* (a literal between -8 and 7) and returns the value which is rounded using the rounding mode determined by the CORCONbits.RND control bit.

#### For example:

register int value asm("A");
int result;

result = \_\_builtin\_sac(value,3);

May generate:

sac.r A, #3, w0

#### Prototype:

int \_\_builtin\_sacr(int value, int shift);

Argument:

valueInteger number to be shifted.shiftLiteral amount to shift.

#### **Return Value:**

Returns the shifted result to the CORCON register.

**Assembler Operator / Machine Instruction:** 

sacr

#### Error Messages:

An error message appears if:

- the result is not an accumulator register
- the shift value is not a literal within range

## \_builtin\_sftac

#### **Description:**

Shifts accumulator by *shift*. The valid shift range is -16 to 16.

For example:

register int result asm("A");

int i;

result = \_\_builtin\_sftac(result,i);

May generate:

sftac A, w0

Prototype:

int \_\_builtin\_sftac(int Accum, int shift);

#### Argument:

Accum Accumulator to shift. shift Amount to shift.

**Return Value:** 

Returns the shifted result to an accumulator.

Assembler Operator / Machine Instruction:

sftac

#### Error Messages:

An error message appears if:

- the result is not an accumulator register
- Accum is not an accumulator register
- the shift value is not a literal within range

## \_builtin\_subab

#### **Description:**

Subtracts accumulators A and B with the result written back to the specified accumulator. For example:

```
register int result asm("A");
register int B asm("B");
result = __builtin_subab(result,B);
```

will generate:

sub A

**Prototype:** 

int \_\_\_\_builtin\_subab(int Accum\_a, int Accum\_b);

#### Argument:

Accum\_aAccumulator from which to subtract.Accum\_bAccumulator to subtract.

**Return Value:** 

Returns the subtraction result to an accumulator.

Assembler Operator / Machine Instruction:

sub

#### **Error Messages:**

An error message appears if the result is not an accumulator register.

#### \_builtin\_tbladdress

#### **Description:**

Returns a value that represents the address of an object in program memory. The argument p must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

#### Prototype:

unsigned long \_\_builtin\_tblpage(const void \*p);

#### Argument:

p object address

#### **Return Value:**

Returns an unsigned long value that represents the address of an object in program memory.

#### Assembler Operator / Machine Instruction:

tbladdress

### \_builtin\_tbladdress

#### Error Messages:

The following error message is produced when this function is used incorrectly:

"Argument to \_\_builtin\_tbladdress() is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if *obj* is object in an executable or read-only section, the following syntax is valid:

unsigned long page = \_\_builtin\_tbladdress(&obj);

## \_builtin\_tbloffset

#### **Description:**

Returns the table page offset of the object whose address is given as a parameter. The argument p must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

#### Prototype:

unsigned int \_\_builtin\_tbloffset(const void \*p);

Argument:

p object address

**Return Value:** 

Returns the table page number offset of the object whose address is given as a parameter.

Assembler Operator / Machine Instruction:

tbloffset

#### Error Messages:

The following error message is produced when this function is used incorrectly:

"Argument to \_\_builtin\_tbloffset() is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if obj is object in an executable or read-only section, the following syntax is valid: unsigned page = \_\_builtin\_tbloffset(&obj);

## \_builtin\_tblpage

#### **Description:**

Returns the table page number of the object whose address is given as a parameter. The argument p must be the address of an object in an EE data, PSV or executable memory space; otherwise an error message is produced and the compilation fails. See the space attribute in **Section 2.3.1 "Specifying Attributes of Variables"** of the "MPLAB<sup>®</sup> C Compiler for PIC24 MCUs and dsPIC<sup>®</sup> DSCs User's Guide" (DS51284).

#### Prototype:

unsigned int \_\_builtin\_tblpage(const void \*p);

#### Argument:

p object address

#### Return Value:

Returns the table page number of the object whose address is given as a parameter.

#### Assembler Operator / Machine Instruction:

tblpage

#### **Error Messages:**

The following error message is produced when this function is used incorrectly:

"Argument to \_\_builtin\_tblpage() is not the address of an object in code, psv, or eedata section".

The argument must be an explicit object address.

For example, if obj is object in an executable or read-only section, the following syntax is valid:

unsigned page = \_\_builtin\_tblpage(&obj);

#### \_builtin\_tblrdh

#### **Description:**

Issues the tblrdh.w instruction to read a word from Flash or EEDATA memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of \_\_builtin\_tbloffset() and \_\_builtin\_tblpage().

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

#### Prototype:

unsigned int \_\_builtin\_tblrdh(unsigned int offset);

#### Argument:

offset desired memory offset

#### **Return Value:**

None.

Assembler Operator / Machine Instruction:

tblrdh

#### Error Messages:

None.

#### \_builtin\_tblrdl

#### **Description:**

Issues the tblrdl.w instruction to read a word from Flash or EEDATA memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of \_\_builtin\_tbloffset() and\_\_builtin\_tblpage().

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

#### Prototype:

unsigned int \_\_builtin\_tblrdl(unsigned int offset);

Argument:

offset desired memory offset

**Return Value:** 

None.

Assembler Operator / Machine Instruction:

tblrdl

Error Messages:

None.

## \_builtin\_tblwth

#### **Description:**

Issues the tblwth.w instruction to write a word to Flash or EEDATA memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of \_\_builtin\_tbloffset() and \_\_builtin\_tblpage().

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

#### Prototype:

void \_\_builtin\_tblwth(unsigned int offset unsigned int data);

#### Argument:

offset desired memory offset data data to be written

**Return Value:** 

None.

Assembler Operator / Machine Instruction:

tblwth

Error Messages:

None.

## \_builtin\_tblwtl

#### **Description:**

Issues the tblrdl.w instruction to write a word to Flash or EEDATA memory. You must set up
the TBLPAG to point to the appropriate page. To do this, you may make use of
\_\_builtin\_tbloffset() and \_\_builtin\_tblpage().

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

#### Prototype:

void \_\_builtin\_tblwtl(unsigned int offset unsigned int data);

#### Argument:

offsetdesired memory offsetdatadata to be written

**Return Value:** 

None.

Assembler Operator / Machine Instruction:

tblwtl

Error Messages:

None.

```
Additional Inline Functions
Example 6-1:
#include "p33fxxxx.h"
volatile long Result_mpy1616;
volatile long Result_addab;
volatile long Result_subab;
volatile long Result_mpy3216;
volatile long Result_div3216;
register int Accu_A asm("A");
register int Accu_B asm("B");
inline static long mpy_32_16 (long, int);
inline static long mpy_32_16 (long x, int y)
 ł
     long result;
     int temp1, temp2;
     temp1 = (x>>1)\&0x7FFF;
     temp2 = x >> 16;
     Accu_A = __builtin_mpy (temp1, y, 0,0,0,0,0,0);
Accu_A = __builtin_sftac (15);
Accu_A = __builtin_mac (temp2, y, 0,0,0,0,0,0,0);
     asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(result) : "w"(Accu_A));
     return result;
int main (void)
     // Variable declarations
     int Input1;
     int Input2;
     int Input3;
     int Input4;
     long Input5;
     int Input6;
     long Input7;
     int Input8;
     // Enable 32-bit saturation, signed and fractional modes for both ACCA
          and ACCB
     CORCON = 0 \times 00C0;
      // Example of 16*16-bit fractional multiplication using ACCA
     Input1 = 32767;
     Input2 = 32767;
     Accu_A = __builtin_mpy (Input1, Input2, 0,0,0,0,0,0);
asm("mov _ACCAL,%0\n\t"
     asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(Result_mpy1616) : "w"(Accu_A));
     // Example of 16*16-bit fractional multiplication using ACCB
     Input3 = 16384;
     Input4 = 16384;
     Accu_B = __builtin_mpy (Input3, Input4, 0,0,0,0,0,0);
asm("mov _ACCBL,%0\n\t"
"mov _ACCBH,%d0" : "=r"(Result_mpy1616) : "w"(Accu_B));
     // Example of 32-bit addition using ACCA (ACCA = ACCA + ACCB)
     Accu_A = __builtin_addab();
asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(Result_addab) : "w"(Accu_A));
     // Example of 32-bit subtraction using ACCB (ACCB = ACCB - ACCA)
     Accu_B = __builtin_subab();
asm("mov _ACCBL,%0\n\t"
"mov _ACCBH,%d0" : "=r"(Result_subab) : "w"(Accu_B));
     // Example of 32*16-bit fractional multiplication using ACCA
     Input5 = 0x7FFFFFF;
Input6 = 32767;
     Result_mpy3216 = mpy_32_16 (Input5, Input6);
     while(1);
```

```
Example 6-2:
               Divide_32_by_16
#include <p33Fxxxx.h>
#include "divide.h"
_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_NONE);
_FWDT(FWDTEN_OFF);
unsigned int divide_(long a, int b) {
  union convert {
    unsigned long 1;
    unsigned int i[2];
  } c;
  int sign;
  unsigned int result;
  c.l = a;
  sign = c.i[1] ^ b;
  if (a < 0) a = (-a);
  if (b < 0) b = -b;
  result = __builtin_divud(a,b);
  result >>= 1;
  if (sign < 0) result = -result;
  return result;
}
int main(void)
{
    unsigned long dividend;
    unsigned int divisor;
    unsigned int quotient;
    dividend = 0x3FFFFFF;
    divisor = 0x7FFF;
    quotient = divide_((long)dividend, (int)divisor);
    while(1);
```

NOTES:



# Section 7. Reference

## HIGHLIGHTS

This section of the manual contains the following major topics:

7.1	Instruction Bit Map	484
7.2	Instruction Set Summary Table	486
7.3	Revision History	496

## 7.1 INSTRUCTION BIT MAP

Instruction encoding for the 16-bit MCU and DSC family devices is summarized in Table 7-1. This table contains the encoding for the MSB of each instruction. The first column in the table represents bits 23:20 of the opcode, and the first row of the table represents bits 19:16 of the opcode. The first byte of the opcode is formed by taking the first column bit value and appending the first row bit value. For instance, the MSB of the PUSH instruction (last row, ninth column) is encoded with 11111000b (0xF8).

**Note:** The complete opcode for each instruction may be determined by the instruction descriptions in **Section 5. "Instruction Descriptions"**, using Table 5-1 through Table 5-12.

#### Table 7-1: Instruction Encoding

ιαυι	e /-I.	11131		ncoun	iy												
									Оро	code<19:16>							
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0000	NOP	BRA CALL GOTO RCALL	CALL	-	GOTO	RETLW	RETFIE RETURN	RCALL	DO <sup>(1)</sup>	REPEAT	-	_	BRA <sup>(1)</sup> (OA)	BRA <sup>(1)</sup> (OB)	BRA <sup>(1)</sup> (SA)	BRA <sup>(1)</sup> (SB)
	0001					SUBR							SUBBR				
	0010									MOV							
	0011	BRA (OV)	BRA (C)	BRA (Z)	BRA (N)	BRA (LE)	BRA (LT)	BRA (LEU)	BRA	BRA (NOV)	BRA (NC)	BRA (NZ)	BRA (NN)	BRA (GT)	BRA (GE)	BRA (GTU)	—
	0100					ADD							ADDC				
	0101					SUB							SUBB				
	0110					AND							XOR				
	0111					IOR							MOV				
	1000									MOV							
A	1001									MOV							
3:2	1010	BSET	BCLR	BTG	BTST	BTSTS	BTST	BTSS	BTSC	BSET	BCLR	BTG	BTST	BTSTS	BSW	BTSS	BTSC
Opcode<23:20>	1011	ADD ADDC	SUB SUBB	AND XOR	IOR MOV	ADD ADDC	SUB SUBB	AND XOR	IOR MOV	MUL.US MUL.UU	MUL.SS MUL.SU	TBLRDH TBLRDL	TBLWTH TBLWTL	MUL	SUB SUBB	MOV.D	MOV
Opo	1100		MAC <sup>(1)</sup> MPY <sup>(1)</sup> MPY.N <sup>(1)</sup> MSC <sup>(1)</sup>		CLRAC <sup>(1)</sup>		MAC <sup>(1)</sup> MPY <sup>(1)</sup> MPY.N <sup>(1)</sup> MSC <sup>(1)</sup>		MOVSAC <sup>(1)</sup>	SFTAC <sup>(1)</sup>	ADD <sup>(1)</sup>	LAC <sup>(1)</sup>	ADD <sup>(1)</sup> NEG <sup>(1)</sup> SUB <sup>(1)</sup>	SAC <sup>(1)</sup>	SAC.R <sup>(1)</sup>		FF1L FF1R
	1101	SL	ASR LSR	RLC RLNC	RRC RRNC	SL	ASR LSR	RLC RLNC	RRC RRNC	DIV.S DIV.U	DIVF <sup>(1)</sup>	_		_	SL	ASR LSR	FBCL
	1110	CP0	CP CPB	CP0	CP CPB		_	CPBGT <sup>(2)</sup> CPBLT <sup>(2)</sup> CPSGT CPSLT	CPBEQ <sup>(2)</sup> CPBNE <sup>(2)</sup> CPSEQ CPSNE	INC INC2	DEC DEC2	COM NEG	CLR SETM	INC INC2	DEC DEC2	COM NEG	CLR SETM
	1111		EDA MA MP	)(1) {C <sup>(1)</sup> C <sup>(1)</sup>		_	_	_	_	PUSH	POP	LNK ULNK	SE ZE	DISI	DAW EXCH SWAP	CLRWDT MOVPAG <sup>(2)</sup> PWRSAV POP.S PUSH.S RESET	NOPR
Note					in dsPIC30F in PIC24E a			33E family de <sup>,</sup> vices.	vices.			L		1	1	-	

Reference

## 7.2 INSTRUCTION SET SUMMARY TABLE

The complete 16-bit MCU and DSC device instruction set is summarized in Table 7-2. This table contains an alphabetized listing of the instruction set. It includes instruction assembly syntax, description, size (in 24-bit words), execution time (in instruction cycles), affected Status bits, and the page number in which the detailed description can be found. Table 1-2 identifies the symbols that are used in the Instruction Set Summary Table.

Note: The instruction cycle counts listed here are for PIC24F, PIC24H, dsPIC30F and dsPIC33F devices. Some instructions require additional cycles in PIC24E and dsPIC33E devices. Refer to Section 3.3 "Instruction Set Summary Tables" and Section 5.4 "Instruction Descriptions" for details.

#### Table 7-2: Instruction Set Summary Table

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
ADD	f {,WREG}	Destination = f + WREG	1	1	-	—	-	—		_	ŷ	ŷ	ţ	$\hat{v}$	Û	99
ADD	#lit10,Wn	Wn = lit10 + Wn	1	1	-	_	_	_	_	—	ţ	Û	ţ	$\hat{v}$	ŷ	100
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	1	1		—	—	—		—	ţ	Û	€	ţ	Û	101
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1	-	_	_	_	_	—	ţ	Û	ţ	$\hat{v}$	ŷ	102
ADD	Acc <sup>(2)</sup>	Add accumulators	1	1	ţ	Û	仓	Û	Û	Ŷ		—	—	—		103
ADD	Wso,#Slit4,Acc	16-bit signed add to accumulator	1	1	ţ	Û	仓	Û	Û	Û	_	_	—	_	_	104
ADDC	f {,WREG}	Destination = f + WREG + (C)	1	1		—	—	—		—	ţ	Û	€	Û	Û	106
ADDC	#lit10,Wn	Wn = Iit10 + Wn + (C)	1	1	-	_	_	_	_	—	ţ	Û	ţ	Û	ŷ	107
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	1	1		_	—	_	_		Û	Û	€	Û	Û	108
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1		_	—	_	-	—	Û	Û	ţ	Û	Û	110
AND	f {,WREG}	Destination = f .AND. WREG	1	1		_	—	_		—		Û	—	$\hat{v}$		112
AND	#lit10,Wn	Wn = lit10 .AND. Wn	1	1	-	_	_	_	_	—	_	Û	—	$\hat{v}$	_	113
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	1	1		_	—	_	-	—	_	Û	—	ţ		114
AND	Wb,Ws,Wd	Wd = Wb .AND. Ws	1	1	-	_	_	_	_	—	_	Û	—	$\hat{v}$	_	115
ASR	f {,WREG}	Destination = arithmetic right shift f, LSb $\rightarrow C$	1	1		_	—	—		—		Û	—	$\hat{v}$	Û	117
ASR	Ws,Wd	Wd = arithmetic right shift Ws, LSb $\rightarrow C$	1	1		—	—	—	_	—	-	Û	-	$\hat{v}$	Û	119
ASR	Wb,#lit4,Wnd	Wnd = arithmetic right shift Wb by lit4, LSb $\rightarrow$ C	1	1		_	_	—		—		Û	_	$\hat{\mathbf{v}}$		121

Legend: 🗘 set or cleared; 🖞 may be cleared, but never set; 🏦 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
ASR	Wb,Wns,Wnd	Wnd = arithmetic right shift Wb by Wns, LSb $\rightarrow$ C	1	1	-	-	_	—	—	-	-	Û		€	_	122
BCLR	f,#bit4	Bit clear f	1	1	_	—	—	—	—	—		_	_	_	—	123
BCLR	Ws,#bit4	Bit clear Ws	1	1	—	-	_	-	-	—	_	-	—	—	—	124
BRA	Expr	Branch unconditionally	1	2	—	—	—	—	-	—	_	—	—	—	—	126
BRA	Wn	Computed branch	1	2	—	—	—	—	-	—	_	—	—	—	—	128
BRA	C,Expr	Branch if Carry	1	1 (2)	—	—	—	—	-	—	_	—	—	—	—	130
BRA	GE,Expr	Branch if greater than or equal	1	1 (2)	—	—	—	—	-	—	_	—	—	—	—	132
BRA	GEU, Expr	Branch if Carry	1	1 (2)	—	—	—	—	-	—	_	—	—	—	—	134
BRA	GT,Expr	Branch if greater than	1	1 (2)	—	—	—	—	-	—	_	—	—	—	—	135
BRA	GTU,Expr	Branch if unsigned greater than	1	1 (2)	-	-		-	-	—	—	-			-	136
BRA	LE,Expr	Branch if less than or equal	1	1 (2)	-	-		-	-	—	_	-		Ι		137
BRA	LEU,Expr	Branch if unsigned less than or equal	1	1 (2)	-	-		-	-	—	_	-		Ι		138
BRA	LT,Expr	Branch if less than	1	1 (2)	-	-		-	-	—	_	-		Ι		139
BRA	LTU,Expr	Branch if not Carry	1	1 (2)	-	-		-	-	—	_	-		Ι		140
BRA	N,Expr	Branch if Negative	1	1 (2)		_		_	_	—		_		Ι		141
BRA	NC,Expr	Branch if not Carry	1	1 (2)	-	-		-	-	—	—	-			-	142
BRA	NN,Expr	Branch if not Negative	1	1 (2)	-	-		-	-	—	—	-			-	143
BRA	NOV,Expr	Branch if not Overflow	1	1 (2)	-	-		-	-	—	—	-			-	144
BRA	NZ,Expr	Branch if not Zero	1	1 (2)	-	-		-	-	—	_	-		Ι		145
BRA	0A, Expr <sup>(2)</sup>	Branch if Accumulator A overflow	1	1 (2)	-	-		-	-	—	_	-		Ι		146
BRA	0B,Expr <sup>(2)</sup>	Branch if Accumulator B overflow	1	1 (2)	-	-		-	-	—	_	-		Ι		147
BRA	OV,Expr	Branch if Overflow	1	1 (2)	-	-		-	-	—	_	-		Ι		148
BRA	SA, Expr <sup>(2)</sup>	Branch if Accumulator A saturated	1	1 (2)	-	-		-	-	—	_	-		Ι		149
BRA	SB, Expr <sup>(2)</sup>	Branch if Accumulator B saturated	1	1 (2)	—	—	—	—	-	—	_	—	—	—	—	150
BRA	Z,Expr	Branch if Zero	1	1 (2)		—		—	—	—		_		_	-	151
BSET	f,#bit4	Bit set f	1	1	—	—	_	—	—	—	—	—	—	—	—	152
BSET	Ws,#bit4	Bit set Ws	1	1	_	—	_	_	—	—	_	_	—	_	_	153
BSW.C	Ws,Wb	Write C bit to Ws <wb></wb>	1	1	—	_	_	_	_	_	—	_	—	—	—	155

#### Instruction Set Summary Table (Continued) Table 7.2.

🕄 set or cleared; 🖟 may be cleared, but never set; 🕆 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged Legend:

SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged. Note 1:

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices. 5:

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	OA <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
BSW.Z	Ws,Wb	Write Z bit to Ws <wb></wb>	1	1	-	—	-	-		—		-	—	-	-	155
BTG	f,#bit4	Bit toggle f	1	1	—	—	—	—		—		—	_	—	—	157
BTG	Ws,#bit4	Bit toggle Ws	1	1	_	—	_	—		—		_	_	—	_	158
BTSC	f,#bit4	Bit test f, skip if clear	1	1 (2 or 3)		_	-			_	Ι	_	_	-	-	160
BTSC	Ws,#bit4	Bit test Ws, skip if clear	1	1 (2 or 3)	—	_	—	_	_	—	_	_	_	_	-	162
BTSS	f,#bit4	Bit test f, skip if set	1	1 (2 or 3)	_	_	_	_	_	—	_	_	_	_	-	164
BTSS	Ws,#bit4	Bit test Ws, skip if set	1	1 (2 or 3)	—	_	—	_	_	—	_	_	_	_	-	166
BTST	f,#bit4	Bit test f to Z	1	1	—	_	—	—	—	—	—	—	_	ŷ	-	168
BTST.C	Ws,#bit4	Bit test Ws to C	1	1	_	_	—	_				—	_	—	€	169
BTST.Z	Ws,#bit4	Bit test Ws to Z	1	1	_	—	—	—	_	—	_	—	—	Û	-	169
BTST.C	Ws,Wb	Bit test Ws <wb> to C</wb>	1	1	_	—	_	_		—	-	—	—	—	€	171
BTST.Z	Ws,Wb	Bit test Ws <wb> to Z</wb>	1	1	_	_	_	—	_	—	_	_	_	Û	-	171
BTSTS	f,#bit4	Bit test f to Z, then set f	1	1	_	—	_	—	_	—	_	_	_	Û	—	173
BTSTS.C	Ws,#bit4	Bit test Ws to C then set	1	1	_	_	_	—	_	—	_	_	_	_	€	175
BTSTS.Z	ZWs,#bit4	Bit test Ws to Z then set	1	1	_	—	_	—	_	—	_	_	_	Û	—	175
CALL	Expr	Call subroutine	2	2	_	_	_	—	_	—	-	-	—	-	-	177
CALL	Wn	Call indirect subroutine	1	2	—	_	—	—		_		—	_	—	-	180
CALL.L	Wn(3)	Call indirect subroutine (long address)	1	4	—	—	—	—		—		—	_	—	—	183
CLR	f	f = 0x0000	1	1		-		—	_	—	_	-		-	-	184
CLR	WREG	WREG = 0x0000	1	1	—	_	—	—		—		_	_	—	—	184
CLR	Wd	Wd = 0	1	1	-	-	—	—		—		-		-	-	185
CLR	Acc,[Wx],Wxd,[Wy],Wyd,AWB <sup>(2)</sup>	Clear accumulator	1	1	0	0	0	0	0	0		—	_	_	—	186
CLRWDT		Clear Watchdog Timer	1	1	_	_	_	_		—		—	_	_	—	188
COM	f {,WREG}	Destination = $\overline{f}$	1	1			—	_	—	—		€		ŷ	_	189
COM	Ws,Wd	$Wd = \overline{Ws}$	1	1	_	_	_	_	_	_		Û	—	Û	—	190

#### Table 7-2: Instruction Set Summary Table (Continued)

Legend: 1 set or cleared; 1 may be cleared, but never set; 1 may be set, but never cleared; 1 always set; 0 always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
CP	f	Compare (f – WREG)	1	1	_	—	_	—	-	_	Û	Û	ţ	ţ	€	191
CP	Wb,#lit5	Compare (Wb – lit5)	1	1	_	-	_	—	_	—	Û	Û	ţ	ţ	ţ	192
CP	Wb,#lit8	Compare (Wb – lit8)	1	1	—	-	—	—	_	—	Û	Û	ţ	ţ	ŷ	193
CP	Wb,Ws	Compare (Wb – Ws)	1	1	-	—	-			—	€	€	ţ	ţ	ţ	194
CP0	f	Compare (f – 0x0000)	1	1	—	—	—	_		_	1	Û	ţ	ţ	1	196
CP0	Ws	Compare (Ws – 0x0000)	1	1	—	—	_			_	1	$\hat{\mathbf{v}}$	ţ	ţ	1	197
CPB	f	Compare with borrow (f – WREG – $\overline{C}$ )	1	1	—	—	_			_	€	$\hat{\mathbf{v}}$	ţ	Û	Û	198
CPB	Wb,#lit5	Compare with borrow (Wb – lit5 – $\overline{C}$ )	1	1	—	—	—	_		_	€	Û	ţ	Û	€	199
CPB	Wb,#lit8	Compare with borrow (Wb – lit8 – $\overline{C}$ )	1	1	—	—	_			_	€	$\hat{\mathbf{v}}$	ţ	Û	Û	200
CPB	Wb,Ws	Compare with borrow (Wb – Ws – $\overline{C}$ )	1	1	—	—	—	_		_	€	Û	ţ	Û	€	201
CPBEQ	Wb,Wn,Expr <sup>(3)</sup>	Compare Wb with Wn, branch if =	1	1 (5)	—	-	—	—	—	—	—	—	—	—	-	203
CPBGT	Wb,Wn,Expr <sup>(3)</sup>	Signed Compare Wb with Wn, branch if >	1	1 (5)	—	-	—	_	_	_		—	—	—	-	204
CPBLT	Wb,Wn,Expr <sup>(3)</sup>	Signed Compare Wb with Wn, branch if <	1	1 (5)	_	_	_	_	_	_	_	_	_	_	_	205
CPBNE	Wb,Wn,Expr <sup>(3)</sup>	Compare Wb with Wn, branch if ≠	1	1 (5)	_	_	_	_	_	—	_	_	_	_	_	206
CPSEQ	Wb,Wn	Compare (Wb with Wn), skip if =	1	1 (2 or 3)		-	-	—		—	-		-	_	-	207
CPSGT	Wb,Wn	Signed Compare (Wb with Wn), skip if >	1	1 (2 or 3)	-	-	-	_	-	—	_	-	-	_	-	211
CPSLT	Wb,Wn	Signed Compare (Wb with Wn), skip if <	1	1 (2 or 3)		-	-	—		—	-		-	_	-	212
CPSNE	Wb,Wn	Compare (Wb with Wn), skip if $\neq$	1	1 (2 or 3)		-	-	_		_			-	_	_	214
DAW.B	Wn	Wn = decimal adjust Wn	1	1	_	-	_	_	_	—	_	_	—	_	ŷ	216
DEC	f {,WREG}	Destination = f - 1	1	1	—	_	—	—		—	€	Û	ţ	ţ	ţ	217
DEC	Ws,Wd	Wd = Ws - 1	1	1	—	-	—	—	_	—	Û	Û	ţ	ţ	ŷ	218
DEC2	f {,WREG}	Destination = f - 2	1	1	—	—	—	—		—	Û	Û	ţ	ţ	ţ	220
DEC2	Ws,Wd	Wd = Ws - 2	1	1	—	_	_	—	_	_	€	Û	€	ţ	ţ	221

\$\$ set or cleared; \$\$ may be cleared, but never set; \$\$ may be set, but never cleared; \$\$ 1' always set; \$\$ o' always cleared; \$\$ --- unchanged

SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

This instruction/operand is only available in PIC24E and dsPIC33E devices.

This instruction/operand is only available in dsPIC30F and dsPIC33F devices.

This instruction/operand is only available in dsPIC33E devices.

#### Table 7-2: Instruction Set Summary Table (Continued)

Legend: Note 1:

2:

3:

4:

5:

6:

() () () 220 () () () 221 Section 7. Reference

Reference

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
DISI	#lit14	Disable interrupts for lit14 instruction cycles	1	1		—	—	—	-	—	I	—	—		-	223
DIV.S	Wm,Wn	Signed 16/16-bit integer divide, $Q \rightarrow Wo, R \rightarrow W1$	1	18	_	—	—	_	-	—	—	Û	Û	Û	ţ	224
DIV.SD	Wm,Wn	Signed 32/16-bit integer divide, $Q \rightarrow Wo, R \rightarrow W1$	1	18	_	_	_	_	-	—	_	Û	Û	Û	€	224
DIV.U	Wm,Wn	Unsigned 16/16-bit integer divide, Q $\rightarrow$ Wo, R $\rightarrow$ W1	1	18		_	_		—	—		0	0	Û	€	226
DIV.UD	Wm,Wn	Unsigned 32/16-bit integer divide, $Q \rightarrow Wo, R \rightarrow W1$	1	18		_			_	—		0	Û	Û	ţ	226
DIVF	<sub>Wm , Wn</sub> (2)	Signed 16/16-bit fractional divide, Q $\rightarrow$ Wo, R $\rightarrow$ W1	1	18		—	—	—	-	—	_	€	Û	Û	ţ	228
DO	#lit14,Expr <sup>(6)</sup>	Do code to PC + Expr, (lit14 + 1) times	2	2		—	—	—	-	—	_	—	—	_	_	230
DO	#lit15,Expr <sup>(4)</sup>	Do code to PC + Expr, (lit15 + 1) times	2	2		_	_		—	—		—	_		—	233
DO	Wn, Expr <sup>(2)</sup>	Do code to PC + Expr, (Wn + 1) times	2	2					_	—				-		235
ED	Wm*Wm,Acc,[Wx],[Wy],Wxd <sup>(2)</sup>	Euclidean distance (no accumulate)	1	1	€	ţ	仓	企	€	仓	Ι	-	-	-	-	239
EDAC	Wm*Wm,Acc,[Wx],[Wy],Wxd <sup>(2)</sup>	Euclidean distance	1	1	ţ	ţ	Û	仓	Û	Ŷ	—	—	—	—	—	241
EXCH	Wns,Wnd	Swap Wns and Wnd	1	1		_	_	_	_	_		_	—	_	_	243
FBCL	Ws,Wnd	Find bit change from left (MSb) side	1	1			-	-	-	-	Ι		-	-	ţ	244
FF1L	Ws,Wnd	Find first one from left (MSb) side	1	1	—	-			-	—	_			—	ţ	246
FF1R	Ws,Wnd	Find first one from right (LSb) side	1	1	_	_	_	_	-	—	_	_	_	_	€	248
GOTO	Expr	Go to address	2	2		—	—	—	-	—	_	—	—	_	—	250
GOTO	Wn	Go to address indirectly	1	2	Ι				_	—						251
GOTO.L	<sub>Wn</sub> (3)	Go to address indirectly (long address)	1	4	1	_	_	_	—	—		—	_		-	253
INC	f {,WREG}	Destination = f + 1	1	1	—	_	—	—	-	—	Û	$\hat{\mathbf{v}}$	Û	Û	Û	254
INC	Ws,Wd	Wd = Ws + 1	1	1		_	_	_	-	—	€	$\hat{\mathbf{v}}$	Û	€	ţ	255
INC2	f {,WREG}	Destination = f + 2	1	1	—	-			-	—	ţ	Û	Û	Û	ţ	257
INC2	Ws,Wd	Wd = Ws + 2	1	1	_	—	—	_	-	—	ţ	Û	Û	Û	ţ	258
IOR	f {,WREG}	Destination = f .IOR. WREG	1	1	-	_	_	_	—	—		Û	_	Û	—	260
IOR	#lit10,Wn	Wn = lit10 .IOR. Wn	1	1	_	_	_	_	-	_	_	Û	_	Û	_	261
IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	1	1	_	_	_	_	-	_	_	Û	_	Û	_	262
IOR	Wb,Ws,Wd	Wd = Wb .IOR. Ws	1	1	_	_	_	_	—	_	_	Û	_	Û	_	263
LAC	Wso,#Slit4, Acc <sup>(2)</sup>	Load accumulator	1	1	Û	Û	仓	仓	Û	仓	_	—	—	_	_	265

#### Table 7-2: Instruction Set Summary Table (Continued)

Legend: 🗘 set or cleared; 🖑 may be cleared, but never set; 🏦 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
LNK	#lit14	Link Frame Pointer	1	1		—	—	_		-		—	—	—	—	267
LSR	f {,WREG}	Destination = logical right shift f, MSb $\rightarrow$ C	1	1	—	—	_	_	_	—	—	0	—	Û	ŷ	269
LSR	Ws,Wd	Wd = logical right shift Ws, MSb $\rightarrow$ C	1	1		—	_		_	_		0	_	€	Û	271
LSR	Wb,#lit4,Wnd	Wnd = logical right shift Wb by lit4, MSb $\rightarrow$ C	1	1	_	—	—	_	_	_	_	Û	—	Û	—	273
LSR	Wb,Wns,Wnd	Wnd = logical right shift Wb by Wns, MSb $\rightarrow$ C	1	1	_	_	_	_		_	_	Û	_	Û	_	274
MAC	Wm*Wn,Acc,[Wx],Wxd,[Wy], Wyd,AWB <sup>(2)</sup>	Multiply and accumulate	1	1	ţ	Û	仓	仓	Û	仓	_	_	_	_	-	275
MAC	Wm*Wm,Acc,[Wx],Wxd,[Wy], Wyd <sup>(2)</sup>	Square and accumulate	1	1	ţ	ţ	仓	仓	ţ	仓		_	_	—	_	277
MOV	f {,WREG}	Move f to destination	1	1	-	_	—	-	_	_	_	€	—	Û	—	279
MOV	WREG, f	Move WREG to f	1	1	_	_	_	_	_	_	_	_	—	_	_	280
MOV	f,Wnd	Move f to Wnd	1	1	-	_	_	_	_	_	_	_	—	—	—	281
MOV	Wns,f	Move Wns to f	1	1	_	—	—	_	_	—		—	—	—	—	282
MOV.B	#lit8,Wnd	Move 8-bit unsigned literal to Wnd	1	1	_	—	—	_	_	—		—	—	—	—	283
MOV	#lit16,Wnd	Move 16-bit literal to Wnd	1	1	_	—	—	_	_	—		—	—	—	—	284
MOV	[Ws+Slit10],Wnd	Move [Ws + Slit10] to Wnd	1	1	_	—	—	_	_	—		—	—	—	—	285
MOV	Wns,[Wd+Slit10]	Move Wns to [Wd + Slit10]	1	1	_	—	—	_	_	—		—	—	—	—	286
MOV	Wso,Wdo	Move Wso to Wdo	1	1	_	—	_	_	_	—	_	—	—	-	—	287
MOV.D	Wns,Wnd	Move double Wns to Wnd:Wnd + 1	1	2	—			—		—	_	-		—	—	289
MOV.D	Wns,Wnd	Move double Wns:Wns + 1 to Wnd	1	2	_	—	_	_	_	—	_	—	—	—	—	289
MOVPAG	#lit10,DSRPAG <sup>(3)</sup>	Move 10-bit literal to DSRPAG	1	1	_			_		_	_	-		—	-	291
MOVPAG	<pre>#lit9,DSWPAG<sup>(3)</sup></pre>	Move 9-bit literal to DSWPAG	1	1		—	—	—	-	—	—	—	—	—	—	291
MOVPAG	#lit8,TBLPAG <sup>(3)</sup>	Move 8-bit literal to TBLPAG	1	1		—	—	—	-	—	—	—	—	—	—	291
MOVPAG	Wn,DSRPAG <sup>(3)</sup>	Move Wn to DSRPAG	1	1	—	—	—	_	_	—		—	—	—	—	292
MOVPAG	Wn,DSWPAG <sup>(3)</sup>	Move Wn to DSWPAG	1	1	—	—	—	—	—	—	_	—	—		—	292
MOVPAG	Wn, TBLPAG <sup>(3)</sup>	Move Wn to TBLPAG	1	1	-	—	—	—	—	—	_	—	—		—	292
MOVSAC	Acc,[Wx],Wxd,[Wy],Wyd,AWB <sup>(2)</sup>	Move [Wx] to Wxd, and [Wy] to Wyd	1	1	_	—	—	_	_	—	_	—	—	—	—	293
MPY	Wm*Wn,Acc,[Wx],Wxd,[Wy], Wyd <sup>(2)</sup>	Multiply Wn by Wm to accumulator	1	1	¢	ŷ	Û	Û	ţ	Û		—	—	_	_	295

#### Table 7-2: Instruction Set Summary Table (Continued)

Legend: 🗘 set or cleared; 🖞 may be cleared, but never set; 🏦 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

**3:** This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	ОВ <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Numbe
MPY	Wm*Wm,Acc,[Wx],Wxd,[Wy], Wyd <sup>(2)</sup>	Square to accumulator	1	1	ŷ	Û	Û	Û	ţ	Û	_	_	—	_	—	297
MPY.N	Wm*Wn,Acc,[Wx],Wxd,[Wy], Wyd <sup>(2)</sup>	-(Multiply Wn by Wm) to accumulator	1	1	0	0	_	_	0	_	_	_	_	_	_	299
MSC	Wm*Wn,Acc,[Wx],Wxd,[Wy], Wyd,AWB <sup>(2)</sup>	Multiply and subtract from accumulator	1	1	ţ	Û	Û	Û	¢	Û	_	_	_		_	301
MUL	f	W3:W2 = f * WREG	1	1	_	_	—	—	_	—	_	_	—	_	<b>—</b>	303
MUL.SS	Wb,Ws,Wnd	{Wnd + 1,Wnd} = signed(Wb) * signed(Ws)	1	1	_	_	—	—	_	—	_	_	—	_	—	305
MUL.SS	Wb,Ws,Acc <sup>(4)</sup>	Accumulator = signed(Wb) * signed(Ws)	1	1	—	—	—	—	—	—	_	—	—	-	—	307
MUL.SU	Wb,#lit5,Wnd	{Wnd + 1,Wnd} = signed(Wb) * unsigned(lit5)	1	1	—	—	—	—	—	—	_	—	—	-	—	308
MUL.SU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = signed(Wb) * unsigned(Ws)	1	1		—	_	—	_	—	—	—	—		—	310
MUL.SU	Wb,Ws,Acc <sup>(4)</sup>	Accumulator = signed(Wb) * unsigned(Ws)	1	1		—	_	—	_	—	—	—	—		—	312
MUL.SU	Wb,#lit5,Acc <sup>(4)</sup>	Accumulator = signed(Wb) * unsigned(lit5)	1	1		—	_	—	_	—	—	—	—		—	314
MUL.US	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsigned(Wb) * signed(Ws)	1	1		—	_	—	_	—	—	—	—		—	315
MUL.US	Wb,Ws,Acc <sup>(4)</sup>	Accumulator = unsigned(Wb) * signed(Ws)	1	1		_	_	—	_	—	_	—	—	_	—	317
MUL.UU	Wb,#lit5,Wnd	{Wnd + 1,Wnd} = unsigned(Wb) * unsigned(lit5)	1	1		_	_	—	_	—	_	—	—	_	—	319
MUL.UU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsigned(Wb) * unsigned(Ws)	1	1		_	_	—	_	—	_	—	—	_	—	320
MUL.UU	Wb,Ws,Acc <sup>(4)</sup>	Accumulator = unsigned(Wb) * unsigned(Ws)	1	1						_	_	-		_		322
MUL.UU	Wb,#lit5,Acc <sup>(4)</sup>	Accumulator = unsigned(Wb) * unsigned(lit5)	1	1		—	—	—		—	—	—	—	—	—	323
MULW.SS	5 Wb,Ws,Wnd <sup>(3)</sup>	Wnd = signed(Wb) * signed(Ws)	1	1		—	—	—		—	—	—	—	—	—	324
MULW.SU	J Wb,Ws,Wnd <sup>(3)</sup>	Wnd = signed(Wb) * unsigned(Ws)	1	1		—	—	—		—	—	—	—	—	—	326
MULW.SU	J Wb,#lit5,Wnd <sup>(3)</sup>	Wnd = signed(Wb) * unsigned(lit5)	1	1				_		_	_	-	-		-	328
MULW.US	5 Wb,Ws,Wnd <sup>(3)</sup>	Wnd = unsigned(Wb) * signed(Ws)	1	1				_		_	_	-	-		-	329
MULW.UU	J Wb,Ws,Wnd <sup>(3)</sup>	Wnd = unsigned(Wb) * unsigned(Ws)	1	1	_	-		-	-	—	—	-	—		—	331
MULW.UU	J Wb,#lit5,Wnd <sup>(3)</sup>	Wnd = unsigned(Wb) * unsigned(lit5)	1	1				-		—	_					332
NEG	f {,WREG}	Destination = $\overline{f} + 1$	1	1	_	_	_	_	_	—	ţ	Û	Û	ţ	Û	333
NEG	Ws,Wd	$Wd = \overline{Ws} + 1$	1	1	—	—	—	_	—	—	ţ	€	Û	¢	Û	333
NEG	Acc <sup>(2)</sup>	Negate accumulator	1	1	Û	Û	Û	仓	ţ	Û	_	—	_	_	_	335
NOP		No operation	1	1	_	_	_		_	_	_	_	_	_	_	336

SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged. Note 1:

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices. This instruction/operand is only available in PIC24E and dsPIC33E devices.

3: 4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	С	Page Number
NOPR		No operation	1	1	-	—	_	_	_	-	—	—	—	_	-	336
POP	f	POP TOS to f	1	1	—	—	—	—	—	—	—	—	—			337
POP	Wdo	POP TOS to Wdo	1	1	—	—	—	—	—	—	—	—	—			338
POP.D	Wnd	POP double from TOS to Wnd:Wnd + 1	1	2	—	—	—	—	—	—	—	—	—			339
POP.S		POP shadow registers	1	1	_	_	_	_	_	_	€	€	ţ	Û	€	340
PUSH	f	PUSH f to TOS	1	1	-	—	—	—	—	—	—	—	—	_	-	341
PUSH	Wso	PUSH Wso to TOS	1	1	—	—	—	—	—	—	—	—	—			342
PUSH.D	Wns	PUSH double Wns:Wns + 1 to TOS	1	2	—	—	—	—	—	—	—	—	—			343
PUSH.S		PUSH shadow registers	1	1	—	—	—	—	—	—	—	—	—			345
PWRSAV	#lit1	Enter Power-saving mode	1	1	—	—	_	—	_	—	_	—	—	_	_	346
RCALL	Expr	Relative call	1	2	-					_	Ι	_		_	_	347
RCALL	Wn	Computed call	1	2	-					-	-	—		_	_	351
REPEAT	#lit14 <sup>(5)</sup>	Repeat next instruction (lit14 + 1) times	1	1	—	—	—	—	—	—	—	—	—			355
REPEAT	#lit15 <sup>(3)</sup>	Repeat next instruction (lit15 + 1) times	1	1	—	—	—	—	—	—	—	—	—			357
REPEAT	Wn	Repeat next instruction (Wn + 1) times	1	1	—	—	—	—	—	—	—	—	—	_	_	359
RESET		Software device Reset	1	1	-					_		—		_	—	363
RETFIE		Return from interrupt enable	1	3 (2)	—	-	-		-	—		ţ	€	Û	ŷ	365
RETLW	#lit10,Wn	Return with lit10 in Wn	1	3 (2)	—	—	—	—	—	—	—	—	—			367
RETURN		Return from subroutine	1	3 (2)	—	—	—	—	—	—	—	—	—	_	_	371
RLC	f {,WREG}	Destination = rotate left through Carry f	1	1	—	—	_	—	_	—	_	ţ	—	Û	Û	373
RLC	Ws,Wd	Wd = rotate left through Carry Ws	1	1	—	_	_	—	_	_	_	Û	_	Û	€	375
RLNC	f {,WREG}	Destination = rotate left (no Carry) f	1	1	_	_	_	_	_	_	_	Û	—	Û	—	377
RLNC	Ws,Wd	Wd = rotate left (no Carry) Ws	1	1	-	_	_	_	_	—	_	€	_	Û	_	379
RRC	f {,WREG}	Destination = rotate right through Carry f	1	1	- 1	—	—	—	—	—	—	Û	—	Û	Û	381
RRC	Ws,Wd	Wd = rotate right through Carry Ws	1	1	—	—	_	—	_	_	_	ţ	—	Û	Û	383
RRNC	f {,WREG}	Destination = rotate right (no Carry) f	1	1	- 1	—	_	—	_	_	_	€	_	Û	_	385
RRNC	Ws,Wd	Wd = rotate right (no Carry) Ws	1	1	- 1	—	_	—	—	_	—	Û	_	Û	_	387
SAC	Acc,#Slit4,Wdo <sup>(2)</sup>	Store accumulator	1	1	- 1	_	_	_	_	_	_	_	_	_	_	389

#### Table 7-2: Instruction Set Summary Table (Continued)

DS70157F-page 493

Legend: 1 set or cleared; 4 may be cleared, but never set; 1 may be set, but never cleared; 1 always set; 0 always cleared; — unchanged

1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

**3:** This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	OA <sup>(2)</sup>	OB <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
SAC.R	Acc,#Slit4,Wdo <sup>(2)</sup>	Store rounded Accumulator	1	1	-	-	-	-	—	-	_	-	-	-	-	391
SE	Ws,Wd	Wd = sign-extended Ws	1	1	—	—	-	—	—	—	—	ŷ	_	Û	ţ	393
SETM	f	f = 0xFFFF	1	1	—	—	—	—	—	_		—	—	—	—	395
SETM	WREG	WREG = 0xFFFF	1	1	_	_	-	_	—	—	—	—	_	_	—	395
SETM	Wd	Wd = 0xFFFF	1	1			-			_	-	-	Ι	-	-	396
SFTAC	Acc,#Slit6 <sup>(2)</sup>	Arithmetic shift accumulator by Slit6	1	1	€	Û	仓	仓	ţ	仓	_		I	Ι	-	397
SFTAC	Acc, Wb <sup>(2)</sup>	Arithmetic shift accumulator by (Wb)	1	1	Û	€	Û	仓	ţ	Ŷ	_	_	_	_	-	398
SL	f {,WREG}	Destination = arithmetic left shift f	1	1	_	—	—	—	_			Û	—	Û	ţ	399
SL	Ws,Wd	Wd = arithmetic left shift Ws	1	1	_	—	—	—	_			Û	—	€	ţ	401
SL	Wb,#lit4,Wnd	Wnd = left shift Wb by lit4	1	1	_	_	—	_	—	—	_	Û	_	€	—	403
SL	Wb,Wns,Wnd	Wnd = left shift Wb by Wns	1	1	_	_	-	_	_	—	_	Û	_	Û	—	404
SUB	f {,WREG}	Destination = f – WREG	1	1	_	_	-	_	—	—	¢	Û	Û	Û	€	405
SUB	#lit10,Wn	Wn = Wn - lit10	1	1	_	—	—	—	—	_	ţ	Û	Û	Û	ţ	406
SUB	Wb,#lit5,Wd	Wd = Wb - lit5	1	1	_	_	-	_	_	_	Û	Û	€	Û	Û	407
SUB	Wb,Ws,Wd	Wd = Wb – Ws	1	1	_	_	—	_	_	—	Û	Û	Û	Û	Û	408
SUB	Acc <sup>(2)</sup>	Subtract accumulators	1	1	€	€	企	企	Û	仓	_	_	_	_	_	410
SUBB	f {,WREG}	destination = f – WREG – $(\overline{C})$	1	1	_	—	—	—	—	_	¢	Û	Û	Û	ţ	411
SUBB	#lit10,Wn	$Wn = Wn - lit10 - (\overline{C})$	1	1	_	_	_	_	_	_	Û	Û	Û	Û	Û	412
SUBB	Wb,#lit5,Wd	$Wd = Wb - lit5 - (\overline{C})$	1	1	_	_	-	_	—	—	ţ	Û	€	Û	ţ	413
SUBB	Wb,Ws,Wd	$Wd = Wb - Ws - (\overline{C})$	1	1	_	_	-	_	_	—	Û	Û	€	Û	Û	415
SUBBR	f {,WREG}	Destination = WREG – f – ( $\overline{C}$ )	1	1	_	—	—	—	—	_	ţ	Û	Û	Û	ţ	417
SUBBR	Wb,#lit5,Wd	$Wd = lit5 - Wb - (\overline{C})$	1	1	_	_	—	—	_	_	¢	Û	Û	Û	Û	418
SUBBR	Wb,Ws,Wd	$Wd = Ws - Wb - (\overline{C})$	1	1	_	_	—	—	_	_	ţ	Û	Û	Û	ţ	420
SUBR	f {,WREG}	Destination = WREG – f	1	1	—	—	l —	_	—	_	Û	Û	Û	Û	Û	422
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	1	1	—	—	t – 1	—	—	_	Û	Û	Û	Û	Û	423
SUBR	Wb,Ws,Wd	Wd = Ws – Wb	1	1	-	—	<u> </u>	_	—	_	Û	Û	€	Û	Û	424
SWAP	Wn	Wn = byte or nibble swap Wn	1	1	_	_	_	_		_	_	_	_	_	_	426

Legend: 1 set or cleared; 4 may be cleared, but never set; 1 may be set, but never cleared; 1 always set; 0 always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.

#### Table 7-2: Instruction Set Summary Table (Continued)

	Assembly Syntax Mnemonic, Operands	Description	Words	Cycles	0A <sup>(2)</sup>	0B <sup>(2)</sup>	SA <sup>(1,2)</sup>	SB <sup>(1,2)</sup>	0AB <sup>(2)</sup>	SAB <sup>(1,2)</sup>	DC	N	ov	z	с	Page Number
TBLRDH	[Ws],Wd	Read high program word to Wd	1	2	—	_	—	—	-	—	_	—	—	—	—	427
TBLRDL	[Ws],Wd	Read low program word to Wd	1	2	—	I	—	_	—	—			—		—	429
TBLWTH	Ws,[Wd]	Write Ws to high program word	1	2	—	I	—	_	—	—			—		—	431
TBLWTL	Ws,[Wd]	Write Ws to low program word	1	2	—	I	—	_	—	—			—		—	433
ULNK		Unlink Frame Pointer	1	1	_	_	-	_	_	—	_	_	—	_	—	435
XOR	f {,WREG}	Destination = f .XOR. WREG	1	1	—	_	—	_	_	—	_	Û	—	Û	_	437
XOR	#lit10,Wn	Wn = lit10 .XOR. Wn	1	1	_		—		_	—		Û	—	Û	—	438
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1	_		—		—	_		Û	_	Û	—	439
XOR	Wb,Ws,Wd	Wd = Wb .XOR. Ws	1	1	—	_	—	_	—	—	-	Û	—	Û	—	440
ZE	Ws,Wnd	Wnd = zero-extended Ws	1	1	_	_	—	_	—	—	_	0	_	Û	1	442

Legend: 1 set or cleared; 4 may be cleared, but never set; 1 may be set, but never cleared; 1 always set; 0 always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, and dsPIC33E devices.

3: This instruction/operand is only available in PIC24E and dsPIC33E devices.

4: This instruction/operand is only available in dsPIC33E devices.

5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F, and dsPIC33F devices.



## 7.3 REVISION HISTORY

## Revision A (May 2005)

This is the initial release of this document.

## **Revision B (September 2005)**

This revision incorporates all known errata at the time of this document update.

## **Revision C (February 2008)**

This revision includes the following corrections and updates:

- Instruction Updates:
  - Updated BRA Instruction (see "BRA")
  - Updated DIVF Instruction (see "DIVF")
  - Updated DO Instruction (see "DO")
  - Updated SUB instruction (see "SUB")

## **Revision D (November 2009)**

This revision includes the following corrections and updates:

- Document renamed from dsPIC30F/33F Programmer's Reference Manual to 16-bit MCU and DSC Programmer's Reference Manual
- Document has been completely redesigned to accommodate all current 16-bit families: dsPIC30F, dsPIC33F, PIC24F and PIC24H

## Revision E (June 2010)

This revision includes the following corrections and updates:

 Information specific to dsPIC33E and PIC24E devices has been added throughout the document

## Revision F (July 2011)

This revision includes the following corrections and updates:

- Added a new section "Built-in Functions"
- Added and updated the cross-references throughout the document
- Updated the bit characteristics from U to U-0 in Register 2-4 and Register 2-6
- Added a note throughout the document specifying the requirement of an additional cycle for read and read-modify-write operations on non-CPU special function registers in dsPIC33E and PIC24E devices
- Updates to formatting and minor text changes were incorporated throughout the document

## INDEX

## Symbols

builtin_addab	447
builtin_btg	
builtin_divmodud	
builtin_divsd	
builtin_edsoffset	458
builtin_edspage	
builtin_mac	
builtin_modsd	
builtin_modud	
builtin_mpy	
builtin_mpyn	
builtin_mulss	
builtin_muluu	
builtin_nop	
builtin_psvoffset	
builtin_sac	
builtin_subab	
builtin_tbladdress	
builtin_tblwth	
builtin_tblwtl	

## Α

Accumulator A, Accumulator B	
Accumulator Access	
Accumulator Selection	
Accumulator Usage	
Addressing Modes for Wd Destination Register	
Addressing Modes for Ws Source Register	
Assigned Working Register Usage	

## в

## **Built-In Functions**

builtin_addab	447
builtin_btg	
builtin_divmodud	
builtin_divsd	454
builtin_edsoffset	458
builtin_edspage	458
builtin_mac	460
builtin_modsd	
builtin_modud	
builtin_mpy	
builtin_mpyn	465
builtin_mulss	468
builtin_muluu	469
builtin_nop	470
builtin_psvoffset	
builtin_sac	
builtin_subab	475
builtin_tbladdress	475
builtin_tblwth	
builtin_tblwtl	
Byte Operations	

## С

Code Exam	ples
-----------	------

'Z' Status bit Operation for 32-bit Addition	77
Base MAC Syntax	
File Register Addressing	53
File Register Addressing and WREG	53
Frame Pointer Usage	73
Illegal Word Move Operations	68

Immediate Addressing	59
Indirect Addressing with Effective Address Update	55
Indirect Addressing with Register Offset	56
Legal Word Move Operations	67
MAC Accumulator WB Syntax	87
MAC Prefetch Syntax	86
Move with Literal Offset Instructions	56
MSC Instruction with Two Prefetches and Accumulate	or
Write Back	87
Normalizing with FBCL	90
Register Direct Addressing	54
Sample Byte Math Operations	65
Sample Byte Move Operations	64
Scaling with FBCL	89
Stack Pointer Usage	71
Unsigned f and WREG Multiply (Legacy MULWF	
Instruction)	80
Using 10-bit Literals for Byte Operands	69
Using the Default Working Register WREG	79
Conditional Branch Instructions	76
Core Control Register	24

## D

Data Addressing Mode Tree	59
Data Addressing Modes	52
DCOUNT Register	20
Default Working Register (WREG)	
Development Support	
DOEND Register	
DOSTART Register	
DSP Accumulator Instructions	
DSP Data Formats	81
DSP MAC Indirect Addressing Modes	57
DSP MAC Instructions	84

## F

File	Register	Addressing	52
------	----------	------------	----

### L

Immediate Addressing	
Operands in the Instruction Set	58
Implied DSP Operands	
Implied Frame and Stack Pointer	
Instruction Bit Map	
Instruction Description Example	98
Instruction Descriptions	
ADD (16-bit Signed Add to Accumulator)	104
ADD (Add Accumulators)	103
ADD (Add f to WREG)	99
ADD (Add Literal to Wn)	100
ADD (Add Wb to Short Literal)	
ADD (Add Wb to Ws)	
ADDC (Add f to WREG with Carry)	106
ADDC (Add Literal to Wn with Carry)	107
ADDC (Add Wb to Short Literal with Carry)	108
ADDC (Add Wb to Ws with Carry)	110
AND (AND f and WREG)	
AND (AND Literal and Wn)	
AND (AND Wb and Short Literal)	114
AND (AND Wb and Ws)	115
ASR (Arithmetic Shift Right by Short Literal)	121
ASR (Arithmetic Shift Right by Wns)	122
ASR (Arithmetic Shift Right f)	
ASR (Arithmetic Shift Right Ws)	119

BCLR (Bit Clear in Ws)         124           BCLR.B (Bit Clear f)         123
$D \cap D \cap D (D; t \cap D; t \cap f)$ (0.0)
BCLR.B (Bit Clear f) 123
BRA (Branch Unconditionally)126
BRA (Computed Branch)
DRA (Computed Dianon)
BRA C (Branch if Carry)130
BRA GE (Branch if Signed Greater Than or Equal) 132
BRA GEU (Branch if Unsigned Greater Than
or Equal)
BRA GT (Branch if Signed Greater Than)
BRA GTU (Branch if Unsigned Greater Than)
BRA LE (Branch if Signed Less Than or Equal) 137
BRA LEU (Branch if Unsigned Less Than or Equal). 138
BRA LT (Branch if Signed Less Than)
BRA LTU (Branch if Not Carry) 142
BRA LTU (Branch if Unsigned Less Than) 140
BRA N (Branch if Negative)141
BRA NN (Branch if Not Negative)
BRA NOV (Branch if Not Overflow)144
BRA NZ (Branch if Not Zero)145
BRA OA (Branch if Overflow Accumulator A)
BRA OB (Branch if Overflow Accumulator B)
BRA OV (Branch if Overflow)148
BRA SA (Branch if Saturation Accumulator A) 149
BRA SB (Branch if Saturation Accumulator B) 150
BRA Z (Branch if Zero)151
BSET (Bit Set f)152
BSET (Bit Set in Ws)153
BSW (Bit Write in Ws)155
BTG (Bit Toggle f) 157
BTG (Bit Toggle in Ws) 158
BTSC (Bit Test f, Skip if Clear) 160
BTSC (Bit Test Ws, Skip if Clear) 162
BTSS (Bit Test f, Skip if Set)164
DTCC (Dit Test 1, okip il Oct)
BTSS (Bit Test Ws, Skip if Set)166
BTST (Bit Test f) 168
BTST (Bit Test in Ws) 169, 171
BTSTS (Bit Test/Set f) 173
BTSTS (Bit Test/Set f)
BTSTS (Bit Test/Set f)         173           BTSTS (Bit Test/Set in Ws)         175           CALL (Call Indirect Subroutine)         180, 181
BTSTS (Bit Test/Set f)         173           BTSTS (Bit Test/Set in Ws)         175           CALL (Call Indirect Subroutine)         180, 181
BTSTS (Bit Test/Set f)         173           BTSTS (Bit Test/Set in Ws)         175           CALL (Call Indirect Subroutine)         180, 181           CALL (Call Subroutine)         177, 178
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL.L (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL.L (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL. (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       192
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       192         CP (Compare Wb with lit8, Set Status Flags)       193
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare Wb with lit5, Set Status Flags)       191         CP (Compare Wb with lit6, Set Status Flags)       193         CP (Compare Wb with lit8, Set Status Flags)       193         CP (Compare Wb with lit8, Set Status Flags)       193         CP (Compare Wb with Ws, Set Status Flags)       194
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear f or WREG)184CLR (Clear Wd)185CLRWDT (Clear Watchdog Timer)188COM (Complement f)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with Ws, Set Status Flags)194CP0 (Compare f with 0x0, Set Status Flags)194
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if Less Than)205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear Wd)185CLRWDT (Clear Watchdog Timer)188COM (Complement f)189COM (Complement Ws)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with Ws, Set Status Flags)194CP0 (Compare Wb with 0x0, Set Status Flags)196CP0 (Compare Wb with 0x0, Set Status Flags)197
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear f or WREG)184CLR (Clear Wd)185CLRWDT (Clear Watchdog Timer)188COM (Complement f)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with Ws, Set Status Flags)194CP0 (Compare f with 0x0, Set Status Flags)194
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if Less Than)205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear Md)185CLR (Clear Wd)185CLR (Clear Wd)188COM (Complement f)189COM (Complement Ws)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with Nt, Set Status Flags)194CP0 (Compare f with 0x0, Set Status Flags)194CP0 (Compare Wb with Nt, Set Status Flags)196CP0 (Compare f with 0x0, Set Status Flags)197CPB (Compare f with WREG using Borrow,197
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if Less Than)205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear f or WREG)184CLR (Clear Wd)185CLRWDT (Clear Watchdog Timer)188COM (Complement f)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with 0x0, Set Status Flags)194CP0 (Compare Wb with 0x0, Set Status Flags)196CP0 (Compare Wb with 0x0, Set Status Flags)197CPB (Compare f with WREG using Borrow, Set Status Flags)198
BTSTS (Bit Test/Set f)173BTSTS (Bit Test/Set in Ws)175CALL (Call Indirect Subroutine)180, 181CALL (Call Subroutine)177, 178CALL (Call Indirect Subroutine Long)183CBSLT (Signed Compare Wb with Wn, Branch if Less Than)205CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear Accumulator, Prefetch Operands)186CLR (Clear Wd)185CLRWDT (Clear Watchdog Timer)188COM (Complement f)190CP (Compare f with WREG, Set Status Flags)191CP (Compare Wb with lit5, Set Status Flags)193CP (Compare Wb with Na, Set Status Flags)194CP0 (Compare f with 0x0, Set Status Flags)197CPB (Compare f with WREG using Borrow, Set Status Flags)198CPB (Compare Wb with lit5 using Borrow,198
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       183         Less Than)       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Md)       185         CLR (Clear Wd)       185         CLR (Complement Ws)       190         CP (Compare Wb with lit5, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Wks, Set Status Flags)       194         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare f with WREG using Borrow,       198         CPB (Compare Wb with lit5 using Borrow,       199, 200
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       183         Less Than)       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLR (Clear Wd)       185         CLR (Clear Wd)       185         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       192         CP (Compare Wb with Ws, Set Status Flags)       193         CP0 (Compare f with 0x0, Set Status Flags)       194         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare f with WREG using Borrow,       198         CPB (Compare Wb with lit5 using Borrow,       199, 200         CPB (Compare Ws with Wb using Borrow,       199, 200
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       183         Less Than)       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Md)       185         CLR (Clear Wd)       185         CLR (Complement Ws)       190         CP (Compare Wb with lit5, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Wks, Set Status Flags)       194         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare f with WREG using Borrow,       198         CPB (Compare Wb with lit5 using Borrow,       199, 200
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement f)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       194         CP0 (Compare Wb with 0x0, Set Status Flags)       196         CP0 (Compare Wb with 0x0, Set Status Flags)       197         CPB (Compare Wb with NREG using Borrow, Set Status Flags)       198         CPB (Compare Wb with lit5 using Borrow, Set Status Flags)       199, 200         CPB (Compare Ws with Wb using Borrow, Set Status Flags)       199, 200
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement f)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Ws, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       196         CP0 (Compare f with 0x0, Set Status Flags)       196         CP0 (Compare f with WREG using Borrow, Set Status Flags)       198         CPB (Compare Wb with lit5 using Borrow, Set Status Flags)       199, 200         CPB (Compare Ws with Wb using Borrow, Set Status Flags)       201         CPBEQ (Compare Wb with Wh, Branch if Equal)       203
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       196         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare Wb with lit5 using Borrow, Set Status Flags)       198         CPB (Compare Wb with Wb using Borrow, Set Status Flags)       201         CPBEQ (Compare Wb with Wh, Branch if Equal)       203         CPBEQ (Compare Wb with Wh, Branch if Equal)       203
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Vd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Wx, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       194         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare Wb with lit5 using Borrow,       3et Status Flags)       198         CPB (Compare Wb with lit5 using Borrow,       3et Status Flags)       199       200         CPB (Compare Wb with Wth Branch if Equal)       203       204       204
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       196         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare Wb with lit5 using Borrow, Set Status Flags)       198         CPB (Compare Wb with Wb using Borrow, Set Status Flags)       201         CPBEQ (Compare Wb with Wh, Branch if Equal)       203         CPBEQ (Compare Wb with Wh, Branch if Equal)       203
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Indirect Subroutine)       177, 178         CALL.L (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear f or WREG)       184         CLR (Clear Wd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement f)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Ws, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       196         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare Wb with lit5 using Borrow, Set Status Flags)       198         CPB (Compare Wb with Wh Branch if Equal)       203         CPBEQ (Compare Wb with Wn, Branch if Equal)       203         CPBEQ (Compare Wb with Wn, Branch if Equal)       204         CPBNE (Signed Compare Wb with Wn, Branch if N
BTSTS (Bit Test/Set f)       173         BTSTS (Bit Test/Set in Ws)       175         CALL (Call Indirect Subroutine)       180, 181         CALL (Call Subroutine)       177, 178         CALL (Call Indirect Subroutine Long)       183         CBSLT (Signed Compare Wb with Wn, Branch if       205         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Accumulator, Prefetch Operands)       186         CLR (Clear Vd)       185         CLRWDT (Clear Watchdog Timer)       188         COM (Complement f)       189         COM (Complement Ws)       190         CP (Compare f with WREG, Set Status Flags)       191         CP (Compare Wb with lit5, Set Status Flags)       193         CP (Compare Wb with Wx, Set Status Flags)       193         CP (Compare Wb with 0x0, Set Status Flags)       194         CP0 (Compare f with 0x0, Set Status Flags)       197         CPB (Compare Wb with lit5 using Borrow,       3et Status Flags)       198         CPB (Compare Wb with lit5 using Borrow,       3et Status Flags)       199       200         CPB (Compare Wb with Wth Branch if Equal)       203       204       204

CPSGT (Signed Compare Wb with Wn,
Skip if Greater Than) 210
CPSGT (Signed Compare Wb with Wn, Skip if Greater
Than) 211
CPSLT (Signed Compare Wb with Wn,
Skip if Less Than)
CPSNE (Signed Compare Wb with Wn,
Skip if Not Equal)
DAW.B (Decimal Adjust Wn)
DEC (Decrement f)
DEC (Decrement Ws)
DEC2 (Decrement f by 2)
DEC2 (Decrement Vs by 2)
DISI (Disable Interrupts Temporarily)
DIV.S (Signed Integer Divide)
DIV.U (Unsigned Integer Divide)
DIVF (Fractional Divide) 228
DO (Initialize Hardware Loop Literal)
DO (Initialize Hardware Loop Wn) 235, 237
ED (Euclidean Distance, No Accumulate) 239
EDAC (Euclidean Distance)
EXCH (Exchange Wns and Wnd)243
FBCL (Find First Bit Change from Left) 244
FF1L (Find First One from Left)
FF1R (Find First One from Right)
GOTO (Unconditional Indirect Jump)
GOTO (Unconditional Jump)
GOTO.L (Unconditional Indirect Jump Long)
INC (Increment f)
INC (Increment Ws)
INC2 (Increment f by 2) 257
INC2 (Increment Ws by 2) 258
IOR (Inclusive OR f and WREG)
IOR (Inclusive OR Literal and Wn) 261
IOR (Inclusive OR Literal and Wn) 261
IOR (Inclusive OR Literal and Wn)
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Short Literal)275MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Wns to f)282
IOR (Inclusive OR Literal and Wn)         261           IOR (Inclusive OR Wb and Short Literal)         262           IOR (Inclusive OR Wb and Ws)         263           LAC (Load Accumulator)         265           LNK (Allocate Stack Frame)         267, 268           LSR (Logical Shift Right by Short Literal)         273           LSR (Logical Shift Right by Wns)         274           LSR (Logical Shift Right by Wns)         274           LSR (Logical Shift Right Ws)         271           MAC (Multiply and Accumulate)         275           MAC (Square and Accumulate)         277           MOV (Move 16-bit Literal to Wn)         284           MOV (Move f to Destination)         279           MOV (Move Wns to [Wd with offset])         286           MOV (Move Wns to f)         282           MOV (Move WREG to f)         280
IOR (Inclusive OR Literal and Wn)         261           IOR (Inclusive OR Wb and Short Literal)         262           IOR (Inclusive OR Wb and Ws)         263           LAC (Load Accumulator)         265           LNK (Allocate Stack Frame)         267, 268           LSR (Logical Shift Right by Short Literal)         273           LSR (Logical Shift Right by Wns)         274           LSR (Logical Shift Right by Wns)         274           LSR (Logical Shift Right f)         269           LSR (Logical Shift Right Ws)         271           MAC (Multiply and Accumulate)         275           MAC (Square and Accumulate)         277           MOV (Move 16-bit Literal to Wn)         284           MOV (Move f to Destination)         279           MOV (Move Wns to [Wd with offset])         286           MOV (Move Wns to f)         282           MOV (Move Wrs to f)         282           MOV (Move Wrs to f)         280           MOV (Move Wrs to f)         280           MOV (Move Ws to Wd)         287
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to the function function)281MOV (Move Ws to f)282MOV (Move Ws to the function function)281MOV (Move Ws to f)282MOV (Move Ws to the function function)281MOV (Move Ws to f)282MOV (Move Ws to the function function)281MOV (Move Ws to the function function)281MOV (Move Ws to Wd)287MOV (Move Ws to Wd)287MOV (Move Ws to Wd)285
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Wns to f)282MOV (Move Ws to the fibre for formal state of the fibre formal state of forma
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Ws to Wd)287MOV (Move Ws to Wd)287MOV (Move Ws to Wd)283MOV.B (Move 8-bit Literal to Wnd)283MOV.D (Double-Word Move from Source to Wnd)283
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)281MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Ws to Md)281MOV (Move Ws to Md)282MOV (Move Ws to Md)283MOV (Move Ws to Wd)283MOV.B (Move Ws to Wd)283MOV.D (Double-Word Move from Source to Wnd)289MOV-PAG (Move Literal to Page Register)291
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])282MOV (Move Ws to f)282MOV (Move Ws to the fister to Wnd)283MOV (Move S-bit Literal to Wnd)283MOV (Move Ws to Wd)287MOV (Move Ws to Wd)283MOV (Move Ws to Ws the fiset to Wnd)283MOV D (Double-Word Move from Source to Wnd)289MOVPAG (Move Ws to Page Register)291MOVPAG (Move Ws to Page Register)292
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])282MOV (Move Ws to Wd)283MOV (Move Ws to Wd)285MOV (Move Ws to Wd)285MOV.D (Double-Word Move from Source to Wnd)283MOV-DAG (Move Ws to Page Register)292MOVASAC (Prefetch Operands and292
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to this offset to Wnd)287MOV (Move Ws to Wd)287MOV (Move Ws to Wd)287MOV (Move Ws to Ws this offset to Wnd)283MOV.D (Double-Word Move from Source to Wnd)289MOVPAG (Move Ws to Page Register)291MOVPAG (Move Ws to Page Register)292MOVSAC (Prefetch Operands and Store Accumulator)293
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])282MOV (Move Ws to f)282MOV (Move Ws to to f)283MOV (Move Ws to to f)285MOV (Move Ws to Wd)287MOV (Move Ws to Wd)287MOV B (Move & bit Literal to Wnd)283MOV-B (Move & bit Literal to Wnd)284MOV-D (Double-Word Move from Source to Wnd)289MOVPAG (Move Ws to Page Register)291MOVPAG (Move Ws to Page Register)292MOVSAC (Prefetch Operands and Store Accumulator)293MPY (Multiply Wm by Wn to Accumulator)295
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])282MOV (Move Ws to f)283MOV (Move Ws to to f)283MOV (Move Ws to Wd)287MOV (Move Ws to Wd)287MOV (Move Ws to Vd)283MOV.B (Move & bit Literal to Wnd)283MOV.D (Double-Word Move from Source to Wnd)289MOVPAG (Move Ws to Page Register)291MOVPAG (Move Ws to Page Register)292MOVAG (Prefetch Operands and Store Accumulator)293MPY (Multiply Wm by Wn to Accumulator)295MPY (Square to Accumulator)297
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])282MOV (Move Ws to f)282MOV (Move Ws to to f)285MOV (Move Ws to the offset to Wnd)287MOV (Move Ws to Wd)287MOV (Move Ws to Wd)285MOV-B (Move Ws to Page Register)291MOVPAG (Move Ws to Page Register)292MOVPAG (Move Ws to Page Register)293MPY (Multiply Wm by Wn to Accumulator)295MPY (Multiply -Wm by Wn to Accumulator)297MPY.N (Multiply -Wm by Wn to Accumulator)299
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move 16 bit Literal to Wn)281MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to to f)283MOV (Move Ws to the fibre to Wnd)285MOV (Move Ws to Wd)287MOV (Move Ws to Wd)285MOV-D (Double-Word Move from Source to Wnd)283MOV-PAG (Move Literal to Page Register)291MOVPAG (Move Ws to Page Register)292MOVAG (Prefetch Operands and Store Accumulator)293MPY (Multiply Wm by Wn to Accumulator)299MSC (Multiply and Subtract from Accumulator)301
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to f)283MOV (Move Ws to to f)283MOV.B (Move Ws to Wd)287MOV (Move Ws to Wd)283MOV.D (Double-Word Move from Source to Wnd)283MOVAG (Move Literal to Page Register)291MOVAG (Move Ws to Page Register)293MPY (Multiply Wm by Wn to Accumulator)293MPY (Multiply Wm by Wn to Accumulator)299MSC (Multiply and Subtract from Accumulator)301MUL (Integer Unsigned Multiply f and WREG)303
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move 16 bit Literal to Wn)281MOV (Move f to Destination)279MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to to f)283MOV (Move Ws to the fibre to Wnd)285MOV (Move Ws to Wd)287MOV (Move Ws to Wd)285MOV-D (Double-Word Move from Source to Wnd)283MOV-PAG (Move Literal to Page Register)291MOVPAG (Move Ws to Page Register)292MOVAG (Prefetch Operands and Store Accumulator)293MPY (Multiply Wm by Wn to Accumulator)299MSC (Multiply and Subtract from Accumulator)301
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to f)283MOV (Move Ws to to f)283MOV.B (Move Ws to Wd)287MOV (Move Ws to Wd)283MOV.D (Double-Word Move from Source to Wnd)283MOVAG (Move Literal to Page Register)291MOVAG (Move Ws to Page Register)293MPY (Multiply Wm by Wn to Accumulator)293MPY (Multiply Wm by Wn to Accumulator)299MSC (Multiply and Subtract from Accumulator)301MUL (Integer Unsigned Multiply f and WREG)303
IOR (Inclusive OR Literal and Wn)261IOR (Inclusive OR Wb and Short Literal)262IOR (Inclusive OR Wb and Ws)263LAC (Load Accumulator)265LNK (Allocate Stack Frame)267, 268LSR (Logical Shift Right by Short Literal)273LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right by Wns)274LSR (Logical Shift Right f)269LSR (Logical Shift Right Ws)271MAC (Multiply and Accumulate)275MAC (Square and Accumulate)277MOV (Move 16-bit Literal to Wn)284MOV (Move f to Destination)279MOV (Move f to Wnd)281MOV (Move Wns to [Wd with offset])286MOV (Move Ws to f)282MOV (Move Ws to to f)283MOV (Move Ws to to f)285MOV.B (Move Ws with offset to Wnd)283MOV.D (Double-Word Move from Source to Wnd)283MOVPAG (Move Literal to Page Register)291MOVPAG (Move Ws to Page Register)292MOVSAC (Prefetch Operands and Store Accumulator)293MPY (Multiply Wm by Wn to Accumulator)293MPY (Multiply -Wm by Wn to Accumulator)299MSC (Multiply and Subtract from Accumulator)301MUL (Integer Unsigned Multiply f and WREG)303MUL.SS (Integer 16x16-bit Signed Multiply with

MUL.SU (Integer 16x16-bit Signed-Unsigned
Multiply with Accumulator Destination)
MUL.SU (Integer 16x16-bit Signed-Unsigned Multiply)310
MUL.SU (Integer 16x16-bit Signed-Unsigned Short
Literal Multiply with Accumulator Destination) 314
MUL.SU (Integer 16x16-bit Signed-Unsigned
Short Literal Multiply)
MUL.US (Integer 16x16-bit Signed-Unsigned
Multiply with Accumulator Destination)
MUL.US (Integer 16x16-bit Unsigned-Signed
Multiply)
MUL.UU (Integer 16x16-bit Unsigned Multiply /rwith Accu-
mulator Destination)
MUL.UU (Integer 16x16-bit Unsigned Multiply) 320
MUL.UU (Integer 16x16-bit Unsigned Short Literal
Multiply with Accumulator Destination)
MUL.UU (Integer 16x16-bit Unsigned
Short Literal Multiply)
MULW.SS (Integer 16x16-bit Signed Multiply
with 16-bit Result) 324
MULW.SU (Integer 16x16-bit Signed-Unsigned
Multiply with 16-bit Result)
MULW.SU (Integer 16x16-bit Signed-Unsigned
Short Literal Multiply with 16-bit Result)
MULW.US (Integer 16x16-bit Unsigned-Signed
Multiply with 16-bit Result)
MULW.UU (Integer 16x16-bit Unsigned Multiply
with 16-bit Result)
MULW.UU (Integer 16x16-bit Unsigned Short
Literal Multiply
with 16-bit Result)
NEG (Negate Accumulator) 335
NEG (Negate f)
NEG (Negate Ws) 333
NEG (Negate Ws)
NOP (No Operation)
NOP (No Operation)
NOP (No Operation)
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         338
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         338
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           POP.S (Pop Shadow Registers)         340
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         342
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Construction of the structure of
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           Wns+1 to TOS)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355           REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push Ws to TOS)         342           PUSH.D (Double Push Wns/         343           Wns+1 to TOS)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355           REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Double Push Wns/         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times). 357         REPEAT (Repeat Next Instruction Wn + 1 Times). 353           RETFIE (Reset)         363           RETFIE (Return from Interrupt)         365, 366
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times). 357         REPEAT (Repeat Next Instruction Wn + 1 Times). 357           REFEAT (Repeat Next Instruction Wn + 1 Times). 356, 366         RETLW (Return from Interrupt)         365, 366
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         371         372
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 355         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         371         372
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         RLC (Rotate Left f through Carry)         373
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         RLC (Rotate Left f through Carry)         373           RLC (Rotate Left Ws through Carry)         375         375
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         357, 349           REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return Nich Literal in Wn)         371, 372         RLC (Rotate Left f through Carry)         373           RLC (Rotate Left f through Carry)         375
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         RLC (Rotate Left f through Carry)         373           RLC (Rotate Left Ws through Carry)         375         375
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETLW (Return with Literal in Wn)         367, 369           RETURN (Return)         371, 372         RLC (Rotate Left f through Carry)         373           RLC (Rotate Left f without Carry)         375         RLNC (Rotat
NOP (No Operation)         336           NOPR (No Operation)         336           POP (Pop TOS to f)         337           POP (Pop TOS to Wd)         338           POP.D (Double Pop TOS to Wnd/         339           Wnd+1)         339           POP.S (Pop Shadow Registers)         340           PUSH (Push f to TOS)         341           PUSH (Push f to TOS)         342           PUSH.D (Double Push Wns/         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         343           PUSH.S (Push Shadow Registers)         345           PWRSAV (Enter Power Saving Mode)         346           RCALL (Computed Relative Call)         351, 353           RCALL (Relative Call)         347, 349           REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357         REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357           REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361         RESET (Reset)         363           RETFIE (Return from Interrupt)         365, 366         RETURN (Return with Literal in Wn)         367, 369           RETURN (Return with Literal in Wn)         367, 369         RETURN (Return)         373           RLC (Rotate Left through Carry)         375         RLNC (Rotat
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Push f to TOS)342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)347, 349REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 363RETFIE (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371, 372RLC (Rotate Left through Carry)373RLC (Rotate Left f without Carry)377RLNC (Rotate Left f without Carry)381RRC (Rotate Right f through Carry)383
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Push f to TOS)342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)347, 349REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 363RETFIE (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371RLC (Rotate Left through Carry)377RLNC (Rotate Left wes without Carry)377RLNC (Rotate Left Wes without Carry)381RRC (Rotate Right f through Carry)383RRNC (Rotate Right f without Carry)385
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Push f to TOS)342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)347, 349REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 363RETFIE (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371RLC (Rotate Left through Carry)377RLNC (Rotate Left wes without Carry)377RLNC (Rotate Left Wes without Carry)381RRC (Rotate Right f through Carry)383RRNC (Rotate Right f without Carry)385
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Double Push Wns/342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)347, 349REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361RESET (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371RLC (Rotate Left f through Carry)377RLNC (Rotate Left f without Carry)379RRC (Rotate Right f through Carry)383RRNC (Rotate Right f without Carry)385RRNC (Rotate Right f without Carry)385
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Push f to TOS)342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)351, 353RCALL (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361RESET (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371, 372RLC (Rotate Left through Carry)373RLC (Rotate Left f without Carry)377RLNC (Rotate Left f without Carry)381RRC (Rotate Right f through Carry)383RRNC (Rotate Right f without Carry)385RRNC (Rotate Right f without Carry)385RRNC (Rotate Right f without Carry)387SAC (Store Accumulator)389
NOP (No Operation)336NOPR (No Operation)336POP (Pop TOS to f)337POP (Pop TOS to Wd)338POP.D (Double Pop TOS to Wnd/339Wnd+1)339POP.S (Pop Shadow Registers)340PUSH (Push f to TOS)341PUSH (Double Push Wns/342PUSH.D (Double Push Wns/343Wns+1 to TOS)343PUSH.S (Push Shadow Registers)345PWRSAV (Enter Power Saving Mode)346RCALL (Computed Relative Call)351, 353RCALL (Relative Call)347, 349REPEAT (Repeat Next Instruction 'lit14 + 1' Times). 357REPEAT (Repeat Next Instruction 'lit15 + 1' Times). 357REPEAT (Repeat Next Instruction Wn + 1 Times).359, 361RESET (Reset)363RETFIE (Return from Interrupt)365, 366RETLW (Return with Literal in Wn)367, 369RETURN (Return)371RLC (Rotate Left f through Carry)377RLNC (Rotate Left f without Carry)379RRC (Rotate Right f through Carry)383RRNC (Rotate Right f without Carry)385RRNC (Rotate Right f without Carry)387

SETM (Set Ws)	395
SFTAC (Arithmetic Shift Accumulator by Slit5)	
SFTAC (Arithmetic Shift Accumulator by Wb)	
SL (Shift Left by Short Literal)	
SL (Shift Left by Wns)	
SL (Shift Left f)	
SL (Shift Left Ws)	
SUB (Subtract Accumulators) SUB (Subtract Literal from Wn)	
SUB (Subtract Short Literal from Wb) SUB (Subtract WREG from f)	
SUB (Subtract WKEG from Wb)	
SUBB (Subtract Short Literal from Wb with Borrow	
SUBB (Subtract Wn from Literal with Borrow)	
SUBB (Subtract WREG and Carry bit from f)	
SUBB (Subtract Ws from Wb with Borrow)	
SUBBR (Subtract f from WREG with Borrow)	
SUBBR (Subtract Wb from Short Literal	
with Borrow)	418
SUBBR (Subtract Wb from Ws with Borrow)	
SUBR (Subtract f from WREG)	
SUBR (Subtract Wb from Short Literal)	
SUBR (Subtract Wb from Ws)	
SWAP (Byte or Nibble Swap Wn)	
TBLRDH (Table Read High)	
TBLRDL (Table Read Low)	
TBLWTH (Table Write High)	431
TBLWTL (Table Write Low)	433
ULNK (De-allocate Stack Frame) 43	5, 436
XOR (Exclusive OR f and WREG)	437
XOR (Exclusive OR Literal and Wn)	
XOR (Exclusive OR Wb and Short Literal)	
XOR (Exclusive OR Wb and Ws)	440
ZE (Zero-Extend Wn)	442
Instruction Encoding Field Descriptors Introduction	
	94
Instruction Set Overview	94 38
Instruction Set Overview Bit Instructions	94 38 45
Instruction Set Overview Bit Instructions Compare/Skip Instructions	94 38 45 46
Instruction Set Overview Bit Instructions Compare/Skip Instructions Control Instructions	94 38 45 46 49
Instruction Set Overview Bit Instructions Compare/Skip Instructions Control Instructions DSP Instructions	94 38 45 46 49 50
Instruction Set Overview Bit Instructions Compare/Skip Instructions Control Instructions DSP Instructions dsPIC30F/33F Instruction Groups	94 38 45 46 49 50 38
Instruction Set Overview Bit Instructions Compare/Skip Instructions Control Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions	94 38 45 46 49 50 38 43
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions	94 38 45 46 49 50 38 43 41
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions Move Instructions	94 38 45 46 49 50 38 43 41 40
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions Move Instructions Program Flow Instructions	
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions Move Instructions Program Flow Instructions Rotate/Shift Instructions	
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions Move Instructions Program Flow Instructions Rotate/Shift Instructions Shadow/Stack Instructions	
Instruction Set Overview Bit Instructions Compare/Skip Instructions DSP Instructions dsPIC30F/33F Instruction Groups Logic Instructions Math Instructions Move Instructions Program Flow Instructions Rotate/Shift Instructions Shadow/Stack Instructions Instruction Set Summary Table	
Instruction Set Overview Bit Instructions	
Instruction Set Overview	
Instruction Set Overview Bit Instructions	
Instruction Set Overview Bit Instructions	
Instruction Set Overview	
Instruction Set Overview	

# 16-bit MCU and DSC Programmer's Reference Manual

lit5
lit8
Slit10
Slit16
Slit4
Slit5
TOS
Wb
Wd
Wm, Wn
٧m <sup>*</sup> Wm 8
۲۴ Wm*Wn
Wn٤
۳۴
Wns
WREG
Ws
Wx٤
۲۴ Wxd
Wy٤
۷٤
Instruction Stalls
DO/REPEAT Loops63
Exceptions
Instructions that Change Program Flow
PSV63
RAW Dependency Detection62
Instruction Symbols
Integer 16x16-bit Signed-Unsigned Short Literal Multiply. 328
Integer 16x16-bit Unsigned Short Literal Multiply
with 16-bit Result
Integer and Fractional Data81
Representation82
Interrupt Priority Level

# M

MAC	
Operations	85
Prefetch Register Updates	85
Prefetches	84
Syntax	85
Write Back	85
MAC Accumulator Write Back Selection	97
MAC or MPY Source Operands (Different Working Regist	ster)97
MAC or MPY Source Operands (Same Working Registe	r). 97
Manual Objective	6
Modulo and Bit-Reversed Addressing Modes	57
MOPAG Destination Selection	97
Multi-Cycle Instructions	39
Multi-Word Instructions	39

## Ν

Normalizing the Accumulator with the FBCL Instruction ..... 90

## ο

Offset Addressing Modes for Wd Destination Register	
(with Register Offset)99	5
Offset Addressing Modes for Ws Source Register	
(with Register Offset)99	5
Ρ	
F	
PIC® Microcontroller Compatibility	9
PRODH	
PRODL Register Pair7	9

Program Counter	19
Programmer's Model	
Register Descriptions	18
PSVPAG Register	19

## R

RCOUNT Register	20
Register Direct Addressing	
Register Indirect Addressing	54
Modes	54
Register Indirect Addressing and the Instruction Set	57
Registers	
CORCON (Core Control)	32, 35
CORCON (Core Control) Register	31, 33
SR (CPU Status)	26, 29
SR (Status) Register	27

### S

Scaling Data with the FBCL Instruction	88
Scaling Examples	89
Shadow Registers	
Automatic Usage	
Software Stack Frame Pointer	
Example	73
Overflow	74
Underflow	75
Software Stack Pointer	
Example	71
Stack Pointer Limit Register (SPLIM)	
Status Register	
DSP ALU Status Bits	
Loop Status Bits	
MCU ALU Status Bits	
Style and Symbol Conventions	7
Document Conventions	7

## т

TBLPAG Register	9
-----------------	---

## U

Using 10-bit Literal Operands	. 69
10-bit Literal Coding	. 69

#### w

Word Move Operations	66
Data Alignment in Memory	66
Working Register Array	18

## X

#### Y

Y Data Space Prefetch Destination Y Data Space Prefetch Operation	
Z	

Z Status Bit77
----------------

NOTES:



# **Worldwide Sales and Service**

#### AMERICAS

Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/ support

Web Address: www.microchip.com

Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455

Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088

**Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075

**Cleveland** Independence, OH Tel: 216-447-0464 Fax: 216-447-0643

**Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924

Detroit Farmington Hills, MI Tel: 248-538-2250 Fax: 248-538-2260

Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453

Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608

Santa Clara Santa Clara, CA Tel: 408-961-6444 Fax: 408-961-6445

Toronto Mississauga, Ontario, Canada Tel: 905-673-0699 Fax: 905-673-6509

### ASIA/PACIFIC

Asia Pacific Office Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon Hong Kong Tel: 852-2401-1200 Fax: 852-2401-3431 Australia - Sydney Tel: 61-2-9868-6733

Fax: 61-2-9868-6755

Tel: 86-10-8569-7000 Fax: 86-10-8528-2104

**China - Chengdu** Tel: 86-28-8665-5511 Fax: 86-28-8665-7889

**China - Chongqing** Tel: 86-23-8980-9588 Fax: 86-23-8980-9500

**China - Hangzhou** Tel: 86-571-2819-3180 Fax: 86-571-2819-3189

**China - Hong Kong SAR** Tel: 852-2401-1200 Fax: 852-2401-3431

**China - Nanjing** Tel: 86-25-8473-2460 Fax: 86-25-8473-2470

**China - Qingdao** Tel: 86-532-8502-7355 Fax: 86-532-8502-7205

**China - Shanghai** Tel: 86-21-5407-5533 Fax: 86-21-5407-5066

**China - Shenyang** Tel: 86-24-2334-2829 Fax: 86-24-2334-2393

**China - Shenzhen** Tel: 86-755-8203-2660 Fax: 86-755-8203-1760

**China - Wuhan** Tel: 86-27-5980-5300 Fax: 86-27-5980-5118

**China - Xian** Tel: 86-29-8833-7252 Fax: 86-29-8833-7256

**China - Xiamen** Tel: 86-592-2388138 Fax: 86-592-2388130

**China - Zhuhai** Tel: 86-756-3210040 Fax: 86-756-3210049

## ASIA/PACIFIC

India - Bangalore Tel: 91-80-3090-4444 Fax: 91-80-3090-4123

India - New Delhi Tel: 91-11-4160-8631 Fax: 91-11-4160-8632

India - Pune Tel: 91-20-2566-1512 Fax: 91-20-2566-1513

**Japan - Yokohama** Tel: 81-45-471- 6166 Fax: 81-45-471-6122

**Korea - Daegu** Tel: 82-53-744-4301 Fax: 82-53-744-4302

Korea - Seoul Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934

Malaysia - Kuala Lumpur Tel: 60-3-6201-9857 Fax: 60-3-6201-9859

**Malaysia - Penang** Tel: 60-4-227-8870 Fax: 60-4-227-4068

Philippines - Manila Tel: 63-2-634-9065 Fax: 63-2-634-9069

**Singapore** Tel: 65-6334-8870 Fax: 65-6334-8850

**Taiwan - Hsin Chu** Tel: 886-3-6578-300 Fax: 886-3-6578-370

Taiwan - Kaohsiung Tel: 886-7-213-7830 Fax: 886-7-330-9305

**Taiwan - Taipei** Tel: 886-2-2500-6610 Fax: 886-2-2508-0102

**Thailand - Bangkok** Tel: 66-2-694-1351 Fax: 66-2-694-1350

## EUROPE

Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4450-2828 Fax: 45-4485-2829

France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

**Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44

**Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781

**Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340

**Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91

**UK - Wokingham** Tel: 44-118-921-5869 Fax: 44-118-921-5820