

# dsPIC30F Programmer's Reference Manual

High Performance Digital Signal Controllers

© 2005 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contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

#### Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

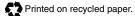
AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance and WiperLock are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.





Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



# **Table of Contents**

ECTION 1. INTRODUCTION	1-1
Introduction	
Manual Objective	
Development Support	
Style and Symbol Conventions	
Instruction Set Symbols	
Related Documents	
ECTION 2. PROGRAMMER'S MODEL	2-1
dsPIC30F Overview	
Programmer's Model	
SECTION 3. INSTRUCTION SET OVERVIEW	3-1
Introduction	
Instruction Set Overview	
Instruction Set Summary Tables	
ECTION 4. INSTRUCTION SET DETAILS	4-1
Data Addressing Modes	
Program Addressing Modes	
Instruction Stalls	
Byte Operations	
Word Move Operations	
Using 10-bit Literal Operands	
Software Stack Pointer and Frame Pointer	
Conditional Branch Instructions	
Z Status Bit	
Assigned Working Register Usage	
DSP Data Formats	
Accumulator Usage	
Accumulator Access	
DSP MAC Instructions	
DSP Accumulator Instructions Scaling Data with the FBCL Instruction	
Normalizing the Accumulator with the FBCL Instruction	
ECTION 5. INSTRUCTION DESCRIPTIONS	5-1
Instruction Symbols	
Instruction Encoding Field Descriptors Introduction	
Instruction Description Example	
Instruction Descriptions	
ECTION 6. REFERENCE	6-1
Data Memory Map	
Core Special Function Register Map	
Program Memory Map	
Instruction Bit Map	
Instruction Set Summary Table	

NOTES:

1



# Section 1. Introduction

### HIGHLIGHTS

This section of the manual contains the following topics:

1.1	Introduction	1-2
1.2	Manual Objective	1-2
1.3	Development Support	1-2
1.4	Style and Symbol Conventions	1-3
1.5	Instruction Set Symbols	1-4
1.6	Related Documents	1-5

#### 1.1 Introduction

Microchip Technology's focus is on products that meet the needs of the embedded control market. We are a leading supplier of:

- 8-bit general purpose microcontrollers (PICmicro<sup>®</sup> MCUs)
- dsPIC30F 16-bit microcontrollers
- · Speciality and standard non-volatile memory devices
- Security devices (KEELOQ<sup>®</sup>)
- Application specific standard products

Please request a Microchip Product Line Card for a listing of all the interesting products that we have to offer. This literature can be obtained from your local sales office, or downloaded from the Microchip web site (www.microchip.com).

#### 1.2 Manual Objective

PICmicro and dsPIC30F devices are grouped by the size of their Instruction Word and Data Path. The current device families are:

- 1. Base-Line: 12-bit Instruction Word length, 8-bit Data Path
- 2. Mid-Range: 14-bit Instruction Word length, 8-bit Data Path
- 3. High-End: 16-bit Instruction Word length, 8-bit Data Path
- 4. Enhanced: 16-bit Instruction Word length, 8-bit Data Path
- 5. dsPIC30F: 24-bit Instruction Word length, 16-bit Data Path

This manual is a software developer's reference for the dsPIC30F 16-bit MCU family of devices. This manual describes the Instruction Set in detail and also provides general information to assist the user in developing software for the dsPIC30F MCU family.

This manual does not include detailed information about the core, peripherals, system integration or device-specific information. The user should refer to the *dsPlC30F Family Reference Manual* for information about the core, peripherals and system integration. For device specific information, the user should refer to the data sheet. The information that can be found in the data sheet 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 dsPIC30F MCU family.

#### 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:

- 1. Code generation
- 2. Hardware/Software debug
- 3. Device programmer
- 4. 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 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. Most format conventions imply a distinction should be made for the emphasized text. The MCU industry has many symbols and non-conventional word definitions/abbreviations. Table 1-1 provides a description for many of the conventions contained in this document.

Table 1-1:	<b>Document Conventions</b>

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	<ol> <li>To force a register/bit to its default state.</li> <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(s) locations in a particular register. One example is SR <ipl2:ipl0> (or IPL&lt;2:0&gt;), which specifies the register and associated bits or bit positions.</ipl2:ipl0>
MSb, MSbit, LSb, LSbit	Indicates the Least Significant or Most Significant bit in a field.
MSByte, MSWord, LSByte, LSWord	Indicates the Least/Most Significant Byte or Word in a field of bits.
Courier Font	Used for code examples, binary numbers and for Instruction Mnemonics in the text.
Times Font	Used for equations and variables.
Times, Bold Font, Italics	Used in explanatory text for items called out from a graphic/equation/example.
Note:	A Note presents information that we wish to re-emphasize, either to help you avoid a common pitfall, or make you aware of operating differences between some device family members. In most instances, a Note is used in a shaded box (as illustrated below), however when referenced to a table, a Note will stand-alone and immediately follow the associated table (as illustrated below Table 1-2).
	Note: This is a Note in a shaded note box.

#### 1.5 Instruction Set Symbols

The Summary Tables in Section 3-2 and Section 6.5, and the instruction descriptions in Section 5.4 utilize the symbols shown in Table 1-2.

#### Table 1-2:Symbols Used in Instruction Summary Tables and Descriptions

Symbol	Description
{}	Optional field or operation
[text]	The location addressed by text
(text)	The contents of text
#text	The literal defined by text
$a\in [b,c,d]$	"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)
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)
Wx	Source Addressing mode and working register for X data bus pre-fetch
Wxd	Destination working register for X data bus pre-fetch
Wy	Source Addressing mode and working register for Y data bus pre-fetch
Wyd	Destination working register for Y data bus pre-fetch
Note: Th	be range of each symbol is instruction dependent. Befer to Section 5 "Instruction

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

#### 1.6 Related Documents

Microchip, as well as other sources, offer additional documentation which can aid in your development with dsPIC30F MCUs. These lists contain the most common documentation, but other documents may also be available. Please check the Microchip web site (www.microchip.com) for the latest published technical documentation.

#### 1.6.1 Microchip Documentation

The following dsPIC30F documentation is available from Microchip at the time of this writing. Many of these documents provide application specific information that gives actual examples of using, programming and designing with dsPIC30F MCUs.

#### 1. dsPIC30F Family Reference Manual (DS70046)

The dsPIC30F Family Reference Manual provides information about the dsPIC30F architecture, peripherals and system integration features. The details of device operation are provided in this document, along with numerous code examples.

#### 2. dsPIC30F Family Overview (DS70043)

This document provides a summary of the available dsPIC30F family variants, including device pinouts, memory sizes and available peripherals.

#### 3. dsPIC30F Data Sheets

The data sheets contain device specific information, such as pinout and packaging details, electrical specifications, and memory maps. Please check the Microchip web site (www.microchip.com) for a list of available device data sheets.

#### 1.6.2 Third Party Documentation

There are several documents available from third party sources around the world. Microchip does not review these documents for technical accuracy. However, they may be a helpful source for understanding the operation of Microchip dsPIC30F devices. Please refer to the Microchip web site (www.microchip.com) for third party documentation related to the dsPIC30F.

NOTES:



## Section 2. Programmer's Model

### HIGHLIGHTS

This section of the manual contains overview information about the dsPIC30F devices. It contains the following major topics:

2.1	dsPIC30F Overview	. 2-2
2.2	Programmer's Model	. 2-3

#### 2.1 dsPIC30F Overview

The dsPIC30F core is a 16-bit (data) modified Harvard architecture with an enhanced instruction set, including support for DSP. The core has a 24-bit instruction word, with a variable length opcode field. The program counter (PC) is 23-bits wide and addresses up to 4M x 24 bits of user program memory space. A single cycle instruction pre-fetch mechanism is used to help maintain throughput and provides predictable execution. The majority of instructions execute in a single cycle, and overhead free program loop constructs are supported using the DO and REPEAT instructions, both of which are interruptible.

The dsPIC30F has 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.

The dsPIC30F instruction set has two classes of instructions: the MCU class of instructions and the DSP class of instructions. These two instruction classes are seamlessly integrated into the architecture and execute from a single execution unit. The instruction set includes many Addressing modes and was designed for optimum C compiler efficiency.

The data space can be addressed as 32K words or 64 Kbytes and 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.

The upper 32 Kbytes of the data space memory map can optionally be mapped into program space at any 16K program word boundary, defined by the 8-bit Program Space Visibility Page (PSVPAG) register. The program to data space mapping feature lets any instruction access program space as if it were data space, which is useful for storing data coefficients.

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.

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 7 Addressing modes are supported for each instruction.

For most instructions, the dsPIC30F 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.

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 bi-directional 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.

The dsPIC30F has a vectored exception scheme with up to 8 sources of non-maskable traps and 54 interrupt sources. Each interrupt source can be assigned to one of seven priority levels.

#### 2.2 Programmer's Model

The programmer's model diagram for the dsPIC30F is shown in Figure 2-1.

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.

Register	Description
ACCA, ACCB	40-bit DSP Accumulators
CORCON	CPU Core Configuration register
DCOUNT	DO Loop Count register
DOEND	DO Loop End Address register
DOSTART	DO Loop Start Address register
PC	23-bit Program Counter
PSVPAG	Program Space Visibility Page Address register
RCOUNT	Repeat Loop Count register
SPLIM	Stack Pointer Limit Value register
SR	ALU and DSP Engine Status register
TBLPAG	Table Memory Page Address register
W0 - W15	Working register array

 Table 2-1:
 Programmer's Model Register Descriptions

#### 2.2.1 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 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.

#### 2.2.2 Default Working Register (WREG)

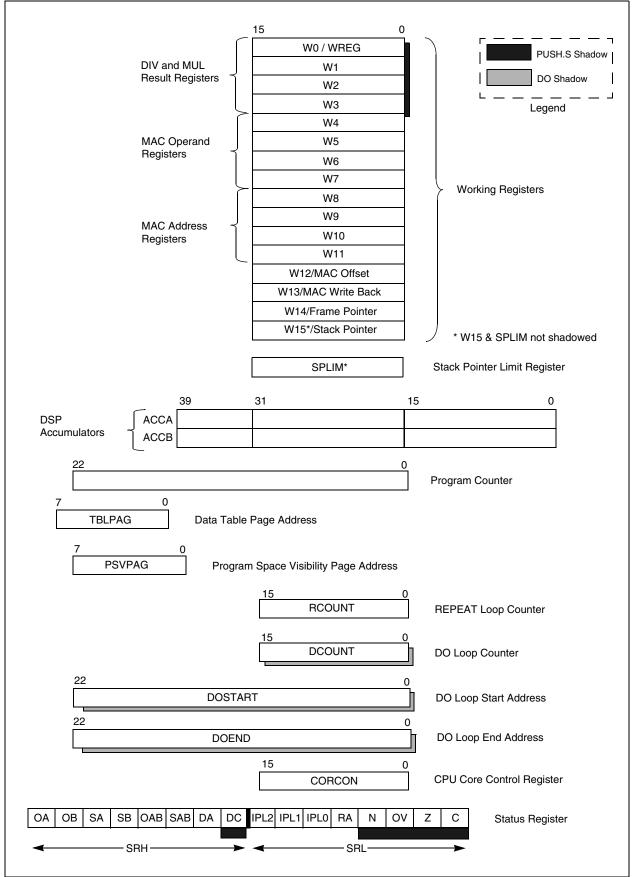
The dsPIC30F instruction set can be divided into two instruction types: working register instructions and file register instructions. The working register instructions use the working register array as data values, or as addresses that point to a memory location. In contrast, file register instructions 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.

#### 2.2.3 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. See **Section 4.7.3 "Software Stack Frame Pointer"** for detailed information about the Frame Pointer.

#### Figure 2-1: Programmer's Model Diagram



#### 2.2.4 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.

#### 2.2.5 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 4.7.5 "Stack Pointer Overflow"** for detailed information about the SPLIM.

#### 2.2.6 Accumulator A, Accumulator B

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)

Refer to **Section 4.12 "Accumulator Usage"** for details on using ACCA and ACCB.

#### 2.2.7 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 0x8000000 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. Refer to the *dsPIC30F Family Reference Manual* for details on accessing the configuration data, Unit ID and Device ID.

#### 2.2.8 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. Refer to the *dsPIC30F Family Reference Manual* for details on accessing program memory with the Table instructions.

#### 2.2.9 PSVPAG Register

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 dsPIC30F 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. Refer to the *dsPIC30F Family Reference Manual* for details on program space visibility.

#### 2.2.10 RCOUNT Register

The 14-bit RCOUNT 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, or the contents of Wn for the "REPEAT Wn" instruction. The REPEAT loop will be executed RCOUNT+1 times.

- **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 dsPIC30F Family Reference Manual for complete details about REPEAT loops.

#### 2.2.11 DCOUNT Register

The 14-bit DCOUNT register 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, or the 14 Least Significant bits of Ws for the "DO Ws, Expr" instruction. The DO loop will be executed DCOUNT+1 times.

Note 1: DCOUNT contains a shadow register. See Section 2.2.16 "Shadow Registers" for information on shadowing.

2: Refer to the dsPIC30F Family Reference Manual for complete details about DO loops.

#### 2.2.12 DOSTART Register

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 following 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: DOSTART has a shadow register. See Section 2.2.16 "Shadow Registers" for information on shadowing.

2: Refer to the dsPIC30F Family Reference Manual for complete details about DO loops.

#### 2.2.13 DOEND Register

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: DOEND has a shadow register. See Section 2.2.16 "Shadow Registers" for information on shadowing.

2: Refer to the dsPIC30F Family Reference Manual for complete details about DO loops.

#### 2.2.14 Status Register

The 16-bit Status register, shown in Register 2-1, maintains status information for instructions which have most recently been executed. 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.

#### 2.2.14.1 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, Zero, Overflow, Negative and Digit Carry (C, Z, OV, N and DC) bits are used to show the immediate status of the MCU ALU. They indicate when an operation has resulted in a carry, zero, overflow, negative result and digit carry, respectively. When a subtract operation is performed, the C flag is used as a Borrow flag.

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 a carry or borrow input (ADDC, CPB, SUBB and SUBBR). See **Section 4.9 "Z Status Bit**" for usage of the Z status bit.

- 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").

#### 2.2.14.2 Loop Status Bits

The DO Active and REPEAT Active (DA, RA) bits are 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. Likewise, 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 begins execution, and it is cleared when the instruction being repeated completes execution for the last time.

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

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'.

#### 2.2.14.3 DSP ALU Status Bits

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 the software developer efficiency in checking 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 SA and SB bits are used to indicate when an operation has generated an overflow out of the Most Significant bit 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 is set to '1', it can only be cleared manually by software, regardless of subsequent DSP operations. When required, it is recommended that the bits be cleared with the BCLR instruction.

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, which utilizes both accumulators. Instead of interrogating the OA and the OB bits independently for arithmetic overflows, a single check of OAB may 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.

#### 2.2.14.4 Interrupt Priority Level Status Bits

The three IPL bits of the SRL, SR<7:5>, and the IPL3 bit, CORCON<3>, set the CPU's Interrupt Priority Level (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 ServiceRoutine 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:** Refer to the dsPIC30F Family Reference Manual for complete details on exception processing.

#### 2.2.15 Core Control Register

The 16-bit CPU Core Control Register (CORCON), shown in Register 2-2, is used to set the configuration of the dsPIC30F CPU. This register provides the ability to:

- map program space into data space
- 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

On device RESET, the CORCON is set to 0x0020, which sets the following mode:

- Program Space not Mapped to Data Space (**PSV** = 0)
- ACCA and ACCB Saturation Disabled (SATA = 0, SATB = 0)
- Data Space Write Saturation Enabled (SATDW = 1)
- Accumulator Saturation mode set to normal (ACCSAT = 0)
- Accumulator Rounding mode set to unbiased (RND = 0)
- DSP Multiplier mode set to signed fractional (**US** = 0, **IF** = 0)

In addition to setting CPU modes, the CORCON contains status information about the DO loop nesting level (**DL**<2:0>) and the **IPL**<3> status bit, which indicates if a trap exception is being processed.

#### 2.2.16 Shadow Registers

A shadow register is used as a temporary holding register and can transfer its contents to or from the associated host register upon some event. 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.

Location	DO	POP.S/PUSH.S
DCOUNT	Yes	—
DOSTART	Yes	—
DOEND	Yes	—
Status Register - DC, N, OV, Z and C bits	_	Yes
W0 - W3	—	Yes

#### Table 2-2: Automatic Shadow Register Usage

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 in the CORCON, CORCON<10:8>.

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

# dsPIC30F Programmer's Reference Manual

	te (SRH):			_		_			
R-0 OA	R-0 OB	R/C-0 SA	R/C-0 SB	R-0 OAB	R/C-0 SAB	R-0	R/V		
DA	UB	54	30	UAB	SAD	DA		bit 8	
								bit 0	
	Low	Byte (SRL):							
	R/V	V-0 R/W	-0 R/W-	0 R	-0	R/W-0	R/W-0	R/W-0	R/W-0
		IPL<2	2:0>	R	A	Ν	OV	Z	С
	bit 7								bit C
oit 15	1 = Accumul	lator A Overflo lator A overflov lator A has not	ved						
bit 14	1 = Accumul	lator B Overflo lator B overflov lator B has not	ved						
bit 13	1 = Accumul	ator A Saturati lator A is satur lator A is not s	ated or has be	en saturate	ed at some	e time			
		his bit may be Ince this bit is s				software.			
oit 12	<b>SB:</b> Accumul 1 = Accumul	ator B Saturati lator B is satur lator B is not s	on bit ated or has be						
		his bit may be Ince this bit is s				software.			
bit 11	<b>OAB:</b> OA    C 1 = Accumul	B Combined A lators A or B h Accumulators A	Accumulator C ave overflowe	Overflow bit d					
bit 10	1 = Accumul	B Combined A lators A or B a Accumulators /	re saturated o	r have beer	n saturated	d at some	time in the	e past	
	<b>2:</b> O	his bit may be ince this bit is s learing this bit	set, it must be	cleared ma		software.			
bit 9	<b>DA:</b> DO Loop 1 = DO loop 0 = DO loop		s						
	Note: Th	nis bit is read c	only.						
bit 8	1 = A carry-o	U Half Carry b out from the M -out from the N	ost Significant						
bit 7-5	111 = CPU Ir 110 = CPU Ir 101 = CPU Ir 100 = CPU Ir 011 = CPU Ir	errupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority nterrupt Priority	<ul> <li>/ Level is 7 (14)</li> <li>/ Level is 6 (14)</li> <li>/ Level is 5 (13)</li> <li>/ Level is 4 (13)</li> <li>/ Level is 3 (11)</li> <li>/ Level is 3 (11)</li> <li>/ Level is 2 (10)</li> </ul>	4) 3) 2) 1) 0)	errupts dis	abled.			

Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL<3> = 1.

#### Register 2-1: SR, Status Register (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 OV: MCU ALU Overflow bit bit 2 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 Note: Refer to Section 4.9 "Z Status Bit" for operation with ADDC, CPB, SUBB and SUBBR instructions. C: MCU ALU Carry/Borrow bit bit 0 1 = A carry-out from the Most Significant bit occurred 0 = No carry-out from the Most Significant bit occurred

Legend:			
R = Readable bit	W = Writable bit	C = Clearable bit	
-n = Value at POR	1 = bit is set	0 = bit is cleared	

Programmer's Model

2

© 2005 Microchip Technology Inc.

# dsPIC30F Programmer's Reference Manual

High Byt	e:										
U		U	U	R/W-0		)/W-0	R-0	R-		/W-0	
—		_	—	US	E	DT		DL<	2:0>		
bit 15										bit 8	
		Low Byte	):								
		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	SATE	B SA	TDW	ACCSA	ΛT	IPL3	PSV	RND	IF
		bit 7									bit C
bit 15-13	Linueo	d									
bit 12		nsigned or s	Signed Mu	ltinlier Moc	le Selec	t hit					
Dit 12	1 = U	Insigned mode	ode enable	d for DSP	multiply	operatior	าร				
bit 11	EDT: E	Early DO Lo	op Termina	ation Contr	ol bit						
		erminate ex lo effect	ecuting DC	D loop at e	nd of cu	rrent itera	ation				
	Not		it will alway	/s read '0'.							
bit 10-8		0>: DO Loc	-		us bits						
	111 =	DO looping	is nested	at 7 levels							
		DO looping									
		DO looping DO looping									
		DO looping									
		DO looping									
		DO looping DO looping			sted (jus	t 1 level)					
	Note	e 1: DL<2:	:1> are rea rst two leve	•	on nest	ina are h	andloc	l by bardy	Naro		
bit 7	SATA:	: ACCA Sat			Jop nesi	ing are n	andiec	a by nard	ware.		
Sit /	-	ccumulator									
	0 = A	ccumulator	A saturation	on disabled	ł						
bit 6		: ACCB Sat									
		ccumulator									
bit 5		W: Data Spa				Saturatior	n Enab	le bit			
		ata space v									
		ata space v									
bit 4		AT: Accumi				t bit					
		.31 saturatio .31 saturatio									
bit 3		Interrupt Pri	-								
	1 = C	PU Interrup	ot Priority L	evel is 8 o	r greater						
	0 = C	PU Interrup	ot Priority L	evel is 7 o	r less (n	o trap exe	ceptior	n activate	d)		
	Not	e 1: This b 2: This b					s (SR<	:7:5>) to f	orm the Cl	PU Interrupt F	riority Level
								,			
bit 2	PSV: F	Program Sp	ace Visibili	ty in Data							-
bit 2	1= P	Program Sp rogram spa rogram spa	ce visible i	n data spa	Space E ce						-

#### 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

Legend:				
R = Readable bit	W = Writable bit	C = Clearable bit	x = bit is unknown	
-n = Value at POR	1 = bit is set	0 = bit is cleared	U = Unimplemented bit,	
			read as '0'	

NOTES:



## Section 3. Instruction Set Overview

### HIGHLIGHTS

This section of the manual contains the following major topics:

3.1	Introduction	. 3-2
3.2	Instruction Set Overview	. 3-2
3.3	Instruction Set Summary Tables	. 3-3

#### 3.1 Introduction

The dsPIC30F instruction set provides a broad suite of instructions, which supports traditional microcontroller applications and a class of instructions, which supports math intensive applications. Since almost all of the functionality of the PICmicro<sup>®</sup> MCU instruction set has been maintained, this hybrid instruction set allows a friendly DSP migration path for users already familiar with the PICmicro microcontroller.

#### 3.2 Instruction Set Overview

The dsPIC30F instruction set contains 84 instructions, which can be grouped into the ten 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 #
Move Instructions	Table 3-2	3-3
Math Instructions	Table 3-3	3-4
Logic Instructions	Table 3-4	3-5
Rotate/Shift Instructions	Table 3-5	3-6
Bit Instructions	Table 3-6	3-7
Compare/Skip Instructions	Table 3-7	3-8
Program Flow Instructions	Table 3-8	3-9
Shadow/Stack Instructions	Table 3-9	3-10
Control Instructions	Table 3-10	3-10
DSP Instructions	Table 3-11	3-10

Table 3-1:dsPIC30F Instruction Groups

Most instructions have several different Addressing modes and execution flows, which require different instruction variants. For instance, there are six unique ADD instructions and each instruction variant has its own instruction encoding. Instruction format descriptions and specific instruction operation are provided in **Section 3. "Instruction Set Overview**". Additionally, a composite alphabetized instruction set table is provided in **Section 6. "Reference**".

#### 3.2.1 Multi-Cycle Instructions

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

- 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. SKIP instructions, which skip over a 2-word instruction, require 3 instruction cycles to execute, with 2 cycles executed as a NOP.
- 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:** Instructions which access program memory as data, using Program Space Visibility, will incur a one or two cycle delay. However, 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 dsPIC30F Family Reference Manual for details.

#### 3.2.2 Multi-Word Instructions

As defined by **Subsection Table 3-2: "Move Instructions**", 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 Instruct	tions			
Assembly	v Syntax	Description	Words	Cycles	Page #
EXCH	Wns,Wnd	Swap Wns and Wnd	1	1	5-115
MOV	f {,WREG} <sup>(see Note)</sup>	Move f to destination	1	1	5-145
MOV	WREG,f	Move WREG to f	1	1	5-146
MOV	f,Wnd	Move f to Wnd	1	1	5-147
MOV	Wns,f	Move Wns to f	1	1	5-148
MOV.B	#lit8,Wnd	Move 8-bit literal to Wnd	1	1	5-149
MOV	#lit16,Wnd	Move 16-bit literal to Wnd	1	1	5-150
MOV	[Ws+Slit10],Wnd	Move [Ws + signed 10-bit offset] to Wnd	1	1	5-151
MOV	Wns,[Wd+Slit10]	Move Wns to [Wd + signed 10-bit offset]	1	1	5-152
MOV	Ws,Wd	Move Ws to Wd	1	1	5-153
MOV.D	Ws,Wnd	Move double Ws to Wnd:Wnd+1	1	2	5-155
MOV.D	Wns,Wd	Move double Wns:Wns+1 to Wd	1	2	5-157
SWAP	Wn	Wn = byte or nibble swap Wn	1	1	5-249
TBLRDH	Ws,Wd	Read high program word to Wd	1	2	5-250
TBLRDL	Ws,Wd	Read low program word to Wd	1	2	5-252
TBLWTH	Ws,Wd	Write Ws to high program word	1	2	5-254
TBLWTL	Ws,Wd	Write Ws to low program word	1	2	5-256
		(MDEO)			

Table 3-2: Move Instructions

**Note:** 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.

	able 3-3: Math Instructions								
Assembly	y Syntax	Description	Words	Cycles	Page #				
ADD	f {,WREG} <sup>(1)</sup>	Destination = f + WREG	1	1	5-7				
ADD	#lit10,Wn	Wn = lit10 + Wn	1	1	5-8				
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	1	1	5-9				
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1	5-10				
ADDC	f {,WREG} <sup>(1)</sup>	Destination = $f + WREG + (C)$	1	1	5-14				
ADDC	#lit10,Wn	Wn = lit10 + Wn + (C)	1	1	5-15				
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	1	1	5-16				
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1	5-17				
DAW.B	Wn	Wn = decimal adjust Wn	1	1	5-95				
DEC	f {,WREG} <sup>(1)</sup>	Destination = $f - 1$	1	1	5-96				
DEC	Ws,Wd	Wd = Ws - 1	1	1	5-97				
DEC2	f {,WREG} <sup>(1)</sup>	Destination = $f - 2$	1	1	5-98				
DEC2	Ws,Wd	Wd = Ws - 2	1	1	5-99				
DIV.S	Wm, Wn	Signed 16/16-bit integer divide	1	18 <sup>(2)</sup>	5-101				
DIV.SD	Wm, Wn	Signed 32/16-bit integer divide	1	18 <sup>(2)</sup>	5-101				
DIV.U	Wm, Wn	Unsigned 16/16-bit integer divide	1	18 <sup>(2)</sup>	5-103				
DIV.UD	Wm, Wn	Unsigned 32/16-bit integer divide	1	18 <sup>(2)</sup>	5-103				
DIVF	Wm, Wn	Signed 16/16-bit fractional divide	1	18 <sup>(2)</sup>	5-105				
INC	f {,WREG} <sup>(1)</sup>	Destination = $f + 1$	1	1	5-124				
INC	Ws,Wd	Wd = Ws + 1	1	1	5-125				
INC2	f {,WREG} <sup>(1)</sup>	Destination = f + 2	1	1	5-126				
INC2	Ws,Wd	Wd = Ws + 2	1	1	5-127				
MUL	f	W3:W2 = f * WREG	1	1	5-169				
MUL.SS	Wb,Ws,Wnd	{Wnd+1,Wnd} = sign(Wb) * sign(Ws)	1	1	5-170				
MUL.SU	Wb,#lit5,Wnd	{Wnd+1,Wnd} = sign(Wb) * unsign(lit5)	1	1	5-172				
MUL.SU	Wb,Ws,Wnd	{Wnd+1,Wnd} = sign(Wb) * unsign(Ws)	1	1	5-174				
MUL.US	Wb,Ws,Wnd	{Wnd+1,Wnd} = unsign(Wb) * sign(Ws)	1	1	5-176				
MUL.UU	Wb,#lit5,Wnd	{Wnd+1,Wnd} = unsign(Wb) * unsign(lit5)	1	1	5-178				
MUL.UU	Wb,Ws,Wnd	{Wnd+1,Wnd} = unsign(Wb) * unsign(Ws)	1	1	5-179				
SE	Ws,Wnd	Wnd = sign-extended Ws	1	1	5-220				
SUB	f {,WREG} <sup>(1)</sup>	Destination = f – WREG	1	1	5-230				
SUB	#lit10,Wn	Wn = Wn - lit10	1	1	5-231				
SUB	Wb,#lit5,Wd	Wd = Wb - lit5	1	1	5-232				
SUB	Wb,Ws,Wd	Wd = Wb – Ws	1	1	5-233				
SUBB	f {,WREG} <sup>(1)</sup>	Destination = f – WREG – $(\overline{C})$	1	1	5-236				
SUBB	#lit10,Wn	$Wn = Wn - lit10 - (\overline{C})$	1	1	5-237				
SUBB	Wb,#lit5,Wd	$Wd = Wb - lit5 - (\overline{C})$	1	1	5-238				
SUBB	Wb,Ws,Wd	$Wd = Wb - Ws - (\overline{C})$	1	1	5-239				
SUBBR	f {,WREG} <sup>(1)</sup>	Destination = WREG – f – $(\overline{C})$	1	1	5-241				
SUBBR	Wb,#lit5,Wd	$Wd = lit5 - Wb - (\overline{C})$	1	1	5-242				
SUBBR	Wb,Ws,Wd	$Wd = Ws - Wb - (\overline{C})$	1	1	5-243				
SUBR	f {,WREG} <sup>(1)</sup>	Destination = WREG – f	1	1	5-245				
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	1	1	5-246				
SUBR	Wb,Ws,Wd	Wd = Ws – Wb	1	1	5-247				
ZE	Ws,Wnd	Wnd = zero-extended Ws	1	1	5-264				

### 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.

Assem	oly Syntax	Description	Words	Cycles	Page #
AND	f {,WREG} <sup>(see Note)</sup>	Destination = f .AND. WREG	1	1	5-19
AND	#lit10,Wn	Wn = lit10 .AND. Wn	1	1	5-20
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	1	1	5-21
AND	Wb,Ws,Wd	Wd = Wb .AND. Ws	1	1	5-22
CLR	f	$\mathbf{f} = 0 \mathbf{x} 0 0 0 0$	1	1	5-75
CLR	WREG	WREG = 0x0000	1	1	5-75
CLR	Wd	$Wd = 0 \times 0000$	1	1	5-76
СОМ	f {,WREG} <sup>(see Note)</sup>	Destination = $\overline{f}$	1	1	5-80
СОМ	Ws,Wd	$Wd = \overline{Ws}$	1	1	5-81
IOR	f {,WREG} <b>(see Note)</b>	Destination = f .IOR. WREG	1	1	5-128
IOR	#lit10,Wn	Wn = lit10 .IOR. Wn	1	1	5-129
IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	1	1	5-130
IOR	Wb,Ws,Wd	Wd = Wb .IOR. Ws	1	1	5-131
NEG	f {,WREG} <sup>(see Note)</sup>	Destination = $\overline{f}$ + 1	1	1	5-181
NEG	Ws,Wd	$Wd = \overline{Ws} + 1$	1	1	5-182
SETM	f	$f = 0 \times FFFF$	1	1	5-221
SETM	WREG	WREG = 0xfff	1	1	5-221
SETM	Wd	Wd = 0xFFFF	1	1	5-222
XOR	f {,WREG} <sup>(see Note)</sup>	Destination = f .XOR. WREG	1	1	5-259
XOR	#lit10,Wn	Wn = lit10 .XOR. Wn	1	1	5-260
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1	5-261
XOR	Wb,Ws,Wd	Wd = Wb .XOR. Ws	1	1	5-262

Table 3-4: Logic Instructions

**Note:** 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.

Assembly SyntaxDescriptionWordsCyclesASRf {,WREG}(see Note)Destination = arithmetic right shift f11ASRWs,WdWd = arithmetic right shift Ws11ASRWb,#lit4,WndWnd = arithmetic right shift Wb by lit411ASRWb,#lit4,WndWnd = arithmetic right shift Wb by Wns11ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11LSRf {,WREG}(see Note)Destination = logical right shift f11LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,#lit4,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws111RRCf {,WREG}(see Note)Destination = rotate right through Carry f11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11 <th colspan="9">Image: Break of the second second</th>	Image: Break of the second								
ASRWs,WdWd = arithmetic right shift Ws11ASRWb,#lit4,WndWnd = arithmetic right shift Wb by lit411ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11LSRf {,WREG}(see Note)Destination = logical right shift f11LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,#lit4,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RRCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-24								
ASRWb,#lit4,WndWnd = arithmetic right shift Wb by lit411ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11LSRf {,WREG}(see Note)Destination = logical right shift f11LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RRCf {,WREG}(see Note)Destination = rotate left (no Carry) f11									
ASRWb,Wns,WndWnd = arithmetic right shift Wb by Wns11LSRf {,WREG}(see Note)Destination = logical right shift f11LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-25								
LSRf {,WREG}(see Note)Destination = logical right shift f11LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCWs,WdWd = rotate left through Carry Ws11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-27								
LSRWs,WdWd = logical right shift Ws11LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCWs,WdWd = rotate left through Carry Ws11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-28								
LSRWb,#lit4,WndWnd = logical right shift Wb by lit411LSRWb,Wns,WndWnd = logical right shift Wb by Wns11LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCWs,WdWd = rotate left through Carry Ws11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-136								
LSRWb,Wns,WndWnd = logical right shift Wb by Wns11RLCf {,WREG}(see Note)Destination = rotate left through Carry f11RLCWs,WdWd = rotate left through Carry Ws11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) Ws11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-137								
RLC       f {,WREG}(see Note)       Destination = rotate left through Carry f       1       1         RLC       Ws,Wd       Wd = rotate left through Carry Ws       1       1         RLC       Ws,Wd       Wd = rotate left through Carry Ws       1       1         RLNC       f {,WREG}(see Note)       Destination = rotate left (no Carry) f       1       1         RLNC       Ws,Wd       Wd = rotate left (no Carry) Ws       1       1         RRC       f {,WREG}(see Note)       Destination = rotate right through Carry f       1       1	5-139								
RLCWs,WdWd = rotate left through Carry Ws11RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-140								
RLNCf {,WREG}(see Note)Destination = rotate left (no Carry) f11RLNCWs,WdWd = rotate left (no Carry) Ws11RRCf {,WREG}(see Note)Destination = rotate right through Carry f11	5-204								
RLNC     Ws,Wd     Wd = rotate left (no Carry) Ws     1     1       RRC     f {,WREG} <sup>(see Note)</sup> Destination = rotate right through Carry f     1     1	5-205								
RRC     f {,WREG} <sup>(see Note)</sup> Destination = rotate right through Carry f     1     1	5-207								
	5-208								
RRC We Wid Wid - rotate right through Carry We 1 1	5-210								
	5-211								
RRNC         f {,WREG}         (see Note)         Destination = rotate right (no Carry) f         1         1	5-213								
RRNCWs,WdWd = rotate right (no Carry) Ws11	5-214								
SL     f {,WREG} <sup>(see Note)</sup> Destination = left shift f     1     1	5-225								
SL         Ws,Wd         Wd = left shift Ws         1         1	5-226								
SL         Wb,#lit4,Wnd         Wnd = left shift Wb by lit4         1         1	5-228								
SL         Wb,Wns,Wnd         Wnd = left shift Wb by Wns         1         1	5-229								

#### Table 3-5: Rotate/Shift Instructions

**Note:** 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.

### **Section 3. Instruction Set Overview**

Assembly	Syntax	Description	Words	Cycles	Page #
	-			-	
BCLR	f,#bit4	Bit clear f	1	1	5-29
BCLR	Ws,#bit4	Bit clear Ws	1	1	5-30
BSET	f,#bit4	Bit set f	1	1	5-54
BSET	Ws,#bit4	Bit set Ws	1	1	5-55
BSW.C	Ws,Wb	Write C bit to Ws <wb></wb>	1	1	5-56
BSW.Z	Ws,Wb	Write $\overline{Z}$ bit to Ws <wb></wb>	1	1	5-56
BTG	f,#bit4	Bit toggle f	1	1	5-58
BTG	Ws,#bit4	Bit toggle Ws	1	1	5-59
BTST	f,#bit4	Bit test f	1	1	5-67
BTST.C	Ws,#bit4	Bit test Ws to C	1	1	5-68
BTST.Z	Ws,#bit4	Bit test Ws to Z	1	1	5-68
BTST.C	Ws,Wb	Bit test Ws <wb> to C</wb>	1	1	5-69
BTST.Z	Ws,Wb	Bit test Ws <wb> to Z</wb>	1	1	5-69
BTSTS	f,#bit4	Bit test f then set f	1	1	5-71
BTSTS.C	Ws,#bit4	Bit test Ws to C then set Ws	1	1	5-72
BTSTS.Z	Ws,#bit4	Bit test Ws to Z then set Ws	1	1	5-72
FBCL	Ws,Wnd	Find bit change from left (MSb) side	1	1	5-116
FF1L	Ws,Wnd	Find first one from left (MSb) side	1	1	5-118
FF1R	Ws,Wnd	Find first one from right (LSb) side	1	1	5-120

Table 2-6: Bit Instruction

				e (see Note)	
Assembl	y Syntax	Description	Words	Cycles <sup>(see Note)</sup>	Page #
BTSC	f,#bit4	Bit test f, skip if clear	1	1 (2 or 3)	5-60
BTSC	Ws,#bit4	Bit test Ws, skip if clear	1	1 (2 or 3)	5-62
BTSS	f,#bit4	Bit test f, skip if set	1	1 (2 or 3)	5-64
BTSS	Ws,#bit4	Bit test Ws, skip if set	1	1 (2 or 3)	5-65
CP	f	Compare (f – WREG)	1	1	5-82
CP	Wb,#lit5	Compare (Wb – lit5)	1	1	5-83
CP	Wb,Ws	Compare (Wb – Ws)	1	1	5-84
CP0	f	Compare (f - 0x0000)	1	1	5-85
CP0	Ws	Compare (Ws - 0x0000)	1	1	5-86
CPB	f	Compare with Borrow (f – WREG – $\overline{C}$ )	1	1	5-87
CPB	Wb,#lit5	Compare with Borrow (Wb – lit5 – $\overline{C}$ )	1	1	5-88
CPB	Wb,Ws	Compare with Borrow (Wb – Ws – $\overline{C}$ )	1	1	5-89
CPSEQ	Wb, Wn	Compare (Wb – Wn), skip if =	1	1 (2 or 3)	5-91
CPSGT	Wb, Wn	Compare (Wb – Wn), skip if >	1	1 (2 or 3)	5-92
CPSLT	Wb, Wn	Compare (Wb – Wn), skip if <	1	1 (2 or 3)	5-93
CPSNE	Wb, Wn	Compare (Wb – Wn), skip if $\neq$	1	1 (2 or 3)	5-94

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

**Note:** 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.

Table 3-8:	. regram r	low Instructions			
Assembly	Syntax	Description	Words	Cycles	Page #
BRA	Expr	Branch unconditionally	1	2	5-31
BRA	Wn	Computed branch	1	2	5-32
BRA	C,Expr	Branch if Carry (no Borrow)	1	1 (2) <sup>(1)</sup>	5-33
BRA	GE,Expr	Branch if greater than or equal	1	1 (2) <sup>(1)</sup>	5-35
BRA	GEU,Expr	Branch if unsigned greater than or equal	1	1 (2) <sup>(1)</sup>	5-33
BRA	GT,Expr	Branch if greater than	1	1 (2) <sup>(1)</sup>	5-37
BRA	GTU,Expr	Branch if unsigned greater than	1	1 (2) <sup>(1)</sup>	5-38
BRA	LE,Expr	Branch if less than or equal	1	1 (2) <sup>(1)</sup>	5-39
BRA	LEU,Expr	Branch if unsigned less than or equal	1	1 (2) <sup>(1)</sup>	5-40
BRA	LT,Expr	Branch if less than	1	1 (2) <sup>(1)</sup>	5-41
BRA	LTU,Expr	Branch if unsigned less than	1	1 (2) <sup>(1)</sup>	5-44
BRA	N,Expr	Branch if Negative	1	1 (2) <sup>(1)</sup>	5-43
BRA	NC,Expr	Branch if not Carry (Borrow)	1	1 (2) <sup>(1)</sup>	5-44
BRA	NN,Expr	Branch if not Negative	1	1 (2) <sup>(1)</sup>	5-45
BRA	NOV,Expr	Branch if not Overflow	1	1 (2) <sup>(1)</sup>	5-46
BRA	NZ,Expr	Branch if not Zero	1	1 (2) <sup>(1)</sup>	5-47
BRA	OA,Expr	Branch if Accumulator A Overflow	1	1 (2) <sup>(1)</sup>	5-48
BRA	OB,Expr	Branch if Accumulator B Overflow	1	1 (2) <sup>(1)</sup>	5-49
BRA	OV,Expr	Branch if Overflow	1	1 (2) <sup>(1)</sup>	5-50
BRA	SA,Expr	Branch if Accumulator A Saturate	1	1 (2) <sup>(1)</sup>	5-51
BRA	SB,Expr	Branch if Accumulator B Saturate	1	1 (2) <sup>(1)</sup>	5-52
BRA	Z,Expr	Branch if Zero	1	1 (2) <sup>(1)</sup>	5-53
CALL	Expr	Call subroutine	2	2	5-73
CALL	Wn	Call indirect subroutine	1	2	5-74
DO	#lit14,Expr	Do code through PC+Expr, (lit14+1) times	2	2	5-107
DO	Wn,Expr	Do code through PC+Expr, (Wn+1) times	2	2	5-109
GOTO	Expr	Go to address	2	2	5-122
GOTO	Wn	Go to address indirectly	1	2	5-123
RCALL	Expr	Relative call	1	2	5-196
RCALL	Wn	Computed call	1	2	5-196
REPEAT	#lit14	Repeat next instruction (lit14+1) times	1	1	5-197
REPEAT	Wn	Repeat next instruction (Wn+1) times	1	1	5-198
RETFIE		Return from interrupt enable	1	3 (2) <sup>(2)</sup>	5-201
RETLW	#lit10,Wn	Return with lit10 in Wn	1	3 (2) <sup>(2)</sup>	5-202
RETURN		Return from subroutine	1	3 (2) <sup>(2)</sup>	5-203

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.

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

Assembly	y Syntax	Description	Words	Cycles	Page #
LNK	#lit14	Link frame pointer	1	1	5-135
POP	f	Pop TOS to f	1	1	5-186
POP	Wd	Pop TOS to Wd	1	1	5-187
POP.D	Wnd	Double pop from TOS to Wnd:Wnd+1	1	2	5-188
POP.S		Pop shadow registers	1	1	5-189
PUSH	f	Push f to TOS	1	1	5-190
PUSH	Ws	Push Ws to TOS	1	1	5-191
PUSH.D	Wns	Push double Wns:Wns+1 to TOS	1	2	5-192
PUSH.S		Push shadow registers	1	1	5-193
ULNK		Unlink frame pointer	1	1	5-258

#### Table 3-10: Control Instructions

Assembly	Syntax	Description	Words	Cycles	Page #
CLRWDT		Clear Watchdog Timer	1	1	5-79
DISI	#lit14	Disable interrupts for (lit14+1) instruction cycles	1	1	5-100
NOP		No operation	1	1	5-184
NOPR		No operation	1	1	5-185
PWRSAV	#lit1	Enter Power Saving mode lit1	1	1	5-194
RESET		Software device RESET	1	1	5-200

#### Table 3-11: DSP Instructions

Assembly	Syntax	Description	Words	Cycles	Page #
ADD	Acc	Add accumulators	1	1	5-11
ADD	Ws,#Slit4,Acc	16-bit signed add to Acc	1	1	5-12
CLR	Acc,Wx,Wxd,Wy,Wyd,AWB	Clear Acc	1	1	5-77
ED	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance (no accumulate)	1	1	5-111
EDAC	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance	1	1	5-113
LAC	Ws,#Slit4,Acc	Load Acc	1	1	5-133
MAC	Wm*Wn,Acc,Wx,Wxd,Wy, Wyd,AWB	Multiply and accumulate	1	1	5-141
MAC	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square and accumulate	1	1	5-143
MOVSAC	Acc,Wx,Wxd,Wy,Wyd,AWB	Move Wx to Wxd and Wy to Wyd	1	1	5-159
MPY	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	Multiply Wn by Wm to Acc	1	1	5-161
MPY	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square to Acc	1	1	5-163
MPY.N	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	-(Multiply Wn by Wm) to Acc	1	1	5-165
MSC	Wm*Wn,Acc,Wx,Wxd,Wy, Wyd,AWB	Multiply and subtract from Acc	1	1	5-167
NEG	Acc	Negate Acc	1	1	5-183
SAC	Acc,#Slit4,Wd	Store Acc	1	1	5-216
SAC.R	Acc,#Slit4,Wd	Store rounded Acc	1	1	5-218
SFTAC	Acc,#Slit6	Arithmetic shift Acc by Slit6	1	1	5-223
SFTAC	Acc,Wn	Arithmetic shift Acc by (Wn)	1	1	5-224
SUB	Acc	Subtract accumulators	1	1	5-235



### **Section 4. Instruction Set Details**

### HIGHLIGHTS

This section of the manual contains the following major topics:

4.1	Data Addressing Modes	4-2
4.2	Program Addressing Modes	4-11
4.3	Instruction Stalls	4-12
4.4	Byte Operations	4-13
4.5	Word Move Operations	4-16
4.6	Using 10-bit Literal Operands	4-19
4.7	Software Stack Pointer and Frame Pointer	4-20
4.8	Conditional Branch Instructions	4-25
4.9	Z Status Bit	4-26
4.10	Assigned Working Register Usage	4-27
4.11	DSP Data Formats	4-30
4.12	Accumulator Usage	4-32
4.13	Accumulator Access	4-33
4.14	DSP MAC Instructions	4-33
4.15	DSP Accumulator Instructions	4-37
4.16	Scaling Data with the FBCL Instruction	4-37
4.17	Normalizing the Accumulator with the FBCL Instruction	4-39

#### 4.1 Data Addressing Modes

The dsPIC30F supports three native Addressing modes for accessing data memory, along with several forms of immediate addressing. Data accesses may be performed using file register, register direct or register indirect addressing, and immediate addressing allows 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, 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.

Table 4-1: dsPIC30F Addressing Modes

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

Note: 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 one to load data from any location in data memory to any working register, and store 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 2.2.2 "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 WREG are specified in the instruction, the operation results are stored in WREG and the contents of the file register are unchanged. Sample instructions which show the interaction between the file register and WREG 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 = 0x1234
  Data Memory 0x27FE = 0x1234
```

```
Example 4-2: File Register Addressing and WREG
```

```
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
```

Instruction Set Details

# 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 which 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, so 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, one may generate flexible looping constructs 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".

Example 4-3: Register Direct Addressing

EXCH W2 Before Instr W2 = 0x3 W3 = 0x0	ruction:	Exchange W2 and W3
After Instruct W2 = 0x0 W3 = 0x3	103D	
Before Instr W0 = 0x9	ruction:	Inclusive-OR 0x44 and W0
After Instruc W0 = 0x9 SL W6	C6E	Shift left W6 by W7, and store to W8
Before Instr W6 = 0x0 W7 = 0x0 W8 = 0x1	ruction: 000C 0008	Shirt left wo by w/, and store to wo
After Instruct W6 = 0x0 W7 = 0x0 W8 = 0x0	000C 008	

#### 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 dsPIC30F are shown in Table 4-2.

	an oot Aut	liessing modes		
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	[++Wn]	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 he 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, that 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.

The MOV with offset instructions (pages page 151 and page 152) provides a literal Note: 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.

Details

```
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
       W1, [--W5], [++W8] ; pre-dec W5, pre-inc W8
ADD
                              ; 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
```

Example 4-4: Indirect Addressing with Effective Address Update

MOV.B [W0+W1], [W7++] ; byte move [W0+W1] to W7, post-inc W7 Before Instruction: W0 = 0x2300W1 = 0x01FEW7 = 0x1000Data Memory 0x24FE = 0x7783Data Memory 0x1000 = 0x11DC After Instruction: W0 = 0x2300W1 = 0x01FEW7 = 0x1001Data Memory 0x24FE = 0x7783Data Memory 0x1000 = 0x1183LAC [W0+W8], A ; load ACCA with [W0+W8] ; (sign-extend and zero-backfill) Before Instruction: W0 = 0x2344 $W8 = 0 \times 0008$  $ACCA = 0 \times 00 7877 9321$ Data Memory 0x234C = 0xE290After Instruction: W0 = 0x2344 $W8 = 0 \times 0008$ ACCA = 0xFF E290 0000Data Memory 0x234C = 0xE290

Example 4-5: Indirect Addressing with Register Offset

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

```
MOV
        [W0+0x20], W1
                             ; move [W0+0x20] to W1
Before Instruction:
    W0 = 0x1200
    W1 = 0x01FE
    Data Memory 0x1220 = 0xFD27
After Instruction:
    W0 = 0x1200
    W1 = 0xFD27
    Data Memory 0x1220 = 0xFD27
        W4, [W8-0x300]
MOV
                           ; move W4 to [W8-0x300]
Before Instruction:
    W4 = 0x3411
    W8 = 0x2944
    Data Memory 0x2644 = 0xCB98
After Instruction:
    W4 = 0x3411
    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 dsPIC30F. 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, 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

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"**, 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 pre-fetches 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.

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]				

Note: As described in Section 4.14 "DSP MAC Instructions", 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

The dsPIC30F 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 non-linear 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 architecture, which can be exploited by any instruction that uses indirect addressing. Refer to the *dsPIC30F Family Reference Manual* for details on using Modulo and Bit-Reversed addressing.

# 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.

	initial depending in the instruction oct
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, CPB, IOR, MUL.SU, MUL.UU, SUB, SUBB, SUBBR, SUBR, XOR
#Slit6	SFTAC
#lit8	MOV.B
#lit10	ADD, ADDC, AND, CP, CPB, IOR, RETLW, SUB, SUBB, XOR
#Slit10	MOV
#lit14	DISI, DO, LNK, REPEAT
#lit16	MOV

Table 4-4:	Immediate O	perands in t	he Instruction Section	et
	minieulale O			-ι

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, W0	; Add 0x10 to W0 (byte mode)
Before Instruction: W0 = 0x12A9	
After Instruction: W0 = 0x12B9	
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 = 0	)x0032
After Instruction: W0 = 0xFFFF W1 = 0x0892	
Data Memory 0x0890 = (	)xFFFE

# 4.1.5 Data Addressing Mode Tree

The Data Addressing modes of the dsPIC30F are summarized in Figure 4-1.

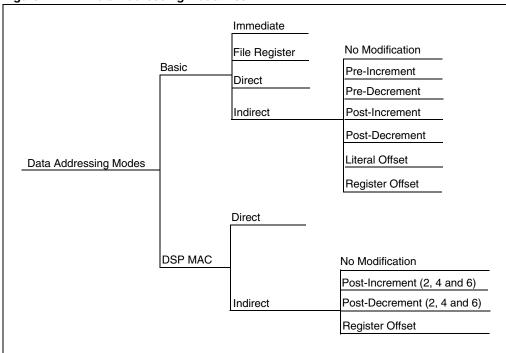


Figure 4-1: Data Addressing Mode Tree

# 4.2 Program Addressing Modes

The dsPIC30F has a 23-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. 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 of the dsPIC30F. 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>
<sub>GOTO Expr</sub> (1) (Unconditional Jump)	PC = lit23	None
GOTO Wn (Unconditional Indirect Jump)	PC = Wn	None
RCALL Expr <sup>(1)</sup> (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 (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 back into the PC.
- 3: After an exception is processed, RETFIE will pop the top-of-stack back into the PC.

# 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 pre-fetched 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 (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] <sup>(2)</sup> MOV.W [W2], W3 <sup>(2)</sup>	
Indirect	Indirect with pre/post-modification	Stall <sup>(1)</sup>	ADD.W W0, W1, [W2] <sup>(2)</sup> MOV.W [W2++], W3 <sup>(2)</sup>	
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 4-0. Raw Dependency Rules (Delection by Rardware	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.

#### 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 dsPIC30F Family Reference Manual for more detailed information about RAW instruction stalls.

## 4.4 Byte Operations

Since the dsPIC30F 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 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 4-2). Example 4-8 shows sample byte move operations and Example 4-9 shows sample byte math operations.

Note:	Instructions which 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 WO
	CLR.B WO

```
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 = 0x1234
  Data Memory 0x1000 = 0x5555
After Instruction:
   W0 = 0x1234
   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 = 0x1002
  Data Memory 0x1000 = 0x3455
```

## Example 4-8: Sample Byte Move Operations

```
Example 4-9: Sample Byte Math Operations
```

```
CLR.B [W6--]
                           ; byte clear [W6], then post-dec W6
Before Instruction:
   W6 = 0x1001
   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 W1
Before Instruction:
   W0 = 0x1234
   W1 = 0xFFFF
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 = 0x1000
  Data Memory 0x1000 = 0x5555
After Instruction:
   W0 = 0x1234
   W1 = 0x5678
   W2 = 0x1001
   Data Memory 0x1000 = 0x55AC
```

Instruction Set Details

# 4.5 Word Move Operations

Even though the dsPIC30F 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 4-2 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: b0 - byte stored at 0x1000 b1 - byte stored at 0x1003 b3:b2 - word stored at 0x1005:1004 (b2 is LSB) b7:b4 - double-word stored at 0x1009:0x1006 (b4 is LSB) b8 - byte stored at 0x100A					

Figure 4-2: Data Alignment in Memory

Note:	extension. How	ch operate in Word mode are not required to use an instruction ever, they may be specified with an optional " $.w$ " or " $.W$ " extension, example, the following instructions are valid forms of a word clear
	CLR	WO
	CLR.w	WO
	CLR.W	WO

```
Example 4-10: Legal Word Move Operations
```

```
MOV
        #0x30, W0
                           ; move the literal word 0x30 to W0
Before Instruction:
   W0 = 0x5555
After Instruction:
   W0 = 0x0030
MOV
        0x1000, W0
                          ; move the word at 0x1000 to W0
Before Instruction:
  W0 = 0x5555
  Data Memory 0x1000 = 0x1234
After Instruction:
  W0 = 0x1234
  Data Memory 0x1000 = 0x1234
MOV
        [WO], [W1++]
                          ; word move [W0] to [W1],
                           ; 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 = 0x1002
  Data Memory 0x1000 = 0xAAAA
  Data Memory 0x1234 = 0xAAAA
```

```
MOV
        0x1001, W0
                         ; move the word at 0x1001 to W0
Before Instruction:
 W0 = 0x5555
 Data Memory 0x1000 = 0x1234
 Data Memory 0x1002 = 0x5678
After Instruction:
 W0 = 0x1234
 Data Memory 0x1000 = 0x1234
 Data Memory 0x1002 = 0x5678
 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 = 0x1234
 Data Memory 0x1000 = 0x5555
 Data Memory 0x1002 = 0x6666
After Instruction:
 W0 = 0x1234
 Data Memory 0x1000 = 0x5555
 Data Memory 0x1002 = 0x6666
 ADDRESS ERROR TRAP GENERATED
 (destination address is misaligned, so MOV is not performed)
MOV
        [WO], [W1++]
                           ; word move [W0] to [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 = 0x1235
 W1 = 0x1002
 Data Memory 0x1000 = 0xAAAA
 Data Memory 0x1234 = 0xAAAA
 Data Memory 0x1236 = 0xBBBB
 ADDRESS ERROR TRAP GENERATED
 (source address is misaligned, so MOV is performed)
```

Example 4-11: Illegal Word Move Operations

# 4.6 Using 10-bit Literal Operands

Several instructions which 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 W0	
ADD.B #0x380, W0 ; ERROR Illegal syntax for byte mo	ode
ADD.B $\#0xFF$ , W0 ; add 255 (or -1) to W0	
ADD.B #0x3FF, W0 ; ERROR Illegal syntax for byte mo	ode
ADD.B #0xF, W0 ; add 15 to W0	
ADD.B #0x7F, W0 ; add 127 to W0	
ADD.B #0x100, W0 ; ERROR Illegal syntax for byte mo	ode

**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 dsPIC30F features 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. This ensures that the SP will point to valid RAM in all dsPIC30F devices 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 W0

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 4-3. 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.

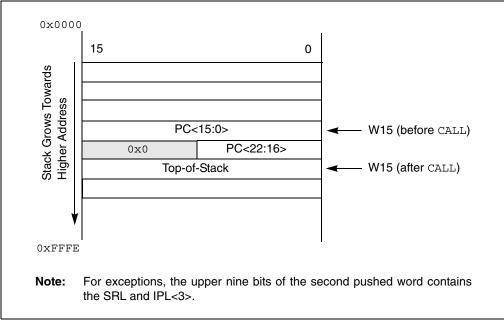
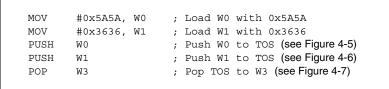


Figure 4-3: Stack Operation for CALL Instruction

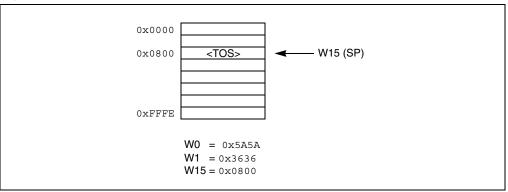
# 4.7.2 Stack Pointer Example

Figure 4-4 through Figure 4-7 show how the software stack is modified for the code snippet shown in Example 4-13. Figure 4-4 shows the software stack before the first PUSH has executed. Note that the SP has the initialized value of  $0 \times 0800$ . Furthermore, the example loads  $0 \times 5A5A$  and  $0 \times 3636$  to W0 and W1, respectively. The stack is pushed for the first time in Figure 4-5 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  $0 \times 0802$ . In Figure 4-6, the contents of W1 are pushed onto the stack, and the new TOS becomes  $0 \times 0804$ . In Figure 4-7, 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  $0 \times 0802$ .

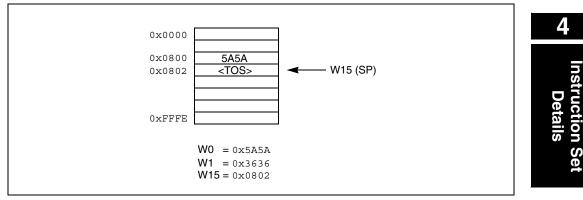
Example 4-13: Stack Pointer Usage











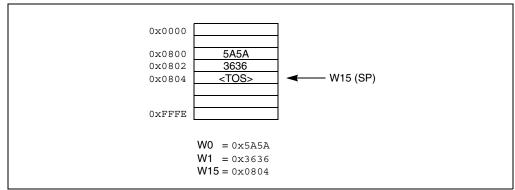
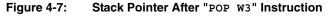
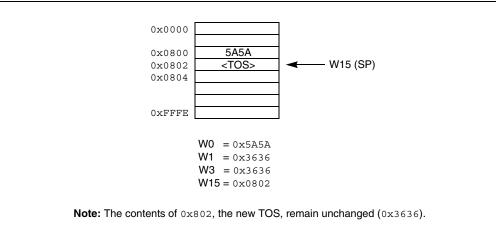


Figure 4-6: Stack Pointer After "PUSH W1" Instruction





# 4.7.3 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  $0 \times 0000$  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.

## 4.7.4 Stack Frame Pointer Example

Figure 4-8 through Figure 4-10 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 dsPIC30F compiler. Figure 4-8 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 4-9). 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 4-10).

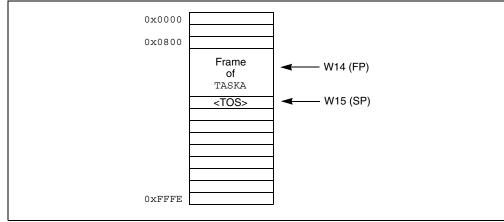
Inside the function "COMPUTE", the FP is used to access the function parameters and temporary (local) variables. [W14+n] will access the temporary variables used by the routine and [W14-n] is used to access the parameters. At the end of the function, the ULNK instruction is used to copy the frame pointer 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 4-9.

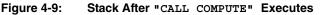
A RETURN instruction will return to the code that called the subroutine. The calling code is responsible for removing the parameters from the stack. The RETURN and POP instructions restore the stack to the state shown in Figure 4-8.

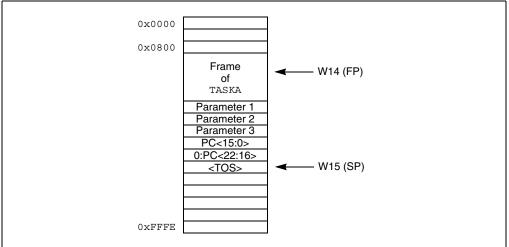
#### Example 4-14: Frame Pointer Usage

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	Wl	;	Pop parameter 2
POP	WO	;	Pop parameter 1
COMPUTE:			
LNK	#4	;	Stack FP, allocate 4 bytes for local variables
ULNK		;	Free allocated memory, restore original FP
RETUR	N	;	Return to TASKA

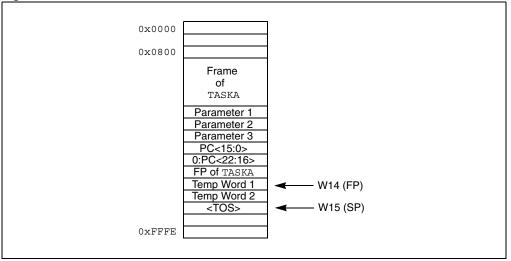












## 4.7.5 Stack Pointer Overflow

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 dsPIC30F Family Reference Manual for more information on the stack error trap.

# 4.7.6 Stack Pointer Underflow

The stack is initialized to  $0 \times 0800$  during RESET. A stack error trap will be initiated should the stack pointer address ever be less than  $0 \times 0800$ .

**Note:** Locations in data space between 0x0000 and 0x07FF are, in general, reserved for core and peripheral special function registers.

# 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, CP0 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 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). 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.

Condition Mnemonic <sup>(1)</sup>	Description	Status Test
С	Carry (not Borrow)	С
GE	Signed greater than or equal	(N&&OV)    (N&&OV)
GEU <sup>(2)</sup>	Unsigned greater than or equal	С
GT	Signed greater than	$(\overline{Z}\&\&\overline{N}\&\&\overline{OV}) \parallel (\overline{Z}\&\&N\&\&OV)$
GTU	Unsigned greater than	C&&Z
LE	Signed less than or equal	Z    (N&&OV)    (N&&OV)
LEU	Unsigned less than or equal	<u>¯</u> C∥Z
LT	Signed less than	(N&&OV)    (N&&OV)
LTU <sup>(3)</sup>	Unsigned less than	C
N	Negative	Ν
NC	Not Carry (Borrow)	ō
NN	Not Negative	N
NOV	Not Overflow	<u>ov</u>
NZ	Not Zero	Z
OA	Accumulator A overflow	OA
ОВ	Accumulator B overflow	OB
OV	Overflow	OV
SA	Accumulator A saturate	SA
SB	Accumulator B saturate	SB
Z	Zero	Z

# 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.

Note: The "Compare and Skip" instructions (CPSEQ, CPSGT, CPSLT, 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 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 MSWord, and the second example generates a zero result for both the LSWord and MSWord.



	result in W5:W4 N2, W4     ; Add LSWord and store to W4
	N3, W5 ; Add MSWord and store to W5
Before 32-bit A	ddition (zero result for MSWord):
W0 = 0x234	
$W1 = 0 \times FFF$	
W2 = 0x39A	
$W3 = 0 \times 001$	.0
$W4 = 0 \times 000$	0
$W5 = 0 \times 000$	0
SR = 0x000	0
After 32-bit Add	dition:
W0 = 0x234	.2
W1 = 0xFFF	0
W2 = 0x39A	A
W3 = 0x001	.0
W4 = 0x5CE	C
$W5 = 0 \times 000$	0
SR = 0x020	1 (DC,C=1)
Before 32-bit A	ddition (zero result for LSWord and MSWord):
W0 = 0xB76	E
W1 = 0xFB7	Έ
W2 = 0x489	2
W3 = 0x048	4
$W4 = 0 \times 000$	
$W5 = 0 \times 000$	
$SR = 0 \times 000$	0
	dition:
After 32-bit Add	· P
After 32-bit Add W0 = 0xB76	
$W0 = 0 \times B76$	'B
$W0 = 0 \times B76$ $W1 = 0 \times FB7$	'В 22
W1 = 0xFB7 W2 = 0x489 W3 = 0x048 W4 = 0x000	7B 22 35 00
W0 = 0xB76 W1 = 0xFB7 W2 = 0x489 W3 = 0x048 W4 = 0x000 W5 = 0x000	7B 22 35 00

#### 4.10 **Assigned Working Register Usage**

The 16 working registers of the dsPIC30F 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 pre-assigned usage. Table 4-9 summarizes these working register assignments, with details provided in subsections Section 4.10.1 "Implied DSP Operands" through Section 4.10.3 "PICmicro<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
W5	MAC Operand
W6	MAC Operand
W7	MAC Operand
W8	MAC Pre-fetch Address (X Memory)
W9	MAC Pre-fetch Address (X Memory)
W10	MAC Pre-fetch Address (Y Memory)
W11	MAC Pre-fetch Address (Y Memory)
W12	MAC Pre-fetch Offset
W13	MAC Write Back Destination
W14	Frame Pointer
W15	Stack Pointer

Table 4-9: **Special Working Register Assignments** 

#### 4.10.1 **Implied DSP Operands**

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 pre-fetch ability, the following 10 register assignments must be adhered to:

- W4-W7 are used for arithmetic operands
- W8-W11 are used for pre-fetch addresses (pointers)
- W12 is used for the pre-fetch 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 pre-fetch ability (described in Section 4.15 "DSP Accumulator Instructions"). The affected instructions are CLR, ED, EDAC, MAC, MOVSAC, MPY, MPY.N and MSC.

The DSP Accumulator class of instructions (described in Section 4.15 "DSP Accumulator Instructions") 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". 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.

Details

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

# 4.10.3.1 Default Working Register WREG

To ease the migration path for users of the Microchip PICmicro families, the dsPIC30F has matched the functionality of the PICmicro instruction sets as closely as possible. One major difference between the dsPIC30F and the PICmicro processors is the number of working registers provided. The PICmicro families only provide one 8-bit working register, while the dsPIC30F provides sixteen, 16-bit working registers. To accommodate for the one working register of the PICmicro MCU, the dsPIC30F 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 dsPIC30F assembler to specify the default working register is similar to that used by the PICmicro 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 which 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
SETM	WREG	; set all bits in the WREG
XOR	RAM100	; XOR RAM100 and WREG, store in RAM100

## 4.10.3.2 PRODH:PRODL Register Pair

Another significant difference between the Microchip PICmicro and dsPIC30F architectures is the multiplier. Some PICmicro families support an 8-bit x 8-bit multiplier, which places the multiply product in the PRODH:PRODL register pair. The dsPIC30F has 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 dsPIC30F still supports the legacy file register multiply instruction (MULWF) with the "MUL{.B} f" instruction (described on page 5-169). 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 to W2
Before Instruction:
  W0 (WREG) = 0x7705
  W2 = 0x1235
  W3 = 0x1000
  Data Memory 0x0100 = 0x1255
After Instruction:
  W0 (WREG) = 0x7705
  W2 = 0x01A9
  W3 = 0 \times 1000
  Data Memory 0x0100 = 0x1255
MUL
        0 \times 100
                    ; (0x100)*WREG (word mode), store to W3:W2
Before Instruction:
  W0 (WREG) = 0x7705
  W2 = 0x1235
  W3 = 0x1000
  Data Memory 0x0100 = 0x1255
After Instruction:
   W0 (WREG) = 0x7705
  W2 = 0xDEA9
  W3 = 0x0885
  Data Memory 0x0100 = 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 5-145) and "MOV{.B} WREG, f" instruction (described on page 5-146) 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 PICmicro 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 0x	1001, WREG	;	move	the	byte	stored	at	location 0x1001 to W0
MOV 0x	1000, WREG	;	move	the	word	stored	at	location 0x1000 to W0
MOV.B WR	EG, TBLPAG	;	move	the	byte	stored	at	W0 to the TBLPAG register
MOV WRI	EG, 0x804	;	move	the	word	stored	at	W0 to location 0x804

Instruction Set Details

# 4.11 DSP Data Formats

# 4.11.1 Integer and Fractional Data

The dsPIC30F 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  $-32768 (0 \times 8000)$  to  $32767 (0 \times 7FFF)$ , including 0. For a 32-bit integer, the data range is  $-2,147,483,648 (0 \times 8000 - 0000)$  to  $2,147,483,647 (0 \times 7FFF)$  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 3.05176x10<sup>-5</sup>. In Normal Saturation mode, the 32-bit accumulators use a 1.31 format, which enhances the precision to 4.6566x10<sup>-10</sup>.

Super Saturation mode expands the dynamic range of the accumulators 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 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 devices for 16-bit, 32-bit and 40-bit registers.

It should be noted that, with the exception of DSP multiplies, the dsPIC30F 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 fractions. 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 devices use 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 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 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 towards 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. This is shown in Figure 4-11 for a positive value and in Figure 4-12 for a negative value.

Converting 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  $2^{N-1}$ . Likewise, to convert an N-bit fraction to an integer, multiply the fractional value by  $2^{N-1}$ .

Figure 4-11: Different Representations of 0x4001

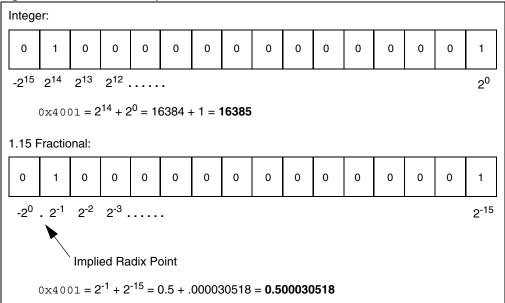


Figure 4-12: Different Representations of 0xC002

Intege	Integer:														
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
-2 <sup>15</sup>	$-2^{15} 2^{14} 2^{13} 2^{12} \dots 2^{0}$											2 <sup>0</sup>			
	0xC0(	02 <b>= -</b> 2	2 <sup>15</sup> + 2	2 <sup>14</sup> +	2 <sup>1</sup> = -3	82768	+ 163	84 + 2	2 = <b>-1</b>	6382					
1.15 I	1.15 Fractional:														
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
-2 <sup>0</sup>	$-2^{0} \cdot 2^{-1} \cdot 2^{-2} \cdot 2^{-3} \cdot \dots \cdot 2^{-15}$										2 <sup>-15</sup>				
	Implied Radix Point														
	$0 \ge 0 \ge -2^0 + 2^{-1} + 2^{-14} = -1.0 + 0.5 + 0.000061035 = -0.499938965$														



# 4.12 Accumulator Usage

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 4-13 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 4-13, 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 4-13) 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, all registers of the accumulator may be used (Item D in Figure 4-13) 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.

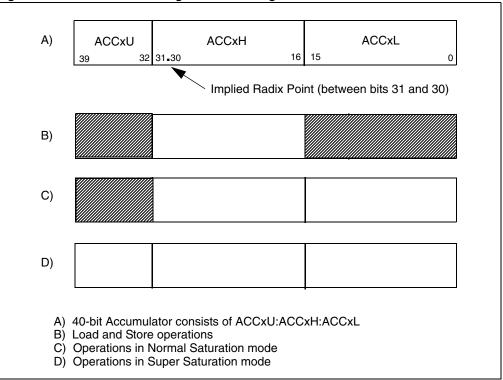


Figure 4-13: Accumulator Alignment and Usage

## 4.13 Accumulator Access

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:** 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).

# 4.14 DSP MAC Instructions

The DSP Multiply and Accumulate (MAC) operations are a special suite of instructions which provide the most efficient use of the dsPIC30F architecture. The DSP MAC instructions, shown in Table 4.14, 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 pre-fetch working registers (MAC Pre-fetches)
- two updates to pre-fetch working registers (MAC Pre-fetch 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 Pre-Fetches

Pre-Fetches (or data reads) are made using the effective address stored in the working register. The two pre-fetches from data memory must be specified using the working registers 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. Allowable destination registers for both pre-fetches 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 pre-fetch 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.

## 4.14.2 MAC Pre-Fetch Register Updates

After the MAC pre-fetches are made, the effective address stored in each pre-fetch 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 pre-fetch 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 pre-fetch register (W9 or W11), or the offset register (W12).

## 4.14.3 MAC Operations

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 pre-fetched from X and Y memory are not actually stored in W4-W7. Instead, only the *difference* of the pre-fetched 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, pre-fetch two values from data memory and store the contents of the other accumulator. Similarly, MOVSAC has the ability to pre-fetch two values from data memory and store the contents of either accumulator.

# 4.14.4 MAC Write Back

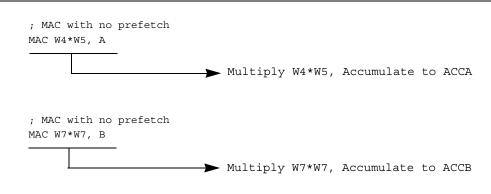
The write back ability of the MAC class of DSP instructions facilitates efficient processing of algorithms. This feature allows one mathematical operation to be performed with one accumulator, and the rounded contents of the other accumulator to be stored in the same cycle. As indicated in Table 4-9, 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 4.14).

# 4.14.5 MAC Syntax

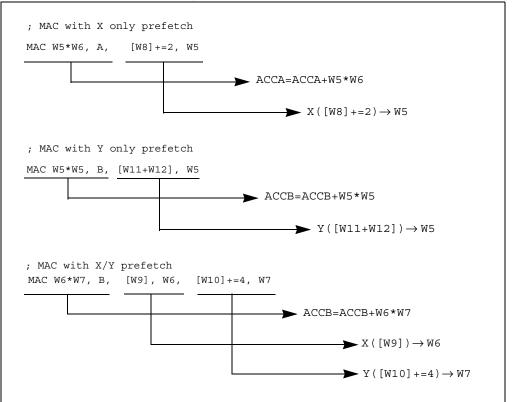
The syntax of the MAC class of instructions can have several formats, which depend on the instruction type and the operation it is performing, with respect to pre-fetches and accumulator write back. With the exception of the CLR and MOVSAC instructions, all MAC class instructions must specify a target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



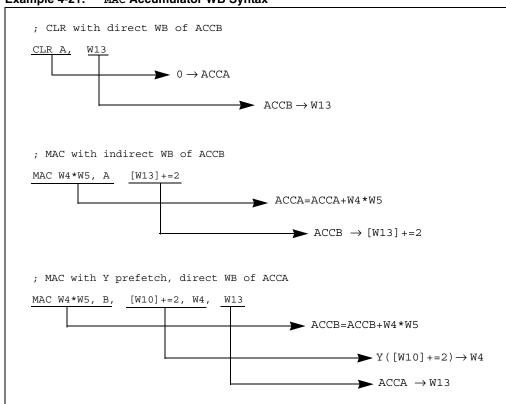


If a pre-fetch is used in the instruction, the assembler is capable of discriminating the X or Y data pre-fetch based on the register used for the effective address. [W8] or [W9] specifies the X pre-fetch and [W10] or [W11] specifies the Y pre-fetch. Brackets around the working register are required in the syntax, and they designate that indirect addressing is used to perform the pre-fetch. 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 pre-fetch, W12 is placed inside the brackets ([W9+W12] for X pre-fetches and [W11+W12] for Y pre-fetches). Each pre-fetch operation must also specify a pre-fetch destination register (W4-W7). In the instruction syntax, the destination register appears before the pre-fetch are shown in Example 4-20.



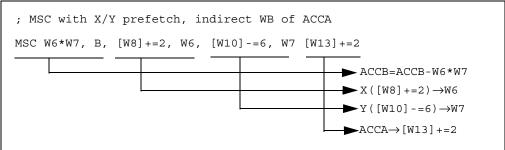


If an accumulator write back is used in the instruction, it is specified last. The write back must use the W13 register, and allowable forms for the write back are "W13" for direct addressing and "[W13]+=2" for indirect addressing with post-increment. By definition, the accumulator not used in the mathematical 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 pre-fetches and a write back is shown in Example 4-22.





Example 4-21: MAC Accumulator WB Syntax

#### 4.15 **DSP Accumulator Instructions**

The DSP Accumulator instructions do not have pre-fetch 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

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 it's 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<sup>™</sup> 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.

S

Word Value Exponent		Full Scale Value (Word Value << Exponent)	
0x0001	14	0x4000	
)x0002	13	0x4000	
x0004	12	0x4000	
x0100	6	0x4000	
)x01FF	6	0x7FC0	
)x0806	3	0x4030	
)x2007	1	0x400E	
x4800	0	0x4800	
x7000	0	0x7000	
x8000	0	0x8000	
x900A	0	0x900A	
xE001	2	0x8004	
xFF07	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 ; determine the exponent to use for scaling FBCL WO, W2 ; store exponent in W2 ; scale the entire data block before processing DO #(BLOCK SIZE-1), SCALE ; move the next data sample to ACCA LAC [W4], A SFTAC A, W2 ; shift ACCA by W2 bits SCALE: ; store scaled input (overwrite original) SAC A, [W4++] ; now process the data ; (processing block goes here)

### 4.17 Normalizing the Accumulator with the FBCL Instruction

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 AccU 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 AccH, as described in **Section 4.12 "Accumulator Usage"**. 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 AccU is in use, or scaling the accumulator up if all of the AccH bits are not being used. To perform such scaling, the FBCL instruction must operate on the AccU byte and it must operate on the AccH 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 an operation in ACCA has just completed (SR intact)
; assume the processor is in super saturation mode
; assume ACCAH 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, WO
                              ; shift ACCA to normalize
```

NOTES:



# HIGHLIGHTS

This section of the manual contains the following major topics:

5.1	Instruction Symbols	. 5-2
5.2	Instruction Encoding Field Descriptors Introduction	.5-2
5.3	Instruction Description Example	5-6
5.4	Instruction Descriptions	5-7

D

nstruction scriptions

### 5.1 Instruction Symbols

All symbols used in Section 5.4 "Instruction Descriptions" are shown 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	Accumulator selection bit: 0=ACCA; 1=ACCB	
aa	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	Pre-Fetch X Operation (see Table 5-6)	
jjjj	Pre-Fetch 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)	
ddd	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	Wb base register select: 0000=W0; 1111=W15	
XX	Pre-Fetch X Destination (see Table 5-7)	
XXXX XXXX XXXX XXXX	16-bit unused field (don't care)	
УУ	Pre-Fetch Y Destination (see Table 5-9)	
Z	Bit test destination: 0=C flag bit; 1=Z flag bit	

ррр	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	<b>Register</b> (with	n Register Offset)
	onoor Addressing	, modeo ioi mo ocaroc	riegiotor (miti	integrator anooty

ggg	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]

# Table 5-6: X Data Space Pre-Fetch Operation

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 Pre-fetch 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 5-7: X Data Space Pre-Fetch Destination

xx	Wxd	
00	W4	
01	W5	
10	W6	
11	W7	

## Table 5-8: Y Data Space Pre-Fetch Operation

iiii	Operation
0000	Wyd=[W10]
0001	Wyd=[W10], W10 = W10 + 2
0010	Wyd=[W10], W10 = W10 + 4
0011	Wyd=[W10], W10 = W10 + 6
0100	No Pre-fetch 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 Pre-Fetch Destination

уу	Wyd
00	W4
01	W5
10	W6
11	W7

 Table 5-10:
 MAC or MPY Source Operands (Same Working Register)

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)

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

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

# 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
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 dsPIC30F Addressing modes. In these circumstances, all possible instruction operands are listed beneath each other (as in the case of op2a, op2b and op2c above). Optional operands are enclosed in braces.
Operands:	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 Operation field summarizes the operation performed by the instruction.
Status Affected:	The Status Affected field describes which bits of the Status Register are affected by the instruction. Status bits are listed by bit position in descending order.
Encoding:	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:	The Examples field contains examples which demonstrate how the instruction operates. "Before" and "After" register snapshots are provided, which allow the user to clearly understand what operation the instruction performs.

# 5.4 Instruction Descriptions

ADD		Add f to WF	REG			
Syntax:	{label:}	ADD{.B}	f	{,WREG}		
Operands:	f ∈ [0 8	191]				
Operation:	(f) + (WRE	EG) $\rightarrow$ destination	tion designat	ed by D		
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	1011	0100	OBDf	ffff	ffff	ffff
Description:	the file rec optional W specified,	ontents of the o gister and plac /REG operand the result is st cored in the file	e the result i I determines ored in WRE	n the destinat the destination	tion register.	The WREG is
	The 'D' bit	selects byte c selects the de select the add	estination (0	for WREG, 1		
		The extension rather than a denote a wor The WREG is	word opera d operation,	tion. You may but it is not re	y use a .w ex equired.	
Words:	1					
Cycles:	1					
Example 1 A	DD.B	RAM100	; Add	WREG to RA	M100 (Byte	mode)
	Before		After			
	Instructio	n	Instructio	n		
WRE			EG CC80	_		
RAM10 S	0 FFC0 R 0000	RAM	100 FF40 SR 0005			
Example 2 AI	DD	RAM200, WRE	G ; Add F	AM200 to W	REG (Word	mode)
WREC RAM20 SF	0 FFC0	WRE RAM2	00 FFC0	(C=1)		

ADD		Add Literal	l to Wn			
Syntax:	{label:}	ADD{.B}	#lit10,	Wn		
Operands:			e operation ord operation			
Operation:	lit10 + (Wn	$) \rightarrow Wn$				
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1011	0000	0Bkk	kkkk	kkkk	dddd
Description:				nd to the conte c into the worki		
	The 'k' bits	specify the li	teral operand	ition (0 for wor I. working registe		
		rather than denote a wo For byte ope value [0:255]	a word opera rd operation, erations, the li ]. See <b>Sectio</b>	instruction de ation. You may but it is not red teral must be s n 4.6 "Using 1 0-bit literal ope	/ use a .w e quired. specified as a <b>0-bit Literal</b>	extension to an unsigned <b>Operands</b> "
Words:	1		U		,	
Cycles:	1					
Example 1	ADD.B	#0xFF, W	7 ; Ad	d -1 to W7	(Byte mode)	)
	Before	9	After			
	Instructi		Instruct			
	W7 12C		W7 12B SR 000			
Example 2	ADD	#0xFF, W	1 ; Ad	d 255 to W1	(Word mode	e)
	Before Instructi W1 12C SR 000	on 0	After Instruct W1 13B SR 000	ion F		

Curatas	(1-11-)		14/1-	ш!: <b>.</b> г	14/-1	
Syntax:	{label:}	ADD{.B}	Wb,	#lit5,	Wd	
					[Wd]	
					[Wd++] [Wd]	
					[++Wd]	
					[++ <b>vv</b> d] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	. 31]				
Operation:	(Wb) + lit	ō → Wd				
Status Affected:	DC, N, O	/, Z, C				
Encoding:	0100	0www	wBqq	qddd	d11k	kkkk
Description:	operand a direct add	and place the	result in the be used for	e destination	5-bit unsigned register Wd. Re egister direct o	egister
	The 'B' bit The 'q' bit The 'd' bit	s select the d s select the a	or word ope lestination A ddress of th	eration (0 for Address mode ne destinatior	word, 1 for byte e.	
	Note:	rather than	a word ope		denotes a byten nay use a .w e t required.	
Words:	1				t loquilou.	
Cycles:	1					
		W0 #01E	W7		nd 31 (Byte	mode)
Example 1	ADD.B	W0, #0x1F	-		e result in	
Example 1	ADD.B Before	WU, HUXIF	-	; Store the	-	
Example 1	Before Instructio	on	Afte	; Store the r tion	-	
Example 1	Before Instructic W0 2290	on	Afte Instruc W0 223	; Store the r tion 90	-	
Example 1	Before Instructio	n 	Afte	; Store the r tion 90 AF	-	
Example 1 Example 2	Before Instructio W0 2290 W7 12C0	n 	Afte Instruc W0 222 W7 122 SR 000	; Store the tion 90 AF 08 (N=1) ; Add W3 a	-	W7 node)
	Before Instruction W0 2290 W7 12C0 SR 0000 ADD Before	0 <b>n</b> ]] W3, #0x6,	Afte Instruc W0 222 W7 122 SR 000 [W4]	; Store the r tion 90 AF 08 (N=1) ; Add W3 a ; Store th er	e result in nd 6 (Word π	W7 node)
	Before Instruction W0 2290 W7 12C0 SR 0000 ADD Before Instruction	m ] W3, #0x6,	Afte Instruc W0 223 W7 127 SR 000 [W4]	; Store the r tion PO AF 08 (N=1) ; Add W3 a ; Store th er etion	e result in nd 6 (Word π	W7 node)
	Before Instruction W0 2290 W7 12C0 SR 0000 ADD Before	on ] ] W3, #0x6, pn	Afte Instruc W0 222 W7 122 SR 000 [W4]	; Store the r tion 90 AF 08 (N=1) ; Add W3 a ; Store th er stion 06	e result in nd 6 (Word π	W7 node)
	Before           Instruction           W0         2290           W7         1200           SR         0000           ADD         Before           Instruction         W3           6006         W4           DDEF         DDEF	on W3, #0x6, on Data	Afte Instruc W0 229 W7 122 SR 000 [W4] [W4] Afte Instruc W3 60	; Store the r tion 90 AF 08 (N=1) ; Add W3 a ; Store th er tion 06 FE	e result in nd 6 (Word π	W7 node)

© 2005 Microchip Technology Inc.

Instruction Descriptions

ADD		Add Wb to	o Ws			
Syntax:	{label:}	ADD{.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) $\rightarrow$ Wd				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	0100	0www	wBqq	qddd	dppp	SSSS
Description:	register Wi direct addr addressing	b and place ressing mus g may be us	the result in	the destination Wb. Either ro d Wd.	the contents o on register Wd egister direct o	. Registe
	The 'q' bits The 'd' bits The 'p' bits	select the c select the c select the c select the c	destination A address of th source Addre address of th	ddress mode e destination ess mode. e source reg	register.	
	10101	rather than		ation. You m	ay use a .we	
Words:	1					
Cycles:	1					
Example 1	ADD.B W	5, W6, W7		ld W5 to W6 Syte mode)	, store res	ult in M
	Before Instruction W5 AB00 W6 0030 W7 FFFF SR 0000	V V	After Instruction V5 AB00 V6 0030 V7 FF30 SR 0000			
Example 2	ADD W	5, W6, W7	-	d W5 to W6 ord mode)	, store res	ult in V
	Before Instruction W5 AB00 W6 0030 W7 FFFF SR 0000	N N		(N=1)		

ADD	Add Accumulators	
Syntax:	{label:} ADD Acc	
Operands:	$Acc \in [A,B]$	
Operation:	$\begin{array}{l} \underline{\text{If } (\text{Acc} = \text{A}):} \\ (\text{ACCA}) + (\text{ACCB}) \rightarrow \text{ACCA} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\$	
Status Affected:	OA, OB, OAB, SA, SB, SAB	
Encoding:	1100 1011 A000 0000 0000	0000
Description:	Add the contents of Accumulator A to the contents of Accumulator place the result in the selected accumulator. This instruction 40-bit addition.	
	The 'A' bit specifies the destination accumulator.	
Words:	1	
Cycles:	1	
Example 1 AD	D A ; Add ACCB to ACCA	
ACCA ACCE SF	ACCB 00 1833 4558	
Example 2 ADD	B ; Add ACCA to ACCB ; Assume Super Saturation mo ; (ACCSAT=1, SATA=1, SATB=1)	
ACCA ACCB SR	00 7654 3210 ACCB 01 5765 5432	·B=1)

Syntax:       {label:}       ADD       Ws,       {#Slit4.}       Acc         [Ws],       [Ws],       [Ws],       [Ws],       [Ws-1],       [-Ws],       [I+Ws],       [Ws+Wb],         Operands:       Ws $\in$ [W0 W15]       [Ws+Wb],       [Ws+Wb],       [Ws+Wb],         Operation:       Shiftsit4 $\in$ [-8 +7]       Acc $\in$ [A,B]       Operation:       Shiftsit4(Extend(Ws)) + (Acc) $\rightarrow$ Acc         Status Affected:       OA, OB, OAB, SA, SB, SAB       Encoding:       1100       1001       Awww       wrrr       rggg         Description:       Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.         The 'A' bit specifies the destination accumulator. The 'a' bits specify the offset register Wb. The 'r' bits specify the source register Ws.       Note:       Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arit left. The contents of the source register are not affected         Words:       1       Cycles:       1       Example 1       ADD       W0, #2, A ; Add W0 right-shifted by 2 to Before		Acc	{#Slit4,}	Ws,	\DD	۱۰۱	/lah/	Syntax:
$[W_{s++}], \\ [W_{s}], \\ [Ws], \\ [++Ws], \\ [W_{s+Wb], \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\$		AUU	\#OIII4,}			-i.j	ίαυς	Symax.
$ \begin{bmatrix} W_{S} - i, \\ [-W_S], \\ [++W_S], \\ [W_{S}+W_{D}], \end{bmatrix} $ Operands: Ws $\in [W0 \dots W15]$ $W_{D} \in [W0 \dots W15]$ $Slit4 \in [-8 \dots +7]$ $Acc \in [A,B]$ Operation: Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) $\rightarrow$ Acc Status Affected: OA, OB, OAB, SA, SB, SAB Encoding: 1100 1001 Awww wrrr rggg Description: Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accumulator. The source operand prior to the The value added to the accumulator may also be shifted by a 4-bid literal before the addition is made. The 'A' bit specifies the destination accumulator. The 'a' bit specify the offset register Wb. The 'r' bits specify the offset register Wb. The 's' bits specify the source register Ws. Note: Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand S								
[Ws],         [++Ws],         [Ws+Wb],         Operands:       Ws ∈ [W0 W15]         Wb ∈ [W0 W15]         Slit4 ∈ [-8 +7]         Acc ∈ [A,B]         Operation:       Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) → Acc         Status Affected:       OA, OB, OAB, SA, SB, SAB         Encoding:       1100       1001       Awww       wrrr       rggg         Description:       Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.         The 'A' bit specifies the destination accumulator.         The 'A' bit specify the offset register Wb.         The 'A' bit specify the source Address mode.         The 's' bits specify the source register Ws.         Note:       Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative								
$[++Ws], [Ws+Wb],$ Operands: Ws $\in$ [W0 W15] Wb $\in$ [W0 W15] Silt4 $\in$ [-8 +7] Acc $\in$ [A,B] Operation: Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) $\rightarrow$ Acc Status Affected: OA, OB, OAB, SA, SB, SAB Encoding: 1100 1001 Awww wrrr rggg Description: Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective ad value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made. The 'A' bit specifies the destination accumulator. The 'a' bits specify the offset register Wb. The 'a' bits specify the source register Wb. The 's' bits specify the source register Ws. Note: Positive values of operand Slit4 represent an arithmet and negative								
[Ws+Wb],         Operands:       Ws ∈ [W0 W15] Wb ∈ [W0 W15] Slit4 ∈ [-8 +7] Acc ∈ [A,B]         Operation:       Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) → Acc         Status Affected:       OA, OB, OAB, SA, SB, SAB         Encoding:       1100       1001       Awww       wrrr       rggg         Description:       Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective ad value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bi literal before the addition is made.         The 'A' bit specifies the destination accumulator. The 'a' bits specify the offset register Wb. The 'r' bits encode the optional shift.         The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.         Note:       Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an aritheft         Words:								
Operands:Ws $\in [W0 \dots W15]$ Wb $\in [W0 \dots W15]$ Slit4 $\in [-8 \dots +7]$ Acc $\in [A,B]$ Operation:Shiftslit4(Extend(Ws)) + (Acc) $\rightarrow$ AccStatus Affected:OA, OB, OAB, SA, SB, SABEncoding:11001001AwwwWrrrrgggDescription:Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.The 'A' bit specifies the destination accumulator. The 'w' bits specify the offset register Wb. The 'r' bits encode the optional shift. The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.Note:Positive values of operand Slit4 represent an arithmet and negative val								
Wb ∈ [W0 W15] Slit4 ∈ [-8 +7] Acc ∈ [A,B]         Operation:       Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) → Acc         Status Affected:       OA, OB, OAB, SA, SB, SAB         Encoding:       1100       1001       Awww       wrrr       rggg         Description:       Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective ad value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.         The 'A' bit specifies the destination accumulator. The 'A' bit specify the offset register Wb. The 'r' bits encode the optional shift. The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.         Note:       Positive values of operand Slit4 represent an arither and negative values of operand Slit4 represent an arither and neg				[1131110],				
Operation:       Shift <sub>Slit4</sub> (Extend(Ws)) + (Acc) → Acc         Status Affected:       OA, OB, OAB, SA, SB, SAB         Encoding:       1100       1001       Awww       wrrr       rggg         Description:       Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.         The 'A' bit specifies the destination accumulator. The 'w' bits specify the offset register Wb. The 'r' bits encode the optional shift.       The 'g' bits select the source Address mode.         The 's' bits specify the source register Ws.       Note:       Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of the source register are not affecte         Words:       1         Cycles:       1         ADD       W0, #2, A       ; Add W0 right-shifted by 2 to					W15]	∈ [W0 ∈ [-8	Wb Slit4	Operands:
Status Affected:OA, OB, OAB, SA, SB, SABEncoding:11001001AwwwwrrrrgggDescription:Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective ad value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.The 'A' bit specifies the destination accumulator. The 'w' bits specify the offset register Wb. The 'r' bits encode the optional shift. The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.Note:Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of the source register are not affectedWords:1Cycles:1ADDW0, #2, A; Add W0 right-shifted by 2 to				.cc) $\rightarrow$ Acc	nd(Ws)) + (A			Operation:
Encoding:11001001AwwwwrrrrgggDescription:Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.The 'A' bit specifies the destination accumulator. The 'A' bit specifies the destination accumulator. The 'r' bits encode the optional shift. The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.Note:Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arith left. The contents of the source register are not affecteWords:1Cycles:1ADDW0, #2, A; Add W0 right-shifted by 2 to								Status Affecte
Description:Add a 16-bit value specified by the source working register to the Significant word of the selected accumulator. The source operand specify the direct contents of a working register or an effective add value specified is added to the Most Significant Word of the accur sign-extending and zero backfilling the source operand prior to the The value added to the accumulator may also be shifted by a 4-bit literal before the addition is made.The 'A' bit specifies the destination accumulator. The 'A' bit specifies the destination accumulator. The 'w' bits specify the offset register Wb. The 'r' bits encode the optional shift. The 'g' bits select the source Address mode. The 's' bits specify the source register Ws.Note:Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet and negative values of the source register are not affecteWords:1Cycles:1ADD W0, #2, A; Add W0 right-shifted by 2 to	SSSS	rggg	wrrr					Encoding:
The 'w' bits specify the offset register Wb.         The 'r' bits encode the optional shift.         The 'g' bits select the source Address mode.         The 's' bits specify the source register Ws.         Note:       Positive values of operand Slit4 represent an arithmet and negative values of operand Slit4 represent an arithmet left. The contents of the source register are not affected         Words:       1         Cycles:       1         Example 1       ADD W0, #2, A ; Add W0 right-shifted by 2 to	umulator, by	d of the accun and prior to the	gnificant Wor source opera	o the Most Sig ackfilling the s cumulator ma	d is added to g and zero b ded to the ac	e specifi -extendir value ad	valu sign The	
and negative values of operand Slit4 represent an arit left. The contents of the source register are not affecte Words: 1 Cycles: 1 Example 1 ADD W0, #2, A ; Add W0 right-shifted by 2 to			/b. node.	set register W ional shift. rce Address n	becify the off code the opt elect the sour	'w' bits s 'r' bits ei 'g' bits s	The The The	
Cycles: 1 Example 1 ADD W0, #2, A ; Add W0 right-shifted by 2 to	ithmetic sh	present an arit	and Slit4 rep	alues of oper	id negative v	a	N	
Example 1 ADD W0, #2, A ; Add W0 right-shifted by 2 to							1	Words:
							1	Cycles:
Before After	o acca	ted by 2 to	right-shift	; Add W0 1	2, A	WO, #	ADD	Example 1
			After		ore	Bef		
Instruction Instruction							_	
W0         8000         W0         8000           ACCA         00         7000         0000         ACCA         00         5000         0000				-			H	

SR

0000

SR

0000

Example 2 ADD [W5++], A

; Add the effective value of W5 to ACCA ; Post-increment W5  $\,$ 

Before Instruction					I	After nstruct	
W5			2000	W5			2002
ACCA	00 0	067	2345	ACCA	00	5067	2345
Data 2000			5000	Data 2000			5000
SR			0000	SR			0000

ADDC	Add f to WREG with Carry
Syntax:	{label:} ADDC{.B} f {,WREG}
Operands:	f ∈ [0 8191]
Operation:	(f) + (WREG) + (C) $\rightarrow$ destination designated by D
Status Affected:	DC, N, OV, Z, C
Encoding:	1011 0100 1BDf ffff ffff ffff
Description:	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). The 'f' bits select the address of the file register.
	<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 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 SUBB. These instructions can only clear Z.</li> </ul>
Words:	1
Cycles:	1
Example 1	DDC.B RAM100 ; Add WREG and C bit to RAM100 ; (Byte mode)
WRE RAM10 S	0 8006 RAM100 8067
Example 2	DDC RAM200, WREG ; Add RAM200 and C bit to the WREG ; (Word mode)
WRE RAM20 S	

ADDC	Add Literal to Wn with Carry
Syntax:	{label:} ADDC{.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) + (C) $\rightarrow$ Wn
Status Affected	DC, N, OV, Z, C
Encoding:	1011 0000 1Bkk kkkk kkkk dddd
Description:	Add the 10-bit unsigned literal operand, the contents of the working register Wn and the Carry bit and place the result back into the working register Wn.
	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.
	<ul> <li>rather than a word operation. You may use a . W extension to 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 2.7 "Using 10-bit Literal Operands" for information on using 10-bit literal operands in Byte mode.</li> <li>3: The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.</li> </ul>
Words:	1
Cycles:	1
Example 1	ADDC.B #0xFF, W7 ; Add -1 and C bit to W7 (Byte mode)
	Before         After           Instruction         Instruction           W7         12C0         W7         12BF           SR         0000         (C=0)         SR         0009         (N,C=1)
Example 2	ADDC #0xFF, W1 ; Add 255 and C bit to W1 (Word mode)
	Before         After           Instruction         Instruction           W1         12C0         W1         13C0           SR         0001         (C=1)         SR         0000

ADDC		Add Wb to	Short Literal	with Carry		
Syntax:	{label:}	ADDC{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	. 31]				
Operation:	-	$5 + (C) \rightarrow Wd$				
Status Affected:	DC, N, O	. ,				
Encoding:	0100	lwww	wBqq	qddd	d11k	kkkk
Description:	operand a Wd. Regis	ontents of the and the Carry I ster direct add ddressing may	pit and place t ressing must	he result in th be used for W	e destination	register
	The 'B' bit The 'q' bit The 'd' bit	ts select the a t selects byte of s select the de s select the ac s provide the l	or word opera estination Add ddress of the o	tion (0 for wor ress mode. destination reg	rd, 1 for byte). gister.	
		denote a wo The Z flag is	a word opera rd operation,	tion. You may but it is not rea DDC, CPB, S	vuse a .wex quired.	tension t
Words:	1					
Cycles:	1					
Example 1	ADDC.B	WO, #0x1F,		dd W0, 31 ar tore the re		-
		)	W7 12 ta 12C0 B0			
Example 2	ADDC	W3, #0x6,		d W3, 6 and ore the res		
۱ Data 0F Data 10	00 DDEE	Data	W4 0 a 0FFE 6 a 1000 D			

ADDC	Add Wb to Ws with Carry						
Syntax:	{label:}	ADDC{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]		
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]					
Operation:	-	$(s) + (C) \rightarrow W(c)$	d				
Status Affected:	DC, N, O\						
Encoding:	0100	lwww	wBqq	qddd	dppp	SSSS	
	The 'B' bit The 'q' bit The 'd' bit The 'p' bit	is select the ac selects byte of s select the de s select the ac s select the so s select the ac	or word opera estination Add Idress of the ource Address	tion (0 for wo dress mode. destination re s mode.	ord, 1 for byte egister.	e).	
		denote a wor The Z flag i	word operat	ion. You may but it is not re ADDC, CP	/ use a .w ex equired.	tension to	
Words:	1						
Cycles:	1						
Example 1 ADD	C.B WO,	,[W1++],[W2+	; Sto		and C bit ( ult in [W2] : W1, W2	-	
W0 W1 W2 Data 0800 Data 1000	0800 1000 AB25	W W W Data 080 Data 100	1 0801 2 1001 0 AB25				

SR

0000

5

SR

0001 (C=1)

```
Example 2
```

W3, [W2++], [W1++]

; Add W3, [W2] and C bit (Word mode) ; Store the result in [W1] ; Post-increment W1, W2

I	Before nstructio	n	lı	After nstruction
W1	1000		W1	1002
W2	2000		W2	2002
W3	0180		W3	0180
Data 1000	8000		Data 1000	2681
Data 2000	2500		Data 2000	2500
SR	0001	(C=1)	SR	0000

AND		AND f and	WREG			
Syntax:	{label:}	AND{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	911				
Operation:	-	•	stination do	signated by D		
-		$(neg) \rightarrow ues$	sination de	signated by D		
Status Affected:	N, Z		0.7.7.5			
Encoding: Description:	1011	0110	OBDf	n of the content	ffff	ffff
	the destination	ation register. 1 register. If V	The optior VREG is sp	of the file registen thal WREG opera the result operation operation that is stored in the	and determine It is stored in	es the
	The 'D' bit	selects the d	estination	eration (0 for wo (0 for WREG, 1 he file register.		
Words:		rather than denote a wo	a word ope ord operation	ne instruction de eration. You may on, but it is not re orking register V	y use a .w e equired.	
Cycles:	1					
Example 1 An	ID.B RAM1	00	; ANI	D WREG to RAM	100 (Byte	mode)
	Before		After			
	Instruction		Instruct	ion		
WRE			EG CC8			
RAM10 SI		RAM	100 FF8 SR 000	<u> </u>		
Example 2 AI	ND RAM200	, WREG	; ANI	D RAM200 to W	IREG (Word	mode)
	Before		After			
	Instruction		Instruct	-		
WRE RAM20 SI	0 12C0	WF RAM	IEG         008           200         12C           SR         000	0		

AND		AND Litera	al and Wd			
Syntax:	{label:}	AND{.B}	#lit10,	Wn		
Operands:		255] for byte 1023] for we W15]		ı		
Operation:	lit10.AND.	$(Wn) \rightarrow Wn$				
Status Affected:	N, Z					
Encoding:	1011	0010	0Bkk	kkkk	kkkk	dddd
Description:	contents o working re	f the working gister Wn. Re	register Wn egister direct	of the 10-bit I and place the addressing n	e result back i nust be used	nto the for Wn.
	The 'k' bits	specify the li	teral operan	ation (0 for w d. working regi	-	e).
	2:	denote a wo For byte op unsigned val	rd operation perations, thue [0:255]. S	ation. You ma , but it is not r ne literal mu See <b>Section</b> 4 on on using	equired. st be specil <b>I.6 "Using 10</b>	ied as an <b>-bit Literal</b>
Words:	1					
Cycles:	1					
Example 1	AND.B #0x8	3, W7	; AND	0x83 to W7	(Byte mode	:)
	Before Instruction W7 12C0 SR 0000		After Instructio N7 1280 SR 0008	n ] (N=1)		
Example 2	AND #0x333	, W1	; AND	0x333 to W2	1 (Word mod	le)
	Before Instruction W1 12D0 SR 0000		After Instructio W1 0210 SR 0000	n ] ]		

AND	AND Wb and Short Literal						
Syntax:	{label:}	AND{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]		
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]					
Operation:	(Wb).AND	.lit5 $\rightarrow$ Wd					
Status Affected:	N, Z						
Encoding:	0110	0www	wBqq	qddd	d11k	kkkk	
Description:	Wb and th Register d	e 5-bit literal a irect addressi	D operation o and place the ing must be us be used for V	result in the c sed for Wb. E	destination re	gister Wd	
	The 'B' bit The 'q' bits The 'd' bits	selects byte of s select the do s select the ac s provide the l The extension rather than a	ddress of the or word opera estination Add ddress of the o literal operand on .B in the i a word operat rd operation, I	tion (0 for wo ress mode. destination re I, a five-bit in nstruction de ion. You may	ord, 1 for byte egister. teger number enotes a byte / use a .w ex	operatic	
Words:	1		ra oporation, i		yunou.		
Cycles:	1						
Example 1	AND.B WO	),#0x3,[W1+	; Store	0 and 0x3 to [W1] increment	(Byte mode Wl	)	
	Before           Instruction           W0         23A5           W1         2211           210         9999           SR         0000	Data 2	After Instruction W0 23A5 W1 2212 2210 0199 SR 0000				
Example 2	AND	W0,#0x1F,W	-	W0 and 0x1 e to W1	F (Word mod	de)	
	Before Instruction W0 6723 W1 7878 SR 0000		After Instruction W0 6723 W1 0003 SR 0000	n     			

AND		And Wb a	nd Ws			
Syntax:	{label:}	AND{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	) W15]				
Operation:	(Wb).ANE	$0.(Ws) \rightarrow Wd$				
Status Affected:	N, Z					
Encoding:	0110	0www	wBqq	qddd	dppp	SSSS
	The 'w' bi The 'B' bit The 'q' bit The 'd' bit The 'p' bit	ts select the a t selects byte s select the d s select the a s select the s	address of th or word ope lestination A ddress of th ource Addre	e base regist ration (0 for v ddress mode e destination	vord, 1 for byte register.	
	Note:	rather than	a word oper		denotes a byte ay use a .w e required.	
Words:	1					
Cycles:	1					
Example 1	AND.B	WO, W1 [W2	; sto	W0 and W1 re to [W2] t-incremen	(Byte mode	)
Data	Before           Instruction           W0         AA55           W1         2211           W2         1001           1000         FFFF           SR         00000	Data	After Instruct W0 AAS W1 222 W2 100 1000 111 SR 000	ion 55 11 02 7F		

Example 2	AND	WO,	[W1++],	W2	; stoi	W0 and [W1], and re to W2 (Word mode) increment W1
	Befor	е			After	
	Instruct	ion		Ir	structio	n
	WO AA5	5		W0	AA55	
	W1 100	0		W1	1002	
	W2 55A	A		W2	2214	
Data	<b>1000</b> 263	4	Data 1	000	2634	
	SR 000	0		SR	0000	
				_		-

ASR		Arithmetic	Shift Right f			
Syntax:	{label:}	ASR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	-				
Operation:	(f<7>) – (f<6:1>) (f<0>) – <u>For word o</u> (f<15>) (f<15>)	$ \Rightarrow \text{Dest} < 7> \\ \Rightarrow \text{Dest} < 6> \\ \Rightarrow \text{Dest} < 5:0> \\ \Rightarrow \text{C} \\ \Rightarrow \text{peration:} \\ \Rightarrow \text{Dest} < 15> \\ \Rightarrow \text{Dest} < 14> \\ ) \Rightarrow \text{Dest} < 13: $				
Chatura Affa ata						
Status Affecte Encoding:	d: N, Z, C	0101	1BDf	ffff	ffff	ffff
Description:	result in the register is s performed, determines stored in W register. The 'B' bit The 'D' bit	ontents of the e destination shifted into the the result is s the destinati /REG. If WRE selects byte of selects the de select the add	register. The e Carry bit of sign-extender on register. If G is not spec or word opera estination (0 f	Least Signific the Status Re d. The optiona WREG is spe cified, the resu tion (0 for wo or WREG, 1 f	ant bit of the egister. After f al WREG ope ecified, the re ult is stored in rd, 1 for byte	file the shift is erand esult is n the file ).
	Note 1:	The extension rather than a	on . B in the in word operated operation,	nstruction de ion. You may but it is not re	use a .wex	
Words:	1					
Cycles:	1					
Example 1	ASR.B RAM40	0, WREG	; ASR RA ; (Byte	AM400 and s mode)	tore to WR	EG
	Before Instruction WREG 0600 AM400 0823 SR 0000	WRE RAM4 S	00 0823	(C=1)		
Example 2	ASR RAM200		; ASR F	AM200 (Wor	d mode)	
F	Before Instruction AM200 8009 SR 0000	RAM2	After Instruction 200 C004 SR 0009	(N, C=1)		

ASR		Arithmetic	Shift Right	Ws		
Syntax:	{label:}	ASR{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	0 W15] 0 W15]				
Operation:	(Ws<7 (Ws<7 (Ws<6 (Ws<0) For word (Ws<1 (Ws<1 (Ws<1	$\begin{array}{c} \hline pperation: \\ >) \rightarrow Wd < 7 > \\ >) \rightarrow Wd < 6 > \\ (1 >) \rightarrow Wd < 6 > \\ (1 >) \rightarrow C \\ \hline operation: \\ 5 >) \rightarrow C \\ (5 >) \rightarrow Wd < 15 \\ (5 >) \rightarrow Wd < 14 \\ (4 : 1 >) \rightarrow Wd < 12 \\ (5 >) \rightarrow C \\ \hline \hline$	>			
	N, Z, C				Γ	
Encoding:	1101	0001	1Bqq	qddd	dppp	ssss
Status Affected: Encoding: Description:	1101 Shift the of the result shifted int the result be used f The 'B' bi The 'G' bi The 'd' bi	contents of the in the destina- to the Carry bi is sign-extend or Ws and Wo it selects byte ts select the d ts select the a	e source regis ation register it of the Statu ded. Either re d. or word oper lestination Ad iddress of the	ster Ws one bit Wd. The Least s Register. Afte gister direct or ation (0 for wor dress mode.	to the right a Significant bi r the shift is p indirect addre	nd place it of Ws is performec essing ma
Encoding:	1101 Shift the of the result shifted int the result be used f The 'B' bi The 'G' bi The 'd' bi The 'd' bi	contents of the in the destina- to the Carry bi is sign-extend or Ws and Wo it selects byte ts select the d ts select the a ts select the s	e source regis ation register it of the Statu ded. Either re d. or word oper lestination Ad iddress of the ource Addres	ster Ws one bit Wd. The Least s Register. Afte gister direct or ation (0 for wor dress mode.	to the right a Significant bi er the shift is p indirect addre rd, 1 for byte) gister.	nd place it of Ws is performec essing ma
Encoding:	1101 Shift the of the result shifted int the result be used f The 'B' bi The 'G' bi The 'd' bi The 'd' bi	contents of the in the destina- to the Carry bi is sign-extend or Ws and Wo to selects byte ts select the d ts select the a ts select the a The extens rather than	e source regis ation register it of the Statu ded. Either re d. or word oper lestination Ad iddress of the ource Address ddress of the ion .B in the a word oper	ster Ws one bit Wd. The Least s Register. Afte gister direct or ation (0 for wor dress mode. destination reg s mode.	to the right a Significant bi er the shift is p indirect addre d, 1 for byte) gister. r. enotes a byte v use a .w e	nd place it of Ws is performed essing ma
Encoding:	1101 Shift the of the result shifted int the result be used f The 'B' bi The 'G' bi The 'q' bi The 'g' bi The 's' bit	contents of the in the destina- to the Carry bi is sign-extend or Ws and Wo to selects byte ts select the d ts select the a ts select the a The extens rather than	e source regis ation register it of the Statu ded. Either re d. or word oper lestination Ad iddress of the ource Address ddress of the ion .B in the a word oper	ster Ws one bit Wd. The Least s Register. After gister direct or dress mode. destination reg source registe o instruction de ation. You may	to the right a Significant bi er the shift is p indirect addre d, 1 for byte) gister. r. enotes a byte v use a .w e	nd place it of Ws is performed essing ma

Example 1

ASR.B [W0++], [W1++]

; ASR [W0] and store to [W1] (Byte mode) ; Post-increment W0 and W1

I	Before nstructior	n I	After nstruction
W0	0600	W0	0601
W1	0801	W1	0802
Data 600	2366	Data 600	2366
Data 800	FFC0	Data 800	33C0
SR	0000	SR	0000

Example 2

ASR W12, W13

; ASR W12 and store to W13 (Word mode)

Before Instruction					
W12 AB01					
W13	0322				
SR	0000				

After Instruction W12 AB01 W13 D580 SR 0009 (N, C=1)

ASR	(lok al.)		ic Shift Right	-		
Syntax:	{label:}	ASR	Wb,	#lit4,	Wnd	
Operands:	Wb ∈ [W0 lit4 ∈ [01 Wnd ∈ [W0	5]				
Operation:		Wnd<15:	15-Shift_Val+ Wnd<15-Shift			
Status Affected:	N, Z					
Encoding:	1101	1110	lwww	wddd	d100	kkkk
Description:	unsigned li	teral and s performed,	tore the result , the result is s	in the destir	register Wb by nation register d. Direct addre	Wnd. Afte
	The 'd' bits	select the	address of th address of th e literal opera	e destination		
	Note:	This instru	ction operate	s in Word mo	ode only.	
Words:	1					
Cycles:	1					
Example 1	ASR W0, #0×	4, Wl	; ASR	W0 by 4 a	nd store to	Wl
	Before Instruction W0 060F W1 1234 SR 0000		After Instruction W0 060F W1 0060 SR 0000	-		
Example 2	ASR W0, #0x	6, W1	; ASR	W0 by 6 ar	nd store to	Wl
	Before Instruction W0 80FF W1 0060 SR 0000		After Instruction W0 80FF W1 FE03 SR 0008	]		
Example 3	ASR W0, #03	xF, W1	; ASR	W0 by 15	and store to	5 W1
	Before Instruction W0 70FF W1 CC26 SR 0000		After Instructi W0 70F1 W1 0000 SR 0000	F 0		

Syntax:	{label:}	ASR	Wb,	Wns,	Wnd	
Operands:	Wb ∈ [W0 Wns ∈ [W Wnd ∈ [W	0W15]				
Operation:	Wb<15> -		ıl I5-Shift_Val+1 Wnd<15-Shift_			
Status Affected:	N, Z					
Encoding:	1101	1110	lwww	wddd	d000	SSSS
Description:	Significant destination	bits of Wns register W	s (up to 15 pos nd. After the s	itions) and shift is perfo	register Wb by t store the result i prmed, the result for Wb, Wns ar	n the ∶is
	The 'd' bit	s select the	address of the address of the address of the	destination	n register.	
		If Wns is	ction operates greater than <sup></sup> FFFF if Wb is r	15, Wnd =	ode only. 0x0 if Wb is po	ositive, ar
Words:	1					
Cycles:	1					
Example 1	ASR WO, WS	5, W6	; ASR	W0 by W5	and store to	W6
	Before		After			
		1	Instructio	-		
	WO 80FF W5 0004		W0 80FF			
			W5 0004 W6 F80E			
	W6 2633 SR 0000		W6 F80E SR 0000	_		
Example 2	ASR W0, W	5, W6	; ASR	W0 by W5	and store to	W6
	Before		After			
	Instruction	า	Instructi	on		
	W0 6688		W0 668	3		
	W5 000A		W5 0002	A		
	W6 FF00		W6 001	Э		
	SR 0000		SR 000	0		
Example 3		W12, W13		W11 by W1	12 and store	to W13
	Before Instructior		After	n		
V	V11 8765	I	Instructio			
				-		
	V12 88E4 V13 A5A5		W12 88E4 W13 F876			
v	CACA U.		VIU F0/0	1		

BCLR		Bit Clear f				
Syntax:	{label:}	BCLR{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0	91] for byte o 90] (even onl 7] for byte op 15] for byte o	y) for word op eration	peration		
Operation:	$0 \rightarrow f < bit4$	>				
Status Affected:	None					
Encoding:	1010	1001	bbbf	ffff	ffff	fffb
Description:	with the Le	it in the file re ast Significan byte operation	t bit (bit 0) an	d advances t	to the Most S	
		select value select the add			e cleared.	
		denote a wor When this in:	t be word alig struction ope	out it is not re ates in Word ned.	equired. I mode, the f	ile register
Words:	1					
Cycles:	1					
Example 1 BC	LR.B 0x800	, #0x7	; Clear	bit 7 in (	)x800	
Data 0800 SF		Data 08	After Instruction 00 666F SR 0000			
Example 2 BC	LR 0x400	, #0x9	; Clear	bit 9 in (	)x400	
Data 040 SF		Data 04	After Instruction 00 A855 SR 0000			

BCLR	Bit Clear in Ws
Syntax:	{label:} BCLR{.B} Ws, #bit4 [Ws], [Ws++], [Ws], [++Ws], [Ws],
Operands:	Ws ∈ [W0 W15] bit4 ∈ [0 7] for byte operation bit4 ∈ [0 15] for word operation
Operation:	$0 \rightarrow Ws < bit4 >$
Status Affected:	
Encoding: Description:	10100001bbbb0B000pppsssClear the bit in register Ws specified by 'bit4'. Bit numbering begins with
Description.	the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 7 for byte operations, bit 15 for word operations). Register direct or indirect addressing may be used for Ws.
	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 address of the source/destination register. The 'p' bits select the source Address mode.
	<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: When this instruction operates in Word mode, the source register address must be word aligned.</li> <li>3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.</li> </ul>
Words:	1
Cycles:	1
Example 1	BCLR.B W2, #0x2 ; Clear bit 3 in W2
	BeforeAfterInstructionInstructionW2F234W2SR0000SR
Example 2	BCLR [W0++], #0x0 ; Clear bit 0 in [W0] ; Post-increment W0
Data	Before         After           Instruction         Instruction           W0         2300         W0         2302           2300         5607         Data 2300         5606           SR         0000         SR         0000

BRA	Bra	nch Unco	nditionally			
Syntax:	{label:} BR/	Ą	Expr			
Operands:	Expr may be a la Expr is resolved					+32767
Operation:	(PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register					
Status Affected:	None					
Encoding:	0011	0111	nnnn	nnnn	nnnn	nnnn
Description:	The program wil of the branch is branches up to 3 resolved by the expression. Afte 2*Slit16, since th	the 2's cor 32K instruc inker from r the branc	nplement nu tions forward the supplied th is taken, th	mber '2*Slit16 d or backward l label, absolu ne new addre	6', which sup d. The Slit16 ute address o ss will be (P0	ports value is or C+2) +
	The 'n' bits are a offset from (PC+		eral that spe	cifies the num	ber of progra	am words
Words:	1					
Cycles:	2					
Example 1	002000 HERE: 002002 002004 002006 002008 00200A THERE: 00200C	BRA TH     	ERE	; B	ranch to T	HERE
	Before Instruction PC 00 2000 SR 0000		PC SR	After Instruction 00 200A 0000		
Example 2	002000 HERE: 002002 002004 002006 002008 00200A THERE: 00200C	BRA THE     	RE+0x2	; Br	anch to TH	ERE+0x2
	Before Instruction PC 00 2000 SR 0000		PC SR	After Instruction 00 200C 0000		
Example 3	002000 HERE: 002002 002004	BRA 0x1  	366	; Br	anch to Ox	1366
	Before Instruction PC 00 2000 SR 0000		PC SR	After Instruction 00 1366 0000		

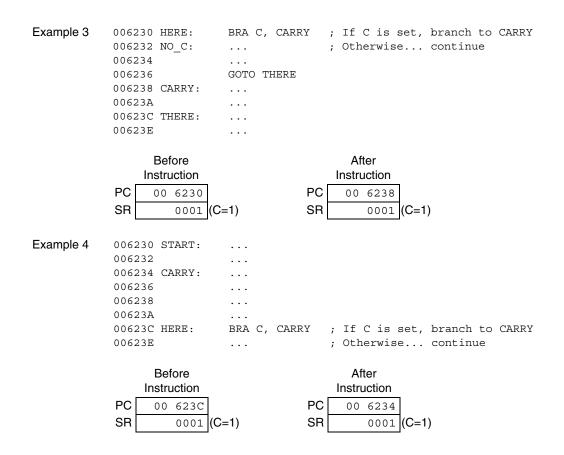
Instruction Descriptions

BRA		Computed E	Branch			
Syntax:	{label:}	BRA	Wn			
Operands:	Wn ∈ [W0	-				
Operation:		$2^*Wn) \rightarrow PC$ struction Regis	ter			
Status Affected:	None					
Encoding:	0000	0001	0110	0000	0000	SSSS
Description:	offset of the supports br instruction	m will branch branch is the anches up to executes, the nented to fetcl	sign-extende 32K instruction new PC will b	ed 17-bit val ons forward o oe (PC+2)+2	ue (2*Wn), w or backward.	hich After this
	The 's' bits	select the add	lress of the s	ource registe	er.	
Words:	1					
Cycles:	2					
Example 1	002000 HERE: 002002  002108 00210A TABLE 00210C	BRA W7    7:	;	Branch fo	rward (2+2	*W7)
	Before           Instruction           PC         00         200           W7         008           SR         000	0	PC W7 SR	After nstruction 00 2108 0084 0000		

BRA C	Bra	anch if Car	rry			
Syntax:	{label:} BF	A	С,	Expr		
Operands:	Expr may be a l Expr is resolved					+32767].
Operation:	Condition = C If (Condition) (PC+2) + 2*S NOP $\rightarrow$ Instr					
Status Affected:	None					
Encoding:	0011	0001	nnnn	nnnn	nnnn	nnnn
Description:	If the Carry flag The offset of the supports branch value is resolve expression. If the branch is	e branch is nes up to 32 d by the lin taken, the r	the 2's comp 2K instruction ker from the new address	lement numb is forward or l supplied labe will be (PC+2	er '2*Slit16', backward. Th I, absolute ac ) + 2*Slit16, s	which ne Slit16 ddress or since the
	PC will have inc becomes a two The 'n' bits are	-cycle instru a 16-bit sig	uction, with a	NOP execute	d in the seco	nd cycle.
	instruction word	IS.				
Words:	1					
Cycles:	1 (2 if branch ta	ken)				
·	002000 HERE: 002002 NO_C: 002004 002006 002008 CARRY: 00200A 00200C THERE: 00200E	BRA C,  GOTO TH  	;	If C is se Otherwise.		
	Before			After		
	Instruction		li	nstruction		
	PC 00 2000 SR 0001 (	C=1)	PC SR	00 2008 0001 <b>(C</b>	C=1)	
Example 2	002000 HERE: 002002 NO_C: 002004 002006 002008 CARRY: 00200A 00200C THERE: 00200E	BRA C,  GOTO TH  	;	If C is se Otherwise.		
	Bef	ore		Aft	er	
	Instruct		_	Instructi		
	PC 00 2000		PC	00 2002		
	SR 0000		SR	0000		

Instruction Descriptions

# dsPIC30F Programmer's Reference Manual



Syntax:	{label:} BRA GE, Expr
Symax.	{iabel.} DHA GE, Expl
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].
Operation:	Condition = (N&&OV)  (!N&&!OV) If (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register
Status Affecte	d: None
Encoding:	0011 1101 nnnn nnnn nnnn nnnn
Description:	If the logical expression (N&&OV)  (!N&&!OV) is true, then the program will branch relative to the next PC. The offset of the branch is the 2'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.
	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.
	<b>Note:</b> The assembler will convert the specified label into the offset be used.
Words:	1
Cycles:	1 (2 if branch taken)
Example 1	007600 LOOP: 007602 007604 007606 007608 HERE: BRA GE, LOOP ; If GE, branch to LOOP 00760A NO_GE: ; Otherwise continue
	Before After Instruction Instruction
	PC 00 7608 PC 00 7600
	SR         0000         SR         0000
Example 2	007600 LOOP: 007602 007604 007606 007608 HERE: BRA GE, LOOP ; If GE, branch to LOOP 00760A NO_GE: ; Otherwise continue
	Before After
	BeforeAfterInstructionInstructionPC007608PC00760A

BRA G	EU		Branch if	Unsigned (	Greater Than	or Equal	
Syntax:		{label:}	BRA	GEU,	Expr		
Operands:		Expr is res	solved by th		lress or expre Slit16 offset t am words.		an offset
Operation:			-				
Status Affected	:	None					
Encoding:		0011	0001	nnnn	nnnn	nnnn	nnnn
Description:		PC. The o which sup The Slit16	ffset of the ports branc	branch is the hes up to 32 solved by the	gram will bran 2's complem K instructions a linker from th	ent number forward or b	'2*Slit16', ackward.
		the PC wil	I have incre then becor	mented to f	dress will be (I etch the next i vcle instruction	nstruction. T	he
		The 'n' bits in instructi		it signed lite	al that specify	the offset fro	om (PC+2)
		Note:	Carry) inst		ntical to the nas the same Slit16.		
Words:		1					
Cycles:		1 (2 if brar	nch taken)				
Example 1	00200 00200 00200 00200 00200	6 8 A C BYPASS:	   GOTO I	U, BYPASS HERE	; to	C is set, BYPASS herwise	
	I PC SR	Before nstruction 00 2000 0001	(C=1)	PC SR	After Instruction 00 200C 0001 ((	C=1)	

BRA GT	• в	Branch if Signed Greater Than						
Syntax:	{label:} B	RA	GT,	Expr				
Operands:	Expr may be Expr is resolv Slit16 ∈ [-327	ed by the li	nker to a Sl		sion.			
Operation:		Z & N & O' Slit16 $\rightarrow$ P struction Re	PC	&&!OV)				
Status Affected:	None							
Encoding:	0011	1100	nnnn	nnnn	nnnn	nnnn		
Description:	program will b 2's compleme instructions for	If the logical expression (!Z&&N&&OV)  (!Z&&!N&&!OV) is true, then the program will branch relative to the next PC. The offset of the branch is the 2'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.						
	If the branch the PC will ha then become cycle.	ave increme	nted to fetcl	n the next ins	truction. The	instruction		
	The 'n' bits ar instruction wo		gned literal	that specify tl	he offset from	ı (PC+2) in		
Words:	1							
Cycles:	1 (2 if branch	taken)						
	002000 HERE: 002002 NO_GT: 002004 002006 002008 00200A 00200C BYPASS: 00200E	BRA GT,   GOTO TH 	. BYPASS IERE		GT, branch erwise			
	Before			After				
	Instruction			nstruction				
	PC 00 2000 SR 0001	(C=1)	PC SR	00 200C 0001 (C	C=1)			

BRA G	ίTU		Branch if U	nsigned Gr	eater Than		
Syntax:		{label:}	BRA	GTU,	Expr		
Operands:		Expr is reso	e a label, ab blved by the l 2768 +327	inker to a SI	ss or express it16, where	ion.	
Operation:							
Status Affecte	ed:	None					
Encoding:		0011	1110	nnnn	nnnn	nnnn	nnnn
Description:		relative to t number '2*' or backwar	he next PC. Slit16', which	The offset of supports bra value is res	rue, then the the branch is anches up to 3 olved by the li n.	the 2's comp 32K instruction	lement ns forward
		the PC will	have increme	ented to fetc	ess will be (PC h the next ins n, with a NOP	truction. The	instruction
		The 'n' bits offset from		literal that s	pecifies the n	umber of inst	ructions
Words:		1					
Cycles:		1 (2 if brand	ch taken)				
Example 1	00200 00200 00200 00200 00200	96 98 9A 9C BYPASS:	   GOTO TH	J, BYPASS HERE		U, branch wise co	
	PC SR	Before Instruction 00 2000 0001	(C=1)		After struction	1)	

BRA L	1	Branch if Signed Less Than or Equal				
Syntax:	{label:}	BRA	LE,	Expr		
Operands:	• •	lved by the	absolute addre e linker to a SI 2767].		ion.	
Operation:	If (Conditior (PC+2) +					
Status Affected	: None					
Encoding:	0011	0100	nnnn	nnnn	nnnn	nnnn
Description:	program wil 2's compler instructions	l branch re nent numb forward or	n (Z  (N&&!O) lative to the ne er '2*Slit16', w backward. Th I, absolute add	ext PC. The of hich supports e Slit16 value	ffset of the bra branches up is resolved b	anch is the to 32K
	PC will have becomes a	e incremen two-cycle i	the new addre ted to fetch the nstruction, wit	e next instruct h a NOP exect	ion. The instruuted in the se	uction ther cond cycle
	The 'n' bits offset from		ed literal that s	pecifies the n	umber of inst	ructions
Words:	1					
Cycles:	1 (2 if brand	h taken)				
Example 1	002000 HERE: 002002 NO_LE: 002004 002006 002008 00200A 00200C BYPASS 00200E	   GOTO	LE, BYPASS	BY	LE, branch PASS herwise	
	Before Instruction PC 00 2000 SR 0001	(C=1)	PC [ SR ]	After Instruction 00 2002 0001 (	C=1)	

BRA L	Branch if Unsigned Less Than or Equal				
Syntax:	{label:} BRA LEU, Expr				
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].				
Operation:	Condition = $ C  Z$ If (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register				
Status Affected	d: None				
Encoding:	0011 0110 nnnn nnnn nnnn nnnn				
Description:	If the logical expression (!C  Z) is true, then the program will branch relative to the next PC. The offset of the branch is the 2'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.				
	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 offset from (PC+2).				
Words:	1				
Cycles:	1 (2 if branch taken)				
Example 1	002000 HERE:       BRA LEU, BYPASS       ; If LEU, branch to BYPA         002002 NO_LEU:        ; Otherwise continue         002004        ;         002006        ;         002008        ;         00200C BYPASS:        ;         00200E        ;				
	Before         After           Instruction         Instruction           PC         00 2000         PC         00 200C           SR         0001         (C=1)         SR         0001         (C=1)				

DS70030F-page 5-40

BRA L	Гв	Branch if Si	gned Less	Than			
Syntax:	{label:} B	RA	LT,	Expr			
Operands:	Expr may be Expr is resolv Slit16 ∈ [-327	ed by the li	nker to a Sli		ion.		
Operation:	If (Condition) (PC+2) + 2	Condition = $(N\&\&!OV)  (!N\&&OV)$ If (Condition) $(PC+2) + 2*Slit16 \rightarrow PC$ $NOP \rightarrow Instruction Register$					
Status Affected	: None						
Encoding:	0011	0101	nnnn	nnnn	nnnn	nnnn	
	complement instructions fo linker from th If the branch the PC will ha then become cycle.	orward or ba e supplied I is taken, the ave increme	ackward. Th abel, absolu e new addre nted to fetch	e Slit16 value te address or ss will be (PC n the next insi	e is resolved l r expression. C+2) + 2*Slit1 truction. The	by the 6, since instruction	
	The 'n' bits a offset from (F	-	literal that s	pecifies the n	umber of inst	ructions	
Words:	1						
Cycles:	1 (2 if branch	taken)					
Example 1	002000 HERE: 002002 NO_LT: 002004 002006 002008 00200A 00200C BYPASS: 00200E	BRA LT    GOTO TI 	, BYPASS HERE	-	T, branch rwise c		
	Before Instruction PC 00 2000 SR 0001	(C=1)		After struction	=1)		

BRA L	U Branch if Unsigned Less Than							
Syntax:	{label:} BRA LTU, Expr							
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].							
Operation:	Condition = !C If (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register							
Status Affected	None							
Encoding:	0011 1001 nnnn nnnn nn	nn						
Description:	The offset of the branch is the 2's complement number '2*Slit16', wh supports branches up to 32K instructions forward or backward. The S	If the Carry flag is '0', then the program will branch relative to the next PC. The offset of the branch is the 2'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.						
	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 instructio offset from (PC+2).	ns						
	Note: This instruction is identical to the BRA NC, Expr (Branch i Carry) instruction and has the same encoding. It will revassemble as BRA NC, Slit16.							
Words:	1							
Cycles:	1 (2 if branch taken)							
Example 1	002000 HERE:       BRA LTU, BYPASS ; If LTU, branch to B         002002 NO_LTU:          002004          002006          002008          002000A       GOTO THERE         00200C       BYPASS:							
	Before Instruction         After Instruction           PC         00 2000 0001 (C=1)         PC         00 2002 0001 (C=1)							

<b>BRA N</b>	Br	anch if Negati	ive					
Syntax:	{label:} BF	RA N,	Exp	ſ				
Operands:	Expr may be a Expr is resolve Slit16 $\in$ [-3276]	ed by the linker			on.			
Operation:		Slit16 $\rightarrow$ PC ruction Registe	er.					
Status Affected	: None							
Encoding:	0011	0011 r	innn n	nnn	nnnn	nnnn		
Description:	PC. The offset which supports Slit16 value is	If the Negative flag is '1', then the program will branch relative to the next PC. The offset of the branch is the 2'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.						
	If the branch is PC will have in becomes a two	cremented to fe	etch the next	instructi	on. The instru	uction ther		
	The 'n' bits are offset from (PC	•	al that specifie	es the nu	umber of inst	ructions		
Words:	1							
Cycles:	1 (2 if branch t	aken)						
Example 1	002000 HERE: 002002 NO_N: 002004 002006 002008 00200A 00200C BYPASS: 00200E	BRA N, BYF   GOTO THERE 		-	N, branch t erwise (			
	Before Instruction PC 00 2000 SR 0008 (N	 N=1)			=1)			

BRA N	IC		Branch if N	lot Carry				
Syntax:		{label:}	BRA	NC,	Expr			
Operands:		Expr is reso	,	linker to a Sli	ss or express t16, where	ion.		
Operation:			-					
Status Affecte	ed:	None						
Encoding:		0011	1001	nnnn	nnnn	nnnn	nnnn	
		If the Carry flag is '0', then the program will branch relative to the next PC. The offset of the branch is the 2'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. If the branch is taken, the new address will be $(PC+2) + 2*Slit16$ , since the						
		PC will hav	e incremente	ed to fetch the	next instructi	ion. The instru	uction then	
		The 'n' bits offset from	0	l literal that s	pecifies the n	umber of inst	ructions	
Words:		1						
Cycles:		1 (2 if bran	ch taken)					
Example 1	0020 0020 0020 0020 0020	006 008 00A 00C BYPASS	   GOTO I	:, BYPASS 'HERE	-	C, branch t rwise co		
	PC SR	Before Instruction 00 2000 0001	) (C=1)		After struction	=1)		

BRA N	IN		Branch if	Not Negative	9		
Syntax:	{	[label:}	BRA	NN,	Expr		
Operands:	I	Expr is res		linker to a S	ess or express lit16, where	sion.	
Operation:							
Status Affecte	ed: I	None					
Encoding:		0011	1011	nnnn	nnnn	nnnn	nnnn
	s a I t	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.	are a signe	,	specifies the n		
Words:	-	1					
Cycles:		1 (2 if bran	ich taken)				
Example 1	002002 002004 002006 002008 002008	S A C BYPASS	   GOTO I	I, BYPASS HERE	-	branch to	
	Ir PC SR	Before nstruction 00 2000 0000	]	Ins	After truction 0 200C 0000		

BRA N	<b>JOV</b> Branch if Not Overflow
Syntax:	{label:} BRA NOV, Expr
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].
Operation:	Condition = $!OV$ If (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register
Status Affecte	ed: None
Encoding: Description:	00111000nnnnnnnnnnnnIf the Overflow flag is '0', then the program will branch relative to the nextPC. The offset of the branch is the 2's complement number '2*Slit16',which supports branches up to 32K instructions forward or backward. TheSlit16 value is resolved by the linker from the supplied label, absoluteaddress 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 offset from (PC+2).
Words:	1
Cycles:	1 (2 if branch taken)
Example 1	002000 HERE:       BRA NOV, BYPASS ; If NOV, branch to BYPASS         002002 NO_NOV:          002004          002006          002008          00200A       GOTO THERE         00200C       BYPASS:         00200E
	Before         After           Instruction         Instruction           PC         00 2000         PC         00 200C           SR         0008         (N=1)         SR         0008

BRA N	Z	Branch if N	ot Zero			
Syntax:	{label:}	BRA	NZ,	Expr		
Operands:	Expr is res	be a label, ab blved by the 2768 +327	inker to a SI	ss or express it16, where	ion.	
Operation:						
Status Affected	l: None					
Encoding:	0011	1010	nnnn	nnnn	nnnn	nnnn
	value is res expression If the branc PC will hav then becon	olved by the h is taken, th e incremente	linker from the e new addre ed to fetch th	s forward or b ne supplied lat ss will be (PC e next instruc n, with a NOP	bel, absolute +2) + 2*Slit16 tion. The inst	address o 6, since the ruction
	cycle. The 'n' bits offset from	•	literal that s	pecifies the n	umber of inst	ructions
Words:	1					
Cycles:	1 (2 if bran	ch taken)				
Example 1	002000 HERE: 002002 NO_NZ: 002004 002006 002008 00200A 00200C BYPASS 00200E	   GOTO TI	, BYPASS HERE		, branch t wise co	
	Before Instruction PC 00 2000 SR 0002	_	Ins	After struction	1)	

BRA (	<b>DA</b> Branch if Overflow Accumulator A
Syntax:	{label:} BRA OA, Expr
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].
Operation:	Condition = OA If (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register
Status Affect	ed: None
Encoding:	0000 1100 nnnn nnnn nnnn nnnn
Description:	If the Overflow Accumulator A flag is '1', then the program will branch relative to the next PC. The offset of the branch is the 2'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.
	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 offset from (PC+2).
	<b>Note:</b> The assembler will convert the specified label into the offset to be used.
Words:	1
Cycles:	1 (2 if branch taken)
Example 1	002000 HERE:       BRA OA, BYPASS       ; If OA, branch to BYPASS         002002 NO_OA:       .       .         002004       .       .         002006       .       .         002008       .       .         00200A       GOTO THERE         00200C       BYPASS:       .         00200E       .       .
	Before         After           Instruction         Instruction           PC         00 2000         PC         00 200C           SR         8800         (OA, OAB=1)         SR         8800         (OA, OAB=1)

BRA O	B		Branch if	Overflow	Acc	umulator B		
Syntax:		{label:}	BRA	OB,		Expr		
Operands:		Expr is res	be a label, a solved by th 32768 +3	e linker to		ss or express 16, where	ion.	
Operation:		. ,						
Status Affecte	d:	None						
Encoding:		0000	1101	nnn	n	nnnn	nnnn	nnnn
Description:		ative to the ber '2*Slit backward.	e next PC. 1 16', which s	The offset upports br value is re	of the anch solve	e branch is th es up to 32K ed by the link	program will e 2's comple instructions f er from the si	ment num- orward or
		the PC wil	I have incre	mented to	fetch	the next inst	C+2) + 2*Slit1 truction. The executed in t	instruction
		The 'n' bits offset from		ed literal th	nat sp	pecifies the n	umber of inst	ructions
Words:		1						
Cycles:		1 (2 if brar	nch taken)					
Example 1	00200 00200 00200 00200 00200	D6 D8 DA DC BYPASS	с      	DB, BYPAS THERE	SS		3, branch † rwise co	
		Before				After		
		Instruction	·			truction		
	PC	00 200	-	PC	0	0 2002		
	SR	880	0 <b>(OA, OAB</b>	8=1) SR		8800 <b>(O</b> A	A, OAB=1)	

BRA O	/	Branch if	Overflow			
Syntax:	{label:}	BRA	OV,	Expr		
Operands:		olved by the	e linker to a	ress or express Slit16, where	ion.	
Operation:		-				
Status Affected:	None					
Encoding:	0011	0000	nnnn	nnnn	nnnn	nnnn
Description:	PC. The off which supp	set of the lorts branch is resolve	oranch is the nes up to 32I d by the link	program will bra 2's complemer K instructions fo er from the supp	nt number '2*S rward or back	Slit16', ward. The
	PC will have	e incremen	ted to fetch t	ress will be (PC he next instruct vith a NOP exect	ion. The instru	uction then
	The 'n' bits offset from		ed literal that	specifies the n	umber of inst	ructions
Words:	1					
Cycles:	1 (2 if brand	ch taken)				
	002000 HERE: 002002 NO_OV 002004 002006 002008 002008 00200A 00200C BYPASS 00200E	   GOTO	OV, BYPASS THERE		V, branch t rwise co	
	Before Instruction PC 00 2000 SR 0002		PC SR	After Instruction	-1)	

<b>BRA S</b>	Α	Branch if Sa	aturation A	ccumulator A	۱.	
Syntax:	{label:}	BRA	SA,	Expr		
Operands:	Expr is reso	e a label, abs blved by the li 2768 +327	nker to a S	ess or express lit16, where	ion.	
Operation:	· · · · ·					
Status Affecte	I: None					
Encoding:	0000	1110	nnnn	nnnn	nnnn	nnnn
	number '2*S or backward label, absol	Slit16', which d. The Slit16 ute address o	supports br value is res or expressio		32K instructio nker from the	ns forward supplied
	PC will have	e incremente	d to fetch th	ess will be (PC- ne next instruc on, with a NOP	tion. The inst	ruction
	The 'n' bits offset from	-	literal that s	specifies the n	umber of inst	ructions
Words:	1					
Cycles:	1 (2 if brand	ch taken)				
Example 1	002000 HERE: 002002 NO_SA: 002004 002006 002008 00200A 00200C BYPASS: 00200E	   GOTO ТН	BYPASS	-	A, branch r	
	Before Instruction PC 00 2000 SR 2400	(SA, SAB=1)	PC	After Instruction	A, SAB=1)	

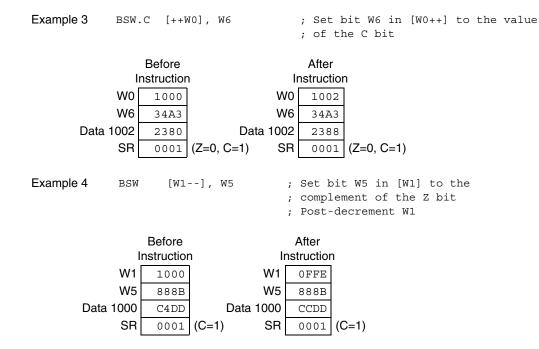
BRA S	B Branch if Saturation Accumulator B					
Syntax:	{label:} BRA SB, Expr					
Operands:	Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].					
Operation:	Condition = SB if (Condition) (PC+2) + 2*Slit16 $\rightarrow$ PC NOP $\rightarrow$ Instruction Register					
Status Affecte	d: None					
Encoding:	0000 1111 nnnn nnnn nnnn nnnn					
Description:	If the Saturation Accumulator B flag is '1', then the program will branch relative to the next PC. The offset of the branch is the 2's complement number '2*Slit16', which supports branches up to 32K instructions forwar 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 instructio then becomes a two-cycle instruction, with a NOP executed in the secon cycle.					
	The 'n' bits are a signed literal that specifies the number of instructions offset from (PC+2).					
Words:	1					
Cycles:	1 (2 if branch taken)					
Example 1	002000 HERE:       BRA SB, BYPASS       ; If SB, branch to BYPASS         002002 NO_SB:        ; Otherwise continue         002004        ;         002006        ;         002008        ;         002000 BYPASS:        ;         00200E        ;					
	Before         After           Instruction         Instruction           PC         00         2000           SR         0000         SR         0000					

BRA Z		Branch if Z	ero			
Syntax:	{label:}	BRA	Ζ,	Expr		
Operands:	Expr is reso		solute addres inker to a Slit '67].		on.	
Operation:						
Status Affected	: None					
Encoding:	0011	0010	nnnn	nnnn	nnnn	nnnn
	supports br value is res expression	anches up to olved by the l	is the 2's con 32K instruction inker from the	ons forward o supplied lab	or backward. <sup>-</sup> el, absolute a	The Slit16 address or
	PC will hav	e incremente	e new address d to fetch the le instruction	next instruct	ion. The instr	uction
	The 'n' bits offset from		literal that sp	ecifies the nu	Imber of instr	uctions
Words:	1					
Cycles:	1 (2 if bran	ch taken)				
Example 1	002000 HERE: 002002 NO_Z: 002004 002006 002008 00200A 00200A 00200C BYPAS 00200E	   GOTO	, BYPASS THERE		Z, branch herwise	
	Before Instruction PC 00 200 SR 000	0	PC SR	After nstruction 00 200C 0002 (2	Z=1)	

BSET		Bit Set f				
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 o peration	peration		
Operation:	$1 \rightarrow f < bit4$	>				
Status Affected:	None					
Encoding:	1010	1000	bbbf	ffff	ffff	fffb
Description:	with the Le	ast Significar	ister f specifi nt bit (bit 0) a ons, bit 15 fo	nd advances	to the Most	
			bit4 of the bi dress of the f		be set.	
		denote a wo When this in address mus	a word operation, rd operation, struction ope at be word ali nstruction opend nd 7.	but it is not r rates in Wor gned.	equired. d mode, the	file register
Words:	1					
Cycles:	1					
Example 1 BSE	ET.B 0x60	1, #0x3	; Set b:	it 3 in Ox	601	
Data 0600 SR	_	Data 06 S	After Instruction 00 FA34 6R 0000			
Example 2 BSE	ET 0x44	4, #0xF	; Set b	it 15 in 0:	x444	
Data 0444 SR		Data 04	After Instruction 44 D604 SR 0000			

BSET		Bit Set in \	Vs			
Syntax:	{label:}	BSET{.B}	Ws,	#bit4		
			[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:		W15] 7] for byte o 15] for word				
Operation:	$1 \rightarrow Ws < b$	it4>				
Status Affected:	None					
Encoding:	1010	0000	bbbb	0B00	0ppp	SSSS
Description:	Least Sign for byte op	ificant bit (bit	s specified by 0) and advar 15 for word op d for Ws.	ices to the Mo	ost Significan	t bit (bit 7
	The 'B' bit The 'p' bits	selects byte select the se	bit4 of the bit or word opera ource Address ddress of the	tion (0 for wo s mode.	rd, 1 for byte	
	2:	rather than denote a wo When this register add	on .B in the a word opera ind operation, instruction op ress must be nstruction op ind 7.	tion. You may but it is not re perates in W word aligned.	y use a .w ex equired. /ord mode, t	xtension t
Words:	1					
Cycles:	1					
Example 1	BSET.B W3,	#0x7	; Set b	it 7 in W3		
	Before		After			
	Instruction			l		
	W3 0026 SR 0000		W3 00A6 SR 0000			
Example 2	BSET [W4++]	, #0x0	-	it 0 in [W4 increment W		
Data 6	Before           Instruction           W4         6700           5700         1734           SR         0000	Data 67	After Instruction W4 6702 '00 1735 SR 0000			

BSW		Bit Write	in Ws			
Syntax:	{label:}	BSW.C BSW.Z	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wb		
Operands:	Ws ∈ [W0 Wb ∈ [W0					
Operation:	<u>For ".C" o</u> C → W <u>For ".Z" o</u>		ault):			
Status Affected:						
Encoding:	1010	1101	Zwww	w000	0ppp	SSSS
	0) and ad Only the f nation bit either reg The 'Z' bit The 'W' bit The 'p' bit	vances to the our Least Sig number. Reg ister direct, o t selects the ts select the s select the	numbering be e Most Signific gnificant bits o gister direct ad r indirect addr C or Z flag as address of the source Addres address of the	ant bit (bit 15) f Wb are used dressing must essing may be source. bit select reg s mode.	of the workin to determine be used for used for We	ng register. the desti- Wb, and
	Note:	This instruc	ction only oper he ". z" operat	ates in Word r	node. If no e	xtension is
Words:	1					
Cycles:	1					
Example 1	BSW.C W2, W	13		bit W3 in W he C bit	12 to the v	value
	Before Instruction W2 F234 W3 111F SR 0002	(Z=1, C=0)	After Instructio W2 7234 W3 111F SR 0002			
Example 2	BSW.Z W2, W	3		oit W3 in W ne Z bit	2 to the c	omplement
	Before           Instruction           W2         E235           W3         0550           SR         0002         (	Z=1, C=0)	After Instructio W2 E234 W3 0550 SR 0002	n       (Z=1, C=0)		



5

BTG		Bit Toggle f				
Syntax:	{label:}	BTG{.B}	f,	#bit4		
Operands:	f ∈ [0 81 bit4 ∈ [0	191] for byte o 190] (even onl . 7] for byte op . 15] for word	y) for word o peration	peration		
Operation:	(f) <bit4> -</bit4>	$\rightarrow$ (f) <bit4></bit4>				
Status Affected:	None					
Encoding:	1010	1010	bbbf	ffff	ffff	fffb
Description:	bit number	file register f i ring begins wit Significant bit (l s.	h the Least S	Significant bit	(bit 0) and ad	dvances to
		s select value select the ad			ggle.	
		rather than a denote a wor When this in address mus When this in between 0 ar	d operation, struction ope t be word ali struction op	but it is not r rates in Wor gned.	equired. d mode, the t	file register
Words:	1					
Cycles:	1					
Example 1 BTG	.B 0x10	001, #0x4	; Toggle	bit 4 in	0x1001	
ا Data 1000 SR	Before nstruction F234 0000	Data 100 SF				
Example 2 BTG	0x16	60, #0x8	; Toggle	bit 8 in 1	RAM660	
lı Data 1660 SR	Before Instruction 5606 0000	Data 1660 SF				

BTG		Bit Toggle	e in Ws			
Syntax:	{label:}	BTG{.B}	Ws,	#bit4		
			[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:		W15] . 7] for byte ( . 15] for word				
Operation:	(Ws) bit4	$\rightarrow$ Ws <bit< td=""><td>4&gt;</td><td></td><td></td><td></td></bit<>	4>			
Status Affected:	None					
Encoding:	1010	0010	bbbb	0B00	0ppp	SSSS
	bit numbe the Most S operations	ring begins v Significant bit s). Register c	vith the Least t (bit 7 for byte direct or indire	omplemented) Significant bit operations, k ct addressing	(bit 0) and ac bit 15 for word may be used	dvances to d
	The 'B' bit The 's' bit	selects byte s select the a	or word oper	position to tes ation (0 for wo source/destir ss mode.	ord, 1 for byte	
	2:	rather than denote a w When this register add	a word opera ord operation instruction of dress must be instruction op	instruction de ation. You ma but it is not re perates in W word aligned perates in Byt	y use a .w e equired. /ord mode, t	xtension t
Words:	1					
Cycles:	1					
Example 1	BTG W2, #02	c0	; Toggl	e bit 0 in	W2	
	Before		After			
	Instruction W2 F234					
	W2 F234 SR 0000		W2 F235 SR 0000			
Example 2	BTG [W0++]	, #0x0		e bit 0 in increment V		
Data 2	Before           Instruction           W0         2300           300         5606           SR         0000	Data 2	After Instruction W0 2302 300 5607 SR 0000			

© 2005 Microchip Technology Inc.

Instruction Descriptions

BTSC	Bit Test f, Skip if Clear
Syntax:	{label:} BTSC{.B} f, #bit4
Operands:	$f \in [0 \dots 8191]$ 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 word operation
Operation:	Test (f) <bit4>, skip if clear</bit4>
Status Affected:	None
Encoding:	1010 1111 bbbf ffff ffff fffb
Description:	Bit 'bit4' in the file register is tested. If the tested bit is '0', the next instruction (fetched during the current instruction execution) is discarded and on the next cycle, a NOP is executed instead. If the tested bit is '1', the next instruction is executed as normal. In either case, the contents of the file register are not changed. For the bit4 operand, bit numbering begins with the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 7 for byte operations, bit 15 for word operations).
	The 'b' bits select value bit4, the bit position to test. The 'f' bits select the address of the file register.
	<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: When this instruction operates in Word mode, the file register address must be word aligned.</li> <li>3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.</li> </ul>
Words:	1
Cycles:	1 (2 or 3)
	002000 HERE:       BTSC.B       0x1201, #2 ; If bit 2 of 0x1201 is 0,         002002       GOTO       BYPASS ; skip the GOTO         002004          002006          002008       BYPASS:         00200A
Data 12	Before         After           Instruction         Instruction           PC         00         2000           00         264F         Data 1200         264F           SR         0000         SR         0000

0 0 0 0 0 0	002000 HERE: 002002 002004 002006 002008 BYPASS: 00200A	BTSC 0x804, GOTO BYPASS   	#14 ; If bit 14 of 0x804 is 0, ; skip the GOTO
	Before Instruction		After Instruction
F	PC 00 2000	PC	00 2004
Data 08	04 2647	Data 0804	2647
S	SR 0000	SR	0000

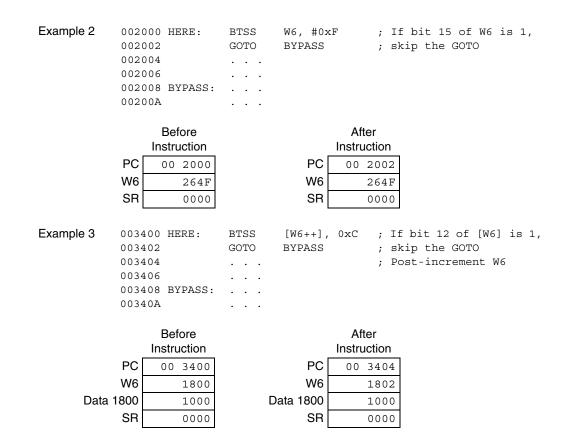
BTSC		Bit Test W	/s, Skip if Cle	ar		
Syntax:	{label:}	BTSC	Ws,	#bit4		
			[Ws],			
			[Ws++],			
			[Ws],			
			[++Ws],			
			[Ws],			
Operands:	Ws ∈ [W0 bit4 ∈ [0 .	-				
Operation:	Test (Ws)	<bit4>, skip i</bit4>	f clear			
Status Affecte	d: None					
Encoding:	1010	0111	bbbb	0000	0ppp	SSSS
	is execute changed. Significan the word. The 'b' bit The 'p' bit	ed as normal For the bit4 t bit (bit 0) ar Either registe s select valu s select the s	ed instead. If the instead. If the instead, bit in operand, bit in advances the divances the direct or incomplete bit4, the bit source Address address of the stion operates.	e, the contents umbering beg o the Most Sig lirect addressi position to tes as mode. source regist	s of Ws are n ins with the L gnificant bit (b ng may be us t. er.	ot east bit 15) of
Words:	1		ction operates		e only.	
Cycles:		f the next ins	truction is skip	oped)		
Example 1	002000 HERE: 002002 002004 002006 002008 BYPAS 00200A	BTSC GOTO  S:	W0, #0x0 BYPASS		bit 0 of W ip the GOTO	-
	Before           Instruction           PC         00         200           W0         264           SR         000	00 ⊧F	PC W0 SR	After Instruction 00 2002 264F 0000		

002 002 002	004 006 008 BYPASS:	BTSC GOTO   	W6, #0xF BYPASS	-	bit 15 of W6 is 0, ip the GOTO
PC W6 SR	264F		PC W6 SR	After Instruction 00 2004 264F 0000	
003 003 003	404 406 408 BYPASS:	BTSC GOTO   	[W6++], BYPASS	; sk:	bit 12 of [W6] is 0, ip the GOTO st-increment W6
	Before			After	
	Instruction			Instruction	
PC	00 3400		PC	00 3402	
W6	1800		W6	1802	
Data 1800	1000	D	ata 1800	1000	
SR	0000		SR	0000	

BTSS		Bit Test f, S	kip if Set			
Syntax:	{label:}	BTSS{.B}	f,	#bit4		
Operands:	f ∈ [0 819 f ∈ [0 819 bit4 ∈ [0 5 bit4 ∈ [0 5	0] (even on 7] for byte op	y) for word op peration	peration		
Operation:	Test (f) <bit4< td=""><td>&gt;, skip if set</td><td></td><td></td><td></td><td></td></bit4<>	>, skip if set				
Status Affected:	None		1			1
Encoding:	1010	1110	bbbf	ffff	ffff	fffb
Description:	instruction ( and on the r next instruct file register with the Lea	fetched durin next cycle, a tion is execu are not char ast Significar	er f is tested. ng the curren NOP is execu ted as norma iged. For the it bit (bit 0) ar on, bit 15 for	t instruction e ted instead. I II. In either ca bit4 operand nd advances	execution) is of f the tested b use, the conte , bit numberin to the Most S	discarded it is '0', the ents of the ng begins
			bit4, the bit p dress of the f		t.	
	2: \ 2: \ 3: \	ather than a denote a wo When this in address mus	on . B in the a word operation, struction ope to be word alion instruction ope and 7.	tion. You may but it is not re rates in Word gned.	/ use a .w e equired. d mode, the t	xtension to file register
Words:	1					
Cycles:	1 (2 or 3 if th	ne next instr	uction is skip	ped)		
. 00'	7100 HERE: 7102 7104	BTSS.B CLR · · ·	0x1401, WREG	#0x1; If k ; don'	oit 1 of 0x t clear WF	
PC Data 1400 SF	0280	D		After struction 00 7104 0280 0000		
00'	7100 HERE: 7102 7104 7106 BYPASS:	BTSS GOTO · · ·	0x890, = BYPASS	#0x9 ; If ] ; ski]	oit 9 of 02 o the GOTO	x890 is 1,
PC Data 0890 SF	0 OOFE		Ir PC Data 0890 SR	After nstruction		

BTSS		Bit Test W	/s, Skip if Set			
Syntax:	{label:}	BTSS	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	#bit4		
Operands:	Ws ∈ [W0 bit4 ∈ [0					
Operation:	Test (Ws)	<bit4>, skip i</bit4>	f set.			
Status Affecte	d: None					
Encoding:	1010	0110	bbbb	0000	0ppp	SSSS
	is execute changed. Significan the word. The 'b' bit The 's' bit	d as normal For the bit4 t bit (bit 0) ar Either registr s select the a s select the a	ed instead. If the In either case operand, bit number advances to er direct or ind value bit4, the address of the source Addres	e, the contents umbering beg o the Most Sig irect addressi bit position to source regist	s of Ws are n ins with the L gnificant bit (b ng may be us test.	ot east vit 15) of
	Note:	This instruc	ction operates	in Word mod	e only.	
Words: Cycles:	1 1 (2 or 3 if	the next ins	truction is skip	pped)		
Example 1	002000 HERE: 002002 002004 002006 002008 BYPAS: 00200A	BTSS GOTO   5:	W0, #0x0 BYPASS	-	bit 0 of W ip the GOTC	-
	PC 00 200 W0 264 SR 000	0 F	PC W0 SR	After Instruction 00 2004 264F 0000		

## dsPIC30F Programmer's Reference Manual



BTST		Bit Test f				
Syntax:	{label:}	BTST{.B}	f,	#bit4		
Operands:	f ∈ [0 8 bit4 ∈ [0	191] for byte o 190] (even onl . 7] for byte op . 15] for word	y) for word o peration	peration		
Operation:	(f) <bit4> -</bit4>	→ Z				
Status Affected:	Z					
Encoding:	1010	1011	bbbf	ffff	ffff	fffb
	are not cha Least Sigr for byte op	he Z flag in the anged. For the hificant bit (bit beration, bit 15	e bit4 operan 0) and advar 6 for word ope	d, bit numbe nces to the M eration).	ring begins w lost Significar	ith the
		s select value select the ad			lested.	
		denote a wor When this in address mus When this in between 0 ar	struction ope t be word ali struction op	rates in Wor gned.	d mode, the f	
Words:	1					
Cycles:	1					
Example 1 BT	ST.B 0x12	201, #0x3		= compleme in 0x1201	ent of	
Data 1200 SF		Data 120 S		(Z=1)		
Example 2 BT	ST 0x1	302, #0x7		= complem in 0x1302		
Data 130 Sl		Data 13	After Instruction 02 F7FF SR 0000			

BTST		Bit Test in V	Vs			
Syntax:	{label:}	BTST.C BTST.Z	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	#bit4		
Operands:	Ws ∈ [W0 bit4 ∈ [0	-				
Operation:	For ".C" op (Ws) <bit For ".Z" op (Ws)<bit< td=""><td><math>4 \rightarrow C</math> eration (defau</td><td><u>lt):</u></td><td></td><td></td><td></td></bit<></bit 	$4 \rightarrow C$ eration (defau	<u>lt):</u>			
Status Affected:	Z or C					
Encoding:	1010	0011	bbbb	Z000	0ppp	SSSS
	Status regist the tested case, the c For the bit4 (bit 0) and register dir The 'b' bits The 'Z' bits The 'p' bits The 's' bits	he compleme ster. If the ". C bit is stored to contents of Ws 4 operand, bit advances to t ect or indirect select value selects the C select the so select the ad	" option of the the Carry fla are not char numbering be he Most Sign addressing n bit4, the bit po or Z flag as d urce Address dress of the s	e instruction i g in the Statu ged. egins with the ificant bit (bit nay be used osition to test estination. mode. source registe	s specified, th us register. In e Least Signif 15) of the wo for Ws. 	ne value of either ficant bit ord. Either
	Note:	This instruction provided, the				xtension is
Words:	1					
Cycles:	1					
Example 1 BTS	ST.C [W0+	+], #0x3		= bit 3 in ncrement W		
W0 Data 1200 SR	FFF7	W Data 120 ⊊=1) S	0 FFF7			
Example 2 BT	ST.Z WO,	#0x7	; Set Z =	= complemen	nt of bit 7	7 in W0
W( SF		W		Z=1)		

Syntax:{label:}BTST.CWs,WbBTST.Z[Ws], [Ws++], [Ws], [++Ws], [Ws],Operands:Ws $\in$ [W0 W15] Wb $\in$ [W0 W15]Operation:For ".C" operation: (Ws)<(Wb)> $\rightarrow$ CFor ".Z" operation (default): (Ws)<(Wb)> $\rightarrow$ ZStatus Affected:Z or CEncoding:10100101Description:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect address of the bit sector The 'p' bits select the address of the bit sector The 's' bits select the address of the source The 's' bits select the address of the source 			
$[Ws++], [Ws], [++Ws], [-Ws], [Ws], [Wb] > \rightarrow CFor ".C" operation: (Ws)<(Wb) > \rightarrow CFor ".Z" operation (default): (Ws)<(Wb) > \rightarrow ZStatus Affected: Z or CEncoding: 1010 0101 Zwww wDescription: The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Least advances to the Most Significant bit (bit 15)Register direct or indirect addressing mayThe 'Z' bit select the address of the bit set The 'p' bits select the address of the source Address mode the 'p' bits select the address of the source Moters in provided, the ". z" operation isWords: 1Cycles: 1Example 1 BTST.C W2, W3 ; Set C = bBefore After$			
$[Ws], [++Ws], [-Ws], [-Ws], \\ Operands: Ws \in [W0 W15] Wb \in [W0 W15] Operation: For ".C" operation: (Ws)<(Wb)> \rightarrow CFor ".Z" operation (default): (Ws)<(Wb)> \rightarrow ZStatus Affected: Z or CEncoding: 1010 0101 Zwww wDescription: The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in the the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Least advances to the Most Significant bit (bit 15) Register direct or indirect addressing mayThe 'Z' bit select the address of the bit set The 'p' bits select the address of the bit set The 'p' bits select the address of the source Address modes the 's' bits select the address of the source Moders in provided, the ". z" operation isWords: 1Cycles: 1Example 1 BTST.C W2, W3 ; Set C = bBefore After$			
$[++Ws], [Ws],$ Operands: Ws $\in$ [W0 W15] Wb $\in$ [W0 W15] Operation: For ".C" operation: (Ws)<(Wb)> $\rightarrow$ C For ".Z" operation (default): (Ws)<(Wb)> $\rightarrow$ Z Status Affected: Z or C Encoding: 1010 0101 Zwww w Description: The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is stored register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed. Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect addressing may The 'Z' bit select the address of the bit set The 'p' bits select the address of the source Note: This instruction only operates i provided, the ". z" operation is Words: 1 Cycles: 1 Example 1 BTST.C W2, W3 ; Set C = b Before After			
[Ws], Operands: Ws ∈ [W0 W15] Wb ∈ [W0 W15] Operation: For ".C" operation: (Ws)<(Wb)> → C For ".Z" operation (default): (Ws)<(Wb)> → Z Status Affected: Z or C Encoding: 1010 0101 Zwww w Description: The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in the the contents of Ws are not changed. Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lead advances to the Most Significant bit to the Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as desting The 'y' bits select the address of the bit set The 'y' bits select the address of the source Note: This instruction only operates i provided, the ". z" operation is Words: 1 Cycles: 1 Example 1 BTST.C W2, W3 ; Set C = b Before After			
Operands: $Ws \in [W0 \dots W15]$ $Wb \in [W0 \dots W15]$ Operation:For ".C" operation: $(Ws) < (Wb) > \rightarrow C$ For ".Z" operation (default): $\overline{(Ws)} < (Wb) > \rightarrow Z$ Status Affected:Z or CEncoding:10100101ZwwwDescription:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". Z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lei advances to the Most Significant bit (bit 16 Register direct or indirect addressing may The 'Z' bit select the address of the bit sec The 'p' bits select the address of the bit sec The 'p' bits select the address of the source Note:Note:This instruction only operates i provided, the ". Z" operation isWords:1Cycles:1Example 1BTST.C W2, W3; Set C = b Before			
Wb $\in [W0 \dots W15]$ Operation:For ".C" operation: (Ws)<(Wb)> $\rightarrow$ C For ".Z" operation (default): (Ws)<(Wb)> $\rightarrow$ ZStatus Affected:Z or CEncoding:10100101ZwwwDescription:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ".Z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect addressing may The 'Z' bit select the address of the bit set The 'p' bits select the address of the bit set The 'p' bits select the address of the source Note:Note:This instruction only operates i provided, the ".z" operation isWords:1Cycles:1Example 1BTST.C W2, W3; Set C = b After			
$\overline{(Ws)<(Wb)> \rightarrow C}$ For ".Z" operation (default): $\overline{(Ws)<(Wb)>} \rightarrow Z$ Status Affected:Z or CEncoding:10100101ZwwwDescription:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". Z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect addressing may The 'Z' bit select the address of the bit set The 'p' bits select the address of the source The 's' bits select the address of the source The 's' bits select the address of the source The 's' bits select the address of the source Note: This instruction only operates i provided, the ". Z" operation isWords:1Cycles:1Example 1BTST.C W2, W3BeforeAfter			
For ".Z" operation (default): (Ws)<(Wb)> $\rightarrow$ ZStatus Affected:Z or CEncoding:10100101ZwwwDescription:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". Z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 16 Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as destin The 'w' bits select the address of the bit set The 'p' bits select the address of the source Note:Note:This instruction only operates i provided, the ". Z" operation isWords:1Cycles:1Example 1BTST.C W2, W3geforeAfter			
$\overline{(Ws)<(Wb)>} \rightarrow Z$ Status Affected:Z or CEncoding:10100101 $\overline{Zwww}$ Description:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 16 Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as destin The 'w' bits select the address of the bit set The 'p' bits select the address of the source Note:Note:This instruction only operates i provided, the ". z" operation isWords:1Cycles:1Example 1BTST.C W2, W3BeforeAfter			
Status Affected:       Z or C         Encoding:       1010       0101       Zwww       w         Description:       The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". Z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.         Only the four Least Significant bits of Wb a number. Bit numbering begins with the Least advances to the Most Significant bit (bit 15) Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as destin The 'w' bits select the address of the bit set The 'p' bits select the address of the source Address mod The 's' bits select the address of the source Note:         Note:       This instruction only operates i provided, the ". Z" operation is         Words:       1         Cycles:       1         Example 1       BTST.C W2, W3       ; Set C = b         Before       After			
Description:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as destin The 'w' bits select the address of the bit se The 'p' bits select the address of the bit se The 'p' bits select the address of the source Note: This instruction only operates i provided, the ". z" operation isWords:1Cycles:1Example 1BTST.C W2, W3; Set C = bBeforeAfter			
Description:The (Wb) bit in register Ws is tested. If the specified, the value of the tested bit is store register. If the ". z" option of the instruction the tested bit is stored to the Zero flag in th the contents of Ws are not changed.Only the four Least Significant bits of Wb a number. Bit numbering begins with the Lea advances to the Most Significant bit (bit 15 Register direct or indirect addressing may The 'Z' bit selects the C or Z flag as destin The 'w' bits select the address of the bit se The 'p' bits select the address of the bit se The 'p' bits select the address of the source Note: This instruction only operates i provided, the ". z" operation isWords:1Cycles:1Example 1BTST.C W2, W3; Set C = bBeforeAfter	w000	0ppp	SSSS
The 'Z' bit selects the C or Z flag as desting The 'w' bits select the address of the bit set The 'p' bits select the source Address more The 's' bits select the address of the source Note: This instruction only operates i provided, the ". z" operation is         Words:       1         Cycles:       1         Example 1       BTST.C W2, W3       ; Set C = b         Before       After	are used east Signi	to determin	e the bit 0) and
The 'w' bits select the address of the bit set The 'p' bits select the source Address mode The 's' bits select the address of the source Note: This instruction only operates in provided, the ". z" operation is         Words:       1         Cycles:       1         Example 1       BTST.C W2, W3       ; Set C = b         Before       After	y be used	for Ws.	
provided, the ". z" operation is Words: 1 Cycles: 1 Example 1 BTST.C W2, W3 ; Set C = b Before After	select regi ode.		
Cycles: 1 Example 1 BTST.C W2, W3 ; Set C = b Before After			extension i
Example 1 BTST.C W2, W3 ; Set C = b Before After			
Before After			
	bit W3 d	of W2	
Instruction Instruction			
W2         F234         W2         F234           W3         2368         W3         2368			

SR 0000

# **Section 5. Instruction Descriptions**

SR

0001 (C=1)

Instruction Descriptions 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
Syntax:	{label:} BTSTS{.B} f, #bit4
Operands:	$f \in [0 \dots 8191]$ 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 word operation
Operation:	$\overline{(f) < bit4>} \rightarrow Z$ 1 $\rightarrow$ (f) < bit4>
Status Affected:	Z
Encoding:	1010 1100 bbbf ffff ffff fffb
Description:	Bit 'bit4' in file register f is tested and the complement of the tested bit is stored to the Zero flag in the Status register. The tested bit is then set to "1" in the file register. For the bit4 operand, bit numbering begins with the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 7 for byte operations, bit 15 for word operations).
	The 'b' bits select value bit4, the bit position to test/set. The 'f' bits select the address of the file register.
	<ul> <li>rather than a word operation. You may use a .w extension to denote a word operation, but it is not required.</li> <li>2: When this instruction operates in Word mode, the file register address must be word aligned.</li> <li>3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.</li> </ul>
Words:	1
Cycles:	1
Example 1 BTST	TS.B 0x1201, #0x3 ; Set Z = complement of bit 3 in 0x1201, ; then set bit 3 of 0x1201 = 1
ا Data 1200 SR Example 2 BTS	
RAM300 SR	BeforeAfterInstructionInstruction8050RAM3000002(Z=1)SR0000

BTSTS		Bit Test/Se	et in Ws			
Syntax:	{label:}	BTSTS.C BTSTS.Z	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	#bit4		
Operands:	Ws ∈ [W0 bit4 ∈ [0					
Operation:	$1 \rightarrow Ws$ For ".Z" of	$it4 \rightarrow C$ s <bit4> peration (defa <math>it4 \rightarrow Z</math></bit4>	<u>uult):</u>			
Status Affected:	Z or C					
Encoding:	1010	0100	bbbb	Z000	0ppp	SSSS
Description:	specified, Status reg the tested cases, the The 'b' bit The 'Z' bit The 'p' bit	the complem ister. If the ". bit is stored to tested bit in s select the v selects the C s select the s	is tested. If th ent of the test C" option of th to the Carry fl Ws is set to " alue bit4, the or Z flag as ource Addres ddress of the	ted bit is store ag in the Stat 1". bit position to destination. s mode.	ed to the Zerc is specified, t us register. Ir o test/set.	flag in the he value of
	Note:		tion only oper e ". z" operat			extension is
Words:	1					
Cycles:	1					
Example 1 BTS	TS.C [WO	++], #0x3	; Set b	= bit 3 i bit 3 in [W increment	0] = 1	
ا W0 Data 1200 SR	Before nstruction 1200 FFF7 0001 (C	Data 1	After Instruction W0 1202 200 FFFF SR 0000	1		
Example 2 BT	STS.Z WO	, #0x7		= compleme , and set b		
W( SF			After Instruction W0 F2BC SR 0002	(Z=1)		

CALL		Call Subr	outine			
Syntax:	{label:}	CALL	Expr			
Operands: Operation:	• •	blved by the $PC$ $\rightarrow$ (TOS) $\rightarrow$ W15 $\rightarrow$ ) $\rightarrow$ (TOS) $\rightarrow$ W15				3606].
Status Affected:	None		giotor			
Encoding:						
1st word	0000	0010	nnnn	nnnn	nnnn	nnn0
2nd word	0000	0000	0000	0000	0nnn	nnnn
Description:	memory (PC+4) is	range. Befo pushed or	Il over the enti- bre the call is m nto the stack. A 3' is loaded inte	hade, the 24- fter the return	bit return add	lress
	The 'n' bits	form the ta	rget address.			
		The linker v be used.	will resolve the	specified exp	pression into	the lit23 to
Words:	2					
Cycles:	2					
Example 1 0260	004	CALL MOV	_FIR W0, W1	; Cal	l _FIR sub	routine
0268 0268	344 _FIR: 346	 MOV # 	0x400, W2	; _FI	R subrouti	ne start
	Before			After		
	Instruction	-	Ir	nstruction		
PC	02 6000	_	PC	02 6844		
W15	A268	_	W15	A26C		
Data A268	FFFF	-	Data A268	6004		
Data A26A	FFFF	-	Data A26A	0002		
SR	0000		SR	0000		
Example 2 072 072		CALL MOV	_G66 W0, W1	; call	routine _G	666
077. 077. 077.	A2A —	INC	W6, [W7++]	; routi	ne start	
	Before			After		
	Instruction	_	_ lı	nstruction		
PC	07 2000		PC	07 7A28		
W15	9004		W15	9008		
Data 9004	FFFF	<u>'</u>	Data 9004	2004		
Data 9006			<b>B</b>			
	FFFF		Data 9006	0007		

CALL			Call Indi	rect Subrou	tine		
Syntax:		{label:}	CALL	Wn			
Operands: Operation:		$ \begin{array}{l} Wn \in [W0 \ .\\ (PC)+2 \to I \\ (PC<15:0>) \\ (W15)+2 \to \\ (PC<23:16: \\ (W15)+2 \to \\ 0 \to PC<22 \end{array} $	PC ) → TOS > W15 >) → TOS > W15				
		(Wn<15:1> NOP $\rightarrow$ Ins					
Status Affecte	ed:	None		- 9			
Encoding:		0000	0001	0000	0000	0000	SSSS
Description:		Before the of the stack. A PC<15:1> a is ignored.	call is mad After the re and PC<2	de, the 24-bi eturn address 2:16> is clea	t return addre s is stacked, V ared. Since PC	ctions of prograr ss (PC+2) is pus Nn<15:1> is load C<0> is always '	shed onto ded into
\A/e vale :			select the	address of t	the source reg	gister.	
Words:		1					
Cycles:		2					
Example 1	00100 00100		CALL W	0	; Call BOO' ; using WO	I subroutine	indirectly
	00160	00 _BOOT:		400, W2 300, W6	; _BOOT st	arts here	
					_	arts here	
	00160		MOV #0x		; _BOOT st. After Instruction	arts here	
	00160	Before	MOV #0x		_ After	-	
	00160 PC W0	Before Instruction	MOV #0x	300, W6 PC W0	After	2	
	00160 PC W0 W15	Before Instruction	MOV #0x	300, W6 РС W0 W15	After Instruction 00 1600	<u>)</u>	
Data 6	PC W0 W15 F00	Before Instruction 00 1002 1600 6F00 FFFF	MOV #0x	PC 90 90 915 94ta 6F00	After Instruction 00 1600 1600 6F04 1004	0 0 1 1	
	PC W0 W15 F02	Before Instruction 00 1002 1600 6F00 FFFF FFFF	MOV #0x	PC PC W0 W15 Data 6F00 Data 6F02	After Instruction 00 1600 1600 6F04 1004	0 0 4 4 0	
Data 6	PC W0 W15 F00	Before Instruction 00 1002 1600 6F00 FFFF	MOV #0x	PC 90 90 915 94ta 6F00	After Instruction 00 1600 1600 6F04 1004	0 0 4 4 0	
Data 6	PC W0 W15 F02	Before Instruction 00 1002 1600 6F00 FFFF FFFF 0000 00	MOV #0x	PC PC W0 W15 Data 6F00 Data 6F02	After Instruction 00 1600 1600 6F04 1004 0000	0 0 4 4 0	indirectly
Data 6 Data 6	00160 PC W0 W15 F00 F02 SR 0042 0042	Before Instruction 00 1002 1600 6F00 FFFF FFFF 0000 00 02 00 _TEST:	MOV #0x	PC W0 W15 Data 6F00 Data 6F02 SR	- After Instruction 00 1600 1600 6F04 1004 0000 0000 ; Call TEST	) ) 4 4 9 ) ) 5 subroutine	indirectly
Data 6 Data 6 Example 2	00160 PC W0 W15 F00 F02 SR 0042 0042 0042 0042 0042 0055	Before Instruction 00 1002 1600 6F00 FFFF FFFF 0000 00 02 00 _TEST:	MOV #0x	PC W0 W15 Data 6F00 Data 6F02 SR	After Instruction 00 1600 1600 6F04 1004 0000 0000 ; Call TEST ; using W7	) 4 4 2 3 5 5 5 5 5 5 5 5 5 6 6 7 5 6 7 5 1 1 1 1 1 1 1 1 1 1 1 1 1	indirectly
Data 6 Data 6 Example 2	PC W0 W15 F02 SF02 0042 0042 0042 0055 0055	Before Instruction 00 1002 1600 6F00 FFFF FFFF 0000 00 02 00 02 02 02 TEST: Before Instruction 00 4200 5500	MOV #0x	PC W0 W15 Data 6F00 Data 6F02 SR W7 W1, W2 W1, W2 W1, W3	After Instruction 00 1600 1600 6F04 1004 0000 ; Call TEST ; using W7 ; _TEST sta ; After Instruction 00 5500	) 4 4 2 3 5 5 5 5 5 5 5 5 5 5 5 5 5	indirectly
Data 6 Data 6 Example 2	PC W0 W15 F00 F02 SR 0042 0042 0042 0055 0055	Before Instruction 00 1002 1600 6F00 FFFF 0000 00 02 00 02 -TEST: 00 Before Instruction 00 4200 5500 6F00	MOV #0x	PC W0 W15 Data 6F00 Data 6F02 SR W7 W1, W2 N1, W2 N1, W3	After Instruction 00 1600 1600 6F04 1004 0000 ; Call TEST ; using W7 ; _TEST sta ; After Instruction 00 5500 5500 6F04	) ) Subroutine arts here	indirectly

CLR		Clear f or W	REG			
Syntax:	{label:}	CLR{.B}	f			
			WREG			
Operands:	f ∈ [0 8 <sup>.</sup>	191]				
Operation:	0  ightarrow destin	nation designa	ted by D			
Status Affected:	None					
Encoding:	1110	1111	OBDf	ffff	ffff	ffff
Description:		contents of a f s specified, the s cleared.				
	The 'D' bit	selects byte c selects the de select the add	estination (0 f	or WREG, 1		
		The extension rather than a denote a wore The WREG i	word operat	ion. You may but it is not re	/ use a .w ex equired.	
Words:	1					
Cycles:	1					
Example 1 CI	R.B RAM2	; 00	Clear RAM2	200 (Byte n	node)	
RAM200 SF		RAM20 SI				
Example 2 C	LR WRE	G ;	Clear WRE	G (Word mo	de)	
WRE( SI		WRE	After Instruction G 0000 R 0000			

CLR		Clear Wd				
Syntax:	{label:}	CLR{.B}	Wd			
			[Wd]			
			[Wd++]			
			[Wd]			
			[++Wd]			
			[Wd]			
Operands:	Wd ∈ [W0	W15]				
Operation:	$0 \to Wd$					
Status Affected:	None					
Encoding:	1110	1011	0Bqq	qddd	d000	0000
Description:		ontents of re may be use	gister Wd. Eitl d for Wd.	ner register d	lirect or indire	ect
	The 'q' bits The 'd' bits <b>Note:</b> Th	select the de select the ac ne extension	or word opera estination Add ddress of the o . B in the instru- ord operation	ress mode. destination re ruction denot	egister. es a byte op	eration
			operation, bu			
Words:	1					
Cycles:	1					
Example 1	CLR.B W2	;	Clear W2 (E	syte mode)		
	Before Instruction		After Instruction			
	W2 3333		<b>N2</b> 3300			
	SR 0000	:	SR 0000			
Example 2	CLR [W0		Clear [W0] Post-increm	ent WO		
Data	Before Instruction W0 2300 2300 5607 SR 0000	۱ Data 23	After Instruction V0 2302 00 0000 SR 0000			

CLR		Clear Acc	umulator, Pre-F	etch Ope	rands	
Syntax: {	label:} CLR	Acc	{,[Wx],Wxd}	{,[Wy]	Wyd}	{,AWB]
			{,[Wx]+=kx,Wxd	} {,[Wy] <sup>.</sup>	+=ky,Wyd}	
			{,[Wx]-=kx,Wxd}	} {,[Wy]∙	-=ky,Wyd}	
			{,[W9+W12],Wx	d} {,[W11	+W12],Wyd}	
Operands:	Wy ∈ [W	3, W9]; kx ∈ [	[-6, -4, -2, 2, 4, 6] ∈ [-6, -4, -2, 2, 4, =2]			
Operation:	$([Wy]) \rightarrow V$	A or B) Nxd; (Wx)+/- Nyd; (Wy)+/- A)) rounded	-ky→Wy			
Status Affected:	OA, OB, S	SA, SB				
Encoding:	1100	0011	A0xx	yyii	iijj	jjaa
Description:	operands the non-s	in preparation	specified accum on for a MAC type imulator results. Id saturate flags	instruction	n and optionatiction of the second	ally store the
	Section 2 optional r contents	1.14.1 "MAC egister direct	and register offse <b>Pre-Fetches</b> ". C or indirect store accumulator, as	Operand A of the cor	WB specifies	the unded
	The 'x' bit The 'y' bit The 'i' bits The 'j' bits	ts select the p ts select the p s select the V s select the V	other accumulato pre-fetch Wxd de pre-fetch Wyd de Vx pre-fetch oper Vy pre-fetch oper accumulator write	stination. stination. ation. ation.		
Words:	1					
Cycles:	1					
Example 1	CLR A, [W8]	+=2, W4, W		with [W	V8], post-: V13	inc W8
	Before			After		
	Instructi			struction		
		F001	W4	1221		
,		2000 2623	W8 W13	2002 5420		
	CCA 00 0067 2			000 0000		
A	CCB 00 5420 3	3 BDD	ACCB 00 5	420 3BDI	)	
A Data 2			ACCB 00 5	420 3BDI 1221		

W7

W8

W10

W13

ACCA

ACCB

SR

Data 2000

Data 3000

Data 4000

C783

2000

3000

4000

2345

1221

FF80

FFC3

0000

00 0067

00 5420 ABDD

Example 2	CLR	B, [W8]+=2, W6,	[W10]+=2, W7,	[W13]+=2 ; C	lear ACCB
				; L	oad W6 with [W8]
				; L	oad W7 with [W10]
				; S	ave ACCA to [W13]
				; P	ost-inc W8,W10,W13
		Before		After	
		Instruction		Instruction	
	W6	F001	W6	1221	

W7

W8

W10

W13

ACCA

ACCB

SR

Data 2000

Data 3000

Data 4000

FF80

2002

3002

4002

2345

1221

FF80

0067

0000

00 0067

00 0000 0000

CLRWD	Г	Clear Watch	ndog Timer			
Syntax:	{label:}	CLRWDT				
Operands:	None					
Operation:	$0 \rightarrow WDT$	count register prescaler A co prescaler B co	ount			
Status Affected:	None					
Encoding:	1111	1110	0110	0000	0000	0000
Description:	prescaler o	contents of the count registers et by configura	s. The Watch	dog Prescal	er A and Pre	scaler B
Words:	1					
Cycles:	1					
Example 1	CLRWDT	; Clear Wat	chdog Time	er		
	Before Instruction SR 0000	SI	After Instruction			

СОМ		Complemer	nt f			
Syntax:	{label:}	COM{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	-				
Operation:		nation designa	ated by D			
Status Affected:	N, Z			Γ		1
Encoding:	1110	1110 ne 1's comple	1BDf	ffff	ffff	ffff
Description:	the result determines stored in V register. The 'B' bit The 'D' bit	in the desti- the destinat /REG. If WRE selects byte o selects the de select the add	ination regis ion register. EG is not spe r word opera estination (0 f	ter. The op If WREG is ecified, the re tion (0 for wo or WREG, 1	tional WREC specified, th esult is stored ord, 1 for byte	G operand le result is d in the file e).
		The extension rather than a denote a wore The WREG is	word operat	tion. You may but it is not re	y use a .w ex equired.	
Words:	1					
Cycles:	1					
Example 1	COM.b RAM	200 ; CC	OM RAM200	(Byte mode)	)	
	SR 0000	RAM20	<b>R</b> 0002 (	<b>Z)</b> AM400 and	store to W	REG
	Before		; (Word After	mode)		
WRE RAM4 S	Instruction	WRE RAM40	Instruction G F7DC 00 0823	N=1)		

		Compleme	ent Ws			
Syntax:	{label:}	COM{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	$\overline{(Ws)} \rightarrow W$	/d				
Status Affected:	N, Z					
Encoding:	1110	1010	1Bqq	qddd	dppp	SSSS
Description:	and place	the result in	the destinatio	contents of th n register Wd. both Ws and	Either regist	
	The 'q' bit The 'd' bit The 'p' bit	s select the c s select the a s select the s	destination Ac address of the source Addres	e destination re	egister.	e).
	Note:	rather than	a word operation	instruction de ation. You may , but it is not re	yusea.we	
Words:	1					
Cycles:	1					
	.B [W0++]		; COM [W0] ; Post-inc:	and store	-	yte mode
Example 1 COM				Lement wo, i	N T	
Example 1 COM	Before		After	Lement WO, N	ΝI	
	Instruction		After Instructio		Ν⊥	
WO	Instruction 2301		After Instructio W0 2302	n	Ν⊥	
W0 W1	Instruction           2301           2400		After Instructio W0 2302 W1 2401	n	ΝL	
WO	Instruction           2301           2400           5607		After Instructio W0 2302 W1 2401 300 5607	n 	ΝL	
W0 W1 Data 2300	Instruction           2301           2400           5607           ABCD	Data 23 Data 24	After Instructio W0 2302 W1 2401 300 5607	n 	ΝL	
W0 W1 Data 2300 Data 2400	Instruction           2301           2400           5607           ABCD           0000	Data 23 Data 24 ++1] ;	After Instructio W0 2302 W1 2401 300 5607 400 ABA9 SR 0008	n     (N=1)   store to [		mode)
W0 W1 Data 2300 Data 2400 SR Example 2 COM	Instruction 2301 2400 5607 ABCD 0000 W0, [W1 Before	Data 23 Data 24 ++1] ;	After Instructio W0 2302 W1 2401 300 5607 400 ABA9 SR 0008 COM W0 and Post-incre After	n ] (N=1) store to [ ment W1		mode)
W0 W1 Data 2300 Data 2400 SR Example 2 COM	Instruction          2301         2400         5607         ABCD         0000         W0, [W1         Before         Instruction	Data 23 Data 24 ++] ; ;	After Instructio W0 2302 W1 2401 300 5607 400 ABA9 SR 0008 COM W0 and Post-incre After Instruction	n ] (N=1) store to [ ment W1		mode)
W0 W1 Data 2300 Data 2400 SR Example 2 COM	Instruction           2301           2400           5607           ABCD           0000           W0, [W1           Before           Instruction           D004	Data 2: Data 2: ++1] ; ; V	After Instructio W0 2302 W1 2401 300 5607 400 ABA9 SR 0008 COM W0 and Post-incre After Instruction V0 D004	n ] (N=1) store to [ ment W1		mode)
W0 W1 Data 2300 Data 2400 SR Example 2 COM	Instruction           2301           2400           5607           ABCD           0000           W0, [W1           Before           Instruction           D004           1000	Data 2: Data 2: ++1] ; ; V	After Instructio           W0         2302           W1         2401           300         5607           400         ABA9           SR         0008           COM W0 and Post-incre         After           Instruction         Y0           V0         D004           V1         1002	n ] (N=1) store to [ ment W1		mode)

СР		Compare f	with WREG,	Set Status F	lags	
Syntax:	{label:}	CP{.B}	f			
Operands:	f ∈ [081	91]				
Operation:	(f) – (WRE	G)				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1110	0011	OBOf	ffff	ffff	ffff
Description:	equivalent stored.	f) – (WREG) a to the SUBWF	instruction, b	ut the result	of the subtrac	ction is not
		selects byte c select the add			ord, 1 for byte	e).
		The extension rather than a denote a wore The WREG is	word operat	ion. You may but it is not re	y use a .w ex equired.	
Words:	1					
Cycles:	1					
Example 1 CP	.B RAM	400 ; Cc	mpare RAM4	00 with WR	EG (Byte m	ode)
WREG RAM40 SI	0 0823	WRI RAM4		(Z=1)		
Example 2 CP	0x12	200 ; Co	mpare (0x1	200) with	WREG (Word	mode)
WRE0 Data 120 SI	0 2277	WRI Data 12		(N=1)		

Syntax:	{label:}	CP{.B}	Wb,	#lit5		
Operands:	Wb ∈ [W lit5 ∈ [0 .	′0 W15] 31]				
Operation:	(Wb) – li	5				
Status Affected:	DC, N, C	V, Z, C				
Encoding:	1110	0001	. 0www	wB00	011k	kkkk
Description:	equivale	nt to the SUI	B instruction, b		ster. This instru of the subtraction for Wb.	
	The 'B' b	it selects by	/te or word op		egister. vord, 1 for byte integer numbe	
	Note:	rather that	an a word ope		denotes a byte ay use a .w e required.	-
Words:	1					
Cycles:	1					
Example 1	CP.B W4,	#0x12	; Compar	e W4 with O	x12 (Byte m	ode)
	Before		After			
	Instruction	ו	Instructio	n T		
	W4 7711 SR 0000		W4 7711 SR 0008	(N=1)		
Example 2	CP W4,	#0x12	; Compar	e W4 with (	)x12 (Word m	node)
	Before Instructio W4 7713	n ]	After Instruct W4 771	ion		

CP		Compare	Wb with Ws, S	Set Status Fla	ags	
Syntax:	{label:}	CP{.B}	Wb,	Ws		
				[Ws]		
				[Ws++]		
				[Ws]		
				[++Ws]		
				[Ws]		
Operands:	Wb ∈ [W0 Ws ∈ [W0					
Operation:	(Wb) – (W	's)				
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	1110	0001	0www	wB00	0ppp	SSSS
Description:	equivalent stored. Re	t to the SUB in egister direct	and update th nstruction, but addressing mu y be used for '	the result of t ust be used for	he subtraction	n is not
	The 'B' bit The 'p' bit	selects byte s select the s	address of the or word opera ource Address ddress of the	tion (0 for wo s mode.	rd, 1 for byte)	).
	Note:	rather than	ion .B in the a word opera ord operation,	tion. You may	yuse a .we	-
Words:	1		,		1	
Cycles:	1					
Example 1 C	CP.B WO,	[W1++]	; Compare [ ; Post-incr		) (Byte mod	e)
	Before		After			
	Instruction		Instructio	_		
	NO ABA9 N1 2000		W0 ABA9 W1 2001			
Data 20						
S	SR 0000			(N=1)		
	CP W5,	W6	; Compare W	6 with W5	(Word mode)	
Example 2	E NJ,					
Example 2 c	Before		After			
·	Before Instruction	n 1	Instructio	n 1		
·	Before	n ]		n ] -		

CP0		Compare f	with 0x0, Set	Status Flag	S	
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:	•	f) – $0 \ge 0$ and units not stored		atus register.	The result of	the
		selects byte c select the add			ord, 1 for byte	).
	Note:	The extension rather than a denote a wor	word operat	tion. You may	yusea.we	
Words:	1					
Cycles:	1					
Example 1 CE	20.B RA	M100 ; (	Compare RAN	4100 with (	)x0 (Byte m	node)
	Before Instructior		After Instructior	<b>1</b>		
RAM10		RAM				
S	R 0000		SR 0008	(N=1)		
Example 2 CP	0 0x1FF	E ; Comp	are (0x1FF	E) with Ox	0 (Word mo	de)
	Before Instruction		After Instruction			
Data 1FFI SF	E 0001	Data 1F				

		Compare	Ws with 0x0, \$	Set Status F	lags	
Syntax:	{label:}	CP0{.B}	Ws			
			[Ws]			
			[Ws++]			
			[Ws]			
			[++Ws]			
			[Ws]			
Operands:	Ws∈[W0	) W15]				
Operation:	(Ws) – 0x	0000				
Status Affected:	DC, N, O	V, Z, C				
Encoding:	1110	0000	0000	0B00	0ppp	SSSS
Description:		n is not store	00 and update ed. Register dir			
	The 'p' bit	s select the	e or word opera source Address address of the	s mode.	-	e).
	Note:	The extens	sion . B in the	instruction d	enotes a bvt	e operatio
			a word operat	tion. You ma	yusea.we	-
Words:	1			tion. You ma	yusea.we	-
Words: Cycles:	1 1		a word operat	tion. You ma	yusea.we	-
Cycles:		denote a w	a word operat	tion. You ma but it is not r with 0 (E	y use a .w e equired.	-
Cycles:	1 PO.B [W4- Before	denote a w	a word operation, ord operation, Compare [W4] Post-decreme After	tion. You ma but it is not r with 0 (E nt W4	y use a .w e equired.	-
Cycles: Example 1 CI	1 PO.B [W4- Before Instruction	denote a w	a word operation, ord operation, Compare [W4] Post-decreme After Instruction	tion. You ma but it is not r with 0 (E nt W4	y use a .w e equired.	-
Cycles: Example 1 CI	1 P0.B [W4- Before Instruction 14 1001	denote a w	a word operation, ord operation, Compare [W4] Post-decreme After Instruction W4 1000	tion. You ma but it is not r with 0 (E nt W4	y use a .w e equired.	-
Cycles: Example 1 CI W Data 100	1 P0.B [W4- Before Instruction 14 1001	denote a w	a word operation, ord operation, Compare [W4] Post-decreme After Instruction W4 1000 000 0034	tion. You ma but it is not r with 0 (E nt W4	y use a .w e equired.	-
Cycles: Example 1 CI W Data 100 S	1 PO.B [W4- Before Instruction (4 1001 00 0034	denote a w	a word operation, ord operation, Compare [W4] Post-decreme After Instruction W4 1000 000 0034	tion. You ma but it is not r with 0 (E nt W4 (Z=1)	y use a .We equired. Byte mode)	extension
Cycles: Example 1 CI W Data 100 S	1 PO.B [W4- Before Instruction 14 1001 00 0034 R 0000 PO [W Before	denote a w -] ; c ; F Data 1	a word operation, ord operation, Compare [W4] Post-decreme After Instruction W4 1000 000 0034 SR 0002 Compare [W After	tion. You ma but it is not r with 0 (E nt W4 (Z=1) 5] with 0	y use a .We equired. Byte mode)	extension
Cycles: Example 1 CI Data 100 S Example 2 CI	1 PO.B [W4- Before Instruction 14 1001 00 0034 R 0000 PO [W Before Instruction	denote a w -] ; c ; F Data 1	a word operation, ord operation, compare [W4] Post-decreme After Instruction W4 1000 000 0034 SR 0002 Compare [W After Instruction	tion. You ma but it is not r with 0 (E nt W4 (Z=1) 5] with 0	y use a .We equired. Byte mode)	extension
Cycles: Example 1 CI Data 100 S Example 2 CI	1 PO.B [W4- Before Instruction /4 1001 00 0034 R 0000 PO [W Before Instruction /5 2400	denote a w -] ; c ; F Data 1	a word operation, ord operation, compare [W4] Post-decreme After Instruction W4 1000 000 0034 SR 0002 Compare [W After Instruction W5 23FE	tion. You ma but it is not r with 0 (E nt W4 (Z=1) 5] with 0	y use a .We equired. Byte mode)	extension

СРВ	Compare	f with WREC	Gusing Borro	ow, Set Statu	ıs Flags	
Syntax:	{label:}	CPB{.B}	f			
_		_				
Operands:	f ∈ [081	-				
Operation:	(f) – (WRE	, , ,				
Status Affected:	DC, N, O	V, Z, C	-			
Encoding:	1110	0011	1B0f	ffff	ffff	ffff
Description:	instruction		– ( <del>C</del> ), and up t to the SUBB i d.			
			or word opera ddress of the f		ord, 1 for byt	e).
	2:	rather than denote a wo The WREG The Z flag is	on . B in the i a word operation, ord operation, is set to work "sticky" for AL can only clear	ion. You may but it is not r ing register V DC, CPB, S	y use a .w e equired. V0.	extension t
Words:	1					
Cycles:	1					
Example 1 CPB	.B RAM4	100 ; Comp	are RAM400	with WREG	using $\overline{C}$ (B	yte mode
	Before		After			
	Instruction		Instruction			
WREG		WR				
RAM400 SR		RAM <sup>2</sup>		(N=1)		
Example 2 CPB	0x12	00 ; Compa	re (0x1200)	with WREG	using $\overline{C}$ (	Word mod
	Before		After			
lı WREG	nstruction	WBI				
	2377					
SR				(C=1)		
Data 1200 SB	2377	Data 12		(C-1)		

СРВ	Compare	Wb with lit5	using Borrov	v, Set Status	Flags	
Syntax:	{label:}	CPB{.B}	Wb,	#lit5		
Operands:	Wb ∈ [W0 lit5 ∈ [0	-				
Operation:	(Wb) – lit5	$-(\overline{C})$				
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	1110	0001	lwww	wB00	011k	kkkk
Description:	is equivale	ent to the SUB	C), and updat is instruction, l addressing mu	out the result	of the subtra	
	The 'B' bit	selects byte	ddress of the or word opera literal operanc	tion (0 for wo	ord, 1 for byte	
		rather than denote a wo The Z flag is	on . B in the a word operation, ord operation, s "sticky" for AI can only clear	tion. You may but it is not re DDC, CPB, S	y use a .w e equired.	xtension to
Words:	1					
Cycles:	1					
Example 1 CI			Compare W4 After Instruction W4 7711 SR 0008		using C (B	yte mode
Example 2 CP	B.B W4, #	0x12 ;	Compare W4	with 0x12	using $\overline{C}$ (By	yte mode
W2 SF			After Instruction V4 7711 SR 0008 (	N=1)		
Example 3 CPI	B W12, a	#0x1F ; C	Compare W12	with 0x1F	using $\overline{C}$ (W	ord mode
W12 SF			After Instruction 12 0020 SR 0003 (2	Z, C=1)		
Example 4 CPF	B W12, ‡	‡0x1F ; C	Compare W12	with 0x1F	using $\overline{C}$ (We	ord mode)
W12 SF		W , C=1) S		C=1)		

	0 1 1 2	-		using Borrow,		0
Syntax:	{label:}	CPB{.B}	Wb,	Ws		
				[Ws]		
				[Ws++] [Ws]		
				[**S] [++Ws]		
				[++vvs] [Ws]		
				[ 113]		
Operands:	Wb ∈ [W0 Ws ∈ [W0					
Operation:	(Wb) – (V	√s) – ( <del>C</del> )				
Status Affected:	DC, N, O	V, Z, C				
Encoding:	1110	0001	lwww	wB00	0ppp	SSSS
Description:	is equival stored. Re	ent to the SUE	BB instruction addressing m	date the Status , but the result nust be used for Ws.	of the subtrac	ction is not
	The 'B' bi The 'p' bit	t selects byte ts select the s	or word oper ource Addres	e Wb source re ation (0 for wor ss mode. Ws source reg	d, 1 for byte)	
	2:	: The Z flag is	•	, but it is not re ADDC,CPB, ar Z.	•	BBR. Thes
Words:	1					
Cycles:	1					
Example 1 C	PB.B W0,		Compare [W1 Post-increm	] with WO us ment W1	sing $\overline{C}$ (Byt	e mode)
V Data 10		]	After Instruction W0 ABAS W1 1000 1000 D0AS SR 0008	9 1 9		
Example 2 C	PB.B WO,		ompare [W1 ost-increm	] with W0 us ent W1	ing $\overline{C}$ (Byt	e mode)
	Before Instruction	n	After Instructio	n		

Example 3 CPB W4, W5

; Compare W5 with W4 using  $\overline{C}$  (Word mode)

(C=1)

I	Before nstructior	ı	I	After nstructior	n
W4	4000		W4	4000	
W5	3000		W5	3000	
SR	0001	(C=1)	SR	0001	(

CPSEQ	Compare Wb with Wn, Skip if Equal (Wb = Wn)
Syntax:	{label:} CPSEQ{.B} Wb, Wn
Operands:	Wb ∈ [W0 W15] Wn ∈ [W0 W15]
Operation:	(Wb) – (Wn) Skip if (Wb) = (Wn)
Status Affected:	None
Encoding:	1110 0111 1www wB00 0000 ssss
Description:	Compare the contents of Wb with the contents of Wn by performing the subtraction (Wb) – (Wn), but do not store the result. If (Wb) = (Wn), the next instruction (fetched during the current instruction execution) is discarded and on the next cycle, a NOP is executed instead. If (Wb) $\neq$ (Wn), the next instruction is executed as normal.
	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 Ws source register.
	<b>Note:</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.
Words:	1
Cycles:	1 (2 or 3 if skip taken)
0 0 0	002000 HERE: CPSEQ.B W0, W1 ; If W0 = W1 (Byte mode), 002002 GOTO BYPASS ; skip the GOTO 002004 002006 002008 BYPASS: 00200A
V V	Before Instruction         After Instruction           PC         00         2000           W0         1001         W0           W1         1000         W1           SR         0000         SR
0	018000 HERE: CPSEQ W4, W8 ; If W4 = W8 (Word mode), 018002 CALL _FIR ; skip the subroutine call 018006 018008
V V	Before Instruction         After Instruction           PC         01         8000           W4         3344         W4           W8         3344         W8           SR         0002         (Z=1)

CPSGT	Signed Compa	ire Wb wi	th Wn, Skip	if Greater TI	nan (Wb > V	/n)
Syntax:	{label:} CPS	SGT{.B}	Wb,	Wn		
Operands:	Wb ∈ [W0 W Wn ∈ [W0 W	-				
Operation:	(Wb) – (Wn) Skip if (Wb) > (\	Wn)				
Status Affected:	None					
Encoding:	1110	0110	0www	wB00	0000	SSSS
Description:	Compare the co subtraction (Wb next instruction discarded and c the next instruct	) – (Wn), (fetched o on the nex	but do not st during the cu t cycle, a NO	ore the resul rrent instruct P is executed	t. If (Wb) > (V ion execution	Nn), the n) is
	The 'w' bits sele The 'B' bit selec The 's' bits sele	cts byte or	word operat	ion (0 for wo	rd, 1 for byte	e).
	rathe	er than a	n.B in the ir word operation d operation, b	on. You may	use a .we	
Words:	1		,		•	
Cycles:	1 (2 or 3 if skip	taken)				
Example 1		PSGT.B OTO · · · · · ·		f W0 > W1 skip the G0	-	e),
	Before Instruction           PC         00         2000           W0         00FF           W1         26FE           SR         0009	N, C=1)	PC W0 W1 SR	After Instruction 00 2006 00FF 26FE 0009	(N, C=1)	
Example 2	018000 HERE: CPS 018002 CAI 018006 018008	L.	W4, W5; If _FIR ; sk	W4 > W5 ( ip the sub		
	Before Instruction           PC         01         8000           W4         2600           W5         2600           SR         0004         (O	V=1)	PC W4 W5 SR	After nstruction 01 8002 2600 2600 0004 ((	DV=1)	

CPSL1	Signed Compare Wb with Wn, Skip if Less Than (Wb < Wn)
Syntax:	{label:} CPSLT{.B} Wb, Wn
Operands:	Wb ∈ [W0 W15] Wn ∈ [W0 W15]
Operation:	(Wb) – (Wn) Skip if (Wb) < (Wn)
Status Affecte	ed: None
Encoding:	1110 0110 1www wB00 0000 ssss
Description:	Compare the contents of Wb with the contents of Wn by performing the subtraction (Wb) – (Wn), but do not store the result. If (Wb) < (Wn), the next instruction (fetched during the current instruction execution) is discarded and on the next cycle, a NOP is executed instead. Otherwise, the next instruction is executed as normal.
	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 Ws source register.
	Note: The extension . B in the instruction denotes a byte operation rather than a word operation. You may use a . w extension t denote a word operation, but it is not required.
Words:	1
Cycles:	1 (2 or 3 if skip taken)
Example 1	002000 HERE: CPSLT.B W8, W9; If W8 < W9 (Byte mode), 002002 GOTO BYPASS; skip the GOTO 002006 002008 00200A BYPASS: 00200C
	Before         After           Instruction         Instruction           PC         00 2000         PC         00 2002           W8         00FF         W8         00FF           W9         26FE         W9         26FE           SR         0008         SR         0008
Example 2	018000 HERE: CPSLT W3, W6 ; If W3 < W6 (Word mode), 018002 CALL _FIR ; skip the subroutine call 018006 018008
	Before Instruction         After Instruction           PC         01         8000           W3         2600         W3         2600           W6         3000         W6         3000           SR         0000         SR         0000

### CDCI T .... .... ... ....

CPSNE	1	Signed Con	npare Wb wi	th Wn, Skip i	f Not Equal (	Wb ≠ Wn)
Syntax:	{label:}	CPSNE{.B}	Wb,	Wn		
Operands:	Wb ∈ [W0 Wn ∈ [W0	-				
Operation:	(Wb) – (Wn Skip if (Wb)					
Status Affected	1: None					
Encoding:	1110	0111	0www	wB00	0000	SSSS
Description:	subtraction instruction ( and on the r	(Wb) – (Wn), fetched durin	but do not st g the current NOP is execu	contents of W ore the result. instruction ex ted instead. C	If (Wb) $\neq$ (Wi ecution) is di	n), the next scarded
	The 'B' bit s	elects byte o	r word operat	Wb source reg ion (0 for wor Vs source reg	d, 1 for byte).	
	I	ather than a	word operation	instruction de tion. You may out it is not rec	use a .w e	-
Words:	1		-			
Cycles:	1 (2 or 3 if s	kip taken)				
Example 1	002000 HERE: 002002 002006 002008 00200A BYPAS 00200C	CPSNE.B GOTO  S:		If W2 != W skip the G	-	de),
	PC 00 200 W2 00F W3 26F SR 000	0 F	PC W2 W3 SR	After Instruction 00 2006 00FF 26FE 0001 (	C=1)	
Example 2	018000 HERE: 018002 018006 018008			If WO != W skip the st		
	Before           Instruction           PC         01         800           W0         300           W8         300           SR         000	0 0 0	PC W0 W8 SR	After Instruction 01 8002 3000 3000 0000		

		Decimal Au	ljust Wn			
Syntax:	{label:}	DAW.B	Wn			
Operands:	Wn ∈ [W0	W15]				
Operation:		0>>9) or (DC $0>) + 6 \rightarrow Wn$				
	Else	0>) → Wn<3:(				
	(Wn<7: Else	l> > 9) or (C = 4>) + 6 → Wn 4>) → Wn<7:⁄	<7:4>			
Status Affected:	Ċ	,				
Encoding:	1111	1101	0100	0000	0000	SSSS
	Carry flag addressing	ult. The Most is used to ind g must be use	icate any dec d for Wn.	cimal rollover.	Register dire	ect
	The 's' bits	s select the ac	ldress of the	source/destir	nation registe	r.
		This instruc	bytes have t tion operates	been added.	node only a	
Words:	1					
Cycles:	1					
Example 1	DAW.B WO	; Dec	imal adjus	t WO		
	Before		After			
	Instruction		Instruction	ו		
	W0 771A SR 0002		W0 7720 SR 0002	(DC=1)		
Example 2	DAW.B W3	; Dec	imal adjus	et W3		
	Before Instructior		After Instruction			
	W3 77AA SR 0000	V	V3 7710 SR 0001 (1			

© 2005 Microchip Technology Inc.

DEC		Decrement	f				
Syntax:	{label:}	DEC{.B}	f	{,WREG}			
<b>A</b>	( <u>10</u> 0.44						
Operands:	f ∈ [0 819	-					
Operation:		$-1 \rightarrow$ destination designated by D					
Status Affected:	DC, N, OV,	, N, OV, Z, C					
Encoding:	1110	1101	OBDf	ffff	ffff	ffff	
Description:	Subtract one from the contents of the file register and place the re destination register. The optional WREG operand determines the tion register. If WREG is specified, the result is stored in WREG is not specified, the result is stored in the file register.				ne destina-		
	The 'D' bit s	selects byte o selects the de select the ado	stination (0 f	or WREG, 1 f			
		The extensio rather than a denote a wore The WREG is	word operat d operation, l	tion. You may out it is not re	/ use a .w ex quired.	•	
Words:	1						
Cycles:	1						
Example 1	DEC.B 0x20	0	; Decrem	ent (0x200	) (Byte mod	le)	
Data 20 S	Before Instruction 00 80FF SR 0000	Data 2	After Instruction 200 80FE SR 0009	) (N,C=1)			
Example 2 D	EC RAM4(	)0, WREG ; ;	Decrement (Word mod		nd store to	WREG	
WRE RAM40 S	-	WRE RAM40 S					

		Decremen	t Ws			
Syntax:	{label:}	DEC{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	(Ws) – 1 -	ightarrow Wd				
Status Affected:	DC, N, O	/, Z, C				
Encoding:	1110	1001	0Bqq	qddd	dppp	SSSS
Description:	result in th	ne destinatior	contents of the register Wd. ed by Ws and '	Either registe		
	The 'q' bit The 'd' bit The 'p' bit	s select the d s select the a s select the s	or word opera lestination Add address of the ource Address address of the	dress mode. destination re s mode.	gister.	).
	r	ather than a	<ul> <li>B in the inst</li> <li>word operation</li> <li>operation, but</li> </ul>	n. You may us	e a .w extens	
Words:	1				incu.	
Cycles:	1					
		], [W8++]	• DEC [W7]	and store	to [W8] (E	syte mod
Example 1 DE	C.B [W/++	]/ []		arement W7		,700 mod
Example 1 DE	Before	], []	; Post-in After			.700 moa
	Before Instruction		; Post-in After Instruction			<i>,,</i> , , , , , , , , , , , , , , , , , ,
W	Before Instruction 7 2301	Ň	; Post-in After Instruction N7 2302			700
W	Before Instruction 7 2301 8 2400	\ \ \	; Post-in After Instruction N7 2302 N8 2401			,
W	Before           Instruction           7         2301           8         2400           0         5607	Ň	; Post-in After Instruction N7 2302 N8 2401 00 5607			,
W7 W8 Data 2300	Before           Instruction           7         2301           8         2400           0         5607           0         ABCD	N Data 23 Data 24	; Post-in After Instruction N7 2302 N8 2401 500 5607			,
W7 W8 Data 2300 Data 2400	Before           Instruction           7         2301           8         2400           0         5607           0         ABCD           3         0000	\ Data 23 Data 24 { ; ; De	; Post-in After Instruction W7 2302 W8 2401 500 5607 .00 AB55	and store t	, W8	-
W7 W8 Data 2300 Data 2400 SF	Before Instruction 7 2301 8 2400 0 5607 0 ABCD 7 0000 C W5, [W6 Before	\ Data 23 Data 24 { ; ; De	; Post-in After Instruction W7 2302 W8 2401 00 5607 00 AB55 SR 0000 crement W5 ost-increment	and store t	, W8	-
W7 W8 Data 2300 Data 2400 SF Example 2 DE	Before Instruction 7 2301 8 2400 0 5607 0 ABCD 7 0000 C W5, [W6 Before Instruction	\ Data 23 Data 24 { ; ; Pc	; Post-in After Instruction W7 2302 W8 2401 00 5607 00 AB55 SR 0000 crement W5 cost-increment After Instruction	and store t	, W8	-
W7 W8 Data 2300 Data 2400 SF	Before           Instruction           7         2301           8         2400           0         5607           0         ABCD           R         0000           C         W5, [W6           Before         Instruction           5         D004	\ Data 23 Data 24 ( ; ; Pc	; Post-in After Instruction W7 2302 W8 2401 00 5607 00 AB55 SR 0000 crement W5 ost-increment	and store t	, W8	-
W7 W8 Data 2300 Data 2400 SF Example 2 DE	Before           Instruction           7         2301           8         2400           0         5607           0         ABCD           2         0000           C         W5, [W6           Before         Instruction           5         D004           6         2000	\ Data 23 Data 24 ( ; ; Pc	; Post-in After Instruction N7 2302 N8 2401 500 5607 00 AB55 SR 0000 crement W5 crement W5 st-increment After Instruction N5 D004 N6 2002	and store t	, W8	-

DEC2		Decrement	f by 2				
Syntax:	{label:}	DEC2{.B}	f	{,WREG}			
Oreanender	f - 10 - 0	1011					
Operands:	f ∈ [0 8	-	:				
Operation:		$) - 2 \rightarrow$ destination designated by D					
Status Affected:		, N, OV, Z, C					
Encoding:	1110	1101	1BDf	ffff	ffff	ffff	
Description:	Subtract two from the contents of the file register and place the result destination register. The optional WREG operand determines the de tion register. If WREG is specified, the result is stored in WREG. If W 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.				ne destina-		
		The extension rather than a wo denote a word o	ord operation	. You may us	e a .w extens		
Words:	1						
Cycles:	1						
Example 1 DE	С2.В 0	x200	; Decreme	ent (0x200)	by 2 (Byt	e mode)	
BeforeAfterInstructionInstructionData 20080FFSR0000SR0009							
Example 2 DI	EC2 F	RAM400, WREG		ent RAM400 CO WREG (Wo	-		
WRE RAM40 S	0 0823	WR RAM4		ı			

DEC2		Decrement	Ws by 2			
Syntax:	{label:}	DEC2{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	(Ws) – 2 -	ightarrow Wd				
Status Affected:	DC, N, O	/, Z, C				
Encoding:	1110	1001	1Bqq	qddd	dppp	SSSS
Description:	result in th	ne destination	ontents of the register Wd. d by Ws and V	Either registe		
	The 'q' bit The 'd' bit The 'p' bit	s select the d s select the a s select the se	or word opera estination Add ddress of the ource Address ddress of the	lress mode. destination re s mode.	egister.	).
	r	ather than a v	. B in the inst vord operatior operation, bu	n. You may us	se a .w exter	
Words:	1					
Cycles:	1					
Example 1 DEC	2.B [W7		DEC [W7] by	72, store ment W7, W8		yte mode
			Post-decrei			
	Before		After			
	Instruction		After Instruction			
W7	Instruction 2301	Ŵ	After Instruction			
	Instruction           2301           2400		After Instruction /7 2300 /8 23FF			
W7 W8	Instruction 2301 2400 0107	N N	After Instruction /7 2300 /8 23FF 00 0107			
W7 W8 Data 2300	Instruction           2301           2400           0107           ABCD	W W Data 230 Data 240	After Instruction /7 2300 /8 23FF 00 0107	J=1)		
W7 W8 Data 2300 Data 2400	Instruction           2301           2400           0107           ABCD           00000	W Data 230 Data 240 S	After Instruction /7 2300 /8 23FF 00 0107 00 ABFF	store to [		mode)
W7 W8 Data 2300 Data 2400 SR	Instruction           2301           2400           0107           ABCD           0000           C2           W5, [W           Before	W Data 230 Data 240 S	After Instruction /7 2300 /8 23FF 00 0107 00 ABFF R 0008 (N C W5 by 2, st-incremen After	store to [		mode)
W7 W8 Data 2300 Data 2400 SR Example 2 DE	Instruction           2301           2400           0107           ABCD           0000           C2           Before           Instruction	W Data 230 Data 240 S N6++] ; DE0 ; Pos	After Instruction 7 2300 78 23FF 00 0107 00 ABFF R 0008 (N C W5 by 2, st-incremen After Instruction	store to [		mode)
W7 W8 Data 2300 Data 2400 SR	Instruction           2301           2400           0107           ABCD           0000           C2 W5, [W           Before           Instruction           50004	W Data 230 Data 240 S N6++] ; DE0 ; Pos	After Instruction /7 2300 /8 23FF 00 0107 00 ABFF R 0008 (N C W5 by 2, st-incremen After	store to [		mode)
W7 W8 Data 2300 Data 2400 SR Example 2 DE	Instruction           2301           2400           0107           ABCD           0000           C2 W5, [W           Before           Instruction           5 D004           6 1000	W Data 230 Data 240 S N6++] ; DE0 ; Pos	After Instruction /7 2300 /8 23FF 00 0107 00 ABFF R 0008 (N C W5 by 2, st-incremen After Instruction V5 D004 V6 1002	store to [		mode)

DISI	Disable Interrupts Temporarily					
Syntax:	{label:} I	DISI	#lit14			
Operands:	lit14 ∈ [0	16383]				
Operation:	lit14 $\rightarrow$ DISI 1 $\rightarrow$ DISI Disable inter	-	14+1) cycles			
Status Affected:	None					
Encoding:	1111	1100	00kk	kkkk	kkkk	kkkk
	<ul> <li>Disable interrupts of priority 0 through priority 6 for (lit14+1) instruction cycles. Priority 0 through priority 6 interrupts are disabled starting in the cycle that DISI executes, and remain disabled for the next (lit 14) cycles. The lit14 value is written to the DISICNT register, and the DISI flag (INTCON2&lt;14&gt;) is set to '1'. This instruction can be used before executing time critical code, to limit the effects of interrupts.</li> <li>Note: This instruction does not prevent priority 7 interrupts and traps from running. See the <i>dsPIC30F Family Reference Manual</i> for</li> </ul>				4) cycles. flag re	
Words:	1	letails.				
Cycles:	1					
(	002000 HERE: 002002 002004	DISI #10			upts for 10 s protected	-
	Before			After		
	Instruction		<u> </u>	nstruction		
	PC 00 2000		PC	00 2002		
DISIC			DISICNT	0100		
INTCO			NTCON2		DISI=1)	
	SR 0000		SR	0000		

DIV.S		Signed Int	eger Divide			
Syntax:	{label:}	DIV.S{W}	Wm, Wn			
		DIV.SD	Wm, Wn			
Operands:		0, W2, W4	word operatic W14] for doul			
Operation:	Wm → <u>If (Wm-</u> 0xFf <u>Else:</u> 0x0 W1:W0	operation (de → W0 <15> = 1): FFF → W1 $\rightarrow$ W1 $\rightarrow$ W1 $\rightarrow$ W0 nder → W1	<u>fault):</u>			
	Wm+1: W1:W0	e operation ([ Wm → W1:W ) / Wn → W0 nder → W1				
Status Affected:	N, OV, Z,	С				
Encoding:	1101	1000	Ottt	tvvv	vW00	SSSS
	W0 and s operation divide ope This instru (with an it remainde otherwise overflow a and clear	ign-extended , Wm+1:Wm i eration is store uction must be reration count r. The N flag v . The OV flag and cleared of ed otherwise.	In the default through W1 to s first copied f ed in W0, and e executed 18 of 17) to gene will be set if th will be set if th will be set if th therwise. The The C flag is alue should n	perform the to W1:W0. The the 16-bit ren times using prate the corn e remainder he divide ope Z flag will be used to imple	operation. In ne 16-bit quot nainder is sto the REPEAT i rect quotient a is negative an eration resulte set if the rem	the doub tient of the red in W nstruction and cleare ed in an ainder is
	operation The 'v' bit The 'W' b	. These bits a is select the L it selects the o	ost Significan re clear for the east Significan dividend size ivisor register.	e word opera nt Word of th (0 for 16-bit,	tion. e dividend.	the doub
	Note 1	: The extens (32-bit) divid	ion . D in the			oublo_wc

DIV.S	Sig	ned Integer Divide
Words:	1	
Cycles:	<b>18 (plus 1 for</b> R	EPEAT execution)
Example 1	REPEAT #17 DIV.S W3, W4	; Execute DIV.S 18 times ; Divide W3 by W4 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         5555           W1         1234           W3         3000           W4         0027           SR         0000	After         Instruction         W0       013B         W1       0003         W3       3000         W4       0027         SR       0000
Example 2	REPEAT #17 DIV.SD W0, W12	; Execute DIV.SD 18 times ; Divide W1:W0 by W12 ; Store quotient to W0, remainder to W1
	Before Instruction W0 2500 W1 FF42 W12 2200	After Instruction W0 FA6B W1 EF00 W12 2200
	SR 0000	SR 0008 (N=1)

Curatavu	(labalı)					
Syntax:	{label:}	DIV.U{W} DIV.UD	Wm, Wn Wm, Wn			
Operands:		0, W2, W4	word operatic W14] for dou			
Operation:	$Wm \rightarrow 0x0 \rightarrow W1:WC$	-	f <u>ault):</u>			
	Wm+1: W1:W0	e operation ([ Wm → W1:V ) / Wns → W0 nder → W1	VO			
Status Affected:	N, OV, Z,	С				
Encoding:	1101	1000	lttt	tvvv	vW00	SSSS
	Wm+1:Wi operation This instru (with an it remainde divide ope will be set	m is first copie is stored in V uction must be eration count r. The N flag v eration resulte t if the remain	erform the divi ed to W1:W0. /0, and the 16 e executed 18 of 17) to gene vill always be ed in an overfle der is 0 and c e algorithm and	The 16-bit quebels bit remainder bit remainder times using erate the corr cleared. The bow and cleared leared otherw	uotient of the er is stored in the REPEAT ect quotient a OV flag will b ed otherwise. vise. The C fl	divide W1. instructio and be set if th The Z fla ag is use
	operation The 'v' bit The 'W' b	These bits a s select the L it selects the c	ost Significan re clear for the east Significa dividend size ivisor register	e word opera nt Word of the (0 for 16-bit,	tion. e dividend.	the doub
	2: 3:	(32-bit) divid extension to Unexpected represented operation (I bit will be s used. Dividing by first cycle of This instruct	ion . D in the dend rather tha denote a wo d results will d in 16 bits. DIV.UD). Wh det and the qu zero will initia f execution. ction is inter	an a word divi rd operation, occur if the This may on en an overflo uotient and re te an arithme	dend. You m but it is not r e quotient c ly occur for w occurs, the emainder sho etic error trap	ay use a equired. an not the doub e OV stat ould not o during t
		boundary.				
Words:	1	boundary.				

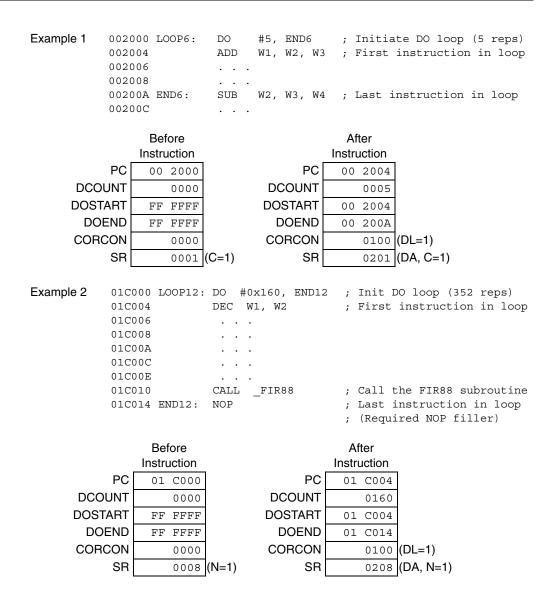
Example 1	REPEAT #17 DIV.U W2, W4	; Execute DIV.U 18 times ; Divide W2 by W4 ; Store quotient to W0, remainder to W1
	Before           Instruction           W0         5555           W1         1234           W2         8000           W4         0200           SR         0000	After Instruction W0 0040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

DIVF		Fractional	Divide			
Syntax:	{label:}	DIVF	Wm, Wn			
Operands:	Wm ∈ [W0 Wn ∈ [W2					
Operation:	$0x0 \rightarrow W0$ $Wm \rightarrow W1$ W1:W0 / V Remainde	$Vn \rightarrow W0$				
Status Affected:	N, OV, Z, (	С				
Encoding:	1101	1001	Ottt	t000	0000	SSSS
	W0 is first divide ope The sign o	cleared and ration is store f the remaine	divisor is stored Wm is copied ed in W0, and der will be the	to W1. The 1 the 16-bit ren same as the	l6-bit quotien nainder is sto sign of the d	t of the bred in W1 ividend.
	(with an ite remainder otherwise. overflow a and cleare	eration count . The N flag The OV flag nd cleared o d otherwise.	e executed 18 of 17) to gene will be set if the will be set if the will be set if the therwise. The The C flag is ralue should ne	erate the corr e remainder i he divide ope Z flag will be used to imple	ect quotient a is negative ar eration resulte set if the rem	and nd cleared ed in an ainder is
			vidend registe ivisor register.			
	2:	or equal to will occur be When this of and remain Dividing by first cycle o	tional divide to Wn. If Wm is ecause the fra occurs, the OV der should not zero will initia f execution. ction is inter	greater than ctional result status bit wil be used. te an arithme	Wn, unexped will be greate I be set and t etic error trap	cted resul er than 1. he quotie during th
Words:	1	boundary.				
Cycles:	18 (plus 1	for REPEAT	execution)			
1	EPEAT #17 IVF W8,	W9 ; D:	kecute DIVF ivide W8 by core quotien	W9	remainder 1	to W1
v v v	Before Instruction V0 8000 V1 1234 V8 1000 V9 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)

DO	Initialize Hardware Loop Literal						
Syntax:	{label:} Do	C	#lit14,	Expr			
Operands:	lit14 $\in$ [0 16383] Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 $\in$ [-32768 +32767].						
Operation:	Push DO shac (lit14) $\rightarrow$ DCO (PC)+4 $\rightarrow$ PC (PC) $\rightarrow$ DOST (PC) + (2*Slit1) Increment DL	UNT ART 6) → DOE	ND		)		
Status Affected:	DA						
Encoding:	0000	1000	00kk	kkkk	kkkk	kkkk	
	0000	0000	nnnn	nnnn	nnnn	nnnn	
	Initiate a no overhead hardware DO loop, which is executed (lit14+1) times. The DO loop begins at the address following the DO instruction, and ends at the address 2*Slit16 instruction words away. The 14-bit count value (lit14) supports a maximum loop count value of 16384, and the 16-bit offset value (Slit16) supports offsets of 32K instruction words in both directions.						
	When this instruction executes, DCOUNT, DOSTART and DOEND are first 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 'k' bits specify the loop count. The 'n' bits are a signed literal that specifies the number of instructions offset from the PC to the last instruction executed in the loop.						
	<ul> <li>Special Features, Restrictions: 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 one time.</li> <li>Using a loop size 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 can NOT be: <ul> <li>an instruction which changes program control flow</li> </ul> </li> </ul>						
	<ul> <li>a DO or REPEAT instruction</li> <li>Unexpected results may occur if any of these instructions are used.</li> </ul>						
			•	•			
					nd supports ditional 5 leve		
	pro Re <b>2:</b> Th	ference Ma	oftware by t anual for deta	he user. See ils.	e the dsPIC3	30F Famil	
Words:	pro Re <b>2:</b> Th	ference Ma e linker will	oftware by t anual for deta	he user. See ils.		30F Famil	

#### dsPIC30F Programmer's Reference Manual



	Initialize Hardware Loop Wn							
Syntax:	{label:}	DO	Wn,	Expr				
Operands:		W15] e an absolute Ived by the lir				+32767]		
Operation:	$\begin{array}{l} (Wn) \rightarrow DC \\ (PC)+4 \rightarrow F \\ (PC) \rightarrow DO \\ (PC) + (2^{*}S) \end{array}$	PC O	ND					
Status Affected:	DA							
	0000	1000	1000	0000	0000	SSSS		
Encoding:	0000	0000	nnnn	nnnn	nnnn	nnnn		
	support a maximum count value of 16384, and the 16-bit offset value (Slit16) supports offsets of 32K instruction words in both directions. When this instruction executes, DCOUNT, DOSTART and DOEND are first 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 'n' bits are a signed literal that specifies the number of instructions offset from (PC+4), which is the last instruction executed in the loop.							
	The followir	<b>atures, Restr</b> ng features an						
	<ol> <li>Using a if these</li> <li>The ve</li> <li>an in</li> <li>a DO</li> </ol>	a loop count of an offset of -2, offsets are us ry <b>last two</b> ins struction whic or REPEAT in ected results r	O will result in -1 or 0 is inv sed. structions of t ch changes pr struction	n the loop bei alid. Unexped he DO loop c rogram contro	ng executed o cted results m an NOT be: ol flow	one time. lay occur		
	<ol> <li>Using a if these</li> <li>The ve</li> <li>an in</li> <li>a DO</li> <li>Unexpense</li> <li>Note 1: 1</li> </ol>	an offset of -2, offsets are us ry <b>last two</b> ins struction whic or REPEAT in	6 0 will result in -1 or 0 is inv sed. structions of t ch changes pr struction nay occur if the ction is interru- an additional See the dsP	n the loop bei alid. Unexped he DO loop o rogram contro hese last inst uptible and su 5 levels may IC30F Family	ng executed o cted results m an NOT be: of flow ructions are u pports 1 level be provided y Reference	one time. ay occur ised. of nesting in softwar Manual fo		
Words:	<ol> <li>Using a if these</li> <li>The ve</li> <li>an in</li> <li>a DO</li> <li>Unexpense</li> <li>Note 1: 1</li> </ol>	an offset of -2, offsets are us ry <b>last two</b> ins struction whic or REPEAT in ected results r The DO instruct Nesting up to by the user. details. The linker will	6 0 will result in -1 or 0 is inv sed. structions of t ch changes pr struction nay occur if the ction is interru- an additional See the dsP	n the loop bei alid. Unexped he DO loop o rogram contro hese last inst uptible and su 5 levels may IC30F Family	ng executed o cted results m an NOT be: of flow ructions are u pports 1 level be provided y Reference	one time. ay occur ised. of nesting in softwar Manual fo		

## dsPIC30F Programmer's Reference Manual

Example 1         002000 LOOP6:         DO         W0, END6         ; Initiate DO loop (N           002004         ADD         W1, W2, W3         ; First instruction in the instreaction in the instruction in the instreaction in the instruction	in loop n loop
Before After	
Instruction	
PC 00 2000 PC 00 2004	
W0 0012 W0 0012	
DCOUNT 0000 DCOUNT 0012	
DOSTART FF FFFF DOSTART 00 2004	
DOEND FF FFFF DOEND 00 2010	
CORCON 0000 CORCON 0100 (DL=1)	
SR 0000 SR 0080 (DA=1)	
Example 2       002000 LOOPA:       DO       W7, ENDA ; Initiate DO loop (1000000000000000000000000000000000000	in loop
Before After	
Instruction	
PC 00 2000 PC 00 2004	
W7 EOOF W7 EOOF	
DCOUNT 0000 DCOUNT 200F	
DOSTART FF FFFF DOSTART 00 2004	
DOEND FF FFFF DOEND 00 2010	
CORCON 0000 CORCON 0100 (DL=1)	
SR 0000 SR 0080 (DA=1)	

Syntax: {la	abel:} ED	Wm*Wm,	Acc,	[Wx],	[Wy],	Wxd
		••••••	7.00,	[Wx]+=kx, [Wx]-=kx,	[Wy]+=ky,	
Operands:	Wx∈[N Wy∈[N	x,B] n ∈ [W4*W4, W /8, W9]; kx ∈ [- /10, W11]; ky ∈ W4 W7]	6, -4, -2, 2, 4	, 6]		
Operation:	([Wx]–[V (Wx)+kx (Wy)+ky	→Wy				
Status Affected:	OA, OB,	OAB, SA, SB,	SAB			<u> </u>
Encoding:	1111	0 0 mm	Alxx	00ii	iijj	jj11
	sign-exte results o Operanc support	n values specifi ended to 40-bit f [Wx] – [Wy] a Is Wx, Wxd and indirect and reg <b>4.14.1 "MAC</b> I	s and stored ire stored in V d Wyd specify gister offset a	in the specifi Vxd, which m / the pre-feto ddressing as	ed accumulat hay be the sai th operations	tor. The me as Wn which
	The 'm' l The 'A' b The 'x' b The 'i' bi	bits select the a bits selects the a bits select the p ts select the W ts select the W	operand regis accumulator for re-fetch differ a pre-fetch o	ter Wm for th or the result. rence Wxd do peration.		
Words:	1					
Cycles:	1					
Example 1	ED W4*W4, A	A, [W8]+=2,	[W10]-=2,	; [W8]-[] ; Post-i:	W4 to ACC W10] to W4 ncrement W8 ecrement W1	3
	Befo			After		
	Instruc		14/4	Instruct		
	N4 N8	009A	W4 W8		0057	
	/10	1100 2300	W0 W10		22FE	
AC			ACCA	00 0000	5CA4	
Data 11		007F	Data 1100		007F	
Data 23	800	0028	Data 2300		0028	

tructi	on		I	nstruct	ion
	009A	W4			0057
	1100	W8			1102
	2300	W10			22FE
A00	0000	ACCA	00	0000	5CA4
	007F	Data 1100			007F
	0028	Data 2300			0028
	0000	SR			0000

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				
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			0008			
ACCB	00	11EF	1F04			
Data 1200			6A7C			
Data 2508			2B3D			
SR			0000			

EDAC	Eucl	idean Distance	!		
Syntax: {labe	el:} EDAC Wm*	Wm, Acc,	[Wx]-=kx,	[Wy], [Wy]+=ky, [Wy]-=ky, [W11+W12]	Wxd
Operands:	Acc ∈ [A,B] Wm*Wm ∈ [W4* Wx ∈ [W8, W9]; Wy ∈ [W10, W11 Wxd ∈ [W4 W	kx ∈ [-6, -4, -2, 2 ]; ky ∈ [-6, -4, -2	2, 4, 6]	I	
Operation:	$(Acc(A \text{ or } B)) + ((Wx)-[Wy]) \rightarrow W$ $(Wx)+kx \rightarrow Wx$ $(Wy)+ky \rightarrow Wy$		cc(A or B)		
Status Affected:	OA, OB, OAB, S	A, SB, SAB		1	
Encoding:	1111 0	Omm Alxx	00ii	iijj	jj10
	sign-extended to results of [Wx] – Operands Wx, W support indirect a Section 4.14.1 " The 'm' bits selec The 'A' bit selects The 'x' bits selec	[Wy] are stored (xd and Wyd spo and register offso <b>MAC Pre-Fetch</b> at the operand res the accumulate	in Wxd, which ecify the pre-fe et addressing a es". egister Wm for pr for the result	may be the s tch operation as described i the square.	ame as Wi s which
	The 'i' bits select The 'j' bits select				
Words:	1				
Cycles:	1				
Example 1 ED	DAC W4*W4, A, [W	[8]+=2, [w10]	-=2, W4	; Square W4 ; add to A0 ; [W8]-[W10 ; Post-inco ; Post-decr	CCA )] to W4 rement W8
	Before		After		
	Instruction	-	Instruct		
W		-	/4	0057	
Wa	-		/8	1102	
W10		W <sup>-</sup>	-	22FE	
ACCA		ACC			
Data 110		Data 110		007F	
Data 230	0 0028	Data 230		0028	

SR

0000

SR

0000

Example 2 EDAC W5\*W5, B, [w9]+=2, [W11+W12], W5 ; Square W5 and

; 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			0008		
ACCB	00	3AD3	1050		
Data 1200			6A7C		
Data 2508			2B3D		
SR			0000		

EXCH		Exchange V	Wns and W	nd		
Syntax:	{label:}	EXCH	Wns,	Wnd		
Operands:	Wns $\in$ [W( Wnd $\in$ [W(					
Operation:	$(Wns) \leftrightarrow (Vns)$	Wnd)				
Status Affected:	None					
Encoding:	1111	1101	0000	0ddd	d000	SSS
Description:		the word cont must be use		working regist nd Wnd.	ers. Register	direct
				e first register. second regist	ter.	
	Note: T	nis instruction	only execu	tes in Word m	ode.	
Words:	1					
Cycles:	1					
Example 1	EXCH W1, W9	; Exc	hange the	contents o	f W1 and W	9
	Before		A	fter		
	Instruction	<u>ו</u>		ruction		
-	V1 55F		W1	A3A3		
	N9 A3A		W9	55FF		
\$	SR 000	0	SR	0000		
Example 2	EXCH W4, W5	; Exc	hange the	contents o	f W4 and W	5
	Before			fter		
	Instruction	ו 		ruction		
-	N4 ABC		W4	4321		
V	N4         ABC           N5         432           SR         000	1	W4 W5 SR	4321 ABCD 0000		

FBCL		Find Firs	t Bit Change	from Left		
Syntax:	{label:}	FBCL	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd		
Operands:	Ws ∈ [W0 Wnd ∈ [W					
Operation:	Max_Shift Sign = (W Temp = (W Shift = 0 While ( (Sl	= 15 s) & 0x8000 /s) << 1 nift < Max_9 Temp << 1 Shift + 1		<b>np &amp;</b> 0x8000)	== Sign) )	
Status Affected:	С		1		1	
Encoding:	1101	1111	0000	0ddd	dppp	SSSS
	negative v Ws and we bit number The next N the Least allows for values up. flag is set.	alue), starti prking towar r result is si Most Signific Significant t the immedia If a bit char When a bit	ng from the Me ds the Least S gn-extended to cant bit after the bit is allocated ate use of Wd nge is not foun change is fou	or a positive va ost Significant bit of o 16-bits and p e sign bit is allo bit number -14 with the SFTA d, a result of -1 nd, the C flag	bit after the f the word op laced in Wn ocated bit nu f. This bit ord c instruction 15 is returned is cleared.	sign bit of berand. Th d. mber 0 ar dering for scalin
	The 'p' bits	s select the	source Addres	e destination re ss mode. e source registe	-	
	Note:	This instru	ction operates	in Word mode	e only.	
Words:	1					
Cycles:	1					
Example 1	FBCL W1, W	9		bit change e result to		in W1
	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

5

FF1L	Find First One from Left				
Syntax:	{label:} FF1L Ws, Wnd [Ws], [Ws++], [Ws], [++Ws], [Ws],				
Operands:	Ws ∈ [W0 W15] Wnd ∈ [W0 W15]				
Operation:	$\begin{array}{l} Max\_Shift = 17\\ Temp = (Ws)\\ Shift = 1\\ While ( (Shift < Max\_Shift) && !(Temp & 0x8000) )\\ Temp = Temp << 1\\ Shift = Shift + 1\\ If (Shift == Max\_Shift)\\ 0 \rightarrow (Wnd)\\ Else\\ Shift \rightarrow (Wnd) \end{array}$				
Status Affected:	с <u>С</u>				
Encoding:	1100 1111 1000 0ddd dppp ssss				
Description:	Finds the first occurrence of a '1' starting from the Most Significant bit of Ws and working towards the Least Significant bit of the word operand. The bit number result is zero-extended to 16-bits and placed in Wnd. Bit numbering begins with the Most Significant bit (allocated number 1) and advances to the Least Significant bit (allocated number 16). A resu of zero indicates a '1' was not found, and the C flag will be set. If a '1' is				
	found, the C flag is cleared.				
	The 'd' bits select the address of the destination register. The 'p' bits select the source Address mode. The 's' bits select the address of the source register.				
	<b>Note:</b> This instruction operates in Word mode only.				
Words:	1				
Cycles:	1				
Example 1	FF1L W2, W5 ; Find the 1st one from the left in W2 ; and store result to W5				
	Before         After           Instruction         Instruction           W2         000A         W2         000A           W5         BBBB         W5         000D           SR         0000         SR         0000				

## dsPIC30F Programmer's Reference Manual

Example 2 FF1L [W2++], W5 ; Find the 1st one from the left in [W2]; and store the result to W5 ; Post-increment W2 Before After Instruction Instruction W2 2000 W2 2002 W5 BBBB W5 0000

SR

0000

0001 (C=1)

Data 2000

Data 2000

SR

0000

0000

5

FF1R	Find First One from Right
Syntax:	{label:} FF1R Ws, Wnd [Ws], [Ws++], [Ws], [++Ws], [Ws],
Operands:	Ws ∈ [W0 W15] Wnd ∈ [W0 W15]
Operation:	$\begin{array}{l} Max\_Shift = 17\\ Temp = (Ws)\\ Shift = 1\\ While ( (Shift < Max\_Shift) && !(Temp & 0x1) )\\ Temp = Temp >> 1\\ Shift = Shift + 1\\ If (Shift == Max\_Shift)\\ 0 \rightarrow (Wnd)\\ Else\\ Shift \rightarrow (Wnd) \end{array}$
Status Affected:	C C
Encoding:	1100 1111 0000 0ddd dppp ssss
Description:	<ul> <li>Finds the first occurrence of a '1' starting from the Least Significant bit of Ws and working towards the Most Significant bit of the word operand. The bit number result is zero-extended to 16-bits and placed in Wnd.</li> <li>Bit numbering begins with the Least Significant bit (allocated number 1) and advances to the Most Significant bit (allocated number 16). A result of zero indicates a '1' was not found, and the C flag will be set. If a '1' is</li> </ul>
	found, the C flag is cleared.
	The 'd' bits select the address of the destination register. The 'p' bits select the source Address mode. The 's' bits select the address of the source register.
	Note: This instruction operates in Word mode only.
Words:	1
Cycles:	1
Example 1	FF1R W1, W9 ; Find the 1st one from the right in W1 ; and store the result to W9 $$
	BeforeAfterInstructionInstructionW1000AW1W9BBBBW9SR0000SR

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

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

GOTO		Unconditic	onal Jump			
Syntax:	{label:}	GOTO	Expr			
Operands:			pression (but n linker to a lit23,		፪ [0 838860	)6].
Operation:	lit23 $\rightarrow$ PC NOP $\rightarrow$ Inst	ruction Reg	ister			
Status Affected:	None					
Encoding:						
1st word	0000	0100	nnnn	nnnn	nnnn	nnn0
2nd word	0000	0000	0000	0000	0nnn	nnnn
Description:	memory ran	ge. The PC Since the PC	nywhere within is loaded with C must always	the 23-bit lite	ral specified ir	n the
	The 'n' bits f	orm the targ	get address.			
		The linker wi used.	ill resolve the s	pecified expr	ession into the	e lit23 to be
Words:	2					
Cycles:	2					
Example 1	026000 026004 -	GOTO MOV	_THERE W0, W1	; J	ump to _THE	ERE
	027844 _THE 027846	RE: MOV	#0x400, W2		ode executi esumes here	
	Before	•		After		
	Instructio			Instruction	1	
	PC 02 60 SR 00	00	PC SR	02 7844 0000		
Example 2	000100 _code	e:		; st	tart of cod	e
	026000 026004	GOTO 	_code+2	; Jı	ump to _cod	e+2
	Before			After		
				Instruction		
	PC 02 60 SR 00		PC SR	00 0102 0000		

GOTO		Unconditio	nal Indirect	Jump		
Syntax:	{label:}	GOTO	Wn			
Operands:	Wn ∈ [W0	W15]				
Operation:	$0 \rightarrow PC < 0 >$	$\rightarrow$ PC<15:1				
Status Affected	None					
Encoding:	0000	0001	0100	0000	0000	SSSS
Description:	Zero is load into PC<15: boundary, V	ed into PC<2 1>. Since the Vn<0> is igno	22:16> and the PC must alword the procession of the procesion of the procession of the procession of the procession of t	ne value spec ways reside o	rds of progran ;ified in (Wn) on an even ac	is loaded
		select the ad	dress of the s	source regist	er.	
Words:	1					
Cycles:	2					
Example 1	006000 006002		74 70, W1	-	np uncondit 16-bit val	-
	007844 _THERE 007846	 : MOV #0 	)x400, W2	•	de executio sumes here	n
	Before Instruction		I	After nstruction		
	W4 7844	1	W4	7844		
	PC 00 6000	_	PC	00 7844		
	SR 0000	)	SR	0000		

INC		Increment f				
Syntax:	{label:}	INC{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f) + 1 $\rightarrow$ de	estination des	ignated by D	)		
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1100	OBDf	ffff	ffff	ffff
Description:	destination destination	Add one to the contents of the file register 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 'D' bit s	selects byte o selects the de select the ado	stination (0 f	or WREG, 1		
		The extensio rather than a denote a wor The WREG is	word operat d operation,	ion. You may but it is not re	y use a .w e equired.	
Words:	1					
Cycles:	1					
Example 1 INC	C.B 0x100	0;	Increment	0x1000 (B	yte mode)	
Data 1000 SR		Data 10	After Instruction 00 8F00 SR 0101	) (DC, C=1)		
Example 2 INC	C 0x1000,	WREG ; ;	Increment (Word mode		d store to	WREG
WREG Data 1000 SR		WRE Data 100 S		(DC, N=1)		

INC		Increment V	Ns			
Syntax:	{label:}	INC{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	(Ws) + 1 -	→ Wd				
Status Affected:	DC, N, OV	', Z, C				
Encoding:	1110	1000	0Bqq	qddd	dppp	SSSS
Description:	the destina	the contents ation register V /s and Wd.				
	The 'q' bits The 'd' bits The 'p' bits	selects byte of s select the de s select the ad s select the so s select the ad	estination Add Idress of the Jurce Address	Iress mode. destination re s mode.	gister.	).
	Note:	rather than a	a word opera	instruction de tion. You may but it is not re	use a .w ex	
Words:	1					
Cycles:	1					
Example 1 II	NC.B W1,	[++W2]	-	ncrement W ment W1 and mode)		W2
	Before		After			
W		,	Instructio	n		
	1 FF7F					
			W1 FF7F			
W	2 2000	,	W2 2001	-		
	2 2000 0 ABCD	N Data 20	W2 2001	(DC, N, OV	=1)	
W Data 200 S	2 2000 0 ABCD	N Data 20	W2 2001 000 80CD SR 010C	W1 and st	·	
W Data 200 S	2 2000 0 ABCD R 0000 NC W1, W2 Before	Data 20 ; ;	W2 2001 000 80CD SR 010C Increment (Word mod After	W1 and st e)	·	
W Data 200 S Example 2 II	2 2000 0 ABCD R 0000 NC W1, W2 Before Instruction	Data 20 ; ;	W2 2001 000 80CD SR 010C Increment (Word mod After Instructio	W1 and st e)	·	
W Data 200 S Example 2 II	2 2000 0 ABCD R 0000 NC W1, W2 Before Instruction 1 FF7F	Data 20	W2 2001 000 80CD SR 010C Increment (Word mod After Instructio W1 FF7F	W1 and st e)	·	
W Data 200 S Example 2 II	2         2000           ABCD         ABCD           R         00000           NC         W1, W2           Before         Instruction           1         FF7F           12         2000	Data 20 ; ;	W2 2001 000 80CD SR 010C Increment (Word mod After Instructio	W1 and st e)	·	

INC2		Increment f	by 2			
Syntax:	{label:}	INC2{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	$(f) + 2 \rightarrow de$	estination des	ignated by D			
Status Affected:	DC, N, OV,	Z, C				
Encoding:	1110	1100	1BDf	ffff	ffff	ffff
Description:	destination destination	Add two to the contents of the file register 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.				ne
	The 'D' bit s	selects byte o selects the de select the ado	stination (0 f	or WREG, 1 f		
		The extensio rather than a denote a wor	word operat	tion. You may	/usea.we	
Words:	1					
Cycles:	1					
Example 1 IN	C2.B 0x10	,	rement 0x1 te mode)	000 by 2		
Data 1000 SF		Data 10	After Instruction 00 8F01 SR 0101	) (DC, C=1)		
Example 2 INC	C2 0x1000,		ncrement 0 Word mode)	-	and store	to WREG
WREG Data 1000 SR		WRE Data 100 S		(DC, N=1)		

		Increment	Ws by 2			
Syntax:	{label:}	INC2{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	(Ws) + 2 -	→ Wd				
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	1110	1000	1Bqq	qddd	dppp	SSSS
Description:		n register Wd.		register Ws ar ct or indirect a	•	
	The 'q' bits The 'd' bits The 'p' bits	s select the de s select the ac s select the so	estination Add ddress of the o ource Address	destination reg	jister.	
	Note:			instruction de	enotes a byte	e operatio
				tion. You may but it is not rec		xtension to
Words:	1					xtension t
	1 1					xtension t
Cycles:	1		rd operation, ; Pre-incr	ement W2 t by 2 and	quired.	
Cycles:	1 INC2.B W1 Before	denote a wo	rd operation, ; Pre-incr ; Incremen ; (Byte mo After	ement W2 t by 2 and de)	quired.	
Cycles: Example 1	1 INC2.B W1 Before Instructio	denote a wo	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instruction	ement W2 t by 2 and de)	quired.	
Cycles: Example 1	1 ENC2.B W1 Before Instructio V1 FF7F	denote a wo	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instruction W1 FF7E	ement W2 t by 2 and de)	quired.	
Cycles: Example 1	1 ENC2.B W1 Before Instructio V1 FF7F V2 2000	denote a wo	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instructio W1 FF7F W2 2001	ement W2 t by 2 and de)	quired.	
Cycles: Example 1 I V V Data 20	1 ENC2.B W1 Before Instructio V1 FF7F V2 2000	denote a wo , [++₩2] n	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instructio W1 FF7F W2 2001	ement W2 t by 2 and de)	<b>quired.</b> store to W	
Cycles: Example 1 I V Data 20 S	1 Before Instructio V1 FF7F V2 2000 00 ABCD	denote a wo , [++₩2] n ] Data 2 W2 ; Ir	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instruction W1 FF7F W2 2001 2000 81CI SR 0100	ement W2 t by 2 and de)	quired. store to Wi =1)	
Cycles: Example 1 I V V Data 20 S	1 ENC2.B W1 Before Instructio V1 FF7F V2 2000 00 ABCD 5R 0000 ENC2 W1, Before	denote a wo , [++₩2] n ] Data 2 W2 ; Ir ; (v	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instruction W1 FF7E W2 2001 2000 81CE SR 0100 hcrement W1 word mode) After	ement W2 t by 2 and de) (DC, N, OV by 2 and s	quired. store to Wi =1)	
Cycles: Example 1 I V Data 200 S Example 2 I	1 ENC2.B W1 Before Instructio V1 FF7F V2 2000 00 ABCD SR 0000	denote a wo , [++₩2] n ] Data 2 W2 ; Ir ; (v	rd operation, ; Pre-incr ; Incremen ; (Byte mo After Instruction W1 FF7F W2 2001 2000 81CT SR 0100 hcrement W1 word mode)	ement W2 t by 2 and de) (DC, N, OV by 2 and s	quired. store to Wi =1)	

© 2005 Microchip Technology Inc.

IOR		Inclusive O	R f and WRE	EG		
	{label:}	IOR{.B}	f	{,WREG}		
Operands:	f ∈ [0 8 <sup>.</sup>	191]				
Operation:	-	$(REG) \rightarrow dest$	nation design	nated by D		
Status Affected:	N, Z	,	Ū			
Encoding:	1011	0111	OBDf	ffff	ffff	ffff
Description:	register W the destination destination WREG is	the logical incl REG and the ation register. In register. If W not specified, selects byte c	contents of th The optional REG is spec the result is s	ne file registe WREG opera ified, the resu stored in the f	r and place th and determin ult is stored in ile register.	ne result in es the WREG. I
		selects the de			for file registe	er).
		The extension rather than a denote a work The WREG i	word operation,	tion. You may but it is not re	y use a .w ex equired.	
Words:	1					
Cycles:	1					
Example 1 IO	R.B 0x10	; 00	IOR WREG t (Byte mode		(Byte mode	e)
	Before		After			
	Instruction		Instructio	n		
WREG		WRI				
Data 1000 SR		Data 10	00 FF34 SR 0000			
	0000			J		
Example 2 IO	R 0x1000	, WREG ; ;	IOR (0x100 (Word mode			
WREG Data 1000 SR	0 FAB	WRI Data 10 (N=1)				

IOR	Inclus	ive OR Literal ar	nd Wn
Syntax:	{label:} IOR{.E	3} #lit10,	Wn
Operands:	lit10 ∈ [0 255] fc lit10 ∈ [0 1023] Wn ∈ [W0 W15]	for word operation	n
Operation:	lit10.IOR.(Wn) $\rightarrow$ V	Wn	
Status Affected:	N, Z		
Encoding:	1011 003	11 OBkk	kkkk kkkk dddo
Description:	and the contents o the working registe	f the working regi er Wn.	peration of the 10-bit literal operand ster Wn and place the result back ir
	The 'B' bit selects The 'k' bits specify The 'd' bits select t	the literal operan	
	denote 2: For byte value [	a word operation e operations, the l 0:255]. See <b>Sec</b>	ation. You may use a .w extensior , but it is not required. literal must be specified as an unsigr tion 4.6 "Using 10-bit Literal Op n using 10-bit literal operands in B
Words:	1		
Cycles:	1		
Example 1	IOR.B #0xAA, W9	; IOR 0xAA ; (Byte mo	
	Before Instruction W9 1234 SR 0000	After Instructi W9 12BI SR 0008	3
Example 2	IOR #0x2AA, W4	; IOR 0x2A ; (Word mo	
	Before		

# dsPIC30F Programmer's Reference Manual

IOR		Inclusive (	OR Wb and S	hort Literal		
Syntax:	{label:}	IOR{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	(Wb).IOR.	lit5 $\rightarrow$ Wd				
Status Affected:	N, Z					
Encoding:	0111	0www	wBqq	qddd	d11k	kkkk
Description:	register W destination	b and the 5-b n register Wd	clusive OR ope bit literal opera l. Register dire indirect addre	nd and place	the result in t g must be use	the d for Wb
	The 'B' bit The 'q' bits The 'd' bits	selects byte s select the d s select the a s provide the The extension	address of the or word opera estination Add ddress of the literal operand ion . B in the	tion (0 for wo lress mode. destination re d, a five-bit in instruction d	ord, 1 for byte egister. teger number. enotes a byte	e operati
			a word opera ord operation,			xtension
Words:	1				•	
Cycles:	1					
Example 1 I	OR.B W1,	#0x5, [W9+	; Store	V1 and 0x5 e to [W9] increment	(Byte mode W9	)
W Data 200	Before           Instruction           /1         AAAA           /9         2000           00         0000           0R         0000	Data 2	After Instruction W1 AAAA W9 2001 2000 00AF SR 0008			
Example 2 I	OR W1, #0	0x0, W9	; IOR W1 wi ; Store to		ord mode)	
V	Before           Instruction           /1         0000           /9         A34D           GR         0000	I	After Instruction W1 0000 W9 0000 SR 0002			

IOR		Inclusive	OR Wb and	l Ws		
Syntax:	{label:}	IOR{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	) W15]				
Operation:	-	.(Ws) $\rightarrow$ Wd				
Status Affected:	N, Z					
Encoding:	0111	0www	wBqq	qddd	dppp	SSSS
	The 'B' bi The 'q' bit The 'd' bit The 'p' bit	t selects byte is select the c is select the a is select the s	or word ope destination A address of th source Addre	ddress mode e destination	word, 1 for byte register.	ə).
	Note:	rather than	a word ope		denotes a by nay use a .w required.	
Words:	1					
Cycles:	1					
Example 1	OR.B W1,	[₩5++], [	;	IOR W1 and Store resul Post-increm		
	Before		Aft			
V		-	Instru W1 AZ			
	V1 AAAA V5 2000	-		AA 01		
	<b>V9</b> 2400	_		01		
Data 20				55		
Data 24				FF		
5	SR 0000		SR 00	08 (N=1)		

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	
In	struction	
W1	AAAA	
W5	5555	
W9	FFFF	
SR	0008	(N=1)

LAC		Load Acc	umulator			
Syntax:	{label:}	LAC	Ws, [Ws], [Ws++], [Ws], [Ws], [++Ws], [Ws+Wb],	{#Slit4,}	Acc	
Operands:	Ws ∈ [W0 Wb ∈ [W0 Slit4 ∈ [-8 Acc ∈ [A,I	) W15] +7]				
Operation:	Shift <sub>Slit4</sub> (E	Extend(Ws)) -	$\rightarrow$ Acc(A or B)			
Status Affected:	OA, OB, O	DAB, SA, SB,	SAB			
Encoding:	1100	1010	Awww	wrrr	rggg	SSSS
	operand ir	ndicates an a	It in the specifies and indicates an rithmetic right sl be 1.15 fraction	nift. The data	a stored in the	source
	operand ir register is sign-exter shifting. The 'A' bit The 'w' bit The 'r' bits The 'g' bit	ndicates an a assumed to l aded (through specifies the s specify the s encode the s select the s	and indicates an	nift. The data al data and is o-backfilled ( cumulator. Nb. e-shift. mode.	a stored in the s s automatically	source
	operand ir register is sign-exter shifting. The 'A' bit The 'w' bit The 'r' bits The 'g' bit	ndicates an a assumed to l aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu	and indicates an rithmetic right sl be 1.15 fractiona bit 39) and zero destination acc offset register V accumulator pri- ource Address	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or	a stored in the s automatically (bits [15:0]), pri n-extension da r causes a sate	or to ata into the
Words:	operand ir register is sign-exter shifting. The 'A' bit The 'k' bit The 'r' bits The 'g' bit The 's' bit	ndicates an a assumed to l aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu	and indicates an rithmetic right sl be 1.15 fractions bit 39) and zero destination acc offset register N accumulator pro- source Address source register ation moves mo- mulator register	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or	a stored in the s automatically (bits [15:0]), pri n-extension da r causes a sate	or to
Words: Cycles:	operand ir register is sign-exter shifting. The 'A' bit The 'k' bit The 'r' bits The 'g' bit The 's' bit <b>Note:</b>	ndicates an a assumed to l aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu	and indicates an rithmetic right sl be 1.15 fractions bit 39) and zero destination acc offset register N accumulator pro- source Address source register ation moves mo- mulator register	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or	a stored in the s automatically (bits [15:0]), pri n-extension da r causes a sate	or to
Cycles:	operand ir register is sign-exter shifting. The 'A' bit The 'A' bit The 'r' bits The 'g' bit The 's' bit <b>Note:</b> 1	ndicates an a assumed to l aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu	and indicates an rithmetic right sl be 1.15 fractions bit 39) and zero e destination acc offset register 1 accumulator pro- source Address source register ation moves mo mulator register overflow and s ; Load AC ; Content ; Post ir	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sigur r (AccxU), or aturation bits CCB with [ is of [W4] icrement W saturatio	a stored in the s s automatically (bits [15:0]), pri n-extension da r causes a satu s will be set. W4] << 3 do not cha:	or to ata into the uration, the
Cycles:	operand ir register is sign-exter shifting. The 'A' bit The 'w' bit The 'r' bits The 'g' bit The 's' bit <b>Note:</b> 1	ndicates an a assumed to I nded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu appropriate	and indicates an rithmetic right sl be 1.15 fractions bit 39) and zero e destination acc offset register N accumulator pro- cource Address source register ation moves more mulator register overflow and s ; Load Ad ; Content ; Post ir ; Assume	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sigur r (AccxU), or aturation bits CCB with [ is of [W4] icrement W saturatio	a stored in the s s automatically (bits [15:0]), pri n-extension da r causes a satu s will be set. W4] << 3 do not cha: 4 n disabled	or to ata into the uration, the
Cycles:	operand ir register is sign-exter shifting. The 'A' bit The 'W' bit The 'r' bits The 'g' bit The 's' bit Note: 1 1 LAC [W4+-	hdicates an a assumed to l inded (through a specifies the ts specify the s encode the s select the s s specify the lf the opera upper Accu appropriate	and indicates an rithmetic right sl be 1.15 fractions bit 39) and zero e destination acc offset register N accumulator pro- cource Address source register ation moves more mulator register overflow and s ; Load Ad ; Content ; Post ir ; Assume	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), on aturation bits CCB with [ :s of [W4] crement W saturatio = 0)	a stored in the s s automatically (bits [15:0]), pri n-extension da r causes a satu s will be set. W4] << 3 do not cha: 4 n disabled r	source or to ata into the uration, the
Cycles: Example 1	operand ir register is sign-exter shifting. The 'A' bit The 'A' bit The 'r' bits The 'g' bit The 'g' bit Note: 1 1 LAC [W4+-	hdicates an a assumed to I aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu appropriate	and indicates an rithmetic right sl be 1.15 fractions be 1.15 fractions e destination acc offset register N accumulator pre- source Address source register ation moves mo- mulator register overflow and s ; Load AC ; Content ; Post ir ; Assume ; (SATB =	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sign r (AccxU), or aturation bits CCB with [ is of [W4] increment W saturatio = 0) Afte	a stored in the s         s automatically         (bits [15:0]), pri         (bits	source or to ata into the uration, the
Cycles: Example 1	operand ir register is sign-exter shifting. The 'A' bit The 'W' bit The 'r' bits The 'g' bit The 's' bit Note: 1 1 LAC [W4+- B Be Inst	hdicates an a assumed to l aded (through specifies the ts specify the s encode the s select the s s specify the If the opera upper Accu appropriate	and indicates an rithmetic right sl be 1.15 fractions be 1.15 fractions e destination acc offset register N accumulator pro- source Address source register ation moves mo mulator register overflow and s ; Load AC ; Content ; Post ir ; Assume ; (SATB =	hift. The data al data and is o-backfilled ( cumulator. Wb. e-shift. mode. Ws. ore than sigur (AccxU), or aturation bits CCB with [ is of [W4] increment W saturatio = 0) Afte Instruct	a stored in the s         s automatically         (bits [15:0]), pri         (bits	source or to ata into the uration, the

Example 2	LAC	[W2],	#7, A	;	Pre-dect Load ACC Contents Assume s (SATA =	CA w s of satu	ith [1 [W2]	do no
		Before Instructio				I	After nstruct	
	W2		4002		W2			4000
	ACCA	00 5125 2	ABCD		ACCA	FF	FF22	1000

9108

1221

0000

Data 4000

Data 4002

SR

> 7 ot change abled

9108

1221

0000

Data 4000

Data 4002

SR

LNK		Alloca	te Stack Frame			
Syntax:	{label:}	LNK	#lit14			
Operands:	lit14 ∈ [0 .	16382	]			
Operation:	$(W14) \rightarrow (W15) + 2$ $(W15) \rightarrow V$ $(W15) + Iit$	→ W15 W14	15			
Status Affected:	None					
Encoding:	1111	101	LO 00kk	kkkk	kkkk	kkk0
	<ul> <li>unsigned 14-bit literal operand. This instruction supports a maximum stack frame of 16382 bytes.</li> <li>The 'k' bits specify the size of the stack frame.</li> <li>Note: Since the stack pointer can only reside on a word boundary of the stack pointer can only of the stack pointer can only on a word b</li></ul>					
	Note:		ust be even.			r bouridary,
Words:	1					
Cycles:	1					
Example 1 LNK	0Ax0#	; Allo	ocate a stack f	frame of 16	50 bytes	
	Befo	re		After		
	Instruc	tion		Instructio	n	
W14		2000	W14	2	2002	
W15		2000	W15		20A2	
Data 2000		0000	Data 2000		2000	
SR		0000	SR	C	0000	

LSR		Logical Sh	nift Right f			
Syntax:	{label:}	LSR{.B}	f	{,WREG}		
Operands:	f∈ [0 8 <sup>-</sup>	-				
Operation:	(f<0>) - For word of $0 \rightarrow De$	st<7> $\rightarrow$ Dest<6:0 $\rightarrow$ C peration: st<15> $\rightarrow$ Dest<14				
Status Affected:	N, Z, C	-1			1	1
Encoding:	1101	0101	OBDf	ffff	ffff	ffff
Description:	in the dest shifted into Significant The option is specified result is st	ination regist the Carry bi bit of the dea al WREG op d, the result is ored in the fil	e file register o er. The Least : t of the Status stination regist erand determi s stored in WF e register. or word opera	Significant bit register. Zero er. nes the destir EG. If WREG	of the file reg is shifted into nation register is not specifi	ister is o the Most r. If WREG ed, the
	The 'D' bit The 'f' bits <b>Note 1:</b>	selects the d select the ad The extensi rather than denote a wo	lestination (0 f ddress of the fi ion .B in the a word opera ord operation, I is set to worki	or WREG, 1 fo le register. instruction de tion. You may out it is not ree	or file register enotes a byte / use a .w e quired.	). e operation
Words:	1					
Cycles:	1					
Example 1	LSR.B 0x6		ically shif te mode)	t right (0>	(600) by on	e
Data 6	Before Instructio 600 55FF SR 0000	-	After Instructi a 600 5571 SR 0001	7		
Example 2	LSR 0x600		Logically s Store to WR (Word mode)	-	(0x600) by	one
Data 6 WR		Dat	After Instructi a 600 55FF /REG 2AFF SR 0001	7		

LSR		Logical S	hift Right W	6		
Syntax:	{label:}	LSR{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	(Ws<0> For word 0 → Wo	d<7> $1>) \rightarrow Wd<6$ $y \rightarrow C$ operation: d<15> $f:1>) \rightarrow Wd<1>$				
	0->	►C				
Status Affected:	N, Z, C	1			T	<u>,                                    </u>
Encoding:	1101	0001	0Bqq	qddd	dppp	SSSS
Description:	the result i shifted into Most Sign	n the destine the Carry b	ation register bit of the State Wd. Either re	ister Ws one b Wd. The Leas us register. Ze egister direct o	st Significant ro is shifted i	bit of Ws is nto the
	The 'q' bits The 'd' bits The 'p' bits	s select the s select the s select the s	destination A address of th source Addre	ration (0 for w ddress mode. e destination r ss mode. e source regis	egister.	e).
	Note:	rather than	a word oper	e instruction d ation. You ma n, but it is not r	yusea.we	•
Words:	1					
Cycles:	1					
Example 1	LSR.B WO, W		SR W0 (Byt tore resul			
	Before Instruction W0 FF03 W1 2378 SR 0000		After Instruction W0 FF03 W1 2301 SR 0001			

Example 2

LSR W0, W1

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

Before				
Instruction				
W0	8000			
W1	2378			
SR	0000			

After Instruction			
W0	8000		
W1	4000		
SR	0000		

					Short Litera		
Syntax:		{label:}	LSR	Wb,	#lit4,	Wnd	
Operands:		Wb ∈ [W0 lit4 ∈ [0 Wnd ∈ [W					
Operation:		$0 \rightarrow Wnd$	→ Shift_Val <15:15-Shift hift_Val> →	:_Val+1> Wnd<15-Shift	_Val:0>		
Status Affected:		N, Z					
Encoding:		1101	1110	0www	wddd	d100	kkkk
Description:		unsigned	literal and st		in the destin	ster Wb by the ation register d.	
		The 'd' bit	s select the	address of th address of th e literal opera	e destination		
		Note:	This instru	ction operate	s in Word mo	de only.	
Words:		1					
Cycles:		1					
Example 1	LSR	W4,	‡14, W5	; LSR W4 ] ; Store re	oy 14 esult to W	5	
		Before		After			
	F	nstruction		Instructio	on		
	W4	C800		W4 C800			
	W5 SR	1200 0000		W5 0003 SR 0000			
Example 2	LSR	W4,	‡1, W5	; LSR W4 ] ; Store re	oy 1 esult to W	5	
		Before struction 0505 F000 0000		After Instruction W4 0505 W5 0282 SR 0000			

LSR		Logical	Shift Right b	y Wns		
Syntax:	{label:}	LSR	Wb,	Wns,	Wnd	
Operands:	Wns ∈ [\	0 W15] V0W15] V0 W15]				
Operation:	$0 \rightarrow Wno$	> → Shift_ I<15:15-Sh Shift_Val> -		ift_Val:0>		
Status Affected:	N, Z					
Encoding:	1101	1110	) Owww	wddd	d000	SSSS
Description:	Significa destinatio Wnd.	nt bits of W on register	ns (only up to Wnd. Direct a	15 positions) ddressing mi	gister Wb by th and store the ust be used for	result in th
	The 'd' b	its select th	e address of t e address of t e address of t	he destinatio	on register.	
			ruction operat greater than		node only. be loaded with	0x0.
Words:	1					
Cycles:	1					
Example 1	LSR W0,	W1, W2	; LSR W0 ; Store 1	by W1 result to	W2	
	Before Instruction		After Instruct			
	W0 C00C		W0 C00			
	W1 0001 W2 2390		W1 000 W2 600			
	W2 2390 SR 0000		W2 600 SR 000			
Example 2	LSR W5,	W4, W3	; LSR W5 ; Store 1	by W4 result to	W3	
	Before		After			
			Instructio			
	W3 DD43 W4 000C		W3 000 W4 000			
	W4 000C W5 0800		W4 000 W5 080			
	SR 0000		SR 000			

MAC			Multiply and	Accumulate			
Syntax:	{label:}	MAC	Wm*Wn, Acc	$\{,[Wx], Wxd\}$	{,[W	/y], Wyd}	{,AWB]
				{,[Wx]+=kx, V	Wxd} {,[W	/y]+=ky, Wyd	}
				{,[Wx]-=kx, V	Vxd} {,[W	/y]-=ky, Wyd}	
				{,[W9+W12],	Wxd} {,[W	/11+W12], W	yd}
Operands:		$\begin{array}{l} Acc \in [x] \\ Wx \in [y] \\ Wy \in [y] \end{array}$	V8, W9]; kx ∈ [ V10, W11]; ky ∈	-6, -4, -2, 2, 4 ፪ [-6, -4, -2, 2	, 6]; Wxd e	≡ [W4 W7]	-
Operation:		(Acc(A ([Wx])– ([Wy])–	[W13, [W13]+= or B)) + (Wm)* → Wxd; (Wx)+kz → Wyd; (Wy)+ky or A)) rounded	(Wn) → Acc(A k→Wx /→Wy	A or B)		
Status Affecte	d:	OA, OB	, OAB, SA, SB	, SAB			
Encoding:		1100	0 mmm	A0xx	yyii	iijj	jjaa
Description:		operand store th	<ul> <li>the contents of ds in preparation</li> <li>e unspecified and is sign-extend</li> <li>allator.</li> </ul>	on for another	MAC type i esults. The	instruction ar 32-bit result	nd optional of the sign
		which s Section optiona	ds Wx, Wxd, W upport indirect 1 4.14.1 "MAC I store of the "c 1 4.14.4 "MAC	and register of <b>Pre-Fetches</b> ther" accumu	offset addr ". Operand lator, as de	essing, as de d AWB speci	escribed in
		The 'A' The 'x' The 'y' The 'i' b The 'j' b	bits select the bit selects the p bits select the p bits select the p bits select the V bits select the V bits select the V	accumulator f bre-fetch Wxd bre-fetch Wyd Vx pre-fetch o Vy pre-fetch o	or the resu destinatio destinatio peration. peration.	ılt. n. n.	ne multiply.
		Note		CORCON<0>	>, determin	es if the mul	tiply is
Words:		Note		CORCON<0> r an integer.	, determin	es if the mul	tiply 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

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

; 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

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

	Befor Instruct		After Instruction				
W4		1000	W4			5BBE	
W5		3000	W5			C967	
W8		0A00	W8			09FE	
W10		1800	W10			1802	
W13		2000	W13			0001	
ACCA	23 5000	2000	ACCA	23	5600	2000	
ACCB	00 0000	8F4C	ACCB	00	0000	1F4C	
Data 0A00		5BBE	Data 0A00			5BBE	
Data 1800		C967	Data 1800			C967	
CORCON		00D0	CORCON			00D0	
SR		0000	SR			8800	(OA, OAB=1)

MAC		Square and	Accumulate				
Syntax: {label:	} MAC	Wm*Wm, Acc				, Wyd}	
			{,[Wx]+=kx, \	-			
			{,[Wx]-=kx, V	•			
			{,[W9+W12],	Wxd}	{,[W11	I+W12], Wyd}	
Operands:	$Acc \in [A, Wx \in Water W$	∈ [W4*W4, W B] 3, W9]; kx ∈ [-( 10, W11]; ky ∈	6, -4, -2, 2, 4,	6]; Wx	- kd ∈ [N		
Operation:	$([Wx]) \rightarrow V$	<sup>-</sup> B)) + (Wm)*(\ Wxd; (Wx)+kx- Wyd; (Wy)+ky-	→Wx	or B)			
Status Affected:	OA, OB,	OAB, SA, SB,	SAB				
Encoding:	1111	0 0 mm	A0xx	УУ	rii	iijj	jjoo
Description:	preparation unspecifie	ne contents of a on for another ed accumulato nded to 40-bits	MAC type inst r results. The	ruction 32-bit	and o result	ptionally store of the signed i	the multiply is
	Operands Wx, Wxd, Wy and Wyd specify optional pre-fetch operations, which support indirect and register offset addressing, as described in <b>Section 4.14.1 "MAC Pre-Fetches"</b> .						
	The 'A' bi The 'x' bi The 'y' bi The 'i' bit	its select the c t selects the a ts select the p ts select the p s select the W s select the W s select the W	ccumulator fo re-fetch Wxd o re-fetch Wyd o x pre-fetch op	r the re destina destina peration	esult. ation. ation. n.	e square.	
	Note:	The IF bit, C or an intege		detern	nines if	the multiply is	fractional
Words:	1						
Cycles:	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)

	Befor Instruc	-		I	After nstruct	
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)

	Befor Instruct			Afte Instru	•••	
W7		76AE	W7		23FF	
W11		2000	W11		1FFE	
ACCA	FE 9834	4500	ACCA	FF 0631	E 0188	
Data 2000		23FF	Data 2000		23FF	
CORCON		00D0	CORCON		00D0	
SR		0000	SR		8800	(OA, OAB=1)
						-

Syntax:	{label:}	MOV{.B}	f	{,WREG}		
Operands:	f ∈ [0 8	191]				
Operation:	-	ination designation	ated by D			
Status Affected:	N, Z	0	,			
Encoding:	1011	1111	1BDf	ffff	ffff	ffff
Description:	The optic WREG is specified, to modify	contents of the nal WREG of specified, th the result is st the status regist selects byte of	perand dete e result is ored back to ster.	ermines the stored in W the file regis	destination REG. If WR ter and the or	register. EG is no nly effect i
	The 'D' bit	selects the ad	stination (0	for WREG, 1		
	2:	The extension rather than a denote a wor The WREG is When moving to Wnd" (pa (W0:W15) to	word opera d operation, s set to work g word data f ge 5-147) in	tion. You may but it is not re ing register V rom file regis struction allo	y use a .w ex equired. V0. ter memory, t ws any worki	ktension t
Words:	1					
Cycles:	1					
Example 1 MOV	.B TMR0,	WREG ; mc	ve (TMR0)	to WREG (B	Byte mode)	
WREG (W0) TMR0 SR	2355	WREG (W0 TMR( SF	2355			
Example 2 MOV	0x800	; up	date SR ba	ased on (02	(800) (Word	d mode)
Data 0800 SR		Data 0800 SF				

MOV		Move WRE	G to f			
Syntax:	{label:}	MOV{.B}	WREG,	f		
Operands: Operation:	f ∈ [0 8 (WREG) -	-				
Status Affected:	None					
Encoding:	1011	0111	1B1f	ffff	ffff	ffff
Description:		contents of the ile register.	e default wor	king register	WREG into t	he
		selects byte of select the ad			ord, 1 for byt	e).
	2:		move. You r but it is not re s set to work g word data nory, the "MO"	nay use a .v equired. ting register \ from the work V Wns to :	v extension t	o denote a array to file 48) instruc-
Words:	1	-				
Cycles:	1					
Example 1 MOV	.B WREG,	0x801	; move WRI	EG to 0x80	1 (Byte mo	de)
WREG (W0) Data 0800 SR	Before Instruction 98F3 4509 0000	WREG (\ Data 0				
Example 2 MOV	WREG,	DISICNT	; move WI	REG to DIS	ICNT	
WREG (W0) DISICNT SR	Before Instruction 00A0 0000 0000	WREG ( DISIO		) )		

MOV		Move f to W	nd			
Syntax:	{label:}	MOV	f,	Wnd		
Operands:	f ∈ [0 65 Wnd ∈ [W					
Operation:	$(f) \to Wnd$					
Status Affected:	None					
Encoding:	1000	Offf	ffff	ffff	ffff	dddd
Description:	register ma be word al	vord contents ay reside anyv igned. Registe select the ade	vhere in the a	32K words of ressing must	data memory	/, but mus
		select the ad			egister.	
Words:		Since the file upper 15 bits assumed to I To move a by to Destina	s of the file re be '0'). yte of data fr	egister addre	ss are encod er memory, t	ed (bit o i he "MOV
Cycles:	1					
Cycles.	1					
Example 1 MOV	CORCON	, W12 ;	move COR	CON to W12		
W12 CORCON SR	Before nstruction 78FA 00F0 0000	W CORCC S		ו		
Example 2 MOV	0x27F	E, W3 ;	move (0x2	7FE) to W3		
W3 Data 27FE SR	Before nstruction 0035 ABCD 0000	Data 27F	After Instruction V3 ABCD FE ABCD SR 0000	ı		

MOV		Move Wns	to f			
Syntax:	{label:}	MOV	Wns,	f		
Operands:	f ∈ [0 6 Wns ∈ [W					
Operation:	$(Wns) \to f$					
Status Affected:	None					
Encoding:	1000	lfff	ffff	ffff	ffff	SSSS
Description:	register. T	word contents he file register out must be wo /n.	may reside	anywhere in t	he 32K words	s of data
		select the ade select the ade		-	er.	
	2: 3:	assumed to I To move a b	e register ad s of the file r be '0'). yte of data to	dress must be egister addre	e word aligne ss are encod nemory, the "	ed (bit 0 is
Words:	1					
Cycles:	1					
Example 1 M	OV W4, X	MDOSRT	; move W4	to XMODSRT		
W XMODSR S	T 1340	XMODS	After Instruction W4 1200 RT 1200 SR 0000	]		
Example 2 M	. 8W VC	0x1222	; move W8	to data ad	dress 0x122	22
W Data 122 S	2 FD88	Data 12	After Instruction W8 F200 222 F200 SR 0000	]		

MOV.B		Move 8-b	it Literal to V	Vnd		
Syntax:	{label:}	MOV.B	#lit8,	Wnd		
Operands:	lit8 $\in$ [0 . Wnd $\in$ [\	255] N0 W15]				
Operation:	lit8 $\rightarrow$ W	nd				
Status Affected:	None					
Encoding:	1011	0011	1100	kkkk	kkkk	dddd
Description:		te of Wnd is		ed into the low Register direct		
			e value of the address of th	literal. e working regis	ster.	
	Note:	This instru must be p	-	es in Byte moo	le and the .E	extensior
Words:	1					
Cycles:	1					
Example 1	MOV.B #0	x17, W5	; load W	5 with #0x17	' (Byte mod	e)
	Before		Afte			
		n T	Instruct			
	W5 7899 SR 0000		W5 781 SR 000			
	<b>3h</b> 0000		<b>3n</b> 000	0		
Example 2	MOV.B #0	xFE, W9	; load W	9 with #0xFE	E (Byte mod	e)
	Before Instructio W9 AB23 SR 0000	n ] ]	Afte Instruct W9 ABF SR 000	tion E		

ΜΟΥ		Move 16-bit	t Literal to W	/nd		
Syntax:	{label:}	MOV	#lit16,	Wnd		
Operands:	lit16 ∈ [-32 Wnd ∈ [W	2768 65535 0 W15]	]			
Operation:	lit16 $\rightarrow$ Wr	nd				
Status Affected:	None					
Encoding:	0010	kkkk	kkkk	kkkk	kkkk	dddd
Description:	The 16-bit be used for	literal 'k' is loa r Wnd.	aded into Wr	d. Register d	irect address	ing must
		s specify the v s select the ad			ster.	
		This instructi The literal ma or unsigned	ay be specifi	ed as a signe		68:32767],
Words:	1					
Cycles:	1					
Example 1 MC	DV #0x423	81, W13	; load W1	3 with #0x	4231	
W1 SI			After Instructio 13 4231 SR 0000	n ] ]		
Example 2 MC	DV #0x4,	W2	; load W2	with #0x4		
W Si Example 3		5	After Instructio V2 0004 SR 0000	n ] with #-10	0.0	
Example 3 MC W SI	Before Instruction 8 23FF	V	FC18 After Instructio V8 FC18 SR 0000			

MOV		Move [Ws	with offset] to	Wnd		
Syntax:	{label:}	MOV{.B}	[Ws+Slit10],	Wnd		
Operands:		512 511] fo 1024 1022	or byte operatio ] (even only) fo		tion	
Operation:	[Ws+Slit10	$] \rightarrow Wnd$				
Status Affected:	None					
Encoding:	1001	0kkk	kBkk	kddd	dkkk	SSSS
Description:	range of S maintain w used for th	lit10 is increa vord address le source, an	lit10] are loade ased to [-1024 . alignment. Reg d direct addres	1022] and gister indirect sing must be	Slit10 must addressing	be even t must be
	The 'B' bit The 'd' bits	selects byte s select the a	value of the lite or word operat ddress of the d ddress of the s	ion (0 for wo	gister.	e).
	2:	than a word word move, In Byte mod Section 4.6	on . B in the ins d move. You m but it is not red le, the range of d <b>"Using 10-bit</b> an address offs	ay use a .w quired. Slit10 is not <b>Literal Ope</b> l	extension t	o denote specified
Words:	1					
Cycles:	1					
Example 1 MO	V.B [W84	+0x13], W10	) ; load W1 ; (Byte m	0 with [W8 ode)	+0x13]	
Wa W10 Data 101A SF	4009	V Data 1	After Instruction W8 1008 V10 4033 01A 3312 SR 0000			
Example 2 MO	V [W4+0	)x3E8], W2	; load W2 ; (Word mo	with [W4+0 de)	x3E8]	
W2 W4 Data 0BE8 SF	0800 5634	Data 0I	After Instruction W2 5634 W4 0800 3E8 5634 SR 0000			

MOV		Move Wns	to [Wd with	offset]		
Syntax:	{label:}	MOV{.B}	Wns,	[Wd+Slit10]		
Operands:		512 511] ir 024 1022	n Byte mode ?] (even only)	in Word mode		
Operation:	$(Wns) \rightarrow [V]$	Vd+Slit10]				
Status Affected:	None					
Encoding:	1001	1kkk	kBkk	kddd	dkkk	SSSS
Description:	of Slit10 is maintain w	increased to ord address	o [-1024 10 alignment. F	Wd+Slit10]. In 22] and Slit10 Register direct a ng must be use	must be eve addressing n	en to nust be
	The 'B' bit The 'd' bits	selects byte select the a	ddress of the	iteral. ration (0 for wo e destination re e destination re	gister.	e).
	2:	than a word word move, In Byte mod Section 4.6	d move. You , but it is not i de, the range <b>5 "Using 10-</b>	instruction den may use a . M required. of Slit10 is not bit Literal Ope ffset from Wd.	reduced as	to denote specified
Words:	1					
Cycles:	1					
Example 1 MO	V.B W0,	[W1+0x7]	; store ; (Byte	W0 to [W1+03 mode)	c7]	
W0 W1 Data 1806 SF	1800 2345	Data 1	After Instructi W0 9015 W1 1800 806 1545 SR 0000	5		
Example 2 MO	V W11, [W	/1-0x400]	; store ; (Word	W11 to [W1-( mode)	)x400]	
W1 W11 Data 0C00 SF	8813 FFEA	N Data 0	After Instructi W1 1000 W11 8813 C00 8813 SR 0000	) 3 3		

MOV		Move Ws t	to Wd			
Syntax:	{label:}	MOV{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws+Wb],	[Wd+Wb]		
Operands:	Ws ∈ [W0 Wb ∈ [W0 Wd ∈ [W0	) W15]				
Operation:	$(Ws) \rightarrow V$	/d				
Status Affected:	None					
Encoding:	0111	lwww	wBhh	hddd	dggg	SSSS
Description:			•	ister into the d essing may be		•
				destination re	9.0.0.	
	-		ource Addres	ss mode. source registe	er.	
	The 's' bit	s select the a The extensi than a wore	ddress of the ion . B in the i d move. You	source registenstruction den may use a .w	otes a byte m	
	The 's' bit Note 1	s select the a The extensi than a word word move When Regi source and	ddress of the ion . B in the i d move. You , but it is not r ster Offset A destination,	source registent nstruction den may use a . M equired. ddressing mon the offset mus	otes a byte m extension to de is used fo t be the sam	o denote or both th
	The 's' bit Note 1 2: 3:	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc	ddress of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W	source registent nstruction den may use a .w equired. ddressing mod	otes a byte m extension to de is used fo t be the sam and Wd. DMOV Ws, [W	o denote or both th e becaus N15++].
Words:	The 's' bit Note 1 2: 3:	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc	ddress of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W	source registent nstruction den may use a . We equired. ddressing mod the offset mus shared by Ws s" translates to	otes a byte m extension to de is used fo t be the sam and Wd. DMOV Ws, [W	o denote a or both the e because v15++].
Words: Cycles:	The 's' bit Note 1: 2: 3: 4:	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc	ddress of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W	source registent nstruction den may use a . We equired. ddressing mod the offset mus shared by Ws s" translates to	otes a byte m extension to de is used fo t be the sam and Wd. DMOV Ws, [W	o denote or both th e becaus N15++].
Cycles:	The 's' bit Note 1 2: 3: 4: 1	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc The instruc	ddress of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W tion "POP Wd	to W4 (Byte	otes a byte m r extension to de is used fo t be the sam and Wd. D MOV Ws, [W MOV [W1!	o denote or both th e becaus N15++].
Cycles:	The 's' bit Note 1: 2: 3: 4: 1 1 1 2: .B [W0 Before	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc The instruc	Address of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W tion "POP Wd Move [W0] Post-decree After	to W4 (Byte ement W0	otes a byte m r extension to de is used fo t be the sam and Wd. D MOV Ws, [W MOV [W1!	o denote or both th e becaus N15++].
Cycles: Example 1 MC	The 's' bit Note 1: 2: 3: 4: 1 1 1 0V.B [W0 Before Instruction	s select the a The extensi than a word word move, When Regi source and the 'w' enco The instruc The instruc	Address of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W tion "POP Wd Move [W0] Post-decree After Instructio	to W4 (Byte ement W0	otes a byte m r extension to de is used fo t be the sam and Wd. D MOV Ws, [W MOV [W1!	o denote or both th e becaus N15++].
Cycles:	The 's' bit Note 1: 2: 3: 4: 1 1 0V.B [W0 Before Instruction 0 0A01	s select the a The extension than a word word move, When Reginsource and the 'w' encord The instruct The instruct ], W4 ;	Address of the ion . B in the i d move. You , but it is not r ster Offset A destination, oding bits are tion "PUSH W tion "POP Wd Move [W0] Post-decree After	to W4 (Byte ement W0	otes a byte m r extension to de is used fo t be the sam and Wd. D MOV Ws, [W MOV [W1!	o denote or both th e becaus N15++].

SR

0000

# **Section 5. Instruction Descriptions**

SR

0000

Instruction Descriptions

; Move [W6] to [W2+W3] (Word mode) ; Post-increment W6 Example 2 MOV [W6++], [W2+W3] Before After Instruction Instruction W2 0800 W2 0800 WЗ 0040 W3 0040 W6 1228 W6 122A Data 0840 Data 0840 9870 0690 Data 1228 Data 1228 0690 0690 SR 0000 SR 0000

MOV.D		Double-W	/ord Move fro	om Source to	Wnd	
Syntax:	{label:}	MOV.D	Wns, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd		
Operands:	Ws∈[W0	/0, W2, W4 . ) W15] /0, W2, W4 .	-			
Operation:	Wns → Wns+1 For indire	addressing Wnd $\rightarrow$ Wnd+1 ct addressing escription				
Status Affected:	None					
Encoding:	1011	1110	0000	0ddd	0ppp	SSSS
	moved to specifies t double-wo by 4 bytes The 'd' bit The 'p' bit	Wnd:Wnd+1 the effective ord. Any pre/ s to accomm s select the s select the	If indirect ad address for th post-incremer odate for the o address of the source Addres	destination re	ed for the sou cant Word of ecrement wil gister.	urce, Ws the
	2:	for informa Wnd must	tion on how do be an even w	rates on doubl ouble-words an orking register. Wnd" translate	re aligned in	memory.
Words:	1					
Cycles:	2					
Example 1	MOV.D W2	,W6 ;	Move W2 to	W6 (Double	mode)	
	Before           Instruction           W2         12FB           W3         9877           W6         9833           W7         FCC6           SR         0000		After Instruct W2 12F W3 987 W6 12F W7 987 SR 000	ion B 7 B 7		

; Move [W7] to W4 (Double mode) ; Post-decrement W7 MOV.D [W7--], W4 Example 2 Before After Instruction Instruction W4 B012 W4 A319 W5 FD89 W5 9927 W7 W7 0900 08FC Data 0900 Data 0900 A319 A319 Data 0902 9927 Data 0902 9927 SR 0000 SR 0000

		Double-W	/ord Move fr	om Wns to D	estination	
Syntax:	{label:}	MOV.D	Wns,	Wnd		
				[Wd]		
				[Wd++]		
				[Wd]		
				[++Wd]		
				[Wd]		
Operands:	-	/0, W2, W4 /0, W2, W4 ) W15]	-			
Operation:	Wns → Wns+1	Wnd $\rightarrow$ Wnd+1	of destination			
		ct addressin escription	g of destinat	on:		
Status Affected:	None					<u> </u>
Encoding:	1011	1110	10qq	qddd	d000	sss0
	Word of the	ne double-we	fies the effec ord. Any pre/	ct addressing tive address fo post-incremen	or the Least S t or pre/post-	Significant decremer
	Word of th will adjust The 'q' bit The 'd' bit	ne double-we Wd by 4 by s select the s select the	fies the effec ord. Any pre/ tes to accom destination A address of th	tive address fo	or the Least S t or pre/post- e double-wor register.	Significant decremer
	Word of th will adjust The 'q' bit The 'd' bit The 's' bit <b>Note 1</b> : 2:	the double-work Wd by 4 by s select the s select the s select the This instri- Figure 4-2 memory. Wnd must The instru	fies the effec ord. Any pre/ tes to accom destination A address of th address of th ruction only for information be an even	tive address for post-incremen modate for the address mode. ne destination	or the Least S t or pre/post- e double-wor register. ster pair. on double-w ble-words ar er.	Significant decremer d. vords. Se e aligned
Worde	Word of th will adjust The 'q' bit The 'd' bit The 's' bit <b>Note 1</b> : 2: 3:	the double-work Wd by 4 by s select the s select the s select the Figure 4-2 memory. Wnd must	fies the effec ord. Any pre/ tes to accom destination A address of th address of th ruction only for information be an even	tive address for post-incremen modate for the address mode. he destination he source regis operates of on on how dou	or the Least S t or pre/post- e double-wor register. ster pair. on double-w ble-words ar er.	Significant decremer d. vords. Se e aligned
	Word of th will adjust The 'q' bit The 'd' bit The 's' bit <b>Note 1</b> : 2:	the double-work Wd by 4 by s select the s select the s select the This instri- Figure 4-2 memory. Wnd must The instru	fies the effec ord. Any pre/ tes to accom destination A address of th address of th ruction only for information be an even	tive address for post-incremen modate for the address mode. he destination he source regis operates of on on how dou	or the Least S t or pre/post- e double-wor register. ster pair. on double-w ble-words ar er.	Significant decremer d. vords. Se e aligned
Words: Cycles: Example 1 MC	Word of th will adjust The 'q' bit The 'd' bit The 's' bit <b>Note 1</b> <b>2</b> <b>3</b>	we double-we Wd by 4 by s select the s select the Figure 4-2 memory. Wnd must The instru [W15++].	fies the effec ord. Any pre/ tes to accom destination <i>A</i> address of th address of th ruction only for information be an even we action PUSH.	tive address for post-incremen modate for the address mode. he destination he source regis operates of on on how dou	or the Least S t or pre/post- e double-wor register. ster pair. on double-w ble-words ar er. lates to MON	Significant decremer d. vords. Se e aligned

Example 2 MOV	.D W4,		decrement W4 to [W	W6 (Double mode) 6]
	Before		After	
	Instructior	ו	Instruction	
W4	100A	W4	100A	
W5	CF12	W5	CF12	
W6	0804	W6	0800	
Data 0800	A319	Data 0800	100A	
Data 0802	9927	Data 0802	CF12	
SR	0000	SR	0000	

Syntax:	{label:}	MOVSAC	Acc	{,[Wx], Wxd}	{,	[Wy], Wyd}	{,AWB}
				{,[Wx]+=kx, V	Vxd}   {,	[Wy]+=ky, Wyd	}
				{,[Wx]-=kx, W	/xd} {,	[Wy]-=ky, Wyd}	
				{,[W9+W12],	Wxd} {,	[W11+W12], W	yd}
Operands:			W11]; ky	y ∈ [-6, -4, -2, 2		d ∈ [W4 W7] Vyd ∈ [W4 W	
Operation:		$([Wx]) \rightarrow Wx$ $([Wy]) \rightarrow Wy$ (Acc(B or A)	/d; (Wy)+	ky→Wy			
Status Affected	l:	None					
Encoding:		1100	0111	A0xx	yyi	i iijj	jjaa
Description:		instruction a Even thoug	and option h an accu an accun	nally store the u umulator opera nulator must be	inspecif tion is no	n for another MA ied accumulato ot performed in ed to designate	r results. this
		which supp Section 4.1 optional sto	ort indired 4.1 "MA re of the	ct and register	offset ad ". Opera llator, as	ptional pre-fetch Idressing, as de and AWB specif described in	scribed in
		The 'x' bits The 'y' bits The 'i' bits s The 'j' bits s	select the select the elect the elect the	e other accumu e pre-fetch Wxc e pre-fetch Wyc Wx pre-fetch c Wy pre-fetch c e accumulator v	destina destina operation operation	ition. 1. 1.	k.
Words:		1					
Cycles:		1					
Example 1	; Fe	tch [W9] t	o W6 to W7,	[W11]+=4, W Post-increm		by 4	
		Before	_			After	
	W6	Instructio		W6	inst	ruction	
	W7		022 200	W7		7811 B2AF	
	W9		800	W9		0800	
	W11		900	W11		1904	
	W13	0	020	W13		3290	
	CCA	00 3290 5	968	ACCA	00 32	90 5968	
Data	-		811	Data 0800		7811	
Data	-		2AF	Data 1900		B2AF	
	SR	0	000	SR		0000	

```
Example 2
```

2 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

	I	Befor nstruct	-		I	Aftei nstruct	
W4			76AE	W4			BB00
W6			2000	W6			52CE
W9			1200	W9			11FE
W11			2000	W11			2000
W12			0024	W12			0024
W13			2300	W13			2302
ACCB	00	9834	4500	ACCB	00	9834	4500
Data 1200			BB00	Data 1200			BB00
Data 2024			52CE	Data 2024			52CE
Data 2300			23FF	Data 2300			9834
SR			0000	SR			0000

MPY	М	ultiply Wm	by Wn to Ac	cumulator		
Syntax: {label:}	MPY W	/m*Wn, Acc	{,[Wx], Wxd} {,[Wx]+=kx, V {,[Wx]-=kx, V {,[W9+W12],	Wxd} {,[Wy Vxd} {,[Wy		
Operands:	$Acc \in [A, E]$ $Wx \in [W8, Wy \in [W10]$	3] , W9]; kx ∈ [-	-6, -4, -2, 2, 4 = [-6, -4, -2, 2	, 6]; Wxd ∈	W5*W7, W6*W [W4 W7] ∈ [W4 W7]	7]
Operation:	$([Wx]) \rightarrow W$	) → Acc(A o /xd; (Wx)+kx /yd; (Wy)+ky	x→Wx			
Status Affected:	OA, OB, O	AB, SA, SB	, SAB			
Encoding:	1100	Ommm	A0xx	yyii	iijj	jj11
Description:	operands i store the u	n preparatio inspecified a sign-extende	n for another	MAC type in esults. The 3	ptionally pre-fet struction and op 2-bit result of th o the specified	otionally
	which sup	port indirect		offset addres	al pre-fetch ope sing, as descril	
	The 'A' bit The 'x' bits The 'y' bits The 'i' bits	selects the a select the p select the p select the W	operand regis accumulator fore-fetch Wxd ore-fetch Wyd /x pre-fetch o /y pre-fetch o	or the result destination destination peration.		ultiply:
	Note:	The IF bit, ( fractional o		, determine	s if the multiply	is
Words:	1					
Cycles:	1					

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 Instructio	on
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)

	I	Befor nstruct			I	After nstruct	
W4			C000	W4			8FDC
W5			9000	W5			0078
W6			671F	W6	1		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	1		0078
CORCON			0000	CORCON			0000
SR			0000	SR			0000

MPY			Square to Ac	cumulator				
Syntax:	{label:}	MPY	Wm*Wm, Acc	{,[Wx]+=kx, {,[Wx]-=kx, \	Wxd} Wxd}	{,[Wy]-	-=ky, Wyd}	
Operands:		Acc ∈   Wx ∈ [	m ∈ [W4*W4, W [A,B] W8, W9]; kx ∈ [- W10, W11]; ky ∈	6, -4, -2, 2, 4,	, 6]; Wx	- d∈[W4	-	
Operation:		([Wx])-	$(Wm) \rightarrow Acc(Ac)$ $\rightarrow Wxd; (Wx)+kx$ $\rightarrow Wyd; (Wy)+ky$	→Ŵx				
Status Affected	d:	OA, OI	B, OAB, SA, SB,	SAB				
Encoding:		111	1 00mm	A0xx	уу	ii	iijj	jj01
Description:		in prep unspec sign-e>	the contents of aration for anoth fied accumulato tended to 40-bit nds Wx, Wxd, W	er MAC type i or results. The s and stored	nstruction 32-bit r in the sp	on and result of pecified	optionally sto f the signed n d accumulato	ore the nultiply is r.
		which s	support indirect a	and register o	ffset ad			
		The 'A' The 'x' The 'y' The 'i'	' bits select the of bit selects the a bits select the p bits select the p bits select the W bits select the W	ccumulator for re-fetch Wxd re-fetch Wyd x pre-fetch op	or the re destina destina peration	sult. tion. tion.	square.	
		Note	: The IF bit, C fractional or	CORCON<0> an integer.	, determ	nines if	the multiply i	S
Words:		1						
Cycles:		1						
Example 1	; So ; Fe	quare W etch [W	5, A, [W9]+=2 N6 and store N9] to W6, Po = 0x0000 (fra	to ACCA st-incremen		-	saturation)	
		B	efore		1	After		
	I	Inst	ruction		Inst	truction		
	W6		6500	W6 W9			365	
				vvy	1	09		
/	W9	00 70	0900		00 /1		902	
	ACCA	00 70	80 0908	ACCA	00 41	FB2 00	000	
Data		00 7C			00 41	7B2 00 B8		

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 = 0x0000 (fractional multiply, no saturation)

	I	Befor nstruct	-		I	Aftei nstruct	
W4			E228	W4			8911
W5			9000	W5			F678
W9			1700	W9			1700
W10			1B00	W10			1B02
W12			FF00	W12			FF00
ACCB	00	9834	4500	ACCB	00	06F5	4C80
Data 1600			8911	Data 1600			8911
Data 1B00			F678	Data 1B00			F678
CORCON			0000	CORCON			0000
SR			0000	SR			0000

MPY.N	Multiply -	Wm by Wn to Ac	cumulator	
Syntax: {lab	el:} MPY.N Wm*Wn, /	Acc {,[Wx], Wxd}	{,[Wy], Wyd}	
		{,[Wx]+=kx, W	/xd} {,[Wy]+=ky, Wyd	}
		{,[Wx]-=kx, W	xd} {,[Wy]-=ky, Wyd	}
		{,[W9+W12], \	Wxd} {,[W11+W12], W	/yd}
Operands:	Acc ∈ [A,B] Wx ∈ [W8, W9]; kx	. ∈ [-6, -4, -2, 2, 4,	; W5*W6; W5*W7; W6 6]; Wxd ∈ [W4 W7] 4, 6]; Wyd ∈ [W4 W	-
Operation:	-(Wm)*(Wn) $\rightarrow$ Acc ([Wx]) $\rightarrow$ Wxd; (Wx) ([Wy]) $\rightarrow$ Wyd; (Wy)	)+kx→Ŵx		
Status Affected:	OA, OB, OAB			
Encoding:	1100 Omm	m Alxx	yyii iijj	jj11
	accumulator results sign-extended to 40	s. The 32-bit result 0-bits and stored to	optionally store the uns of the signed multiply o the specified accum ers Wm and Wn for the	is ılator.
	The 'A' bit selects the The 'x' bits select the The 'y' bits select the The 'i' bits select the The 'j' bits select the	he pre-fetch Wxd c he pre-fetch Wyd c ne Wx pre-fetch op	lestination. lestination. eration.	
	Note: The IF b or an in		determines if the multip	oly is fractiona
Words:	1			
Cycles:	1			
Example 1	; Multiply W4*W5, n	W8]+=2, W4, [W] negate the resu	ult and store to A	
	; Fetch [W10] to W1 ; Fetch [W10] to W1 ; CORCON = 0x0001		ent W10 by 2	
	; Fetch [W10] to W	5, Post-increme	ent W10 by 2	
	; Fetch [W10] to W ; CORCON = 0x0001 Before	5, Post-increme	ent W10 <sup>by 2</sup> ply, no saturation <b>After</b>	
	; Fetch [W10] to W3 ; CORCON = 0x0001 Before Instruction W4 3023 W5 1290	5, Post-increme (integer multip W4 W5	ent W10 by 2 ply, no saturation After Instruction	
	; Fetch [W10] to W3 ; CORCON = 0x0001 Before Instruction N4 3023 N5 1290 N8 0B00	5, Post-increme (integer multip W4 W5 W8	After Instruction 0054 0B02	
	Fetch [W10] to W3 CORCON = 0x0001 Before Instruction W4 3023 W5 1290 W8 0B00 10 2000	5, Post-increme (integer multip W4 W5 W8 W10	After Instruction 0054 660A 0B02 2002	
N N W AC	Fetch [W10] to W3 CORCON = 0x0001 Before Instruction N4 3023 N5 1290 N8 0B00 10 2000 CA 00 0000 2387	5, Post-increme (integer multip W4 W5 W8 W10 ACCA	After Instruction 0054 660A 0B02 2002 FF FC82 7650	
	Fetch [W10] to W3 CORCON = 0x0001 Before Instruction N4 3023 N5 1290 N8 0B00 10 2000 CA 00 0000 2387 00 0054	5, Post-increme (integer multip W4 W5 W8 W10	After Instruction 0054 660A 0B02 2002	
N N W AC4 Data 0E	Fetch [W10] to W3 CORCON = 0x0001 Before Instruction W4 3023 W5 1290 W8 0B00 10 2000 CA 00 0000 2387 00 0054 00 660A	5, Post-increme (integer multip W4 W5 W8 W10 ACCA Data 0B00	After Instruction 0054 660A 0B02 2002 FF FC82 7650 0054	

5

```
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		After Instruct	
W4	3023	W4		0054
W5	1290	W5		660A
W8	0B00	W8		0B02
W10	2000	W10		2002
ACCA	00 0000 2387	ACCA	FF F904	ECA0
Data 0B00	0054	Data 0B00		0054
Data 2000	660A	Data 2000		660A
CORCON	0000	CORCON		0000
SR	0000	SR		0000

Syntax: {label:}	MSC W	/m*Wn, Acc	{,[Wx], Wxd}	{,	[Wy], V	Vyd}	{,AWB}
			{,[Wx]+=kx, W	xd} {,	[Wy]+=	⊧ky, Wyd}	
			{,[Wx]-=kx, Wx	۲d} {,	[Wy]-=	ky, Wyd}	
			{,[W9+W12], V	Vxd} {,	[W11+'	W12], Wyd}	
Operands:	$\begin{array}{l} Acc \in [A, \\ Wx \in [Wx] \\ Wy \in [Wx] \end{array}$	B] 8, W9]; kx ∈	V4*W6, W4*W7 [-6, -4, -2, 2, 4, ∈ [-6, -4, -2, 2, =2]	6]; Wxo	d ∈ [W	4 W7]	[7]
Operation:	(Acc(A or ([Wx])→ ([Wy])→		$(Wn) \rightarrow Acc(A)$ $(x \rightarrow Wx)$ $(x \rightarrow Wy)$	or B)			
Status Affected:	OA, OB,	OAB, SA, SE	B, SAB				
Encoding:	1100	0 mmm	Alxx	yyi	i	iijj	jjaa
Description:	operands store the	in preparation unspecified s sign-extend	of two working r on for another M accumulator res ded to 40-bits ar	IAC type sults. Th	e instru he 32-b	uction and o bit result of t	ptionally he signe
	which sup Section optional s	oport indirect 4.14.1 "MAC store of the "	Wy and Wyd spo t and register of <b>C Pre-Fetches</b> ". other" accumula <b>C Write Back</b> ".	fset ado . Opera	dressin Ind AW	ig as descril 'B specifies	oed in
	The 'A' bi The 'x' bi The 'y' bi The 'i' bit The 'j' bit	t selects the ts select the ts select the s select the s select the v	e operand regist accumulator for pre-fetch Wxd c pre-fetch Wyd c Wx pre-fetch op Wy pre-fetch op accumulator wr	r the readestinat destinat destinat eration eration	sult. tion. tion.		ultiply.
	Note:		CORCON<0>,	determ	nines if	the multiply	is
Marda	4	tractional of	or an integer.				
Words:	1						
Cycles:	1						

```
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)

		Befor struct			I	After nstruct	
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			

	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		

After

Syntax:	{label:}	MUL{.B}	f			
Operands:	f ∈ [0 8	191]				
Operation:	For byte o		0 M0			
	For word	)<7:0> * (f)<7: operation:	$0> \rightarrow W2$			
		$(f) \rightarrow W2:V$	<b>V</b> 3			
Status Affected:	None					
Encoding:	1011	1100	OBOf	ffff	ffff	ffff
Description:	register and and the re executed the Most S Least Sign	e default work nd place the re sult are interp in Byte mode, Significant Wo nificant Word o selects byte o	esult in the W reted as uns the 16-bit re rd of the 32- of the 32-bit i	/2:W3 registe igned intege sult is stored bit result is s result is store	er pair. Both rs. If this inst in W2. In We tored in W3, ed in W2.	operands ruction is ord mode and the
		select the ad			, <b>,</b>	
Words:	3:	The WREG i The IF bit, C This is the or multiply.	ORCON<0>	, has no effe	ct on this ope	
Cycles:	1					
Example 1 MUL	.B 0x800	; Multip]	Ly (0x800)	*WREG (Byt	e mode)	
	Before		After			
WREG (W0)	9823	WREG (W	Instruction 0) 9823	1		
W120 (W0) W2	FFFF		<b>/2</b> 13B0			
	FFFF	W				
W3						
Data 0800	2690	Data 080	<b>)0</b> 2690			
	2690 0000		00 2690 R 0000			
Data 0800	0000	S	R 0000	WREG (Word	l mode)	
Data 0800 SR Example 2 <sup>MUL</sup>	0000 TMR1 Before	S	R 0000 Ly (TMR1)* After		ł mode)	
Data 0800 SR Example 2 <sup>MUL</sup>	0000 TMR1 Before Instruction	S; Multip	R 0000 Ly (TMR1)* After Instruction		l mode)	
Data 0800 SR Example 2 <sup>MUL</sup> WREG (W0)	0000 TMR1 Before Instruction F001	S; Multip: WREG (W	R 0000 Ly (TMR1)* After Instruction 0) F001		ł mode)	
Data 0800 SR Example 2 <sup>MUL</sup>	0000 TMR1 Before Instruction	S ; Multip: WREG (W W	R 0000 Ly (TMR1)* After Instruction		ł mode)	
Data 0800 SR Example 2 <sup>MUL</sup> WREG (W0) W2	0000 TMR1 Before Instruction F001 0000	S ; Multip: WREG (W W	R         0000           Ly         (TMR1)*           After         Instruction           0)         F001           /2         C287           /3         2F5E		ł mode)	

MUL.SS	Integer 16x16-bit Signed Multiply					
Syntax:	{label:}	MUL.SS	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W0		. W12]			
Operation:	-		Ws) $\rightarrow$ Wnd:	Vnd+1		
Status Affected:	None	, <b>c</b> (	,			
Encoding:	1011	1001	lwww	wddd	dppp	SSSS
	compleme for Wb and used for W The 'w' bits The 'd' bits The 'p' bits	nt signed int I Wnd. Regis I's. I's select the a Is select the a Is select the s	and the result egers. Regist ster direct or r address of the address of the source Address address of the	er direct add egister indire base regist lower destir s mode.	ressing must ect addressin er. nation registe	be used g may be
	2: 3: 4:	Since the pl an even we how double Wnd may n	tion operates roduct of the n orking registen words are al ot be W14, si CORCON<0>	nultiplication r. See Figure igned in mer nce W15<0>	is 32-bits, Wi e 4-2 for info nory. • is fixed to ze	rmation o ero.
Words:	1					
Cycles: Example 1	1 MUL.SS WO, W		; Multiply ; Store the		o W12:W13	
V	Before Instruction W0 9823 W1 67DC V12 FFFF V13 FFFF SR 0000	v W W	After Instruction V0 9823 V1 67DC 12 D314 13 D5DC SR 0000			

Example 2	MUL.SS	W2,	[W4],	WO
-----------	--------	-----	-------	----

; Pre-decrement W4 ; Multiply W2\*[W4] ; Store the result to W0:W1

I	Before Instructior	ı	After 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

5

MUL.SU	J	Integer 16x16-bit Signed-Unsigned Short Literal Multi					
Syntax:	{label:}	MUL.SU	Wb,	#lit5,	Wnd		
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wnd ∈ [W	-	W12]				
Operation:	signed (W	/b) * unsigned	lit5 $\rightarrow$ Wnd:	Wnd+1			
Status Affected	None						
Encoding:	1011	1001	0www	wddd	d11k	kkkk	
Description:	result in tw the result register), a The Wb o compleme integer. Re The 'w' bit	e contents of vo successive is stored in W and the Most S perand and th ent signed inte egister direct a s select the a s select the ac	working regind (which m Significant W result Who ger. The liter addressing n ddress of the	sters. The Le ust be an evo ord of the res a are interpre ral is interpre nust be used a base regist	east Significa en numbered sult is stored ted as a two' ted as an un for Wb and V er.	nt Word of I working in Wnd+1. s signed Wnd.	
	Note 1: 2: 3:		ion operates oduct of the r rking register words are al ot be W14, si	in Word mo nultiplication r. See Figure igned in mer nce W15<0>	de only. is 32-bits, Wi e 4-2 for info nory. • is fixed to ze	rmation on ero.	
Words:	1						
Cycles:	1						
Example 1	MUL.SU W0, ‡	‡0x1F, W2	; Multiply ; Store th	WO by lit e result t			
	Before		After				

I	Before nstructior	ı	After Instruction
W0	C000	W0	C000
W2	1234	W2	4000
W3	C9BA	W3	FFF8
SR	0000	SR	0000

Example 2	MUL.SU	W2,	#0x10,			y W2 by literal 0x10 he result to W0:W1
		ore iction			After Instruction	n
	W0 A	BCD		WC	2400	
	W1 8	9B3		W1	000F	
	W2 F	240		W2	F240	
	<b>SR</b> 0	000		SF	0000	]

MUL.SU		Integer 16	x16-bit Sign	ed-Unsigne	d Multiply	
Syntax:	{label:}	MUL.SU	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W		W12]			
Operation:	signed (W	/b) * unsigne	d (Ws) $ ightarrow$ Wr	nd:Wnd+1		
Status Affected:	None					
Encoding:	1011	1001	0www	wddd	dppp	SSSS
	The Wb c complement unsigned Wnd. Reg The 'w' bi The 'w' bi The 'bi The 'p' bi The 's' bit <b>Note 1</b> 2:	perand and the signed in integer. Reg gister direct of the select	t Significant W the result Wn teger. The Ws ister direct ad r register indi address of the address of the source Addre address of the ction operates roduct of the orking registe e-words are a not be W14, s	d are interpre s operand is idressing mu rect address e base regis e lower desti ss mode. e source regi s in Word mo multiplication er. See Figur ligned in me	eted as a two interpreted a st be used fo ing may be u ter. nation registe ster. ode only. n is 32-bits, W re 4-2 for info mory.	's s an r Wb and sed for Ws er. Ind must be prmation or
		: The IF bit,	CORCON<0>	>, has no effe	ect on this op	eration.
Words:	1					
Cycles: Example 1 MUI	1 L.SU W8, Before Instruction		; Multiply ; Store the After Instruction	e result t	o W0:W1	
WO		١	<b>NO</b> 0000			
W1			<b>N1</b> F100			
W8			N8 F000			
W9 Data 178C		۱ Data 17	N9         178C           8C         F000			
SR			SR 0000			
			L			

Example 2 MUI	.SU W2,	/	Pre-Increment W3 Multiply W2*[W3] Store the result to W4:W5				
	Before	After					
	Instruction	I	Instruction				
W2	0040	W2	0040				
W3	0280	W3	0282				
W4	1819	W4	1A00				
W5	2021	W5	0000				
Data 0282	0068	Data 0282	0068				
SR	0000	SR	0000				

MUL.US	Integer 16x16-bit Unsigned-Signed Multiply								
Syntax:	{label:}	MUL.US	Wb,	Ws,	Wnd				
				[Ws],					
				[Ws++],					
				[Ws],					
				[++Ws],					
				[Ws],					
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W	-	. W12]						
Operation:	-	(Wb) * signe	-	nd:Wnd+1					
Status Affected:	None	( ) 0	( )						
Encoding:	1011	1000	1www	wddd	dppp	SSSS			
	register), The Wb c and the re integer. R Register c The 'w' bi The 'a' bi The 'p' bi The 's' bit <b>Note 1</b> : 2:	and the Most perand is inte esult Wnd are legister direct direct or regis ts select the a ts select the a this instruct Since the p an even we how double the Wnd may m	Significant V erpreted as a a interpreted addressing ter indirect a address of th address of th address of th source Addre address of th enderss of th enderss of th enderss of th address of th addre	Vord of the re in unsigned i as a two's co must be use iddressing m he base regis e lower desti ass mode. e source reg s in Word mo multiplication er. See Figu aligned in me since W15<0	ination registe ister. ode only. n is 32-bits, W re 4-2 for info	in Wnd+1 /s operand gned Wnd. or Ws. er. /nd must b rmation of			
Words:	1								
Cycles:	1								
Example 1 MU	L.US WO,			W0*[W1] ( e result t	unsigned-s o W2:W3	igned)			
	Before Instruction	1	After Instructio	n ]					

Example 2	MUL.US	W6,	[W5++],	W10	;	; Mult. W6*[W5] (unsigned-signed) ; Store the result to W10:W11 ; Post-Increment W5
	Bef					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	SR	0000

MUL.U	U	Integer 16	x16-bit Ur	signed Sho	rt Literal Mult	iply				
Syntax:	{label:}	MUL.UU	Wb,	#lit5,	Wnd					
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wnd ∈ [V		W12]							
Operation:	unsigned	(Wb) * unsig	ned lit5 $\rightarrow$	Wnd:Wnd+1						
Status Affected	: None									
Encoding:	1011	1000	0www	wddd	d11k	kkkk				
Description:	result in to the result register), Both ope	wo successiv is stored in N and the Most rands and the	e working i Nnd (which Significan e result are	registers. The n must be an t Word of the ninterpreted a	al, and store the e Least Signific even numbere result is stored as unsigned int Vb and Wnd.	ant Word o d working I in Wnd+1				
	The 'd' bi	The 'w' bits select the address of the base register. The 'd' bits select the address of the lower destination register. The 'k' bits define a 5-bit unsigned integer literal.								
	2:	an even w how double Wnd may r	oroduct of the orking regi e-words are not be W14	ne multiplicat ster. See Fig e aligned in r , since W154	ion is 32-bits, V gure 4-2 for inf	ormation o zero.				
Words:	1									
Cycles:	1									
Example 1	MUL.UU WO,	#0xF, W12			literal 0xF t to W12:W13	3				
	Before           Instruction           W0         2323           W12         4512           W13         7821           SR         0000	W W	Afte Instruct W0 232 (12 0F0 (13 000 SR 000	tion 3 D 2						
Example 2	MUL.UU W7,	#0x1F, W0			literal 0x1B t to W0:W1	7				
	Before           Instruction           W0         780B           W1         3805           W7         F240           SR         0000	N N	Afte Instruct W0 550 W1 001 W7 F24 SR 000	tion :0 D 0						

MUL.UU	J	Integer 16x16-bit Unsigned Multiply						
Syntax:	{label:}	MUL.UU	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd			
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wnd ∈ [W0		. W12]					
Operation:	-		red (Ws) $\rightarrow$ V	Vnd:Wnd+1				
Status Affected:	None							
Encoding:	1011	1000	0www	wddd	dppp	SSSS		
	Both sourc integers. F Register di The 'w' bits The 'd' bits The 'p' bits	e operands a legister direct irect or indirects s select the a s select the a s select the s	Significant W and the resul addressing address of the ddress of the ource Address ddress of the	t are interpre must be use g may be use base regist lower destir s mode.	ted as unsig d for Wb and ed for Ws. er. nation registe	ned I Wnd.		
Words:	2: 3: 4: 1	Since the pr an even wo how double Wnd may no	tion operates oduct of the r rking registe -words are al ot be W14, si CORCON<0>	nultiplication r. See Figure igned in mer nce W15<0>	is 32-bits, Wi e 4-2 for info nory. • is fixed to ze	rmation o ero.		
Cycles: Example 1	1 MUL.UU W4, W		Multiply W4 Store the r			ned)		
	Before Instruction W0 FFFF W2 2300 W3 00DA W4 FFFF SR 0000	v v v	After Instruction V0 FFFF V2 0001 V3 FFFE V4 FFFF SR 0000	1				

mple 2 MUI	.UU W0,	;		he resu	(unsigned-unsigned llt to W4:W5 : W1	d)
	Before		After			
	Instruction	ו	Instruction	n		
W0	1024	WO	1024			
W1	2300	W1	2302			
W4	9654	W4	6D34			
W5	BDBC	W5	0D80			
Data 2300	D625	Data 2300	D625			
SR	0000	SR	0000			
		-		-		

Example 2

NEG		Negate f				
Syntax:	{label:}	NEG{.B}	f	{,WREG}		
Operands:	f ∈ [0 8	191]				
Operation:	-	destination de	signated by	D		
Status Affected:	DC, N, O\		0 ,			
Encoding:	1110	1110	OBDf	ffff	ffff	ffff
Description:	place the determine	the 2's compl result in the d s the destinat VREG. If WR	estination regition register.	gister. The op If WREG is s	otional WREC	G operand result is
	The 'D' bit	selects byte selects the c select the ac	lestination (0	for WREG,		
	2:		a word opera ord operation	tion. You ma , but it is not	y use a .w ex required.	
Words:	1					
Cycles:	1					
Example 1 NEG.	B 0x880,		egate (0x8 tore resul	-	mode)	
lr WREG (W0) Data 0880 SR	Before nstruction 9080 2355 0000	WREG (W0 Data 0880 SF	2355	l=1)		
Example 2 NEG	0x1200	; N	egate (0x1	200) (Word	d mode)	
lı Data 1200 SR	Before Instruction 8923 0000	Data 1200 SF				

NEG		Negate W	S			
Syntax:	{label:}	NEG{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:		0 W15] 0 W15]				
Operation:	(Ws) + 1	ightarrow Wd				
Status Affected:	DC, N, O	V, Z, C				
Encoding:	1110	1010	0Bqq	qddd	dppp	SSSS
Description:	and place	e the result in	the destinati	e contents of on register W for both Ws	d. Either regi	-
	The 'q' bi The 'd' bi The 'p' bi	ts select the o ts select the a ts select the s	destination A address of th source Addre	ration (0 for v ddress mode e destination ess mode. e source regis	register.	rte).
		rather than a	word operati	nstruction der on. You may but it is not re	use a .wext	-
Words:	1		a operation,		quireu.	
Cycles:	1					
Example 1 NEG.B	W3, [W4	-	ate W3 and t-incremen	store to t W4	[W4] (Byte	mode)
	Before		After			
-	struction					
W3 W4	7839 1005	V V	<b>/3</b> 7839 <b>/4</b> 1006			
Data 1004	2355	Data 100				
SR	0000		R 0008 (	N=1)		
Example 2 NEG [	W2++], [	; N		nt W4 (Word and store ent W2		
lr W2 W4	Before estruction 0900 1002	N N				
Data 0900 Data 1000 SR	870F 5105 0000	Data 090 Data 100 S				

NEG		Negate Acc	unuator			
Syntax:	{label:}	NEG	Acc			
Operands:	$Acc \in [A,B]$					
Operation:	<u>If (Acc = A):</u> -ACCA →					
	<u>Else:</u> -ACCB →	ACCB				
Status Affected:	OA, OB, OA	AB, SA, SB,	SAB			
Encoding:	1100	1011	A001	0000	0000	0000
Description:	accumulator	Compute the 2's complement of the contents of the specified accumulator. Regardless of the Saturation mode, this instruction operates on all 40-bits of the accumulator.				
	The 'A' bit s	pecifies the	selected acc	cumulator.		
Words:	1					
-	1					
Cycles: Example 1 NEC	GA;N		lt to ACCA	A saturation	1)	
-	GA;N	tore resul	lt to ACCA		1)	
-	G A ; N ; S ; C	tore resul ORCON = 0:	lt to ACCA	saturation		
Example 1 NEC	GA;N;S;C	tore resultore resultore resultore n	lt to ACCA x0000 (no ACCA	saturation	n	
Example 1 NEC ACCA CORCON	G A ; N ; S ; C Before Instructior 00 3290 5 0	tore resul ORCON = 0: 9C8 000	lt to ACCA x0000 (no ACCA CORCON	Saturation After Instructio FF CD6F A	n 1638 1000	
Example 1 NEC	G A ; N ; S ; C Before Instructior 00 3290 5 0	tore resul ORCON = 0: n 9C8	lt to ACCA x0000 (no ACCA	Saturation After Instructio FF CD6F A	n .638	
ACCA CORCON	G A ; N ; S ; C Before Instructior 00 3290 5 0 0	tore resul ORCON = 0: 9C8 000	ACCA ACCA CORCON SR	Saturation After Instructio FF CD6F A	n 1638 1000	
Example 1 NEC ACCA CORCON SR	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 3 B ; N ; S	n 9C8 000 000 egate ACC tore resu	ACCA ACCA CORCON SR It to ACCE	Saturation	n .638 .000 .000	
Example 1 NEC ACCA CORCON SR	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 3 B ; N ; S	n 9C8 000 000 egate ACC tore resu	ACCA ACCA CORCON SR It to ACCE	Saturation After Instructio FF CD6F A C C	n .638 .000 .000	
Example 1 NEC ACCA CORCON SR	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 3 B ; N ; S	n 9C8 000 000 egate ACC tore resu	ACCA ACCA CORCON SR It to ACCE	Saturation	n .638 .000 .000	
Example 1 NEC ACCA CORCON SR Example 2 NEC	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 0 3 B ; N ; S ; C	TORCON = 0: ORCON = 0: 000 000 000 Cegate ACCI tore resul ORCON = 0:	ACCA ACCA CORCON SR 1t to ACCE	After Instructio FF CD6F A C C C C C C C C C C C C C C C C C C C	n .638 .000 .000 	
Example 1 NEC ACCA CORCON SR Example 2 NEC	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 0 0 3 B ; N ; S ; C Before Instruction FF F230 1	n 9C8 000 000 egate ACC tore resu 0RCON = 0: n 0DC	ACCA ACCA CORCON SR It to ACCE x00C0 (nor ACCB	Saturation	n 638 0000 0000 ttion) n F24	
Example 1 NEC ACCA CORCON SR Example 2 NEC	G A ; N ; S ; C Before Instruction 00 3290 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	n PC8 000 000 regate ACCI tore resul ORCON = 0: n	ACCA ACCA CORCON SR 1t to ACCE	Saturation	n .638 .000 .000 	

NOP			No Operatio	on			
Syntax:		{label:}	NOP				
Operands:		None					
Operation:		No Operat	tion				
Status Affecte	ed:	None					
Encoding:		0000	0000	xxxx	xxxx	xxxx	xxxx
Description:		No Operat	tion is perform	ied.	•		
		The 'x' bits	s can take any	value.			
Words:		1					
Cycles:		1					
Example 1	PC SR	Before Instruction	ecute no op	In	After struction 00 1094 0000		
Example 2	NOF	); ex	cecute no op	peration			
	РС	Before Instruction	1		After struction		

NOPR		No Operatio	on			
Syntax:	{label:}	NOPR				
Operands:	None					
Operation:	No Opera	tion				
Status Affected:	None					
Encoding:	1111	1111	xxxx	xxxx	xxxx	xxxx
Description:	No Opera	tion is perform	ied.	11		
	The 'x' bit	s can take any	/ value.			
Words:	1	-				
Cycles:	1					
Example 1 NOP	Before			After		
	nstruction	7		struction		
PC	00 2430	]	PC	00 2432		
PC SR		]				
-	00 2430	ecute no op	PC SR	00 2432		
SR	00 2430 0000 R ; ex Before	ecute no op	PC SR	00 2432 0000		
Example 2 NOP	00 2430 0000 R ; ex Before Instruction	cecute no or	PC SR	After struction		
SR	00 2430 0000 R ; ex Before	] xecute no op	PC SR	00 2432 0000		

POP		Pop TOS to	f			
Syntax:	{label:}	POP	f			
Operands:	f ∈ [0 65	534]				
Operation:	$\begin{array}{c} (W15)-2 \rightarrow \\ (TOS) \rightarrow f \end{array}$	→ W15				
Status Affected:	None					
Encoding:	1111	1001	ffff	ffff	ffff	fff0
Description:	The stack pointer (W15) is pre-decremented by 2 and the Top-of-Sta (TOS) word is written to the specified file register, which may reside anywhere in the lower 32K words of data memory.					
	The 'f' bits	select the ad	dress of the	file register.		
		This instruct				
Words:	<b>2</b> :	The file regis	ster address	must be wor	d aligned.	
Cycles:	1					
Example 1 POP	0x1230	; Рор ТО	S to 0x123	0		
In	Before struction		After Instruction			
W15_ Data 1004	1006 A401	W15 Data 1004				
Data 1004	2355	Data 100-				
SR	0000	SF				
Example 2 POP	0x880	; Рор ТО	S to 0x880			
	Before struction 2000 E3E1 A090 0000	W15 Data 0880 Data 1FFE SF	A090 A090			

POP	Pop TOS to Wd					
Syntax:	{label:}	POP	Wd			
			[Wd]			
			[Wd++]			
			[Wd]			
			[Wd]			
			[++Wd]			
			[Wd+Wb]			
Operands:	Wd ∈ [W0 Wb ∈ [W0					
Operation:	(W15)-2 – (TOS) $\rightarrow$					
Status Affected:	None					
Encoding:	0111	lwww	w0hh	hddd	d100	1111
Description:	(TOS) wo		) is pre-decrei Wd. Either re			
	The 'h' bit	s select the de	offset register	lress mode.		
			ddress of the		-	
	Note 1:	This instruct This instruct	ddress of the ion operates i tion is a spec MOV [W15	n Word mo	de only. of the "MOV	
Words:	Note 1:	This instruct This instruct instruction (	ion operates i tion is a spec	n Word mo	de only. of the "MOV	
	Note 1: 2:	This instruct This instruct instruction (	ion operates i tion is a spec	n Word mo	de only. of the "MOV	
	Note 1: 2:	This instruct This instruct instruction (	ion operates i tion is a spec MOV [W15	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP	Note 1: 2: 1 1 W4 Before	This instruct This instruct instruction ( MOV.	tion operates i tion is a spec MOV [W15 S to W4 After	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP	Note 1: 2: 1 1 W4 Before nstruction	This instruct This instruct instruction ( MOV.	tion operates i tion is a spec MOV [W15 S to W4 After Instruction	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP I W4	Note 1: 2: 1 1 W4 Before nstruction EDA8	This instruct This instruct instruction ( MOV. ; Pop TO	tion operates i tion is a spec MOV [W15 S to W4 After Instruction 4 C45A	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15	Note 1: 2: 1 1 W4 Before nstruction EDA8 1008	This instruct This instruct instruction ( MOV. ; Pop TO W2 W15	ion operates i tion is a spec MOV [W15 S to W4 After Instruction C45A 5 1006	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP I W4	Note 1: 2: 1 1 W4 Before nstruction EDA8 1008	This instruct This instruct instruction ( MOV. ; Pop TO	tion operates i tion is a spec MOV [W15 S to W4 After Instruction C45A 5 1006 5 C45A	n Word mo	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006	Note 1:         2:           1         1           W4         Before           nstruction         EDA8           1008         C45A           0000         0000	This instruct This instruct instruction ( MOV. ; Pop TO W4 W15 Data 1006 SF ; Pre-in	tion operates i tion is a spec MOV [W15 S to W4 After Instruction C45A 5 1006 5 C45A	in Word mod ific version 5], wd). It	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006 SR Example 2 POP	Note 1:         2:           1         1           W4         Before           nstruction         EDA8           1008         C45A           0000         [++W10]           Before         Before	This instruct This instruct instruction ( MOV. ; Pop TO W4 W15 Data 1006 SF ; Pre-in	tion operates it tion is a spec MOV [W15 S to W4 After Instruction C45A 0000 crement W10 S to [W10] After	in Word mod ific version 5], wd). It	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006 SR Example 2 POP	Note 1: 2: 1 1 W4 Before nstruction EDA8 1008 C45A 0000 [++W10] Before nstruction	This instruct This instruct instruction ( MOV. ; Pop TO W4 W15 Data 1006 SF ; Pre-in ; Pop TO	tion operates it tion is a spec MOV [W15 S to W4 After Instruction CC45A CC4	in Word mod ific version 5], wd). It	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006 SR Example 2 POP	Note 1: 2: 1 1 W4 Before nstruction EDA8 1008 C45A 0000 [++W10] Before nstruction	This instruct This instruct instruction ( MOV. ; Pop TO W4 W15 Data 1006 SF ; Pre-in	tion operates i tion is a spec MOV [W15 S to W4 After Instruction C45A 0000 Crement W10 S to [W10] After Instruction 0 0E04	in Word mod ific version 5], wd). It	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006 SR Example 2 POP	Note 1:           2:           1           1           W4           Before           nstruction           EDA8           1008           C45A           0000           [++W10]           Before           nstruction           0E02           1766	This instruct This instruct instruction (i MOV. ; Pop TO W4 W18 Data 1006 SF ; Pre-in ; Pop TO W10	tion operates in tion is a spec MOV [W15 S to W4 After Instruction C45A 1006 C45A 0000 crement W10 S to [W10] After Instruction 0 E04 5 1764	in Word mod ific version 5], wd). It	de only. of the "MOV	
Cycles: Example 1 POP I W4 W15 Data 1006 SR Example 2 POP I W10 W15	Note 1:           2:           1           1           W4           Before           nstruction           EDA8           1008           C45A           0000           [++W10]           Before           nstruction           0E02           1766           E3E1	This instruct This instruct instruction ( MOV. ; Pop TO W4 W15 Data 1006 SF ; Pre-in ; Pop TO W10 W15	tion operates it tion is a spec MOV [W15 S to W4 After Instruction CC45A 0000 Crement W10 S to [W10] After Instruction O E04 1764 C7B5	in Word mod ific version 5], wd). It	de only. of the "MOV	

Instruction Descriptions

POP.D		Double Pop	TOS to Wn	d:Wnd+1		
Syntax:	{label:}	POP.D	Wnd			
Operands: Operation:	Wnd $\in$ [Wr (W15)-2 $\rightarrow$ (TOS) $\rightarrow$ W (W15)-2 $\rightarrow$ (TOS) $\rightarrow$ W	Vnd+1 → W15	. W14]			
Status Affected:	None	ind .				
Encoding:	1011	1110	0000	0ddd	0100	1111
Description:	Wnd:Wnd- Least Sign the stack p	vord is poppe +1. The Most ificant Word i pointer (W15)	Significant W s stored to W gets decrem	Vord is stored Ind. Since a d ented by 4.	d to Wnd+1, double-word	and the
	The 'd' bits	s select the ad	ddress of the	destination	register pair.	
	2:	This instruct information of Wnd must b This instruct instruction (M MOV.D.	on how doub e an even wo	le-words are orking registe fic version of	aligned in m er. the "MOV.D	ws, Wnd"
Words:	1					
Cycles:	2					
Example 1 POP.	D W6	; Doub	le pop TOS	to W6		
In W6 W7 W15 Data 084C Data 084E SR	Before struction 07BB 89AE 0850 3210 7654 0000	W6 W7 W15 Data 084C Data 084E SF	7654 084C 3210 7654			
Example 2 POP.	D WO	; Doub	le pop TOS	to WO		
In W0 W1 W15 Data 0BB8 Data 0BBA SR	Before struction 673E DD23 0BBC 791C D400 0000	W0 W15 Data 0BB8 Data 0BBA SF	D400 0BB8 791C D400			

POP.S		Pop Shado	w Registers	i		
Syntax:	{label:}	POP.S				
Operands:	None					
Operation:	Pop shad	ow registers				
Status Affected:	DC, N, O	-				
Encoding:	1111	1110	1000	0000	0000	0000
Description:	primary re	es in the shado egisters. The f N and DC Sta	ollowing regi	sters are affe		
		The shadow only be acce The shadow	essed with Pt	JSH.S and P	OP.S.	They may
Words:	1					
Cycles:	1					
Example 1 PO		Pop the sha (See PUSH.S	5		ents of sha	adows)
	Before		After			
	Instruction		Instruction	l		
W			<b>VO</b> 0000			
W			V1 1000			
W			<b>V2</b> 2000			
W			<b>V3</b> 3000			
SI	R 00E0 (IF	'L=/)	SR 00E1	(IPL=7, C=1)	)	

Note:	After instruction execution	, contents of shadow	registers are NOT modified.
-------	-----------------------------	----------------------	-----------------------------

PUSH		Push f to T	os			
Syntax:	{label:}	PUSH	f			
Operands:	f ∈ [0 65	534]				
Operation:	(f) → (TO (W15)+2 –					
Status Affected:	None					
Encoding:	1111	1000	ffff	ffff	ffff	fff0
Description:	(TOS) loca	nts of the spe ation and ther gister may res	the stack po	ointer (W15)	is incremente	ed by 2.
	The 'f' bits	select the ad	dress of the	file register.		
		This instruct The file regis				
Words:	1					
Cycles:	1					
Example 1 PUSH	0x2004	; Pu	sh (0x2004	) to TOS		
	Before		After			
In W15	struction	W15	Instruction			
Data 0B00	0B00 791C	Data 0B00				
Data 2004	D400	Data 2004				
SR	0000	SF	0000			
Example 2 PUSH	0xC0E	; Pus	h (0xC0E)	to TOS		
	Before struction		After Instruction			
W15	0920	W15				
Data 0920	0000	Data 0920				
Data 0C0E	67AA	Data 2004				
SR	0000	SF	0000			

PUSH		Push Ws	s to TOS
Syntax:	{label:}	PUSH	Ws
			[Ws]
			[Ws++]
			[Ws]
			[Ws]
			[++Ws]
			[Ws+Wb]
Operands:	Ws ∈ [W0 Wb ∈ [W0	-	
Operation:	(Ws) → ( (W15)+2 ·		
Status Affected:	None		
Encoding:	0111	lwww	w001 1111 1ggg ssss
Description:			re written to the Top-of-Stack (TOS) location and r (W15) is incremented by 2.
	The 'g' bit	s select the	e offset register Wb. source Address mode.
	Note 1:	This instru	
Words:	Note 1:	This instru	uction operates in Word mode only. uction is a specific version of the "MOV Ws, Wd
Words: Cycles:	Note 1: 2:	This instru This instru instruction	uction operates in Word mode only. uction is a specific version of the "MOV Ws, Wd
	Note 1: 2: 1 1	This instru This instru instruction MOV.	-
Cycles:	Note 1: 2: 1 1	This instru This instru instruction MOV.	uction operates in Word mode only. uction is a specific version of the "MOV Ws, Wo (MOV Ws, [W15++]). It reverse assembles a
Cycles: Example 1 PUS	Note 1: 2: 1 1 5H W2 Before Instruction	This instru This instru instruction MOV.	After Instruction
Cycles: Example 1 PUS W2	Note 1: 2: 1 1 5H W2 Before Instruction 2 6889	This instru This instru instruction MOV. ; Pus	After Instruction N2 6889
Cycles: Example 1 PUS W2 W15	Note 1: 2: 1 1 5H W2 Before Instruction 2 6889 5 1566	; This instru This instru instruction MOV. ; Pus	After Instruction M2 6889 15 1568
Cycles: Example 1 PUS W2	Note 1: 2: 1 1 SH W2 Before Instruction 2 6889 1566 5 0000	This instru This instru instruction MOV. ; Pus V W Data 15	After Instruction M2 6889 15 1568
Cycles: Example 1 PUS W2 W15 Data 1566	Note 1: 2: 1 1 5H W2 Before Instruction 2 6889 5 1566 5 0000 0000	This instru This instru instruction MOV. ; Pus V W Data 15 S	After Instruction N2 6889 15 1568 66 6889
Cycles: Example 1 PUS W2 W15 Data 1566 SF	Note 1: 2: 1 1 3H W2 Before Instruction 2 6 8 9 1566 0000 3 0000 3 5 H [W5+W10 Before	This instru This instru instruction MOV. ; Pus V W Data 15 S	After Instruction N2 6889 15 1568 66 6889 SR 0000 sh [W5+W10] to TOS After
Cycles: Example 1 PUS W15 Data 1566 SF Example 2 PUS	Note 1:           1           1           SH         W2           Before           Instruction           2         6889           5         0000           5         0000           5         0000           5         0000           5         0000           5         0000           5         0000           5         0000           5         0000           5         0000	This instru This instru instruction MOV. ; Pus V Data 15 S	After Instruction Sh [W5+W10] to TOS After Instruction After MOV WS, [W15++]). It reverse assembles a Mathematical State (MOV WS, W15++]). It reverse assembles a Mathematical State (MOV WS, [W15++]). It reverse assembles as Mathematical State (MOV WS, [W15++]). It reverse
Cycles: Example 1 PUS W15 Data 1566 SF Example 2 PUS	Note 1: 2: 1 1 5H W2 Before Instruction 2 6889 5 1566 6 0000 5 00000 5 00000 5 00000 5 00000000	This instru This instru instruction MOV. ; Pus V Data 15 3 0] ; Pus	After Instruction (MOV WS, [W15++]). It reverse assembles a After Instruction N2 6889 (15 1568 66 6889 SR 0000 sh [W5+W10] to TOS After Instruction N5 1200
Cycles: Example 1 PUS W15 Data 1566 SF Example 2 PUS	Note 1:         1         1         5H       W2         Before         Instruction         2       6889         5       1566         0000         5H       (W5+W10)         Before         Instruction         5       1200         0004	This instru This instru instruction MOV. ; Pus V Data 15 S 0] ; Pus V W W	After Instruction Sh [W5+W10] to TOS After Instruction After MOV WS, [W15++]). It reverse assembles a Mathematical State (MOV WS, W15++]). It reverse assembles a Mathematical State (MOV WS, [W15++]). It reverse assembles as Mathematical State (MOV WS, [W15++]). It reverse
Cycles: Example 1 PUS W15 Data 1566 SF Example 2 PUS W10	Note 1: 2: 1 1 3H W2 Before Instruction 2 6889 1566 0000 3 0000 3 00000 3 0000 3 0000 3 00000000	This instru This instru instruction MOV. ; Pus V Data 15 S 0] ; Pus V W W	After Instruction MV WS, [W15++]). It reverse assembles a After Instruction M2 6889 15 1568 66 6889 SR 0000 sh [W5+W10] to TOS After Instruction M5 1200 10 0044 15 0808
Cycles: Example 1 PUS W15 Data 1566 SF Example 2 PUS W10 W15	Note 1: 2: 1 1 5H W2 Before Instruction 2 6889 5 1566 6 0000 5H [W5+W10 Before Instruction 5 1200 0044 0 806 2 216F 4 B20A	This instru This instru instruction MOV. ; Pus V Data 15 3 0] ; Pus V W W Data 08 Data 12	After Instruction Sh [W5+W10] to TOS After Instruction M2 6889 SR 0000 Sh [W5+W10] to TOS After Instruction M5 1200 10 0044 15 0808 06 B20A

Instruction Descriptions

PUSH.D		Double Pus	sh Wns:Wns	+1 to TOS		
Syntax:	{label:}	PUSH.D	Wns			
Operands: Operation:	Wns ∈ [We (Wns) → ( (W15)+2 - (Wns+1) - (W15)+2 -	→ W15 → (TOS)	W14]			
Status Affected:	None	7 111				
Encoding:	1011	1110	1001	1111	1000	sss0
Description:	Least Sign Significant	vord (Wns:Wi ificant word ( word (Wns+: rd is pushed,	Wns) is push 1) is pushed	ned to the TC to the TOS I	OS first, and tast. Since a	he Most
	The 's' bits	s select the ac	dress of the	source regis	ster pair.	
	2:	This instruct for informatic Wns must be This instruct instruction (I as MOV.D.	on on how do e an even wo ion is a speci	ouble-words orking registe ific version of	are aligned i er. f the "MOV.D	n memory. Wns, Wd"
Words:	1					
Cycles:	2					
Example 1 PUSH	.D W6	; Pu	sh W6:W7 t	o TOS		
lr W6 W7 W15 Data 1240 Data 1242 SR	Before astruction C451 3380 1240 B004 0891 0000	W6 W7 W15 Data 1240 Data 1242 SF	3380 1244 C451 3380			
Example 2 PUSH	.D W10	; P	ush W10:W1	1 to TOS		
اr W10 W11 W15 Data 0C08 Data 0C0A SR	Before astruction 80D3 4550 0C08 79B5 008E 0000	W10 W11 W15 Data 0C08 Data 0C0A SF	4550 0C0C 8 80D3 4550			

PUSH.S		Push Shade	ow Register	s		
Syntax:	{label:}	PUSH.S				
Operands:	None					
Operation:		dow registers				
Status Affected:	None	low registere				
Encoding:	1111	1110	1010	0000	0000	0000
Description:	shadow re	nts of the prime gisters. The f DV, N and DC	ollowing regi	sters are sha		
		The shadow only be acce The shadow	ssed with Pt	JSH.S and P	OP.S.	They may
Words:	1					
Cycles:	1					
Example 1	PUSH.S ; P	Push primary	y register	s into sha	adow regist	ers
	Before		After			
	Instruction		Instruction			
	<b>WO</b> 0000	V	/0 0000			
	W1 1000	W	/1 1000			
	<b>W2</b> 2000	V	<b>/2</b> 2000			
	<b>W3</b> 3000	V	<b>/3</b> 3000			
	SR 0001 (C	=1) S	R 0001	(C=1)		



PWRSA	V Enter Power Saving Mode
Syntax:	{label:} PWRSAV #lit1
Operands: Operation:	$\begin{array}{l} \text{lit1} \in [0,1] \\ 0 \rightarrow \text{WDT count register} \\ 0 \rightarrow \text{WDT prescaler A count} \\ 0 \rightarrow \text{WDT prescaler B count} \\ 0 \rightarrow \text{WDTO (RCON<4>)} \\ 0 \rightarrow \text{SLEEP (RCON<3>)} \\ 0 \rightarrow \text{IDLE (RCON<2>)} \\ \hline \text{If (lit1 = 0):} \\ \hline \text{Enter SLEEP mode} \\ \hline \hline \text{Else:} \end{array}$
Status Affected:	Enter IDLE mode None
Encoding:	1111 1110 0100 0000 0000 000k
Description:	Place the processor into the specified Power Saving mode. If $lit1 = 0$ , SLEEP mode is entered. In SLEEP mode, the clock to the CPU and peripherals are shutdown. If an on-chip oscillator is being used, it is also shutdown. If $lit1 = 1$ , IDLE mode is entered. In IDLE mode, the clock to the CPU shuts down, but the clock source remains active and the peripherals continue to operate.
	This instruction resets the Watchdog Timer Count register and the Prescaler Count registers. In addition, the WDTO, SLEEP and IDLE flags of the Reset System and Control (RCON) register are reset.
	<ul> <li>Note 1: The processor will exit from IDLE or SLEEP through an interrupt, processor RESET or Watchdog Time-out. See the dsPIC30F Data Sheet for details.</li> <li>2: If awakened from IDLE mode, IDLE (RCON&lt;2&gt;) is set to '1' and the clock source is applied to the CPU.</li> <li>3: If awakened from SLEEP mode, SLEEP (RCON&lt;3&gt;) is set to '1' and the clock source is started.</li> <li>4: If awakened from a Watchdog Time-out, WDTO (RCON&lt;4&gt;) is set to '1'.</li> </ul>
Words:	1
Cycles:	1
Example 1	PWRSAV #0 ; Enter SLEEP mode
Example 2	Before     After       Instruction     Instruction       SR     0040       (IPL=2)     SR       PWRSAV     #1       ; Enter IDLE mode
	BeforeAfterInstructionInstructionSR0020(IPL=1)SR0020(IPL=1)

RCALL		Relative Ca	11			
Syntax:	{label:}	RCALL	Expr			
Operands:	• •		e address, lab inker to a Slit <sup>-</sup>	•		32767
Operation:	(W15) + 2 (PC) + (2 *	ightarrow (TOS) ightarrow W15 ightarrow (TOS)				
Status Affected:	None					
Encoding:	0000	0111	nnnn	nnnn	nnnn	nnnn
Description:	from the cu pushed on sign-extend	rrent PC. Bei to the stack. A	with a range o fore the call is After the return ue (2 * Slit16) n the PC.	made, the re address is s	eturn address stacked, the	(PC+2) is
		are a signed ords) from (P	literal that sp C+2).	ecifies the siz	e of the relat	ive call (in
	Note:		le, this instruc			
Words:	1					
Cycles:	2					
01	.2004 .2006 .2458 _Task	RCALL ADD  1: SUB	_Task1 W0, W1, W2 W0, W2, W3		l _Task1 sk1 subrou	tine
01	_245A					
	Before		1.	After		
P		_	PC	nstruction		
W1			W15	01 2458		
Data 081			Data 0810	2006		
Data 081			Data 0010	0001		
	R 000		SR	0000		
	)620E )6210	RCALL MOV 	_Init W0, [W4++		l _Init	
	07000 _Init 07002 _	: CLR	₩2	; _In	it subrout	ine
	Before		1	After		
				nstruction		
P W1			PC W15	00 7000 0C54		
Data 0C5			Data 0C50	6210		
Data 0C5 Data 0C5			Data 0C50	0000		
S			SR	0000		
0		<u> </u>		0000		

RCALL		Compute	ed Relative	Call		
Syntax:	{label:}	RCALL	Wn			
Operands: Operation:	$ \begin{array}{l} Wn \in [W0 \\ (PC) + 2 \rightarrow \\ (PC < 15:0 > \\ (W15) + 2 \rightarrow \\ (PC < 22:16 \\ (W15) + 2 \rightarrow \\ (PC) + (2 * \\ NOP \rightarrow Ins \end{array} $	$ \begin{array}{l} \rightarrow PC \\ ) \rightarrow (TOS \\ \rightarrow W15 \\ >) \rightarrow (TOS \\ \rightarrow W15 \\ (Wn)) \rightarrow \end{array} $	S) PC			
Status Affected:	None		0			
Encoding:	0000	0001	0010	0000	0000	SSSS
Description:	The range of PC. Before stack. After (2 * (Wn)) is	of the call is the call is the return added to	s 32K progra made, the re address is s the contents	specified by the w m words forward of turn address (PC- tacked, the sign-e of the PC and the st be used for Wr	br back from +2) is pushed extended 17- e result is sto	the current d onto the pit value
	The 's' bits	select the	address of	the source regist	er.	
Words:	1					
Cycles:	2					
Example 1 00FF8 00FF8	3E )8 )A	INC   RCALL MOVE	W2, W3 W6 W4, [W10	; RCAL	ination of L with W6	E RCALL
01000		MOVE	W4, [WIU			
PC W6 W15 Data 1004 Data 1006 SR	Before Instruction 01 000A FFC0 1004 98FF 2310 0000		PC W6 W15 Data 1004 Data 1006 SR	After Instruction 00 FF8C FFC0 1008 000C 0001 0000		
Example 2 00030 00030		RCALL FF1L 	W2 W0, W1	; RCAL	L with W2	
00049 00049		 CLR 	W2	; Dest:	ination of	RCALL
PC W2 W15 Data 1004 Data 1006 SR	Before Instruction 00 0302 00A6 1004 32BB 901A 0000		PC W2 W15 Data 1004 Data 1006 SR	After         Instruction         00       0450         00       00A6         1008         0304         0000         0000		

REPEAT	Repeat Next Instruction 'lit14+1' Times
Syntax:	{label:} REPEAT #lit14
Operands:	lit14 ∈ [0 16383]
Operation:	(lit14) $\rightarrow$ RCOUNT (PC)+2 $\rightarrow$ PC Enable Code Looping
Status Affected:	RA
Encoding:	0000 1001 00kk kkkk kkkk kkkk
Description:	Repeat the instruction immediately following the REPEAT instruction (lit14 + 1) times. The repeated instruction (or target instruction) is held 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.
	<ol> <li>Special Features, Restrictions:</li> <li>When the repeat literal is '0', REPEAT has the effect of a NOP and the RA bit is not set.</li> <li>The target REPEAT instruction can NOT be:         <ul> <li>an instruction that changes program flow</li> <li>a DO, DISI, LNK, MOV.D, PWRSAV, REPEAT or UNLK instruction</li> <li>a 2-word instruction</li> </ul> </li> </ol>
	<ul> <li>a 2-word instruction</li> <li>Unexpected results may occur if these target instructions are use</li> </ul>
	<b>Note:</b> The REPEAT and target instruction are interruptible.
Words:	1
Cycles:	1
Example 1 0004	
	Before After
PC	Instruction         Instruction           00 0452         PC         00 0454
RCOUNT	0000432 PC 000434 0000 RCOUNT 0009
SR	0000 SR 0010 (RA=1)
Example 2 0008	
PC RCOUNT SR	Before         After           Instruction         Instruction           00         089E         PC         00         08A0           0000         RCOUNT         03FF           0000         SR         0010         (RA=1)

REPEAT		Repeat Ne	xt Instructio	n Wn+1 Tim	es	
Syntax:	{label:}	REPEAT	Wn			
Operands:	Wn ∈ [W0	W15]				
Operation:	(PC)+2 $\rightarrow$	>) $\rightarrow$ RCOUN PC ode Looping	ΙT			
Status Affected:	RA					
Encoding:	0000	1001	1000	0000	0000	SSSS
Description:	(Wn<13:0	e instruction i >) times. The he instructior	instruction to	be repeated	d (or target in	struction)
	lower 14-b the target is execute	instruction e bits of Wn. RC instruction. W d one more t with the instr	COUNT is de /hen RCOUN ime, and ther	cremented w IT equals zer n normal inst	ith each exec o, the target ruction exect	cution of
	The 's' bits	s specify the	Wn register t	hat contains	the repeat co	ount.
	•	eatures, Res n (Wn) = 0, Ri et.		e effect of a	NOP and the	RA bit is
	2. The ta	arget REPEAT	instruction	can NOT be:		
	• an	instruction the	at changes p	rogram flow		
	inst	O, DISI, I truction		, PWRSAV,	REPEAT OF	JLNK
		-word instruc				
	Unex	pected result	s may occur i	if these targe	t instructions	are used.
	Note:	The REPEA'	I and target	instruction a	re interruptib	le.
Words:	1					
Cycles:	1					
Example 1 000A2		ſ ₩4 [₩0++],		Execute CO Vector con		cimes
PC W4 RCOUNT SR	Before Instruction 00 0A26 0023 0000 0000	- - - -		After struction 00 0A28 0023 0023 0010 (R	A=1)	

	0089E)008A0	REPEAT TBLRDL	W10 [W2++], [W3++]		TBLRD (W10+1) times t (0x840)
		Before truction		After Instruction	
	PC C	0 089E	PC	00 08A0	]
W	/10	00FF	W10	00FF	
RCOU	NT	0000	RCOUNT	00FF	
	SR	0000	SR	0010	(RA=1)

RESET		Reset			
Syntax:	{label:}	RESET			
Operands: Operation:	condition.	egisters that are affect (RCON<6>)	ted by a MCLI	R Reset to their	RESET
Status Affected:	OA, OB, C	AB, SA, SB, SAB, DA	, DC, IPL<2:0	)>, RA, N, OV, Z	Z, C
Encoding:	1111	1110 0000	0000	0000	0000
Description:	and periph set to '0', t	ction provides a way t eral registers will take he location of the RES , will be set to '1' to in	e their power-c	on value. The Po truction. The SV	C will be VR bit,
	Note:	Refer to the <i>dsPICs</i> power-on value of all		Reference Mani	ual for the
Words:	1				
Cycles:	1				
Example 1 00202	2A RESET	; Execute sc	oftware RESI	ET	
	Before		After		
РСГ	Instruction	PC	Instruction	1	
wo	00 202A 8901	W0	00 0000		
W1	08BB	W1	0000		
W2	B87A	W2	0000		
W3	872F	W3	0000		
W4	C98A	W4	0000		
W5	AAD4	W5	0000		
W6	981E	W6	0000		
W7	1809	W7	0000		
W8	C341	W8	0000		
W9 W10	90F4	W9 W10	0000		
W10	F409 1700	W10	0000		
W12	1008	W12	0000		
W13	6556	W13	0000		
W14	231D	W14	0000		
W15	1704	W15	0800		
SPLIM	1800	SPLIM	0000		
TBLPAG	007F	TBLPAG	0000		
PSVPAG	0001	PSVPAG	0000	1	
CORCON	00F0	CORCON	0020	(SATDW=1)	
RCON	0000	RCON	0040	(SWR=1)	
SR	0021	(IPL, C=1) SR	0000		

RETFIE	F	Return from Interrupt			
Syntax:	{label:} F	RETFIE			
Operands:	None				
Operation:	$\begin{array}{l} (TOS{<}7{>}) \rightarrow \\ (TOS{<}6{:}0{>}) \\ (W15){-}2 \rightarrow V \\ (TOS{<}15{:}0{>}) \end{array}$	) → (SR<7:0>) • (IPL3, CORCON<3>) → (PC<22:16>)			
Status Affected:	IPL<3:0>, R/	A, N, OV, Z, C			
Encoding:	0000	0110 0100	0000	0000	0000
Description:	the low byte	Interrupt Service Routin of the Status register, IP yte of the PC. The stack of the PC.	L<3> (CORC	CON<3>) an	d the Mos
	e <b>2:</b> B	estores the Interrupt Pr xecution was processed lefore RETFIE is execu- nust be cleared in softwa	l. uted, the ap	propriate int	terrupt fla
147 1					
	1 3 (2 if except	tion pending)			
	3 (2 if except	tion pending) ; Return from IS			
Cycles:	<b>3 (2 if excep</b>	; Return from IS	R		
Cycles:	3 (2 if except A26 RETFIE Before	; Return from IS	R After		
Cycles: Example 1 0002	3 (2 if except A26 RETFIE Before Instruction	; Return from IS	R After struction		
Cycles: Example 1 0002 PC	3 (2 if except A26 RETFIE Before Instruction 00 0A26	; Return from IS PC 0 W15 Data 0830	R After struction		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101	; Return from IS PC (0 W15 Data 0830 Data 0832	R After struction 01 0230 0830 0230 8101		
PC W15 Data 0830 Data 0832 CORCON	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0001	; Return from IS PC (0 W15 Data 0830 Data 0832 CORCON	R struction 01 0230 0830 0230 8101 0001		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 00001 0000	; Return from IS PC (0 W15 Data 0830 Data 0832	R struction 01 0230 0830 0230 8101 0001 0081 (IP	<sup>.</sup> L=4, C=1)	
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832 CORCON SR	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 00001 0000	; Return from IS PC 0 W15 Data 0830 Data 0832 CORCON SR	R struction 01 0230 0830 0230 8101 0001 0081 (IP		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832 CORCON SR	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0000 8101 0000	; Return from IS PC (0 W15 Data 0830 Data 0832 CORCON SR ; Return from IS	R struction 01 0230 0830 0230 8101 0001 0081 (IP R		
Cycles: Example 1 0007 PC W15 Data 0830 Data 0832 CORCON SR Example 2 0080 PC	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0000 8101 0000 050 RETFIE Before	; Return from IS PC ( W15 Data 0830 Data 0832 CORCON SR ; Return from IS PC (	R Struction 01 0230 0830 0230 8101 0001 0081 (IP R After		
Cycles: Example 1 0007 PC W15 Data 0830 Data 0832 CORCON SR Example 2 0080 PC W15	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0001 0000 050 RETFIE Before Instruction	; Return from IS PC ( W15 Data 0830 Data 0832 CORCON SR ; Return from IS PC ( W15	R After struction 01 0230 0830 0230 8101 0001 0081 (IP R After struction		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832 CORCON SR Example 2 0080 PC W15 Data 0922	3 (2 if except           A26         RETFIE           Before         Instruction           00         0A26           0834         0230           8101         0001           0000         0000           050         RETFIE           Before         Instruction           000         8050	; Return from IS PC (0 W15 Data 0830 Data 0832 CORCON SR ; Return from IS Nts W15 Data 0922	R struction 01 0230 0830 0230 8101 0001 0081 (IP R After struction 00 7008		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832 CORCON SR Example 2 0080 PC W15 Data 0922 Data 0924	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0001 0000 050 RETFIE Before Instruction 00 8050 0926 7008 0300	; Return from IS PC (0 W15 Data 0830 Data 0832 CORCON SR ; Return from IS Nts Data 0922 Data 0924	R After struction 01 0230 0830 0230 8101 0001 0001 0081 (IP R After struction 00 7008 0922 7008 0300		
Cycles: Example 1 0002 PC W15 Data 0830 Data 0832 CORCON SR Example 2 0080 PC W15 Data 0922	3 (2 if except A26 RETFIE Before Instruction 00 0A26 0834 0230 8101 0001 0000 050 RETFIE Before Instruction 00 8050 0926 7008	; Return from IS PC (0 W15 Data 0830 Data 0832 CORCON SR ; Return from IS Nts W15 Data 0922	R struction 01 0230 0830 0230 8101 0001 0001 (IP R After struction 00 7008 0922 7008 0300 0300 0000		

RETLW		Return with	1 Literal in	Wn			
Syntax:	{label:}	RETLW{.B}	#lit10,	Wn			
Operands:	-	. 255] for byte . 1023] for we W15]	•				
Operation:	(W15)-2 <i>—</i> ;	PC<22:16>) → W15 PC<15:0>)					
Status Affected:	None						
Encoding:	0000	0101	0Bkk	kkkk		kkkk	dddd
Description:	in Wn. The signed liter pointer (W	m subroutine software sta ral is stored ir 15) is decrem	nck is poppe n Wn. Since nented by 4	ed twice to i two pops a	restore are ma	the PC de, the s	and the stack
	The 'k' bits	selects byte of specify the version of the select the advised the select the advised the select the	alue of the	literal.			∕te).
		The extension rather than a denote a wo For byte open value [0:255 ands" for int mode.	a word oper ord operatio rations, the ij. See <b>Sec</b>	ration. You i n, but it is n literal must l <b>tion 4.6 "L</b>	may us lot requ be spec <b>Ising</b> 1	e a .we uired. cified as a I <b>0-bit Li</b>	extension to an unsigned <b>teral Oper</b>
Words:	1						
Cycles:	3 (2 if exce	eption pending	g)				
Example 1 0004	140 RETLW	.B #0xA, W	0 ; Ret	urn with	0xA :	in WO	
	Before			After			
50	Instruction			nstruction	I		
PC	Instruction 00 0440		PC	nstruction 00 7006			
wo	Instruction           00         0440           9846		PC W0	nstruction 00 7006 980A			
W0 W15	Instruction 00 0440 9846 1988		PC W0 W15	nstruction 00 7006 980A 1984			
W0 W15 Data 1984	Instruction 00 0440 9846 1988 7006		PC W0 W15 ata 1984	nstruction 00 7006 980A 1984 7006			
W0 W15	Instruction 00 0440 9846 1988		PC W0 W15	nstruction 00 7006 980A 1984			
W0 W15 Data 1984 Data 1986	Instruction 00 0440 9846 1988 7006 0000 0000	Da	PC W0 W15 ata 1984 ata 1986 SR	nstruction 00 7006 980A 1984 7006 0000	0x230	) in W2	:
W0 W15 Data 1984 Data 1986 SR	Instruction 00 0440 9846 1988 7006 0000 0000 50A RETLW Before	Da	PC W0 W15 ata 1984 ata 1986 SR W2 ; Ret	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After	0x230	) in W2	
W0 W15 Data 1984 Data 1986 SR Example 2 0005	Instruction 00 0440 9846 1988 7006 0000 0000 50A RETLW Before Instruction	Da	PC W0 W15 ata 1984 ata 1986 SR W2 ; Ret	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After nstruction	0x230	) in W2	
W0 W15 Data 1984 Data 1986 SR Example 2 0005	Instruction 00 0440 9846 1988 7006 0000 50A RETLW Before Instruction 00 050A	Da	PC W0 W15 ata 1984 ata 1986 SR W2 ; Ret	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After nstruction 01 7008	0x230	) in W2	
W0 W15 Data 1984 Data 1986 SR Example 2 0005 PC W2	Instruction           00         0440           9846         1988           1988         7006           0000         0000           50A         RETLW           Before Instruction           00         050A           00         050A	Da	PC W0 W15 ata 1984 ata 1986 SR W2 ; Ret	Anstruction           00         7006           980A         1984           7006         0000           00000         0000           curn with         After           nstruction         01         7008           0230         0230	0x230	) in W2	:
W0 W15 Data 1984 Data 1986 SR Example 2 0005 PC W2 W15	Instruction 00 0440 9846 1988 7006 0000 50A RETLW Before Instruction 00 050A 0993 1200	Da #0x230,	PC	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After nstruction 01 7008 0230 11FC	0x230	) in W2	
W0 W15 Data 1984 Data 1986 SR Example 2 0005 PC W2 W15 Data 11FC	Instruction           00         0440           9846         1988           7006         0000           0000         0000           50A         RETLW           Before Instruction           00         050A           000         050A           1200         7008	Da #0x230, *	PC W0 W15 ata 1984 ata 1986 SR W2 ; Ret W2 ; Ret U2 W2 W15 tta 11FC	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After nstruction 01 7008 0230 11FC 7008	0x230	) in W2	1
W0 W15 Data 1984 Data 1986 SR Example 2 0005 PC W2 W15	Instruction 00 0440 9846 1988 7006 0000 50A RETLW Before Instruction 00 050A 0993 1200	Da #0x230, *	PC	nstruction 00 7006 980A 1984 7006 0000 0000 curn with After nstruction 01 7008 0230 11FC	0x230	) in W2	:

RETURN		Return					
Syntax:	{label:}	RETURN					
Operands:	None						
Operation:	$(TOS) \rightarrow (I)$ $(W15)-2 \rightarrow (TOS) \rightarrow (I)$	$\begin{array}{l} (W15)\text{-}2 \rightarrow W15 \\ (TOS) \rightarrow (PC\text{-}22\text{:}16\text{-}) \\ (W15)\text{-}2 \rightarrow W15 \\ (TOS) \rightarrow (PC\text{-}15\text{:}0\text{-}) \\ \text{NOP} \rightarrow \text{Instruction Register} \end{array}$					
Status Affected:	None						
Encoding:	0000	0110 000	0 0000	0000	0000		
Description:		n subroutine. The so nce two pops are ma ed by 4.					
Words:	1						
Cycles:	3 (2 if exce	ption pending)					
Example 1 001A	06 RETUR	N ; Return f	rom subroutin	e			
	Before		After				
PC	Instruction	PC	Instruction				
FC W15	1248	W15					
Data 1244	0004	Data 1244					
Data 1246	0001	Data 1246	0001				
_			0001				
SR	0000	SR					
SR Example 2 0054		SR		e			
L		SR	0000	e			
L	04 RETUR	SR N ; Return f PC	0000 rom subroutin After Instruction 00 0966	e			
Example 2 0054	04 RETUR Before Instruction	SR N ; Return f	0000 rom subroutin After Instruction 00 0966	e			
Example 2 0054	Before Instruction	SR N ; Return f PC	0000 rom subroutin After Instruction 00 0966 0906	e			
Example 2 0054 PC [ W15 ]	Before Instruction 00 5404 090A	SR N ; Return f PC W15	0000 rom subroutin After Instruction 00 0966 0906 0966 0966 0000	e			

RLC	LC Rotate Left f through Carry							
Syntax:	{label:}	RLC{.B}	f	{,WREG}				
Operands:	f ∈ [0	8191]						
Operation:	$(C) \rightarrow (f < 6:0>)$ $(f < 7>)$ <u>For word</u> $(C) \rightarrow$	<u>operation:</u> Dest<0> )>) → Dest<1;						
		∢						
Status Affected:	N, Z, C							
Encoding:	1101	0110	1BDf	ffff	ffff	ffff		
	of the Sta destination The option	g and place th itus Register i on, and it is the nal WREG op specified, the	is shifted into en overwritter perand detern	the Least Sig with the Mos nines the des	nificant bit o st Significan tination regi	of the t bit of Ws ster. If		
	specified	, the result is a the result is a	stored in the f	ile register.				
	The 'D' b	it selects byte it selects the o s select the a	destination (0	for f, 1 for W		(ie).		
			a word opera ord operation	tion. You may but it is not i	y use a .w e required.			
Words:	1							
Cycles:	1							
Example 1 RI	C.B 0x123	3 ; Rot	ate Left w	/ C (0x123	3) (Byte 1	mode)		
Data 123 S		Data 123 Si		l, C=1)				
Example 2 RI	uC 0x820,		cate Left w pre result		) (Word m	ode)		
WREG (W0 Data 082 S	0 216E	WREG (W0 Data 082 C=1) SF	<b>0</b> 216E	S=0)				

RLC		Rotate Le	ft Ws throu	gh Carry		
Syntax:	{label:}	RLC{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:		0 W15] 0 W15]				
Operation:	$(C) \rightarrow (Ws<6)$ $(Ws<7)$ $(Ws<7)$ $(C) \rightarrow (Ws<1)$					
Status Affected:	_ <b>C</b> N, Z, C					
Encoding:	1101	0010	1Bqq	qddd	dppp	SSSS
Description:	the Carry Carry flag Wd, and register d The 'B' bi The 'q' bi The 'd' bi The 'p' bi	flag and pla g of the Statu it is then ove lirect or indirect it selects byte ts select the ts select the ts select the	ce the result s register is s rwritten with ect addressin e or word ope destination A address of th source Addre	register Ws on in the destinat shifted into the the Most Signi ng may be used eration (0 for w Address mode. ne destination ess mode. ne source regis	ion register <sup>1</sup> Least Signif ficant bit of <sup>1</sup> d for Ws and vord, 1 for by register.	Wd. The ficant bit of Ws. Either Wd.
	Note:	rather thar	a word oper	e instruction de ration. You may	/usea.we	
Wordo	4	denote a v	vora operatio	n, but it is not	required.	
Words: Cycles:	1					
	с.в wo, w			w/ C (WO) ( esult in W3	Byte mode	)
	Before Instruction		After Instruction			

Section 5	. Instruction	Descriptions
-----------	---------------	--------------

W0

WЗ

SR

9976

5879

0001 (C=1)

W0

WЗ

SR

9976

58ED

0009 (N=1)

Descriptions

Instruction

Example 2 RLC  $\mbox{[W2++]},\mbox{[W8]}$  ; Rotate Left w/ C  $\mbox{[W2]}$  (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 8082 Data 2008 C041 Data 2008 C041 SR SR 0001 (C=1) 0009 (N, C=1)

RLNC		Rotate Lef	t f without C	arry		
Syntax:	{label:}	RLNC{.B}	f	{,WREG}		
Operands:	f∈ [0 8	191]				
Operation:	(f<7>) -	pperation: ) → Dest<7:1 → Dest<0> operation:	>			
	(f<14:0	>) → Dest<1: → Dest<0>	5:1>			
Status Affected:	N, Z					
Encoding:	1101	0110	OBDf	ffff	ffff	ffff
Description:	result in th	e contents of ne destinatior Significant bi	register. The	e Most Signifi	cant bit of f is	s stored in
	WREG is	nal WREG op specified, the the result is s	e result is stor	red in WREG		
	The 'D' bi	t selects byte t selects the c s select the a	destination (0	for WREG, 1		
			a word opera ord operation	tion. You ma , but it is not	y use a .w ex required.	
Words:	1					
Cycles:	1					
Example 1 RLM	NC.B 0x123	33 ; Ro	otate Left	(0x1233) (	Byte mode)	
	Before		After			
D.+. 1000	Instruction	Data 400	Instruction			
Data 1232 SF		Data 123 SI		<b>l</b> =1)		
Example 2 RLM	NC 0x820,		otate Left core result		lord mode)	
WREG (W0) Data 0820		WREG (WC Data 082	· · · · · · · · · · · · · · · · · · ·			
SF				C=0)		

RLNC		Rotate Left Ws without Carry					
Syntax:	{label:}	RLNC{.B}	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]			
Operands:	Ws ∈ [W0 Wd ∈ [W0						
Operation:	(Ws<7: <u>For word</u> (Ws<14	$\begin{array}{l} \begin{array}{l} \begin{array}{c} \begin{array}{c} \begin{array}{c} \begin{array}{c} \begin{array}{c} \begin{array}{c} \begin{array}{c} \begin{array}{c$	15:1>				
Status Affected:	N, Z						
Encoding:	1101	0010	0Bqq	qddd	dppp	SSSS	
Description:	place the Ws is stor affected. Ws and V	result in the d red in the Lea Either registe /d.	lestination reg ast Significan r direct or inc	egister Ws on gister Wd. The t bit of Wd, ar lirect address	e Most Signif nd the Carry ing may be u	icant bit o flag is not used for	
	The 'q' bit The 'd' bit The 'p' bit	s select the c s select the a s select the s	destination Ad address of the source Addre	ration (0 for b ddress mode. e destination ss mode. e source regis	register.	rd).	
	Note:	rather than	a word opera	instruction de ation. You may	y use a .we		
Words:	1						
Cycles:	1						
Example 1	RLNC.B W0, N		cate Left ore the rea	(WO) (Byte sult in W3	mode)		
	Before Instruction W0 9976 W3 5879 SR 0001 (C	W W S=1) SF	3 58EC	N, C=1)			

Example 2	RLNC	[W2++]	, [W8]	; Pc	st-i	ncre	t [W2] ment W2 lt in	2	mode)
	В	Before			Af	fter			
	Ins	truction		Instruction					
	W2	2008		W	2 2	200A			
	W8	094E		W	<b>3</b> 0	)94E			
Data 0	94E	3689	Data	a 0941	E 8	8083			
Data 2	8008	C041	Dat	a 200	<b>3</b> C	2041			
	SR	0001 <b>(C</b>	<b>;</b> =1)	SF	R 0	009	(N, C=1	)	

RRC		Rotate Right f through Carry						
Syntax:	{label:}	RRC{.B}	f	{,WREG}				
Operands:	f ∈ [0 8	3191]						
Operation:	$\begin{array}{c} (C) \rightarrow \\ (f < 7:1 > \\ (f < 0 >) \end{array}$ $\begin{array}{c} For \ word \\ (C) \rightarrow \end{array}$	<u>operation:</u> Dest<15> >) → Dest<1						
	>	->C						
Status Affected:	N, Z, C							
Encoding:	1101	0111	1BDf	ffff	ffff	ffff		
Description:	Carry flag of the Sta	and place th tus Register	ne result in the	ter f one bit to e destination the Most Sig n with the Lea	register. The nificant bit of	Carry flag the		
	WREG is	specified, th		mines the des red in WREG file register.				
	The 'D' b	t selects the		ration (0 for b ) for WREG, 1 e file register.				
		rather than denote a w	a word operation	instruction de ation. You ma n, but it is not rking register	y use a .w ex required.			
Words:	1							
Cycles:	1							
Example 1	RC.B 0x123	3 ; Ro	tate Right	w/ C (0x12	233) (Byte	mode)		
Data 12	Before Instruction 32 E807 SR 0000	Data 123 S	After Instruction 32 7407 R 0000					
Example 2	RC 0x820,		tate Right ore result	w/ C (0x82 in WREG	20) (Word n	node)		
WREG (W Data 08	<b>20</b> 216E	WREG ( Data 0 C=1)	<b>820</b> 216E	-				

RRC		Rotate Right Ws through Carry					
Syntax:	{label:}	RRC{.B}	Ws,	Wd			
			[Ws],	[Wd]			
			[Ws++],	[Wd++]			
			[Ws],	[Wd]			
			[++Ws],	[++Wd]			
			[Ws],	[Wd]			
Operands:	Ws ∈ [W0 Wd ∈ [W0						
Operation:	(C) → (Ws<7 (Ws<0 <u>For word</u> (C) → (Ws<1	$\begin{array}{c} \hline pperation: \\ Wd<7> \\ (1>) \rightarrow Wd<6 \\ (1>) \rightarrow C \\ operation: \\ Wd<15> \\ (1>) \rightarrow Wd<6 \\ (1>) \rightarrow C \\ (1>) \rightarrow C$					
Status Affected:	N, Z, C						
Encoding:	1101	0011	1Bqq	qddd	dppp	SSSS	
Description:	the Carry Carry flag Wd, and i	Rotate the contents of the source register Ws one bit to the right through the Carry flag and place the result in the destination register Wd. The Carry flag of the Status Register is shifted into the Most Significant bit of Wd, and it is then overwritten with the Least Significant bit of Ws. Either register direct or indirect addressing may be used for Ws and Wd.					
	The 'q' bi The 'd' bi The 'p' bi	ts select the ts select the ts select the	destination A address of th source Addr	eration (0 for w Address mode. ne destination ess mode. ne source regis	register.	te).	
	Note:	rather than	a word ope	e instruction de ration. You ma n, but it is not	y use a .we	•	
Words:	1						
Cycles:	1						
Example 1	RRC.B W0, W		-	c w/ C (WO) esult in W3	(Byte mode	e)	
	Before Instruction W0 9976 W3 5879 SR 0001 (C	V	After Instruction /0 9976 /3 58BB R 0008				

Instruction Descriptions Example 2 RRC [W2++], [W8] ; Rotate Right w/ C [W2] (Word mode) ; Post-increment W2 ; Store result in [W8] After Before Instruction Instruction W2 W2 2008 200A W8 W8 094E 094E Data 094E Data 094E E020 3689 Data 2008 C041 Data 2008 C041 SR 0001 (C=1) SR 0009 (N, C=1)

RRNC		Rotate Rig	ht f without	Carry		
Syntax:	{label:}	RRNC{.B}	f	{,WREG}		
Operands:	f∈[08	-				
Operation:	(f<0>)	pperation: ) → Dest<6:0 → Dest<7> operation:	>			
		>) → Dest<14 → Dest<15>	1:0>			
	►					
Status Affected:	N, Z	-	1	1	<b>-</b>	r
Encoding:	1101	0111	OBDf	ffff	ffff	ffff
Description:	result in th	e contents of t ne destination Significant bit	register. The	Least Signif	icant bit of f is	s stored in
	WREG is	nal WREG op specified, the the result is s	result is stor	red in WREG		
	The 'D' bi	t selects byte t selects the c s select the ac	lestination (0	for WREG, 1		
			a word opera ord operation	tion. You mag , but it is not	y use a .w ex required.	
Words:	1					
Cycles:	1					
Example 1 RRI	NC.B 0x12	33 ; Ro	tate Right	(0x1233)	(Byte mode	e)
	Before		After			
	Instruction		Instruction			
Data 1232 SF		Data 1232 SF				
Example 2 RRI	NC 0x820,	WREG ; Ro ; St	tate Right ore result		Word mode)	
WREG (W0 Data 0820	·	WREG (V Data 08		n   		
SF	R 0001 (C	C=1)	SR 0001	(C=1)		

RRNC		Rotate Right Ws without Carry					
Syntax:	{label:}	RRNC{.B}	Ws,	Wd			
			[Ws],	[Wd]			
			[Ws++],	[Wd++]			
			[Ws],	[Wd]			
			[++Ws],	[++Wd]			
			[Ws],	[Wd]			
Operands:	Ws ∈ [W0 Wd ∈ [W0						
Operation:	(Ws<0> For word (Ws<15	$1>) \rightarrow Wd<6$ $\rightarrow) \rightarrow Wd<7>$	14:0>				
Status Affected:	N, Z						
Encoding:	1101	0011	0Bqq	qddd	dppp	SSSS	
Description:	place the of Ws is s	Rotate the contents of the source register Ws one bit to the right and place the result in the destination register Wd. The Least Significant bit of Ws is stored in the Most Significant bit of Wd, and the Carry flag is not affected. Either register direct or indirect addressing may be used for Ws and Wd.					
	The 'q' bit The 'd' bit The 'p' bit	s select the c s select the a s select the s	lestination Address of the	ration (0 for v ddress mode. e destination ss mode. e source regis	register.	te).	
	Note:	rather than	a word opera	instruction de ation. You ma 1, but it is not	y use a .we	-	
Words:	1		·		·		
Cycles:	1						
Example 1	RRNC.B W0, W			t (WO) (Byt esult in W3			
	Before Instruction W0 9976 W3 5879	W W					
	SR 0001 (C			C=1)			

Example 2	RRNC	[W2++],	[W8]	; Po	tate Ri st-incr ore res	emen	t W2	mode)
	В	efore			After			
	Inst	ruction		I	nstructior	า		
	W2	2008		W2	200A			
	W8	094E		W8	094E			
Data 0	94E	3689	Data	094E	E020			

Data 2008

SR

C041

0008 (N=1)

Data 2008

SR

C041

0000

Syntax:	{label:}	SAC	Acc,	{#Slit4,}	Wd [Wd]	
					[Wd++] [Wd] [Wd] [++Wd] [Wd+Wb]	
Operands:	Acc ∈ [A,E Slit4 ∈ [-8 Wb, Wd ∈	-	5]			
Operation:	Shift <sub>Slit4</sub> (A (Acc[31:16	acc) (optiona 6]) $\rightarrow$ Wd	l)			
Status Affected:	None					
Encoding:	1100	1100	Awww	wrrr	rhhh	dddd
	or indirect The 'A' bit The 'w' bit The 'r' bits The 'h' bits	addressing specifies the s specify the s encode the s select the	ates an arithm may be used e source accu e offset registe optional accu destination Ac e destination r	for Wd. umulator. er Wb. umulator pre ddress mode		ister direc
	2:	This instruction instruction accumulate If Data Writ 1), the value	ction stores th SAC.R may b or contents. te saturation is	ne truncated be used to st s enabled (S Vd is subject	contents of A contents of A ore the round ATDW, CORC to saturation	.cc. The led CON<5>,
Words:	1					
Cycles:	1					
; St	ght shift ore resul	ACCA by				
	Before			After		
	Instructi			Instructi	on	
л <i></i> - Г		·				
W5		B900	W5	00 1005	0120	
W5 ACCA CORCON	00 120F	B900 FF00 0010	W5 ACCA CORCON	00 120F	0120 FF00 0010	

Example 2 SAC B, #-4, [W5++]
; Left shift ACCB by 4
; Store result to [W5], Post-increment W5
; CORCON = 0x0010 (SATDW = 1)

Before Instruction					I	After nstruct	
W5			2000	W5			2002
ACCB	FF	C891	8F4C	ACCB	FF	C891	1F4C
Data 2000			5BBE	Data 2000			8000
CORCON			0010	CORCON			0010
SR			0000	SR			0000

5

SAC.R		Store Ro	unded Accum	ulator		
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	,	al)			
Status Affected:	None	- <u>1</u> / / 11G				
Encoding:	1100	1101	Awww	wrrr	rhhh	dddd
	is -8:7, wh positive op (Convention Either regins The 'A' bit The 'A' bit The 'r' bits The 'h' bits	ere a negat berand indic onal or Com ster direct of specifies th s specify th s encode the s select the	ntents of AccH tive operand in cates an arithm vergent) is set or indirect add ne source accu e offset registe e optional accu destination Ace e destination r	adicates an a netic right shi by the RND ressing may imulator. er Wb. umulator pre- ddress mode	rithmetic left s ft. The Round bit, CORCON be used for V shift.	shift and ding mod N<1>.
	2:	This instru instruction accumulat If Data Wr = 1), the v	iction does not iction stores th SAC may be it for contents. ite saturation i alue stored to hift is performe	e rounded co used to store s enabled (S Wd is subject	ontents of Acoustic the truncated ATDW, COR	c. The d CON<5>
Words:	1					
Words: Cycles:	1 1					
Cycles: Example 1 SAC ; R. ; Si	1	ACCA by led result	to W5			
Cycles: Example 1 SAC ; R. ; Si	<b>1</b> .R A, #4, ight shift tore round	ACCA by led result :0010 (SAT	to W5	After		
Cycles: Example 1 SAC ; R: ; S <sup>1</sup> ; Co	1 .R A, #4, ight shift tore round DRCON = 0x	ACCA by led result :0010 (SAT	to W5 TDW = 1)	After Instructio	'n	
Cycles: Example 1 SAC ; R ; S ; C W5	1 .R A, #4, ight shift tore round DRCON = 0x Before Instructi	ACCA by led result coolo (SAT on B900	to W5 TDW = 1) W5	Instructio	0121	
Cycles: Example 1 SAC ; R: ; S <sup>1</sup> ; Co	1 .R A, #4, ight shift tore round DRCON = 0x Before Instructi	ACCA by led result :0010 (SAT on	to W5 TDW = 1)	Instruction		

Example 2	SAC.R	в, #-4,	[W5++]
		1	2 2 2 1 1 1

; Left shift ACCB by 4

; Store rounded result to [W5], Post-increment W5

; CORCON =  $0 \times 0010$  (SATDW = 1)

Before Instruction					I	Aftei nstruct	
W5			2000	W5			2002
ACCB	FF	F891	8F4C	ACCB	FF	F891	8F4C
Data 2000			5BBE	Data 2000			8919
CORCON			0010	CORCON			0010
SR			0000	SR			0000

SE		Sign-Exter	nd Ws			
Syntax:	{label:}	SE	Ws, [Ws], [Ws++], [Ws], [++Ws],	Wnd		
			[Ws],			
Operands:	Ws ∈ [W0 Wnd ∈ [W	-				
Operation:	Ws<7:0> <u>If (Ws&lt;7&gt;</u> 0xFF -	$\rightarrow$ Wnd<7:0>				
	<u>Else:</u> 0 → Wn	d<15:8>				
Status Affected:	N, Z, C	1		1		
Encoding:	1111	1011	0000	0ddd	dppp	SSSS
Description:	register di direct add	rect or indire	Ws and stor ct addressing be used for \ ag.	may be used	d for Ws, and	l registe
	The 'p' bit	s select the s	ddress of the ource Addres ddress of the	s mode.	-	
		. w extension The source	on converts a n. Ws is addres dification is b	sed as a byte		
Words:	1			-		
Cycles:	1					
Example 1 SE	W3, W4 ;	Sign-exte	end W3 and	store to W	4	
	Before		After			
W3	Instruction 7839	W	Instruction 3 7839			
W3 W4		W				
SR		SI		C=1)		
Example 2 SE [	[W2++], W12	-	extend [W2] .ncrement W		to W12	
	Before Instruction		After			
	INSTRUCTION		Instruction			
		\^/	2 0901			
W2	0900	W W1				
	0900	W W1 Data 090	2 FF8F			

# dsPIC30F Programmer's Reference Manual

SETM		Set f or WR	EG				
Syntax:	{label:}	SETM{.B}	f				
			WREG				
Operands:	f ∈ [0 81	91]					
Operation:	For byte operation: $0xFF \rightarrow$ destination designated by DFor word operation: $0xFFFF \rightarrow$ destination designated by D						
Status Affected:	None						
Encoding:	1110	1111	1BDf	ffff	ffff	ffff	
Description:	All the bits of the specified register are set to '1'. If WREG is specified the bits of WREG are set. Otherwise, the bits of the specified file register are set.						
	The 'D' bit	selects byte of selects the d select the ad	estination (0	for WREG, 1			
			word operat rd operation,	ion. You may but it is not	vuse a .wex required.		
Words:	1			0 0			
Cycles:	1						
Example 1 SETM.	3 0x891	; Set Ox	891 (Byte	mode)			
Data 0890 SR	Before nstruction 2739 0000	Data 0890 SF					
Example 2 SETM	WREG	; Set WR	EG (Word m	ode)			
lı WREG (W0) SR	Before nstruction 0900 0000	WREG (WO					

SETM		Set Ws				
Syntax:	{label:}	SETM{.B}	Wd			
			[Wd]			
			[Wd++]			
			[Wd]			
			[++Wd]			
			[Wd]			
Operands:	Wd ∈ [W0	0 W15]				
Operation:	<u>For byte o</u>	<u>operation:</u> → Wd for byte	operation			
		operation:	oporation			
	0xFFF	$F \rightarrow Wd$ for w	ord operation	1		
Status Affected:	None	-	1	I	I	I
Encoding:	1110	1011	1Bqq	qddd	d000	0000
Description:		s of the specif ddressing ma			Either registe	er direct or
	The 'q' bit	ts selects byte ts select the d ts select the a	estination Ad	dress mode		yte).
	Note:	rather than	on . B in the a word opera ord operation	tion. You ma	yusea.we	
Words:	1		· · · · · · · · · · · · · · · · · · ·			
Cycles:	1					
Example 1 SETM	.B W13	; Set W1	.3 (Byte mo	de)		
	Before		After			
	Instruction		Instruction			
W1		W1	_			
SI	R 0000	SI	<b>R</b> 0000			
Example 2 SETM	[W6]	] ; Pre-de ; Set [W		(Word mod	le)	
	Before Instruction		After Instruction			
W		W				
Data 124		Data 124	FFFF			

		Arithmetic		nulator by SI	it6	
Syntax:	{label:}	SFTAC	Acc,	#Slit6		
Operands:	Acc ∈ [A,B Slit6 ∈ [-16					
Operation:	Shift <sub>k</sub> (Acc)	$\rightarrow$ Acc				
Status Affected:	OA, OB, O	AB, SA, SB,	SAB			
Encoding:	1100	1000	A000	0000	01kk	kkkk
Description:	signed, 6-b shift range a positive c	it literal and is -16:16, wl	store the res nere a negat ates a right s	of the specified sult back into t ive operand ir shift. Any bits	the accumulandicates a lef	ator. The t shift and
				or the result. f bits to be sh	ifted.	
		CORCON< the accumu If the shift modification	7> or SATB, lator is subje amount is g	or the target a CORCON<6> ct to saturation reater than 1 hade to the	), the value on. 6 or less th	stored to an -16, n
Words:	1					
Cycles:	1					
Example 1 SFTA ; Ar ; St		to ACCA		12		
Example 1 SFTA ; Ar ; St	AC A, #12 rithmetic p core result DRCON = 0x( Before	to ACCA 0080 (SATA		After		
Example 1 SFTZ ; Ar ; St ; CC	AC A, #12 rithmetic result DRCON = 0x( Before Instructio	to ACCA 0080 (SATA n	. = 1)	After Instruction		
Example 1 SFTA ; Ar ; St ; CC ACCA	AC A, #12 rithmetic result core result DRCON = 0x0 Before Instructio 00 120F F	to ACCA 0080 (SATA n FF00	( = 1)	After Instruction	OFF	
; A1 ; St ; CC	AC A, #12 rithmetic p core result DRCON = 0x0 Before Instructio 00 120F F	to ACCA 0080 (SATA n	. = 1)	After Instruction 00 0001 2 0		
Example 1 SFTA ; Ar ; St ; CC ACCA CORCON SR Example 2 SFTA ; Ar ; St	AC A, #12 rithmetic p core result DRCON = 0x0 Before Instructio 00 120F F	to ACCA 0080 (SATA 7F00 0080 0000 	ACCA CORCON SR ACCB by 1	After Instruction	0FF 080	
Example 1 SFTA ; Ar ; St ; CC ACCA CORCON SR Example 2 SFTA ; Ar ; St	AC A, #12 rithmetic p core result DRCON = 0x0 Before Instructio 00 120F F 00 C C C C C C C C C C C C C C C C C C	to ACCA (SATA (SATA (SATA (SATA) (SATA) (SATA) (SATB) (SATB)	ACCA CORCON SR ACCB by 1	After Instruction 00 0001 2 0 0 0 After	0FF 080	
Example 1 SFTA ; Ar ; St ; CC ACCA CORCON SR Example 2 SFTA ; Ar ; St ; CC	AC A, #12 rithmetic p core result DRCON = 0x0 Before Instructio 00 120F F 00 C C C C C C C C C C C C C C C C C C	to ACCA 0080 (SATA 0080 0080 0000 0000 0000 0000 0000 0	ACCA CORCON SR ACCB by 1 = 1)	After Instruction 00 0001 2 0 0 0 After Instruction	0FF 080 000	
Example 1 SFTA ; Ar ; St ; CC ACCA CORCON SR Example 2 SFTA ; Ar ; St	AC A, #12 rithmetic p core result DRCON = 0x( Before Instruction 00 120F F ( 00 120F F) ( 00 120	to ACCA (SATA (SATA (SATA (SATA (SATA (SATB (SATB (SATB (SATB (SATB) (SATB) (SATB)	ACCA CORCON SR ACCB by 1 = 1)	After Instruction 00 0001 2 0 0 0 After	0FF 080 000	

	P	rithmetic	Shint Accu	mulator by W	D	
Syntax:	{label:} S	FTAC	Acc,	Wb		
Operands:	$Acc \in [A,B]$ $Wb \in [W0 \dots$	W15]				
Operation:	Shift <sub>(Wb)</sub> (Acc	$) \rightarrow Acc$				
Status Affected:	OA, OB, OA	3, SA, SB,	SAB			
Encoding:	1100	1000	A000	0000	0000	SSSS
Description:	store the resu Wb are used where a nega	ult back int to specify ative value	o the accurr the shift arr indicates a	of the specified nulator. The Le nount. The shif left shift and a h are shifted o	ast Significa t range is -16 a positive val	nt 6 bits c 5:16, ue
				for the source, e shift count re		
	C th <b>2:</b> If m	ORCON< the accumul the shift a odification	7> or SATB, lator is subje amount is g	for the target a CORCON<6> ect to saturatic preater than 1 nade to the r.	<ol> <li>the value and</li> <li>on.</li> <li>or less that</li> </ol>	stored to an -16, r
Words:	1					
Cycles:	1					
European and another						
; S1	AC A, W0 rithmetic sh tore result ORCON = 0x00 Before Instruction FF	to ACCA 00 (satu	-	After Instructio	n FFC	
; A: ; S: ; C: W0 ACCA	rithmetic sh tore result DRCON = 0x00 Before Instruction	to ACCA 00 (satu FC	ration di W0 ACCA	After Instructio	FFC	
; A: ; Sf ; CO WO ACCA CORCON	rithmetic sh tore result DRCON = 0x00 Before Instruction FF 00 320F AB 00	to ACCA 00 (satu FC 09 00	w0 ACCA CORCON	After Instruction F 03 20FA B 0	FFC 090 000	A.D. 1)
; A: ; Si ; Co WO ACCA CORCON SR Example 2 SFTA ; An ; St	rithmetic sh tore result DRCON = 0x00 Before Instruction FF 00 320F AB	to ACCA 00 (satu FC 09 00 00 00 ift ACCB	W0 ACCA CORCON SR by (W12)	After Instruction F 03 20FA B 0	FFC 090	AB=1)
; A: ; Si ; Cd W0 ACCA CORCON SR Example 2 SFTZ ; Ai ; St	rithmetic sh tore result DRCON = 0x00 Before Instruction FF 00 320F AB 00 00 AC B, W12 rithmetic sh core result	to ACCA 00 (satu FC 09 00 00 00 ift ACCB	W0 ACCA CORCON SR by (W12)	After Instruction F 03 20FA B 0	FFC 090 000	AB=1)
; A: ; S: ; CO W0 ACCA CORCON SR Example 2 SFT7 ; An ; St ; CO	rithmetic sh tore result ORCON = 0x00 Before Instruction FF 00 320F AB 00 00 00 AC B, W12 rithmetic sh core result ORCON = 0x00	to ACCA 00 (satu FC 09 00 00 00 ift ACCB	W0 ACCA CORCON SR by (W12) = 1)	After Instruction 03 20FA B 0 8	FFC 090 000 800 (OA, O	AB=1)
; A: ; Si ; Cd W0 ACCA CORCON SR Example 2 SFTA ; An ; St ; Cd W12	rithmetic sh tore result DRCON = 0x00 Before Instruction 00 320F AB 00 00 AC B, W12 rithmetic sh tore result DRCON = 0x00 Before Instruction 00	to ACCA 00 (satu FC 09 00 00 00 00 ift ACCB 40 (SATB	W0 ACCA CORCON SR by (W12) = 1) W12	After Instruction 03 20FA B 0 0 8 8 After Instruction	FFC 090 000 800 (OA, O	AB=1)
; A: ; Si ; Cd W0 ACCA CORCON SR Example 2 SFTZ ; A1 ; St ; Cd W12 ACCB	rithmetic sh tore result ORCON = 0x00 Before Instruction FF 00 320F AB 00 00 00 AC B, W12 rithmetic sh tore result ORCON = 0x00 Before Instruction 00 FF FFF1 8F4	to ACCA 00 (satu FC 09 00 00 00 00 00 00 00 00 00 00 00 00	W0 ACCA CORCON SR by (W12) = 1) W12 ACCB	After Instruction 03 20FA B 0 0 8 0 8 0 8 7 8 7 8 7 8 7 9 9 9 9 9 9 9 9 9 9 9 9	FFC 090 800 (OA, O	AB=1)
; A: ; Si ; Cd W0 ACCA CORCON SR Example 2 SFTA ; An ; St ; Cd W12	rithmetic sh tore result DRCON = 0x00 Before Instruction 00 320F AB 00 00 AC B, W12 rithmetic sh tore result DRCON = 0x00 Before Instruction 00	to ACCA 00 (satu FC 09 00 00 00 00 00 00 00 00 00 00 00 00	W0 ACCA CORCON SR by (W12) = 1) W12	After Instruction 03 20FA B 0 0 8 0 8 8 0 0 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	FFC 090 800 (OA, O	AB=1)

SL		Shift Left	f			
Syntax:	{label:}	SL{.B}	f	{,WREG}		
Operands:	f ∈ [0 8 <sup>.</sup>	191]				
Operation:	$0 \rightarrow De$ For word (f<15>)					
	[]◀	←0				
Status Affected:	N, Z, C	-1	1	1	1	
Encoding:	1101	0100	OBDf	ffff	ffff	ffff
Description:	result in tl register is	he destination shifted into t	e file register n register. The he Carry bit c Significant bit	e Most Signif of the Status	icant bit of th register, and	e file zero is
	WREG is	specified, the	perand detern e result is stor stored in the t	ed in WREG	-	
	The 'D' bi	t selects the	or word oper destination (0 ddress of the	for WREG, 2	-	
		rather than denote a w	ion . B in the i a word opera ord operation i is set to wor	tion. You ma , but it is not	y use a .w ex required.	
Words:	1					
Cycles:	1					
Example 1 SL.B	0x909	; Shift le	eft (0x909)	(Byte mod	le)	
ا Data 0908 SR	Before Instruction 9439 0000	Data 090 S		C=1)		
Example 2 SL	0x1650,		Shift left Store resu			è)
lı WREG (W0) Data 1650 SR	Before nstruction 0900 4065 0000	WREG (W0 Data 165 S	0 4065	l=1)		

SL		Shift Left	Ws			
Syntax:	{label:}	SL{.B}	Ws,	Wd		
			[Ws],	[Wd]		
			[Ws++],	[Wd++]		
			[Ws],	[Wd]		
			[++Ws],	[++Wd]		
			[Ws],	[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0	-				
Operation:	$0 \rightarrow W$ For word (Ws<18	$(>) \rightarrow C$ $(>) \rightarrow Wd < d < 0 >$ $(>) \rightarrow C$ $(>) \rightarrow C$ $(+) \rightarrow Wd$				
		-0				
Status Affected: Encoding:	N, Z, C	0000	0Bqq	qddd	dppp	
Description:	the result shifted int Least Sig	in the destir o the Carry nificant bit o	he source reg nation register bit of the Stat f Wd. Either r	ister Ws one I Wd. The Mos us register, an egister direct o	t Significant b d 0 is shifted	it of Ws i into the
	may be used for Ws and Wd. 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 address of the destination register. The 'p' bits select the source Address mode. The 's' bits select the address of the source register.					
	Note:	rather that	n a word oper	e instruction de ation. You may n, but it is not	yusea.we	
Words:	1					
Cycles:	1					
Example 1 SL.B	W3, W4		left W3 (By result to W			
W3	Before Instruction	١	After Instruction W3 78A9			

# dsPIC30F Programmer's Reference Manual

Example 2 SL [	W2++], [W	; Store		to [W12]
	Before		After	
I	nstruction	I	nstructior	ו
W2	0900	W2	0902	
W12	1002	W12	1002	
Data 0900	800F	Data 0900	800F	
Data 1002	6722	Data 1002	001E	
SR	0000	SR	0001	(C=1)

SL		Shift Left by	y Short Lite	ral		
Syntax:	{label:}	SL	Wb,	#lit4,	Wnd	
Operands:	Wb ∈ [W0 lit4 ∈ [0 <sup>-</sup> Wnd ∈ [W	5]				
Operation:	Wnd<15:8	→ Shift_Val Shift_Val> = W _Val-1:0> = 0	/b<15-Shift_'	Val:0>		
Status Affected:	N, Z					
Encoding:	1101	1101	0www	wddd	d100	kkkk
Description:	literal and shifted out	te contents of store the resu of the source /b and Wnd.	ult in the des	tination regis	ter Wnd. Any	/ bits
	The 'd' bit	s select the a s select the ac s provide the l	ddress of the	destination	register.	er.
	Note:	This instruct	ion operates	in Word mo	de only.	
Words:	1					
Cycles:	1					
Example 1 SL	W2, #4, W2		left W2 b result to	-		
	Before Instruction V2 78A9 GR 0000	W2 SR		l=1)		
Example 2 SL	W3, #12, W		left W3 b result to	-		
V	Before           Instruction           V3         0912           V8         1002           SR         0000	W3 W8 SF	2000			

SL		Shift Left b	y Wns			
Syntax:	{label:}	SL	Wb,	Wns,	Wnd	
Operands:	Wb ∈ [W0 Wns ∈ [W Wnd ∈ [W	0W15]				
Operation:	Wnd<15:5	$\rightarrow$ Shift_Val Shift_Val> = W Val-1:0> = 0	/b<15-Shift_	_Val:0>		
Status Affected:	N, Z					
Encoding:	1101	1101	0www	wddd	d000	SSSS
Description:	Significant the destination	ne contents of t bits of Wns ( ation register legister direct	(only up to 1 Wnd. Any b	5 positions) a its shifted ou	and store the tof the source	result in e registe
	The 'd' bit	s select the a select the a select the a select the a	ddress of the	e destination	register.	
	2:	This instruct If Wns is gre				0x0.
Words:	1					
Cycles:	1					
Example 1 SL	WO, W1, W2		left W0 l result to	by W1<0:4> o W2		
	Before		After			
	Instruction		Instruction			
WC		W				
W1 W2		W <sup>-</sup> W2				
SF		SF				
Example 2 SL	W4, W5, W6			by W5<0:4>		
		; Store	result to	0 W6		
	Before		After			
	Instruction		Instruction			
W4		W				
W5	. —	W				
W6 SF		We SF				
36	0000	51	₹ 0000			

SUB		Subtract W	/REG from t	f		
Syntax:	{label:}	SUB{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	91]				
Operation:	(f) – (WRE	G) $\rightarrow$ destination	ation designa	ated by D		
Status Affected:	DC, N, OV	Z, C				
Encoding:	1011	0101	OBDf	ffff	ffff	ffff
Description:	contents of destination destination If WREG is	e contents o the specifier register. The register. If V not specifie	d file register e optional W VREG is spe d, the result	r, and place f REG operan cified, the re is stored in t	the result in d determine sult is stored he file regist	the s the d in WREG. er.
	The 'D' bit	selects byte selects the d select the ac	estination (0	for WREG,		
		The extension rather than a denote a work The WREG	a word opera ord operation	tion. You ma , but it is not	y use a . w e required.	
Words:	1					
Cycles:	1					
Example 1 SUB.B		; Sub. WRE ; Store re		-	te mode)	
	Before		After			
-	nstruction		Instruction			
WREG (W0)	7804	WREG (W				
Data 1FFE SR	9439	Data 1FF		(NL C-1)		
34	0000	2	0009	(N, C=1)		
Example 2 SUB	0xA04, WR		. WREG fro re result		(Word mod	le)
Ir WREG (W0) Data 0A04 SR	Before nstruction 6234 4523 0000	WREG (W Data 0A	4523	n (N=1)		

SUB		Subtract L	iteral from V	Wn		
Syntax:	{label:}	SUB{.B}	#lit10,	Wn		
Operands:			te operation ord operatio	n		
Operation:	(Wn) – lit1	m O  ightarrow  m Wn				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	1011	0001	0Bkk	kkkk	kkkk	dddd
Description:	working re	gister Wn, a	nd store the	operand from result back in st be used for	the working	
	The 'k' bits	specify the	or word ope literal operar ddress of th		ister.	
		rather than denote a wo For byte op unsigned va eral Operat	a word opera ord operatior erations, the alue [0:255].	instruction de ation. You may n, but it is not literal must b See <b>Section</b> rmation on us	vuse a .wex required. e specified a <b>4.6 "Using</b> "	ktension to Is an <b>10-bit Lit-</b>
Words:	1					
Cycles:	1					
Example 1 SUB.B	#0x23, W	-	o. 0x23 fro ore result	om W0 (Byte to W0	mode)	
W0 SR	Before nstruction 7804 0000	W		N=1)		
Example 2 SUB	#0x108,	-	o. 0x108 f: pre result	rom W4 (Wor to W4	d mode)	
W4 SR	Before nstruction 6234 0000	W. SI		C=1)		

SUB		Subtract S	Short Literal	from Wb		
Syntax:	{label:}	SUB{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	(Wb) – lit5	$\rightarrow$ Wd				
Status Affected:	DC, N, OV	′, Z, C				
Encoding:	0101	0www	wBqq	qddd	d11k	kkkk
Description:	register W direct addı	b, and place	the result in the used for	the destination	the contents of the contents of the contents of the content of the	l. Register
	The 'B' bit The 'q' bits The 'd' bits	selects byte s select the of s select the a provide the The extens rather than	destination A address of the literal opera ion . B in the a word opera	ration (0 for v ddress mode e destination nd, a five-bit instruction d ation. You ma	word, 1 for by register. integer numb lenotes a byte ay use a .we:	er. e operation
		denote a w	ord operatior	n, but it is not	required.	
Words:	1					
Cycles:	1					
Example 1 SUB.E	3 W4, #0x1		Sub. 0x10 Store res		Byte mode)	
W4 W5 SR Example 2 SUB	7804	W W S , [W2++]	<b>/5</b> 7872 <b>R</b> 0005 ((	<b>OV,C=1)</b> x8 from W0 result to	(Word mode	3)
W0 W2		w	; Post-in After Instruction 0 F230	ncrement W		
Data 2004 SR		Data 200 S	04 F228 R 0009 (I	N, C=1)		

SUB		Subtract W	s from Wb			
Syntax:	{label:}	SUB{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	W15]				
Operation:	(Wb) – (Ws	s) $\rightarrow$ Wd				
Status Affected:	DC, N, OV	, Z, C				
Encoding:	0101	0www	wBqq	qddd	dppp	SSSS
Description:	Subtract the contents of the source register Ws from the contents of the base register Wb and place the result in the destination register Wd. Register direct addressing must be used for Wb. Either register direct of indirect addressing may be used for Ws and Wd. The 'w' bits select the address of the base register. The 'B' bit selects byte or word operation (0 for word, 1 for byte). The 'q' bits select the address of the destination register. The 'd' bits select the address of the destination register. The 'b' bits select the address of the destination register. The 'b' bits select the address of the destination register. The 'b' bits select the address of the source register.					er Wd. er direct or te).
	Note:	rather than a		tion. You ma	lenotes a byte ay use a .we: required.	-
Words:	1					
Cycles:	1					
Example 1 SUB.B	W0, W1,		. W1 from re result	-	mode)	
lr W0 W1 SR	Before nstruction 1732 7844 0000	WC W <sup>-</sup> SF	7844	DC, N=1)		

Example 2 SUB	W7, [W8++],	[W9++]	; Sub. [W8] from W7 (Word mode) ; Store result to [W9] ; Post-increment W8 ; Post-increment W9
	Before		After
	Instruction	I	nstruction
W7	2450	W7	2450
W8	1808	W8	180A
W9	2020	W9	2022
Data 1808	92E4	Data 1808	92E4
Data 2022	A557	Data 2022	916C
SR	0000	SR	010C (DC, N, OV=1)

SUB	Subt	ract Accumulator	S		
Syntax:	{label:} SUB	Acc			
Operands:	$Acc \in [A,B]$				
Operation:	If (Acc = A): ACCA – ACCI Else: ACCB – ACCA				
Status Affected:	OA, OB, OAB, S	A, SB, SAB			
Encoding:	1100 1	011 A011	0000	0000	0000
Description:		tents of the unspec the result back int n.			
	The 'A' bit specif	ies the destination	accumulator.		
Words:	1				
Cycles:	1				
Example 1 SUB	; Store	ract ACCB from A the result to DN = 0x0000 (no	ACCA	1)	
	Before Instruction		After Instructio	n	
ACCA	76 120F 098A	ACCA	52 1EFC 4	D73	
ACCB	23 F312 BC17	ACCB	23 F312 B	C17	
CORCON	0000	CORCON	-	000	
SR	0000	SR	1	100 <b>(OA, C</b>	)B=1)
Example 2 SUB	; Store	act ACCA from A the result to N = 0x0040 (SAT	ACCB		
	Before Instruction		After Instruction		
ACCA	FF 9022 2EE1	ACCA	FF 9022 2E		
ACCB	00 2456 8F4C	АССВ	00 7FFF FF	FF	
CORCON	0040	CORCON	00	040	

SUBB	Subtract WREG and Carry bit from f						
Syntax:	{label:} SUBB{.B} f {,WREG}						
Operands: Operation: Status Affected:	f ∈ [0 8191] (f) – (WREG) – ( $\overline{C}$ ) → destination designated by D DC, N, OV, Z, C						
Encoding:	1011 0101 1BDf ffff ffff ffff						
Description:	Subtract the contents of the default working register WREG and the Borrow flag (Carry flag inverse, $\overline{C}$ ) from the contents of the specified file register 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). The 'f' bits select the address of the file register.						
	<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> <li>3: The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.</li> </ul>						
Words:	1						
Cycles:	1						
Example 1 SUBB.	B 0x1FFF ; Sub. WREG and $\overline{C}$ from (0x1FFF) (Byte mode) ; Store result to 0x1FFF						
WREG (W0) Data 1FFE SR	9439 Data 1FFE 8F39						
Example 2 SUBB	0xA04, WREG ; Sub. WREG and $\overline{C}$ from (0xA04) (Word mode) ; Store result to WREG						
WREG (W0) Data 0A04 SR	6235 Data 0A04 6235						

SUBB	Subtract Wn from Literal with Borrow
Syntax:	{label:} SUBB{.B} #lit10, Wn
Operands:	lit10 $\in$ [0 255] for byte operation lit10 $\in$ [0 1023] for word operation Wn $\in$ [W0 W15]
Operation:	$(Wn) - Iit10 - (\overline{C}) \rightarrow Wn$
Status Affected:	DC, N, OV, Z, C
Encoding:	1011 0001 1Bkk kkkk kkkk dddd
Description:	<ul> <li>Subtract the unsigned 10-bit literal operand and the Borrow flag (Carry flag inverse, C) 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.</li> <li>The 'B' bit selects byte or word operation (0 for word, 1 for byte).</li> <li>The 'k' bits specify the literal operand.</li> <li>The 'd' bits select the address of the working register.</li> </ul>
	<ul> <li>rather than a word operation. You may use a .w extension to 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 Literal Operands" for information on using 10-bit literal operands in Byte mode.</li> <li>3: The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.</li> </ul>
Words:	1
Cycles:	1
Example 1 SUBB.	B #0x23, W0 ; Sub. 0x23 and $\overline{C}$ from W0 (Byte mode) ; Store result to W0
WC SR Example 2 SUBB	
W4 SF	BeforeAfterInstructionInstruction6234W4612C

SUBB		Subtract S	Short Liter	al from Wb	with Borrow	
Syntax:	{label:}	SUBB{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	lit5 ∈ [0	0 W15] 31] 0 W15]				
Operation:	(Wb) – lit	$5 - (\overline{C}) \rightarrow Wd$	l			
Status Affected:	DC, N, O	V, Z, C				
Encoding:	0101	lwww	wBqq	qddd	d11k	kkkk
Description:	ription: Subtract the 5-bit unsigned literal operand and the Borrow flag inverse, C) from the contents of the base register Wb result in the destination register Wd. Register direct address used for Wb. Either register direct or indirect addressing m for Wd.				register Wb an direct addressi	d place th ng must b
	The 'B' b The 'q' bi The 'd' bi The 'k' bi	ts select the c ts select the a ts provide the	or word o destination address of literal ope	peration (0 for Address mo the destination rand, a five-b	or word, 1 for b de.	ber.
	2	rather than denote a w The Z flag i	a word op ord operat s "sticky" f	eration. You r ion, but it is r	may use a . We not required. PB, SUBB and	xtension
Words:	1					
Cycles:	1					
Example 1 SUBB	.B W4, #0			LO and $\overline{C}$ free free free free free free free fre	rom W4 (Byte 5	mode)
W W SI	5 7804	W W S	5 7871	]		
Example 2 SUBB	WO, #0x8		; Store 1	$x^8$ and $\overline{C}$ for the second seco		l mode)
W W Data 200 Si	2 2004 4 A557	W W Data 200 Z=1) S	2 2006 4 0000	-		

SUBB	Subtract Ws from Wb with Borrow					
Syntax:	{label:}	SUBB{.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	$(s) - (\overline{C}) \rightarrow W$	/d			
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	0101	lwww	wBqq	qddd	dppp	SSSS
	place the addressin may be us The 'w' bit The 'B' bit The 'q' bit The 'd' bit The 'p' bit	result in the o g must be us sed for Ws an is select the a	lestination re ed for Wb. F id Wd. address of th or word ope estination A ddress of th ource Addre	egister Wd. F legister direc e base regis ration (0 for ddress mode e destinatior ss mode.	word, 1 for by e. n register.	ddressing
		rather than to denote a The Z flag is	a word operative word	ation. You m ion, but it is ADDC,CPE	denotes a byte ay use a .w not required. 3, SUBB and	extensior
Words:	1					
Cycles: Example 1 SUBB.F	1 3 WO, W1,		b. W1 and ore resul		) (Byte mode	e)
W0 W1 SR	Before nstruction 1732 7844 0000	Wi W SF	After Instruction 0 17ED 1 7844	DC, N=1)		

Example 2 SUBB	W7, [W8++], [N	; S ; P	tore res ost-inci	and C from W7 sult to [W9] rement W8 rement W9	(Word mode)
	Before		After		
I	nstruction	I	nstruction		
W7	2450	W7	2450		
W8	1808	W8	180A		
W9	2022	W9	2024		
Data 1808	92E4	Data 1808	92E4		
Data 2022	A557	Data 2022	916C		
SR	0000	SR	010C	(DC, N, OV=1)	

SUBBR		Subtract f f	rom WREG	i with Borro	w	
Syntax:	{label:}	SUBBR{.B}	f	{,WREG}		
Operands: Operation:	f ∈ [0 8 (WREG) -	191] · (f) – ( $\overline{C}$ ) → de	estination de	esignated by	D	
Status Affected:	DC, N, O\	/, Z, C				
Encoding:	1011	1101	1BDf	ffff	ffff	ffff
Description:	(Carry flag in the desi destination	ne contents of g inverse, C) fr tination registen n register. If W s not specified	rom the cont er. The optio /REG is spe	tents of WRE mal WREG o cified, the re	G, and place perand dete sult is stored	e the result rmines the I in WREG.
	The 'D' bit	selects byte of selects the de select the ad	estination (0	for WREG,		
	2:	The extension rather than a denote a wo The WREG is The Z flag is These instru	word opera rd operation s set to wor "sticky" for	tion. You ma , but it is not king register ADDC, CPB	y use a .w e required. W0.	extension to
Words:	1					
Cycles:	1					
Example 1 SUBBR	.B 0x803	; Sub. (0x8 ; Store res			G (Byte mo	de)
l WREG (W0) Data 0802 SR	Before nstruction 7804 9439 0002 (Z	WREG (W Data 080 =1) S	-			
Example 2 SUBBR	0xA04, WF	EG ; Sub. ; Store	(0xA04) ar result to		WREG (Word	mode)
l WREG (W0) Data 0A04 SR	Before nstruction 6234 6235 0000	WREG (M Data 0A S	04 6235	]		

SUBBR		Subtract W	b from Sho	ort Literal wit	th Borrow	
Syntax:	{label:}	SUBBR{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	31]				
Operation:	-	$-(\overline{C}) \rightarrow Wd$				
Status Affected:	DC, N, O					
Encoding:	0001	1www	wBqq	qddd	d11k	kkkk
Description:	flag invers destinatio	ne <u>contents</u> of se, C) from the n register Wd. ster direct or ir	e 5-bit unsig Register di	ned literal an rect address	d place the re ing must be u	esult in the
	The 'B' bil The 'q' bit The 'd' bit	s select the a selects byte s select the de s select the ad s provide the l	or word ope estination Ad ddress of the	ration (0 for v ddress mode e destination	word, 1 for by register.	
			a word opera rd operatior s "sticky" for	ation. You ma n, but it is not ADDC,CPB	y use a .wex required.	ctension t
Words:	1					
Cycles:	1					
Example 1 SUBB	R.B W0, #0	)x10, W1 ; ;	Sub. W0 an Store rest		0x10 (Byte	mode)
W W SF	<b>1</b> 786A	W0 W1 , C=1) SF	7800	DC, Z, C=1)		
Example 2 SUBB	R W0, #0x8	-		sult to [W		mode)
W W Data 200 Si	2 2004 4 A557	W0 W2 Data 2004 =1) SF	2 2006 FFFE	DC, N=1)		

SUBBR	Subtract Wb from Ws with Borrow					
Syntax:	{label:}	SUBBR{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	) W15]				
Operation:	(Ws) – (W	$'$ b) − ( $\overline{C}$ ) → We	d			
Status Affected:	DC, N, O	/, Z, C				
Encoding:	0001	lwww	wBqq	qddd	dppp	SSSS
	be used fo Ws and W The 'w' bit The 'B' bit The 'q' bit The 'd' bit The 'p' bit	in the destinat or Wb. Registe /d. ts select the ac s selects byte of s select the de s select the ac s select the ac s select the ac	er direct or in ddress of the pr word oper estination Ad ddress of the purce Addre	ndirect addre e base regis ration (0 for ddress mode e destination ss mode.	essing may be ster. word, 1 for by e. n register.	e used for
		The extension rather than a denote a wo The Z flag is These instru	word operation rd operation "sticky" for	ition. You ma , but it is no ADDC, CPB	ay use a .wex t required.	ctension to
Words:	1					
Cycles:	1					
Example 1 SUBB	R.B WO, WI		ub. W0 and core resul		1 (Byte moo	le)
W W SI	1 7844	W0 W1 SR	7844	C=1)		

ł	

Example 2 SUBBR	W7, [W8++],[	;	Store rea	and C from [W8] sult to [W9] rement W8 rement W9	(Word mode)
I	Before nstruction		After Instruction	ı	
W7	2450	V	<b>17</b> 2450		
W8	1808	V	<b>/8</b> 180A		
W9	2022	V	/9 2024		
Data 1808	92E4	Data 180	<b>)8</b> 92E4		
Data 2022	A557	Data 202	2 <b>2</b> 6E93		
SR	0000	S	<b>R</b> 0005	(OV, C=1)	

SUBR		Subtract f	from WREC	à		
Syntax:	{label:}	SUBR{.B}	f	{,WREG}		
Operands:	f ∈ [0 81	911				
Operation:	-	(f) $\rightarrow$ destina	tion design:	ated by D		
Status Affected:	DC, N, OV,		debigin			
Encoding:	1011	1101	0BDf	ffff	ffff	ffff
Description:	the default destination destination	e contents of working regis register. The register. If W not specified	ster WREG, e optional W /REG is spe	and place th REG operan cified, the re	e result in th d determine: sult is stored	ne s the d in WREG
	The 'D' bit	selects byte o selects the d select the ad	estination (	for WREG,		
		The extension rather than a denote a wo The WREG	word opera	tion. You ma ı, but it is not	y use a . w e required.	•
Words:	1					
Cycles:	1					
Example 1 SUBR.B	0x1FFF		0x1FFF) fi result to	com WREG () 0x1FFF	Byte mode)	
	Before		After			
Ir	struction		Instruction			
WREG (W0)	7804	WREG (W0)				
Data 1FFE SR	9439 0000	Data 1FFE SF				
Example 2 SUBR	0xA04, WRI	-	(0xA04) : e result (	from WREG to WREG	(Word mode	e)
lr WREG (W0) Data 0A04 SR	Before Instruction 6234 6235 0000	WREG (W0) Data 0A04 SR	6235	N=1)		

SUBR		Subtract V	Vb from Sh	ort Literal		
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, O	V, Z, C				
Encoding:	0001	0www	wBqq	qddd	d11k	kkkk
Description:	literal ope Register o	erand, and pla	ace the resu sing must b	It in the dest e used for WI	rom the unsig ination registe o. Either regist	r Wd.
	The 'B' bi The 'q' bi The 'd' bi	ts select the c ts select the a	or word op lestination address of t	eration (0 for Address moc he destinatio	r word, 1 for by le.	
	Note:	rather than	a word ope		denotes a byten a .w e .w e .w e	
Words:	1			,	·	
Cycles:	1					
Example 1 SUBR	.B W0, #0			from 0x10 sult to W1	(Byte mode)	
Wi W SF	<b>1</b> 786A	W W S	1 7800	n (DC, Z, C=1	)	
Example 2 SUBR	W0, #0x	8, [W2++]	; Store	0 from 0x8 result to ncrement W		)
Wi Wi Data 2004		W		1		

SUBR		Subtract W	b from Ws			
Syntax:	{label:}	SUBR{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	$ \begin{array}{l} Wb \in \ [W0 \\ Ws \in \ [W0 \\ Wd \in \ [W0 \end{array} \end{array} $	W15]				
Operation:	(Ws) – (Wt	$(b) \rightarrow Wd$				
Status Affected:	DC, N, OV	Z, C				
Encoding:	0001	0www	wBqq	qddd	dppp	SSSS
	Register di indirect add The 'w' bits The 'B' bit The 'q' bits The 'd' bits The 'p' bits The 's' bits	rect addressi dressing may select the a selects byte of select the do select the ad select the ad	ng must be be used for ddress of the or word oper estination Ac ddress of the burce Address ddress of the	used for Wb. YWs and Wd e base regist ration (0 for v ddress mode e destination ss mode. e source regi	er. word, 1 for by register. ster.	er direct c
	Note:	rather than a	a word opera		enotes a byte ay use a .w e required.	
Words:	1					
Cycles:	1					
Example 1 SUBR.B	WO, W1,		b. W0 from ore result	n W1 (Byte to W0	mode)	
	Before struction 1732 7844 0000	WC W1 SF	7844	C=1)		

Example 2 SUBR	W7, [W8++],	, [W9++]	; Sub. W7 from [W8] (Word mode) ; Store result to [W9] ; Post-increment W8 ; Post-increment W9
	Before		After
I	nstruction		nstruction
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 Ni	bble Swap V	Vn		
Syntax:	{label:}	SWAP{.B}	Wn			
Operands:	Wn∈[	W0 W15]				
Operation:	(Wn <u>For wo</u>	<u>te operation:</u> )<7:4> ↔ (Wn)< <u>rd operation:</u> )<15:8> ↔ (Wn)				
Status Affecte	d: None					
Encoding:	111	1 1101	1B00	0000	0000	SSSS
	Signific Wn is u	of Wn are swapp cant Byte of Wn unchanged. Reg	are swapped ister direct ac	, and the Mc ddressing mi	est Significan ust be used f	t Byte of for Wn.
		bits select the a	•	•		<i>j</i> ,
	Note	rather than	on . B in the a word opera ord operation	tion. You ma	ay use a .we	
Words:	1					
Cycles:	1					
Example 1	SWAP.B WO	; Nibble sw	ap (W0)			
	Before Instruction W0 AB87 SR 0000	n   Wi   SF	-			
Example 2	SWAP WO	; Byte swap	(WO)			
	Before Instruction W0 8095 SR 0000	n   Wi   SF				

TBLRDH		Table Read H	ligh						
Syntax:	{label:}	TBLRDH{.B}	[Ws],	Wd					
			[Ws++],	[Wd]					
			[Ws],	[Wd++]					
			[++Ws],	[Wd]					
			[Ws],	[++Wd]					
				[Wd]					
Operands:	$Ws \in [W0]$ $Wd \in [W0]$								
Operation:	For byte op If (LSB(V 0 → V Else Progra For word o	Ws) = 1) Vd am Mem [(TBL	.PAG),(Ws)]	<23:16> → W	d				
		Mem [(TBLPA	AG),(Ws)] <2	$23:16> \rightarrow Wd \cdot$	<7:0>				
Status Affected:	None								
Encoding:	1011	1010	1Bqq	qddd	dppp	SSSS			
	store it to the destination register Wd. The target word address of progra memory is formed by concatenating the 8-bit Table Pointer register, TBLPAG<7:0>, with the effective address specified by Ws. Indirect addressing must be used for Ws, and either register direct or indirect addressing may be used for Wd.								
	In Word mode, zero is stored to the Most Significant Byte of the destination register (due to non-existent program memory) and the third program memory byte (PM<23:16>) at the specified program memory address is stored to the Least Significant Byte of the destination register.								
	register (du memory by	ue to non-existe te (PM<23:16:	ent program >) at the spe	memory) and cified program	n memory ac	ogram			
	register (du memory by stored to th In Byte mo not word al non-exister memory by	ue to non-existe te (PM<23:16:	ent program >) at the spe icant Byte of address dep stored to the mory). If Ws >) at the spe	memory) and cified program the destination pends on the c destination re is word aligne	n memory ac on register. ontents of W egister (due ed, the third	ogram Idress is Vs. If Ws to program			
	register (du memory by stored to the In Byte mo- not word al non-exister memory by stored to the The 'B' bits The 'q' bits The 'd' bits The 'p' bits	te to non-exist te (PM<23:16: ne Least Signifi de, the source ligned, zero is nt program me te (PM<23:16:	ent program >) at the spe icant Byte of address dep stored to the mory). If Ws >) at the spe register. word operate stination Add iress of the cource Address	memory) and cified program the destination e destination re- is word aligned cified program tion (0 for wor ress mode. destination (da mode.	n memory ac on register. ontents of W egister (due ed, the third n memory ac d, 1 for byte ata) register.	ogram Idress is Vs. If Ws to program Idress is			
	register (du memory by stored to the In Byte mo- not word al non-exister memory by stored to the The 'B' bits The 'q' bits The 'd' bits The 'p' bits	te to non-existence (PM<23:16) te (PM<23:16) te Least Signifience igned, zero is not program me te (PM<23:16) te destination of selects byte or select the des select the des select the add select the source	ent program >) at the spe icant Byte of address dep stored to the mory). If Ws >) at the spe register. word operation tination Add tress of the construction tress of the so n . B in the in move. You m	memory) and cified program the destination dends on the contract destination re- is word aligned cified program tion (0 for wor ress mode. destination (da source (address struction dence hay use a .w	a memory ac on register. ontents of W egister (due ed, the third a memory ac d, 1 for byte tta) register. ss) register. otes a byte n	ogram Idress is Vs. If Ws to program Idress is ).			
Words:	register (du memory by stored to the In Byte mo- not word al non-exister memory by stored to the The 'B' bits The 'G' bits The 'G' bits The 's' bits	te to non-exist te (PM<23:16: te Least Signified, the source ligned, zero is the program me te (PM<23:16: te destination is selects byte or select the des select the add select the add select the add The extension than a word r	ent program >) at the spe icant Byte of address dep stored to the mory). If Ws >) at the spe register. word operation tination Add tress of the construction tress of the so n . B in the in move. You m	memory) and cified program the destination dends on the contract destination re- is word aligned cified program tion (0 for wor ress mode. destination (da source (address struction dence hay use a .w	a memory ac on register. ontents of W egister (due ed, the third a memory ac d, 1 for byte tta) register. ss) register. otes a byte n	ogram Idress is Vs. If Ws to program Idress is ).			

Example 1 TBL	RDH.B [WO]		PM (TBLPAG:[ to [W1] increment W1	· · · · ·
	Before		After	
	Instruction		Instruction	
W0	0812	W0	0812	
W1	0F71	W1	0F72	
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 TBL	RDH [W6++]	; Store t		]) (Word mode)
	Before		After	
	Instruction		Instruction	
W6	3406	W6	3408	
W8	65B1	W8	0029	
Program 00 3406	29 2E40	Program 00 3406	29 2E40	
TBLPAG	0000	TBLPAG	0000	
SR	0000	SR	0000	

5

Instruction Descriptions

TBLRDL		Table Read L	.ow			
Syntax:	{label:}	TBLRDL{.B}	[Ws],	Wd		
			[Ws++],	[Wd]		
			[Ws],	[Wd++]		
			[++Ws],	[Wd]		
			[Ws],	[++Wd]		
				[Wd]		
Operands:	Ws ∈ [W0 Wd ∈ [W0					
Operation:	Else Progra For word o	Vs) = 1) am Mem [(TBl am Mem [(TBl	LPAG),(Ws)]	$<7:0> \rightarrow Wd$	d	
Status Affected:	None					
Encoding:	1011	1010	0Bqq	qddd	dppp	SSSS
Description:	store it to th memory is TBLPAG<7 addressing	ontents of the ne destination formed by con ':0>, with the of must be used may be used	register Wd. ncatenating t effective add d for Ws, and	The target w he 8-bit Table ress specified	ord address of Pointer regis	of program ster, ect
	destination contents of memory wo word aligne	ode, the lower register. In By Ws. If Ws is i ord (PM<15:7: ed, the first by ie destination	yte mode, the not word align >) is stored to te of the prog	e source addr ned, the seco o the destinat	ess depends and byte of the ion register. If	on the e program <sup>f</sup> Ws is
	The 'q' bits The 'd' bits The 'p' bits	selects byte o select the de select the ad select the sol select the add	stination Add dress of the o urce Address	ress mode. destination (d mode.	ata) register.	r byte).
	Note:	The extension than a word				
		word move, b	out it is not re	quired.		
Words:	1	word move, b	out it is not re	quired.		

# **Section 5. Instruction Descriptions**

Example 1 TBL	RDL.B [W0+	+], W1 ; Read PM ; Store t ; Post-in		10]) (Byte mode)
	Before		After	
	Instruction		Instruction	
WO	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 TBL	RDL [W6],	; Store	PM (TBLPAG: to W8 increment W	:[W6]) (Word mode) N8
	Before		After	
	Instruction		Instruction	
W6	3406	W6	3408	
W8	1202	W8	1204	
Data 1202	658B	Data 1202	2E40	
Program 00 3406	29 2E40	Program 00 3406	29 2E40	
TBLPAG	0000	TBLPAG	0000	
SR	0000	SR	0000	

	Table Write H	ligh			
{label:}	TBLWTH{.B}	Ws,	[Wd]		
		[Ws],	[Wd++]		
		[Ws++],	[Wd]		
		[Ws],	[++Wd]		
		[++Ws],	[Wd]		
		[Ws],			
If (LSB(					
Else					
		em [(TBLPAC	G),(Wd)]<23:1	6>	
		Mem [(TBL	PAG) (Wd)] <2	23.16>	
None			///////////////////////////////////////	_0.10>	
1011	1011	1Bqq	qddd	dppp	SSSS
Significant program m register, TI direct or in	Word of progra nemory is forme BLPAG<7:0>, v idirect addressi	am memory. ed by concat vith the effec	The destination of the destination of the destination of the second seco	on word addr bit Table Poir specified by V	ess of nter Vd. Eithe
the upper using a We	byte of progran d that is word a	n memory (P ligned in Byt	M<23:16>). T e mode or Wo	his may be p ord mode. If B	erforme yte mod
The 'q' bits The 'd' bits The 'p' bits	s select the des s select the add s select the sou	tination Add lress of the c rce Address	ress mode. lestination (ac mode.	ddress) regist	
Note:	The extension than a word r word move, b	nove. You m	nay use a .w		
Note:		nove. You m	nay use a .w		
	$Ws \in [W0]$ $Wd \in [W0]$ For byte o If (LSB( NOP Else (Ws)) For word c (Ws)<7: None 1011 Store the c Significant program m register, T direct or in must be us Since prog the upper using a Wa is used wit The 'B' bit The 'd' bits The 'p' bits	{label:} TBLWTH{.B} Ws $\in$ [W0 W15] Wd $\in$ [W0 W15] For byte operation: If (LSB(Wd) = 1) NOP Else (Ws) $\rightarrow$ Program Me For word operation: (Ws)<7:0> $\rightarrow$ Program Me 1011 1011 Store the contents of the Significant Word of program register, TBLPAG<7:0>, v direct or indirect addressi must be used for Wd. Since program memory is the upper byte of program using a Wd that is word a is used with a Wd that is in The 'B' bit selects byte or The 'q' bits select the des The 'd' bits select the add The 'p' bits select the sources	[Ws], [Ws++], [Ws], [++Ws], [Ws], [Ws]	{label:} TBLWTH{.B} Ws, [Wd] [Ws], [Wd++] [Ws++], [Wd] [Ws], [++Wd] [++Ws], [Wd] [Ws], Ws $\in$ [W0 W15] Wd $\in$ [W0 W15] For byte operation: If (LSB(Wd) = 1) NOP Else (Ws) $\rightarrow$ Program Mem [(TBLPAG),(Wd)]<23:1 For word operation: (Ws)<7:0> $\rightarrow$ Program Mem [(TBLPAG),(Wd)]<23:1 None 1011 1011 1Bqq qddd Store the contents of the working source register W Significant Word of program memory. The destinati program memory is formed by concatenating the 8- register, TBLPAG<7:0>, with the effective address a direct or indirect addressing may be used for Ws, a must be used for Wd. Since program memory is 24-bits wide, this instruct the upper byte of program memory (PM<23:16>). T using a Wd that is word aligned in Byte mode or Wc is used with a Wd that is not word aligned, no opera The 'B' bit selects byte or word operation (0 for word The 'g' bits select the address of the destination (address mode. The 'd' bits select the address of the destination (address mode. The 'g' bits select the address of the destination (address mode. The 'g' bits select the source Address mode.	{label:} TBLWTH{.B} Ws, [Wd] [Ws], [Wd++] [Ws], [++Wd] [++Ws], [Wd] [Ws], Ws $\in$ [W0 W15] Wd $\in$ [W0 W15] For byte operation: If (LSB(Wd) = 1) NOP Else (Ws) $\rightarrow$ Program Mem [(TBLPAG),(Wd)]<23:16> For word operation: (Ws)<7:0> $\rightarrow$ Program Mem [(TBLPAG),(Wd)]<23:16> None 1011 1011 1Bqq qdd dppp Store the contents of the working source register Ws to the Most Significant Word of program memory. The destination word addr program memory is formed by concatenating the 8-bit Table Poin register, TBLPAG<7:0>, with the effective address specified by V direct or indirect addressing may be used for Ws, and indirect addressing may be used for Ws, and indirect address ing may be used for Ws, and indirect addressing may be used for Wd. Since program memory is 24-bits wide, this instruction can only the upper byte of program memory (PM<23:16>). This may be pusing a Wd that is word aligned in Byte mode or Word mode. If B is used with a Wd that is not word aligned, no operation is perfor The 'B' bit selects byte or word operation (0 for word, 1 for byte) The 'd' bits select the destination Address mode. The 'd' bits select the address of the destination (address) register

Example 1 TBL	WTH.B [W0+	; to PM	[W0] (I Latch Hig] increment V	h (TBLPAG:[W1])
	Before		After	
	Instruction		Instruction	
W0	0812	WO	0812	
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	are not u	0	SH memory	ntents of program memory is programmed using the Reference Manual.
Example 2 TBL	NTH W6, [	; to PM L	6 (Word atch High crement W8	mode) (TBLPAG:[W8])
	Before		After	
	Instruction		Instruction	
W6	0026	W6	0026	
W8	0870	W8	0872	
Program 00 0870	22 3551	Program 00 0870	26 3551	
TBLPAG	0000	TBLPAG	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 dsPIC30F Family Reference Manual.

0000

SR

TBLWTL	ı	Table Write L	-000			
Syntax:	{label:}	TBLWTL{.B}	Ws,	[Wd]		
			[Ws],	[Wd++]		
			[Ws++],	[Wd]		
			[Ws],	[++Wd]		
			[++Ws],	[Wd]		
			[Ws],			
Operands:	Ws ∈ [W0 . Wd ∈ [W0 .					
Operation:	For byte op If (LSB(V					
			em [(TBLPA	G),(Wd)] <15:8	}>	
	Else	-				
	- (Ws) <u>For word o</u> r	•	em [(TBLPA	G),(Wd)] <7:0>	•	
		Program Mem	[(TBLPAG),	(Wd)] <15:0>		
Status Affected:	None					
Encoding:	1011	1011	0Bqq	qddd	dppp	SSSS
Encoding:	1011 Store the co Word of pro memory is TBLPAG<7	Dontents of the v ogram memory formed by con ':0>, with the e dressing may b	working sour y. The destin catenating t effective add	qddd rce register Ws ation word ad he 8-bit Table ress specified Ws, and indired	to the Least dress of progr Pointer regist by Wd. Eithe	Significat ram er, r direct o
	1011 Store the co Word of pro memory is TBLPAG<7 indirect ado used for Wo In Word mo Byte mode, If Wd is not memory (P	ontents of the y ogram memory formed by con (:0>, with the e dressing may b d. ode, Ws is stor , the Least Sig word aligned	working sour y. The destin icatenating t effective add be used for N red to the low nificant bit o , Ws is store Vd is word a	ce register Ws ation word add he 8-bit Table ress specified	to the Least dress of prog Pointer regist by Wd. Eithe ct addressing program men es the destina d byte of prog	Significa ram er, r direct o must be nory. In ation byto gram
Encoding:	1011 Store the co Word of pro memory is TBLPAG<7 indirect add used for We In Word mo Byte mode, If Wd is not memory (P program me The 'B' bit s The 'q' bits The 'd' bits The 'p' bits	ontents of the y ogram memory formed by con (0>, with the e dressing may b d. ode, Ws is stor the Least Sig word aligned M<15:8>). If V emory (PM<7: selects byte or select the des select the ado select the sou	working sour working sour y. The destin icatenating t iffective add be used for N red to the low nificant bit o , Ws is store Vd is word al 0>). word opera stination Add dress of the ource Address	rce register Ws ation word add he 8-bit Table ress specified Ws, and indired wer 2 bytes of f Wd determin d to the secon ligned, Ws is s tion (0 for word lress mode. destination (ac	to the Least of program register address of program menes the destinated byte of program to the fill of the fill o	Significat ram er, r direct o must be nory. In ation byte gram rst byte o
Encoding:	1011 Store the co Word of pro memory is TBLPAG<7 indirect add used for We In Word mo Byte mode, If Wd is not memory (P program me The 'B' bits The 'd' bits The 'd' bits The 's' bits Note:	pontents of the p ogram memory formed by con (0>, with the e dressing may b d. ode, Ws is stor the Least Sig word aligned M<15:8>). If V emory (PM<7: selects byte or select the des select the des select the ado select the ado	working sour working sour y. The destin icatenating t iffective add be used for N red to the low nificant bit o , Ws is store Vd is word al 0>). word opera stination Add tress of the o urce Address tress of the s n . B in the i love. You ma	rce register Ws ation word add he 8-bit Table ress specified Ws, and indired wer 2 bytes of f Wd determin d to the secon ligned, Ws is s tion (0 for word ress mode. destination (ad s mode. source (data) r nstruction den ay use a . W ext	to the Least of the constraints	Significat ram er, r direct o must be nory. In ation byte gram rst byte o er.
Encoding:	1011 Store the co Word of pro memory is TBLPAG<7 indirect add used for We In Word mo Byte mode, If Wd is not memory (P program me The 'B' bits The 'd' bits The 'd' bits The 's' bits Note:	pontents of the p ogram memory formed by con (0>, with the e dressing may b d. ode, Ws is stor the Least Sig word aligned M<15:8>). If V emory (PM<7: selects byte or select the des select the des select the ado select the ado select the ado the extension than a word m	working sour working sour y. The destin icatenating t iffective add be used for N red to the low nificant bit o , Ws is store Vd is word al 0>). word opera stination Add tress of the o urce Address tress of the s n . B in the i love. You ma	rce register Ws ation word add he 8-bit Table ress specified Ws, and indired wer 2 bytes of f Wd determin d to the secon ligned, Ws is s tion (0 for word ress mode. destination (ad s mode. source (data) r nstruction den ay use a . W ext	to the Least of the constraints	Significat ram er, r direct o must be nory. In ation byte gram rst byte o er.

Example 1 TBI	WTL.B W0,			mode) IBLPAG:[W1])
	Before		After	
	Instruction		Instruction	
WO	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	are not u procedure	<pre>(pdated until the FLA e described in the dsP [W8] ; Write [ ; to PM I</pre>	SH memory IC30F Family	
	Before		After	
	Instruction		Instruction	1
W6		W6	1600	
W8		W8	7208	
Data 1600		Data 1600	0130	
Program 01 7208		Program 01 7208	09 0130	
TBLPAG	0001	TBLPAG	0001	

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 dsPIC30F Family Reference Manual.

0000

SR

ULNK		De-allocate	Stack Fram	ne		
Syntax:	{label:}	ULNK				
Operands:	None					
Operation:	$W14 \rightarrow W$ (W15)-2 – (TOS) $\rightarrow V$	→ W15				
Status Affected:	None					
Encoding:	1111	1010	1000	0000	0000	0000
Description:	sequence. (W15) equ	ction de-alloo The stack fra al to the fram rame pointer	ame is de-all ne pointer (W	ocated by se	etting the sta	ck pointer
Words:	1					
Cycles:	1					
	; Unlir Before struction 2002 20A2 2000 0000	uk the stac W14 W18 Data 2000 SF	After Instruction 4 2000 5 2000 0 2000			
Example 2 ULNK	; Unlir	nk the stac	k frame			
	Before		After			
In W14 [	struction 0802	W14	Instruction			
W14 W15	0802	W15				
Data 0800	0800	Data 0800				
SR	0000	SF	<b>R</b> 0000			
_						

# **Section 5. Instruction Descriptions**

XOR		Exclusive (	OR f and Wi	REG		
Syntax:	{label:}	XOR{.B}	f	{,WREG}		
Operands:	f ∈ [0 8	1911				
Operation:	-	VREG) $\rightarrow$ des	stination des	ignated by D		
Status Affected:	N, Z	,		.g		
Encoding:	1011	0110	1BDf	ffff	ffff	ffff
Description:	default wo register ar WREG op specified,	the logical exp orking register and place the re- perand determ the result is s tored in the fill	WREG and esult in the c ines the des tored in WR	the contents lestination regis	of the specif gister. The op ter. If WREG	ied file otional is
	The 'D' bit	t selects byte t selects the d s select the ac	estination (0	for WREG, 1		
			a word opera ord operation	tion. You may , but it is not	vuse a .wex required.	
Words:	1					
Cycles:	1					
Example 1 XOR.B	0x1FFF		x1FFF) and result to	l WREG (Byt 0x1FFF	e mode)	
	Before		After			
Ir WREG (W0)	nstruction 7804	WREG (W0	Instruction			
Data 1FFE	9439	Data 1FFE				
SR	0000	SF		N=1)		
Example 2 XOR	0xA04, V		(0xA04) a re result	and WREG (W to WREG	ord mode)	
lr WREG (W0) Data 0A04 SR	Before nstruction 6234 A053 0000	WREG (W0 Data 0A04 SR	A053	<b>√</b> =1)		

XOR		Exclusive	OR Literal a	nd Wn		
Syntax:	{label:}	XOR{.B}	#lit10,	Wn		
Operands:		1023] for v	rte operation word operatio	n		
Operation:	lit10.XOR	.(Wn) $\rightarrow$ Wn				
Status Affected:	N, Z					
Encoding:	1011	0010	1Bkk	kkkk	kkkk	dddd
Description:	operand a	and the conte e working re	nts of the wo	peration of the rking register gister direct a	Wn and store	e the resu
	The 'k' bit	s specify the	literal operar	ration (0 for w nd. e working reg	-	te).
Worda		unsigned v eral Oper	alue [0:255].	ne literal mu See <b>Section</b> Information c	4.6 "Using	10-bit Li
Words: Cycles:	1 1					
Example 1 XOR.		-	OR 0x23 and tore result	d W0 (Byte to W0	mode)	
W			After Instruction /0 7827 R 0000			
Example 2 XOR	#0x108, W		OR 0x108 an tore result	nd W4 (Word t to W4	l mode)	
W. SI			After Instruction /4 603C R 0000			

XOR		Exclusive OR Wb and Short Literal						
Syntax:	{label:}	XOR{.B}	Wb,	#lit5,	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]			
Operands:	Wb ∈ [W0 lit5 ∈ [0 Wd ∈ [W0	. 31]						
Operation:	(Wb).XOF	R.lit5 $\rightarrow$ Wd						
Status Affected:	N, Z							
Encoding:	0110	lwww	wBqq	qddd	d11k	kkkk		
Description:	register W the destin	/b and the ur ation registe	nsigned 5-bit r Wd. Regist	literal opera er direct add	the contents ond and place the ressing must be used in the rest of	he result i be used fo		
	The 'B' bi The 'q' bit The 'd' bit	is select the as select the as provide the The extens rather than	e or word ope destination A address of th e literal opera sion . B in the a word oper	eration (0 for address mod ne destination and, a 5-bit ir e instruction ation. You m	word, 1 for by e. n register. hteger number denotes a byte ay use a .we	e operatic		
Marda	4	denote a w	ord operatio	n, but it is no	t required.			
Words: Cycles:	1							
Oycles.	I							
Example 1	KOR.B W4, #0	Ox16, W5		4 and 0x14 result to	4 (Byte mode 5 W5	e)		
V	Before           Instruction           V4         C822           V5         1200           SR         0000	١	After Instruction N4 C822 N5 1234 SR 0000	n     				
Example 2	KOR W2, #0	Ox1F, [W8+	; Stor	W2 by 0x11 e result f -increment		e)		
V	Before           Instruction           V2         8505           V8         1004           04         6628		After Instruction N2 8505 N8 1006					

XOR	Exclusive OR Wb and Ws					
Syntax:	{label:}	XOR{.B}	Wb,	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wd [Wd] [Wd++] [Wd] [++Wd] [Wd]	
Operands:	Wb ∈ [W0 Ws ∈ [W0 Wd ∈ [W0	) W15]				
Operation:	-	R.(Ws) → Wd				
Status Affected:	N, Z					
Encoding:	0110	lwww	wBqq	qddd	dppp	SSSS
	The 'B' bi The 'q' bit The 'd' bit The 'p' bit	ts select the a	or word ope destination A address of the source Addre	ration (0 for ddress mode e destinatior ss mode.	word, 1 for by e. ı register.	rte).
	Note:	rather than		ation. You m	denotes a byt ay use a .we t required.	
Words:	1					
Cycles:	1					
Example 1 XOF	R.B W1, [T	N5++], [W9-	; Sto:	W1 and [W re result t-incremen		
W1 W5 W9 Data 2000 Data 2600 SR	2000 2600 115A 0000	V V Data 20 Data 26		n (N=1)		

Example 2	XOR	W1, W5	, W9				(Word mode) 1lt to W9
		Before			After		
	In	struction		In	struction		
	W1	FEDC		W1	FEDC		
	W5	1234		W5	1234		
	W9	A34D		W9	ECE8		
	SR	0000		SR	0008	(N=1)	

# dsPIC30F Programmer's Reference Manual

ZE		Zero-Exten	d Wn			
Syntax:	{label:}	ZE	Ws, [Ws], [Ws++], [Ws], [++Ws], [Ws],	Wnd		
Operands:	Ws ∈ [W0 Wnd ∈ [W	) W15] /0 W15]				
Operation:	-	$\rightarrow$ Wnd<7:0>				
Status Affected:	N, Z, C					
Encoding:	1111	1011	10qq	qddd	dppp	SSSS
	a 16-bit v. Wnd. Eith and regis cleared au positive.	nd the Least s alue and store er register dir ter direct addr nd the C flag i	e the result in ect or indirec essing must s set, becaus	the destinati t addressing be used for V e the zero-ex	on working re may be used Vnd. The N fl	egister I for Ws, lag is
	The 'd' bit The 'p' bit	is select the d is select the a is select the s is select the a	ddress of the ource Addres	destination r s mode.	-	
		This operati .wextensio The source address mo	n.	essed as a		
Words:	1					
Cycles:	1					
Example 1 ZE	W3, W4	; zero-exte ; Store res	nd W3 ult to W4			
	Before		After			
14	Instruction /3 7839	W	Instruction 3 7839			
	<b>14</b> 1005	W W				
S	<b>R</b> 0000	SF	R 0001 (C	5=1)		
Example 2 ZE	[W2++], W1:	; Store	xtend [W2] to W12 ncrement W	2		
W1 Data 090		W2 W12 Data 0900 SF	2 008F D 268F	S=1)		

6



# Section 6. Reference

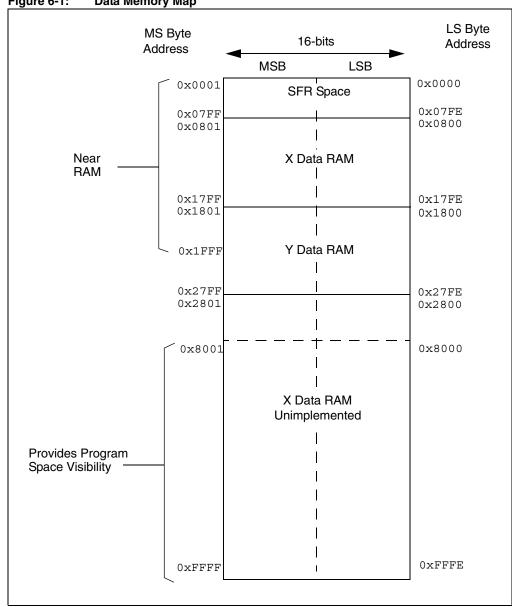
## HIGHLIGHTS

This section of the manual contains reference information for the dsPIC30F. It consists of the following sections:

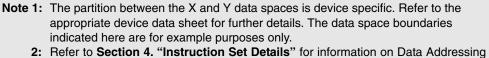
6.1	Data Memory Map	6-2
6.2	Core Special Function Register Map	6-3
6.3	Program Memory Map	6-6
6.4	Instruction Bit Map	6-7
6.5	Instruction Set Summary Table	6-9

### 6.1 **Data Memory Map**

A sample dsPIC30F data memory map is shown in Figure 6-1.



### Figure 6-1: **Data Memory Map**



- modes, performing byte accesses and word alignment requirements.
- 3: Refer to the dsPIC30F Family Reference Manual for information on accessing program memory through data address space.

### 6.2 Core Special Function Register Map

The Core Special Function Register Map is shown in Table 6-1. Please refer to the dsPIC30F Data Sheet for complete register descriptions and the memory map of the remaining special function registers.

Table 6-1:	dsPIC30	dsPIC30F Core Register Map	gister Ma <sub>l</sub>	٥														
Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	2 Bit 11	1 Bit 10	0 Bit 9	9 Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET State
WO	0000								W0 (WREG)	EG)								0000 0000 0000
W1	0002								W1									0000 0000 0000 0000
W2	0004								W2									0000 0000 0000 0000
W3	0006								W3									0000 0000 0000 0000
W4	0008								W4									0000 0000 0000 0000
W5	A000								W5									0000 0000 0000 0000
WG	000C								W6									0000 0000 0000 0000
2M	3000								W7									0000 0000 0000 0000
W8	0010								W8									0000 0000 0000 0000
6M	0012								M9									0000 0000 0000 0000
W10	0014								W10									0000 0000 0000 0000
W11	0016								W11									0000 0000 0000 0000
W12	0018								W12									0000 0000 0000 0000
W13	001A								W13									0000 0000 0000 0000
W14	001C								W14									0000 0000 0000 0000
W15	001E								W15									0000 1000 0000 0000
SPLIM	0020								SPLIM	٧								0000 0000 0000 0000
ACCAL	0022								ACCAL									0000 0000 0000 0000
ACCAH	0024								ACCAH	н								0000 0000 0000 0000
ACCAU	0026		S	Sign-extension		of ACCA<39>	-6						ACCAU	_				0000 0000 0000 0000
ACCBL	0028								ACCBL	_								0000 0000 0000 0000
ACCBH	002A								ACCBH	н								0000 0000 0000 0000
ACCBU	002C		S	Sign-extension		of ACCB<39>	9>						ACCBU					0000 0000 0000 0000
PCL	002E								PCL									0000 0000 0000 0000
PCH	0030	Ι	Ι	Ι	I				Ι	Ι			đ	PCH				0000 0000 0000 0000
TBLPAG	0032		Ι										TBLPAG	(5				0000 0000 0000 0000
PSVPAG	0034		I										PSVPAG	<u>رم</u>				0000 0000 0000 0000
RCOUNT	9600								RCOUNT	Ļ								XXXX XXXX XXXX XXXX
DCOUNT	0038								DCOUNT	۲T								XXXX XXXX XXXX XXXX
DOSTARTL	003A								DOSTARTL	ЗТL								XXXX XXXX XXXX XXXX
DOSTARTH	003C	I	Ι	Ι					Ι	Ι				DOSTARTH	폰			0000 0000 00xx xxxx
DOENDL	003E								DOENDL	۲.								XXXX XXXX XXXX XXXX
DOENDH	0040	Ι	Ι	I	I				I					DOENDH	<u>т</u>			0000 0000 00xx xxxx
SR	0042	OA	OB	SA	SB	OAB	3 SAB	BA	8	IPL2	IPL1	IPL0	RA	z	S	z	υ	0000 0000 0000 0000

# dsPIC30F Programmer's Reference Manual

Table 6-1: dsPIC30F Core Register Map (Continued)

		asi ioon one register map (our mea	קושוניו ויוועף		(2001)													
Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11 Bit 10	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3 Bit 2 Bit 1	Bit 2	Bit 1	Bit 0	RESET State
CORCON	0044	I	I		SN	EDT	DL2	DL1	DLO	SATA	SATB	SATDW	ACCSAT	IPL3	PSV	RND	≞	0000 0000 0010 0000
MODCON	0046	XMODEN	YMODEN				BWM<3:0>	3:0>			ΥW	YWM<3:0>			XWM<3:0>	3:0>		0000 0000 0000 0000
XMODSRT	0048							AMOD	XMODSRT<15:0>	2:0>								XXXX XXXX XXXX XXXX
XMODEND	004A							XMOD	XMODEND<15:0>	2:0>								XXXX XXXX XXXX XXXX
YMODSRT	004C							YMOD	rMODSRT<15:0>	2:0>								XXXX XXXX XXXX XXXX
YMODEND	004E							YMOD	YMODEND<15:0>	2:0>								XXXX XXXX XXXX XXXX
XBREV	0020	NBREN							XBRE	XBREV<14:0>								XXXX XXXX XXXX XXXX
DISICNT	0052									DISICNT<13:0>	T<13:0>							0000 0000 0000 0000
Reserved	0054 - 007E		ļ											I				0000 0000 0000 0000

### 6.3 Program Memory Map

A sample dsPIC30F program memory map is shown in Figure 6-2.

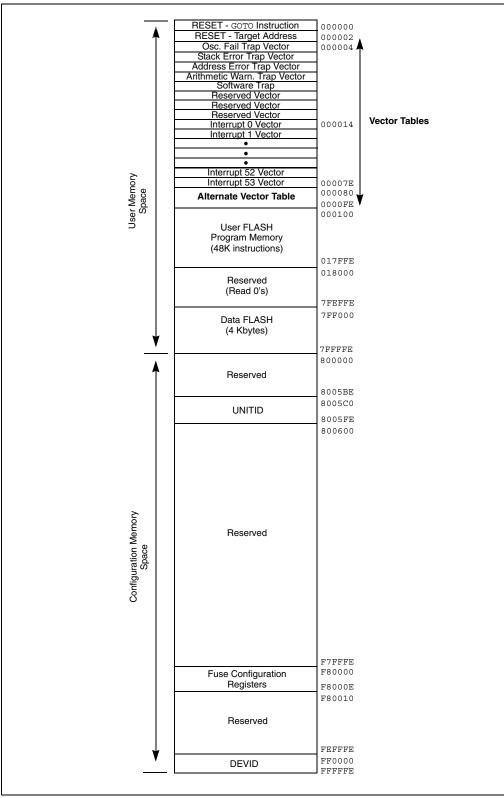


Figure 6-2: Program Space Memory Map

### 6.4 Instruction Bit Map

Instruction encoding for the dsPIC30F is summarized in Table 6-2. This table contains the encoding for the Most Significant Byte 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 Most Significant Byte 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 5.2 through Table 5-12.

		IIII	BRA (SB)										BTSC	MOV	FF1L FF1R	FBCL	CLR SETM	NOPR
		0111	BRA (SA)			BRA (GTU)							BTSS	MOV.D	I	ASR LSR	COM NEG	CLRWDT PWRSAV POP.S PUSH.S RESET
		TOTT	BRA (OB)			BRA (GE)							MSB	SUB	SAC.R	SL	DEC DEC2	DAW EXCH SWAP
		0011	BRA (OA)			BRA (GT)							BTSTS	MUL	SAC		INC INC2	DISI
		TIOI	1	SUBBR		BRA (NN)	ADDC	SUBB	XOR	MOV			BTST	ТВГМТL ТВLWTH	ADD NEG SUB	-	CLR SETM	SE ZE
		1010	1			BRA (NZ)							BTG	TBLRDL TBLRDL	LAC	Ι	COM NEG	LNK
		1001	REPEAT			BRA (NC)							BCLR	MUL.SU MUL.SS	ADD	DIVF	DEC DEC2	РОР
	Opcode<19:16>	1000	Oq		MOV	BRA (NOV)					MOV	MOV	<b>BSET</b>	MUL.UU MUL.US	SFTAC	n'na S'NIQ	INC INC	HSNd
	Opo	TTT0	RCALL			BRA							BTSC	10R MOV	MOVSAC	RRC RRNC	CPSEQ	I
		0110	RETFIE RETURN			BRA (LEU)							BTSS	AND XOR		RLC RLNC	CPSGT CPSLT	l
		1010	RETLW			BRA (LT)							BTST	SUB SUBB	MAC MPY MPY.N MSC	ASR LSR	I	
ling		0010	GOTO	SUBR		BRA (LE)	ADD	SUB	AND	IOR			BTSTS	ADD ADDC		SL	Ι	l
dsPIC30F Instruction Encoding		TT00	I			BRA (N)							BTST	NOM	CLRAC	RRC	CPB CP	
Istructio		0100	CALL			BRA (Z)							BTG	AND XOR		RLC RLNC	CP0	ED EDAC MAC MPY
IC30F Ir		1000	BRA CALL GOTO RCALL			BRA (C)							BCLR	SUB SUBB	MAC MPY MPY.N MSC	ASR LSR	CPB CPB	- <u>-</u> 2 2
dsP		0000	don			BRA (OV)							BSET	ADD ADDC		SL	CP0	
Table 6-2:			0000	1000	0100	1100	0100	1010	0110	1110	1000	1001	1010	1011	0011	TOTT	1110	IIII
Tabl											<02:	6<23	bcod	0				

# dsPIC30F Programmer's Reference Manual

### 6.5 Instruction Set Summary Table

The complete dsPIC30F instruction set is summarized in Table 6-3. 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 which are used in the Instruction Set Summary Table.

Table 6-3:	3: dsPIC30F Instruction Set Summary Table	et Summary Table				-	•		·	•	-	-	-	-	-	
	Assembly Syntax Mnemonic,Operands	Description	Words	Cycles	OA	OB	SA	SB	OAB	SAB	В	z	S	N	υ	Page #
ADD	f {,WREG}	Destination = f + WREG	-	-	I	I	Ι	Ι	Ι	Ι	⇔	⇔	⇔	⇔	\$	5-7
ADD	#lit10,Wn	Wn = lit10 + Wn	٦	1							€	⇔	Ŷ	⇔	£ €	5-8
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	٦	1							⇔	⇔	Ŷ	⇔	€ €	5-9
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1							Û	Û	Û	Û	€ £	5-10
ADD	Acc	Add accumulators	٦	1	Ŷ	€	Û	Ŷ	€	Û	I				9	5-11
ADD	Ws,#Slit4,Acc	16-bit signed add to accumulator	٦	1	Ŷ	€	Û	Ŷ	€	Ŷ	I				9	5-12
ADDC	f {,WREG}	Destination = $f + WREG + (C)$	1	1							Û	Û	Û	仓	€ £	5-14
ADDC	#lit10,Wn	Wn = lit10 + Wn + (C)	٢	1		I	1				€	⇔	Ŷ	₽	€ \$	5-15
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	٦	1							⇔	⇔	Ŷ	⇔	€ €	5-16
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1							Û	ţ	Û	分	€ €	5-17
AND	f {,WREG}	Destination = f .AND. WREG	1	1							I	Û		Û	-	5-19
AND	#lit10,Wn	Wn = lit10 .AND. Wn	٦	1		I	Ι		Ι		Ι	¢		¢	-	5-20
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	٦	1							I	⇔		⇔	9	5-21
AND	Wb,Ws,Wd	Wd = Wb .AND. Ws	1	1								Û		Û	-	5-22
ASR	f {,WREG}	Destination = arithmetic right shift f	٦	1							Ι	⇔		€	3 ()	5-24
ASR	Ws,Wd	Wd = arithmetic right shift Ws	٢	1		I	I				Ι	⇔		€	3 Û	5-25
ASR	Wb,#lit4,Wnd	Wnd = arithmetic right shift Wb by lit4	1	1								Û		Û	-	5-27
ASR	Wb,Wns,Wnd	Wnd = arithmetic right shift Wb by Wns	٢	1		I	1				I	⇔		⇔	0	5-28
BCLR	f,#bit4	Bit clear f	-	-	Ι	Ι	I		Ι		Ι				<u>ل</u>	5-29
BCLR	Ws,#bit4	Bit clear Ws	٦	-	Ι	Ι	Ι		Ι				Ι		-	5-30
BRA	Expr	Branch unconditionally	-	2		I						I			1	5-31
BRA	Wn	Computed branch	-	2		I			I			Ι			1	5-32
BRA	C, Expr	Branch if Carry	٦	1 (2)	Ι	Ι	Ι		Ι	Ι			Ι		-	5-33
BRA	GE,Expr	Branch if greater than or equal	-	1 (2)		I	I		I			I			1	5-35
BRA	GEU,Expr	Branch if Carry	-	1 (2)		I			I			Ι			1	5-36
BRA	GT,Expr	Branch if greater than	۲	1 (2)	I	I	Ι	1	Ι	I	Ι	Ι	I	1	1	5-37
BRA	GTU,Expr	Branch if unsigned greater than	۲	1 (2)	Ι	Ι		Ι	Ι	Ι		Ι		I		5-38
BRA	LE,Expr	Branch if less than or equal	۲	1 (2)	I	I	I		Ι	1	I	Ι	I	1	1	5-39
BRA	LEU,Expr	Branch if unsigned less than or equal	۲	1 (2)	Ι	Ι	I	Ι	Ι	Ι		Ι	I	Ι	1	5-40
BRA	LT,Expr	Branch if less than	۲	1 (2)	I	I	I		Ι	I		Ι	I	I	1	5-41
BRA	LTU,Expr	Branch if not Carry	۲	1 (2)	I	I	I		Ι	1	I	Ι	I	1	1	5-42
BRA	N, Expr	Branch if Negative	۲	1 (2)	Ι	Ι	I	Ι	Ι	Ι		Ι	I	Ι	1	5-43
BRA	NC,Expr	Branch if not Carry	۲	1 (2)	Ι	I	Ι		Ι	I		I	I	I	1	5-44
BRA	NN,Expr	Branch if not Negative	۲	1 (2)	Ι	Ι	Ι	Ι	Ι	I		I	I	I	1	5-45
Legend: Note:	<ul> <li>set or cleared; U may be cleared, bu</li> <li>SA, SB and SAB are only modified if</li> </ul>	set or cleared; $ eqtin may be cleared, but never set;  heta may be set, but never cleared; '1' always set; SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged$	'1' always set; ise unchanged.		'0' always cleared;	ared;		Inged								

# Table 6-3: dsPIC30F Instruction Set Summary Table

lable 6-3:	c dsPlC30F Instruction Set Summary 1ap	et summary lable (continued)	-											_		ſ
	Assembly Syntax Mnemonic,Operands	Description	Words	Cycles	OA	OB	SA	SB	OAB	SAB	R	z	٥	z	ш С	Page #
BRA	NOV,Expr	Branch if not Overflow	۲	1 (2)	I	I	I	1	I	1	1	1	1	1	- 5	5-46
BRA	NZ,Expr	Branch if not Zero	٦	1 (2)	I	I	I	Ι	Ι	I					- 2	5-47
BRA	OA, Expr	Branch if Accumulator A overflow	٦	1 (2)	I	I	I	I	Ι	I					- 5	5-48
BRA	OB,Expr	Branch if Accumulator B overflow	٦	1 (2)	Ι	I	I	Ι	Ι	I					- 5	5-49
BRA	OV,Expr	Branch if Overflow	۲	1 (2)	I	I		I	I						- 2	5-50
BRA	SA,Expr	Branch if Accumulator A saturated	۲	1 (2)	I	I	I	I	I						- 5	5-51
BRA	SB,Expr	Branch if Accumulator B saturated	-	1 (2)	Ι	I	I	I	I	I		I	I		- 2	5-52
BRA	Z, Expr	Branch if Zero	-	1 (2)	Ι	Ι	Ι	Ι	Ι	Ι		Ι	Ι		- 2	5-53
BSET	f,#bit4	Bit set f	۲	1	I	I	I	I	Ι				Ι		- 5	5-54
BSET	Ws,#bit4	Bit set Ws	٢	٢	I	I		I	I			1			- 5	5-55
BSW.C	Ws,Wb	Write C bit to Ws <wb></wb>	٦	1		I	I	I	I			Ι			- 5	5-56
BSW.Z	Ws,Wb	Write Z bit to Ws <wb></wb>	1	1	I	I	I	I	I	I					- 2	5-56
BTG	f,#bit4	Bit toggle f	۲	1	I	I		I	I						- 2	5-58
BTG	Ws,#bit4	Bit toggle Ws	۲	٢	I	I		I	I						- 5	5-59
BTSC	f,#bit4	Bit test f, skip if clear	1	1 (2 or 3)	I	I	I	I	Ι						- 2	5-60
BTSC	Ws,#bit4	Bit test Ws, skip if clear	٢	1 (2 or 3)	I	I	I	I	I	I					- 5	5-62
BTSS	f,#bit4	Bit test f, skip if set	٦	1 (2 or 3)		I		I	I			Ι			- 5	5-64
BTSS	Ws,#bit4	Bit test Ws, skip if set	٢	1 (2 or 3)	Ι	I	Ι	Ι	Ι	I					- 5	5-65
BTST	f,#bit4	Bit test f	٢	1			I							¢	- 2	5-67
BTST.C	Ws,#bit4	Bit test Ws to C	٦	1	I	I	I	I	Ι						ţ 5	5-68
BTST.Z	Ws,#bit4	Bit test Ws to Z	٦	1	I		I		I					¢	- 2	5-68
BTST.C	Ws,Wb	Bit test Ws <wb> to C</wb>	-	-	I	I	I	I	1		1				€ \$	5-69
BTST.Z	Ws,Wb	Bit test Ws <wb> to Z</wb>	٦	1	Ι		I	I	Ι					Ŷ	- 2	5-69
BTSTS	f,#bit4	Bit test then set f	-	-		I	I	I	Ι		1			¢	- 2	5-71
BTSTS.C	Ws,#bit4	Bit test Ws to C then set	-	-	I		I	I	I		1				ţ 5	5-72
BTSTS.Z	Ws,#bit4	Bit test Ws to Z then set	٦	1			I		I					Ŷ	- 5	5-72
CALL	Expr	Call subroutine	2	2	I	I	I	I	Ι	I					- 5	5-73
CALL	Wn	Call indirect subroutine	٦	2	Ι	Ι	Ι	Ι	Ι	Ι		Ι			- 2	5-74
CLR	f	$f = 0 \times 0000$	-	-	Ι	I	I	I	I						1	5-75
CLR	WREG	<b>WREG</b> = $0 \times 0000$	-	-	Ι	Ι	Ι	Ι	Ι	Ι		Ι	Ι		1	5-75
CLR	Wd	Wd = 0	٦	٦	Ι	Ι	Ι	Ι	Ι	Ι		Ι			- 5	5-76
CLR	Acc,Wx,Wxd,Wy,Wyd,AWB	Clear Accumulator	-	-	0	0	0	0	0	0					1	5-77
CLRWDT		Clear Watchdog Timer	-	-				I							1	5-79
COM	f {,WREG}	Destination = $\overline{f}$	-	-								€		€	2	5-80
COM	Ws,Wd	Wd = <u>Ws</u>	٦	1	Ι	Ι	Ι	Ι	Ι			¢		¢	- 5	5-81
Legend: Note:	3 set or cleared; $3$ may be cleared, but never set; SA, SB and SAB are only modified if the correspo	$\hat{\mathbf{T}}$ may be set, but never cleared; anding saturation is enabled, otherwi	'1' always set; se unchanged.		'0' always cleared;		— unchanged	langed								

Table 6-3: dsPIC30F Instruction Set Summary Table (Continued)

	Assembly Syntax Mnemonic.Operands	Description	Words	Cycles	AO	ОВ	SA	SB	OAB	SAB	R	z	5	И	с	Page #
СР	<b>-</b>	Compare (f – WREG)	-	-	1	1	1	1	1	1	Ŷ	Ŷ	Ĥ	£	Ŷ	5-82
СР	Wb,#lit5	Compare (Wb – lit5)	-	-	1	1	1	1	I		> ⇔	) \$	> ⇔	> ⇔		5-83
СР	Wb,Ws	Compare (Wb – Ws)	-	۲	I	I		I			⇔	⇔	⇔	⇔	⇔	5-84
CPO	f	<b>Compare (f</b> – 0x0000)	-	-	I	Ι	I	Ι	Ι	I	-	⇔	⇔	⇔	-	5-85
СРО	Ws	<b>Compare (Ws</b> – 0x0000)	-	-	1	Ι	1	Ι	Ι	Ι	-	⇔	⇔	⇔	-	5-86
CPB	f	Compare with borrow $(f - WREG - \overline{C})$	-	-	Ι		1		I	Ι	⇔	⇔	≎	⇔	⇔	5-87
CPB	Wb,#lit5	Compare with borrow (Wb – lit5 – $\overline{C}$ )	-	-	1	1	1	1	Ι	Ι	⇔	⇔	⇔	⇔	⇔	5-88
СРВ	Wb,Ws	Compare with borrow (Wb – Ws – $\overline{C}$ )	-	-	Ι		1		Ι	Ι	⇔	⇔	⇔	⇔	⇔	5-89
CPSEQ	Wb, Wn	Compare (Wb with Wn), skip if =	-	1 (2 or 3)						Ι						5-91
CPSGT	Wb, Wn	Signed Compare (Wb with Wn), skip if >	۲	1 (2 or 3)	I	I	I	I			I		I	I		5-92
CPSLT	Wb, Wn	Signed Compare (Wb with Wn), skip if <	٢	1 (2 or 3)	I								I			5-93
CPSNE	Wb, Wn	Signed Compare (Wb with Wn), skip if ≠	٢	1 (2 or 3)	I	I	Ι	I			I		I	I		5-94
DAW.B	Wn	Wn = decimal adjust Wn	٢	1	I	I	I	I		I	I	I	I	1	¢	5-95
DEC	f {,WREG}	Destination = $f - 1$	-	1					I		¢	¢	€	⇔	Ŷ	5-96
DEC	Ws,Wd	Wd = Ws - 1	-	-	Ι		I		Ι	Ι	⇔	€	⇔	⇔	€	5-97
DEC2	f {,WREG}	Destination = $f - 2$	-	1					I		¢	¢	€	⇔	Ŷ	5-98
DEC2	Ws,Wd	Wd = Ws - 2	٢	1	I	I	I	I			¢	€	¢	€	¢	5-99
DISI	#lit14	Disable interrupts for lit14 instruction cycles	۲	۲		I	Ι	I	Ι	Ι						5-100
DIV.S	Wm, Wn	Signed 16/16-bit integer divide	٢	18	Ι							⇔	Û	€	¢	5-101
DIV.SD	Wm, Wn	Signed 32/16-bit integer divide	-	18								€	€	€	Ŷ	5-101
DIV.U	Wm, Wn	Unsigned 16/16-bit integer divide	-	18					I			0	0	⇔	Ŷ	5-103
DIV.UD	Wm, Wn	Unsigned 32/16-bit integer divide	٢	18	I	I	I	I			I	0	¢	€	¢	5-103
DIVF	Wm, Wn	Signed 16/16-bit fractional divide	٢	18								Û	Û	Û	Û	5-105
DO	#lit14, Expr	Do code to PC+Expr, (lit14+1) times	2	2		I		I								5-107
DO	Wn, Expr	Do code to PC+Expr, (Wn+1) times	2	2	Ι		Ι		Ι			Ι	I		Ι	5-109
ED	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance (no accumulate)	۲	1	Û	Û	Û	Û	ţ	Û						5-111
EDAC	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance	٢	1	Û	Û	Û	Û	Û	Û						5-113
EXCH	Wns,Wnd	Swap Wns and Wnd	۲	1	I	I	I	I			I		I	I		5-115
FBCL	Ws,Wnd	Find bit change from left (MSb) side	٢	1	I										Û	5-116
FF1L	Ws,Wnd	Find first one from left (MSb) side	٢	1	I	I	I	I		I	I	I	I	1	Ŷ	5-118
FF1R	Ws,Wnd	Find first one from right (LSb) side	٢	1	Ι	Ι		Ι	Ι					Ι	Û	5-120
GOTO	Expr	Go to address	2	2					Ι	Ι						5-122
GOTO	Wn	Go to address indirectly	-	2	Ι					I	I			I	I	5-123

Martine Symmetry         Martine Symmetry<	Assembly Synthy         Description         Mode         Code         Sol<															-		
			Assembly Syntax Mnemonic,Operands	Description	Words	Cycles	OA	ОВ	SA	SB	OAB	SAB	В	z	2	Я		Page #
Www         We	W.W.M.         W.M. M.         W.M. M.M.         W.M. M.M.         W.M. M.M.	INC	f {,WREG}	Destination = f + 1	-	-	I	Ι	I	Ι	Ι	I	⇔	⇔	⇔	⇔		5-124
	I.UMEG()         Demindione 1+2         I	INC	Ws,Wd	Wd = Ws + 1	۲	٦			I	_			Ŷ	⇔	¢	⇔		5-125
WixNot         Mode Mode 2           WixNot         We inform         We informed in the formed in the forme	Word         Word         Word         Word         Word         Word         S	INC2	f {,WREG}	f +	۲	٦			Ι				Ŷ	⇔	Ŷ	€		5-126
(WREe)         Destination = LOR, WREG         1	I (WREG)         Destination = ()OR, WREG         I = 1 <th<< td=""><td>INC2</td><td>Ws,Wd</td><td>Wd = Ws + 2</td><td>-</td><td>-</td><td> </td><td>I</td><td>I</td><td> </td><td>I</td><td>I</td><td>⇔</td><td>⇔</td><td>⇔</td><td>⇔</td><td></td><td>5-127</td></th<<>	INC2	Ws,Wd	Wd = Ws + 2	-	-		I	I		I	I	⇔	⇔	⇔	⇔		5-127
mintolun         maintolun         maintolun <th< td=""><td>mitto.         Mail mitto.         <t< td=""><td>IOR</td><td>f {,WREG}</td><td>Destination = f .IOR. WREG</td><td>۲</td><td>٦</td><td> </td><td>Ι</td><td>Ι</td><td>—</td><td> </td><td>I</td><td> </td><td>⇔</td><td> </td><td>€</td><td>1</td><td>5-128</td></t<></td></th<>	mitto.         Mail mitto. <t< td=""><td>IOR</td><td>f {,WREG}</td><td>Destination = f .IOR. WREG</td><td>۲</td><td>٦</td><td> </td><td>Ι</td><td>Ι</td><td>—</td><td> </td><td>I</td><td> </td><td>⇔</td><td> </td><td>€</td><td>1</td><td>5-128</td></t<>	IOR	f {,WREG}	Destination = f .IOR. WREG	۲	٦		Ι	Ι	—		I		⇔		€	1	5-128
Wunkikud         We wulde	WD/Mailed         WD = WD. DCR IIIS         WD = WD. DCR IIIS         WD = WD. DCR IIIS         WD = WD. DCR WD         WD = WD. D	IOR	#lit10,Wn	Wn = lit10 .IOR. Wn	-	-			I	I				⇔		⇔	22	5-129
WUMANUG         Wet who LOR, we         Wet we         I	Why Word         Wee Word	IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	-	-	I	I	Ι	I	I	I	I	⇔		⇔	ى ا	5-130
We #SIR4, Acc:         Load Accumulator         1         1         2         3         3         3         3         3           1(#144)         I (#Iffate)         Dim/itame poinder         1 <td>We shale, Acc         Lead Accumulator         1         1         2         2         2         1         <th1< th="">         1</th1<></td> <td>IOR</td> <td>Wb,Ws,Wd</td> <td>Wd = Wb .IOR. Ws</td> <td>۲</td> <td>+</td> <td>I</td> <td>Ι</td> <td>Ι</td> <td>—</td> <td> </td> <td>I</td> <td> </td> <td>⇔</td> <td> </td> <td>€</td> <td>1</td> <td>5-131</td>	We shale, Acc         Lead Accumulator         1         1         2         2         2         1 <th1< th="">         1</th1<>	IOR	Wb,Ws,Wd	Wd = Wb .IOR. Ws	۲	+	I	Ι	Ι	—		I		⇔		€	1	5-131
#if14         Link frame pointer         1 <th1< th="">         1         <th1< th="">         1</th1<></th1<>	iff14       Implicate	LAC	Ws,#Slit4, Acc	Load Accumulator	-	-	⇔	ţ	₽	Û	¢	¢	Ι				ى ا	5-133
I, (MREG)         Destination logical right shift         1 <th1< th="">         1</th1<>	(i,WREG)       Destination = logical right shift // S       I <th< td=""><td>LNK</td><td>#lit14</td><td>Link frame pointer</td><td>۲</td><td>٢</td><td> </td><td>Ι</td><td>I</td><td> </td><td>I</td><td>I</td><td>I</td><td> </td><td> </td><td> </td><td>1</td><td>5-135</td></th<>	LNK	#lit14	Link frame pointer	۲	٢		Ι	I		I	I	I				1	5-135
We/We/         We/We/         Mode logical right shift Work         I	W6.Wd         Wd = logical right shift WGs         1 <th1< th="">         1         <th1< th=""> <th< td=""><td>LSR</td><td>f {,WREG}</td><td>Destination = logical right shift f</td><td>٢</td><td>٦</td><td> </td><td>-</td><td>I</td><td> </td><td> </td><td>I</td><td>I</td><td>0</td><td> </td><td>⇔</td><td></td><td>5-136</td></th<></th1<></th1<>	LSR	f {,WREG}	Destination = logical right shift f	٢	٦		-	I			I	I	0		⇔		5-136
Wb #illity.Mod         Wo a legical right shift.Wb by lifte         1 <th1< th="">         1         <th1< th="">         1</th1<></th1<>	Wb #iftd, Wind         Who Wind         Who Wind         Who Wind         Who Wind         Who Wind         Who Wind	LSR	Ws,Wd	Wd = logical right shift Ws	٦	1			Ι					0		€		5-137
Wb,Wre,WrdWh,Wre,WrdWnd = logical right shit Wb by WrsIndII <td>Wb.Wms.Mmd         Wm and the logical right shift Wb by Mms         Ind         Logical right shift Wb by Mms         Logical right shift Shif</td> <td>LSR</td> <td>Wb,#lit4,Wnd</td> <td>Wnd = logical right shift Wb by lit4</td> <td>٢</td> <td>٦</td> <td>l</td> <td>I</td> <td>I</td> <td></td> <td>I</td> <td>I</td> <td>I</td> <td>⇔</td> <td>I</td> <td>⇔</td> <td>1</td> <td>5-139</td>	Wb.Wms.Mmd         Wm and the logical right shift Wb by Mms         Ind         Logical right shift Wb by Mms         Logical right shift Shif	LSR	Wb,#lit4,Wnd	Wnd = logical right shift Wb by lit4	٢	٦	l	I	I		I	I	I	⇔	I	⇔	1	5-139
Wm Wm, Acc, Wx, Wxd, Wy, Wyd, AWBMutiply and accumulate111 <th1< th="">11&lt;</th1<>	Winn'A.dc.W.X.Wod.W.Y.Mod.         Multiply and accumulate         11         1 <th1< th="">         1         1         <th1< td=""><td>LSR</td><td>Wb,Wns,Wnd</td><td>Wnd = logical right shift Wb by Wns</td><td>٢</td><td>٦</td><td> </td><td> </td><td>I</td><td>_</td><td> </td><td> </td><td> </td><td>⇔</td><td> </td><td>⇔</td><td>22</td><td>5-140</td></th1<></th1<>	LSR	Wb,Wns,Wnd	Wnd = logical right shift Wb by Wns	٢	٦			I	_				⇔		⇔	22	5-140
Wm*m.facc.Wx,Wxd,Wy,Wd,Square and accmulate11 <th1< th=""><th< td=""><td>Wm*Wn.Acc,Wx,Wad,Wy,Wd,Square and accumulate11133333411111<math>t(WREG)</math>Move for destination11</td><td>MAC</td><td>Wm*Wn,Acc,Wx,Wxd,Wyd,AWB</td><td>Multiply and accumulate</td><td>۲</td><td>٦</td><td>¢</td><td>ţ</td><td>Û</td><td>Ų</td><td>Ŷ</td><td>Û</td><td> </td><td> </td><td> </td><td> </td><td>22</td><td>5-141</td></th<></th1<>	Wm*Wn.Acc,Wx,Wad,Wy,Wd,Square and accumulate11133333411111 $t(WREG)$ Move for destination11	MAC	Wm*Wn,Acc,Wx,Wxd,Wyd,AWB	Multiply and accumulate	۲	٦	¢	ţ	Û	Ų	Ŷ	Û					22	5-141
f(WEG)Move free destination $1$		MAC	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd,	Square and accumulate	٢	1	Û	Û	Û	Û	Û	Û					1	5-143
WREGAMore WREGA forMore WREGA for11 <th1< th=""><th1< td=""><td>WREG#         Move WREG tof         11         <th1< th=""> <th1< th="">         1</th1<></th1<></td><td>MOV</td><td>f {,WREG}</td><td>Move f to destination</td><td>-</td><td>-</td><td>I</td><td>Ι</td><td>Ι</td><td>I</td><td>I</td><td>Ι</td><td>I</td><td>⇔</td><td> </td><td>⇔</td><td>1</td><td>5-145</td></th1<></th1<>	WREG#         Move WREG tof         11         1 <th1< th=""> <th1< th="">         1</th1<></th1<>	MOV	f {,WREG}	Move f to destination	-	-	I	Ι	Ι	I	I	Ι	I	⇔		⇔	1	5-145
f,MndMove fto WndII<	tWnst1Move for ModeMove Move ModeMove ModeMove Move ModeMove ModeMove Move ModeMove Move ModeMove Move ModeMove Move ModeMove Move ModeMove ModeMove ModeMove Move ModeMove Mode	MOV	WREG,f	Move WREG to f	۲	۲		Ι	I	Ι	Ι	I	I				1	5-146
Wns,f         Move Write of         Move Write of         Move Write of         Move Write of         Move Beht unsigned literal to Wnd         1 <th1< td=""><td>Wns,fMore Wins to fMore Wins to fMore Wins to fMore Wins to fIII<!--</td--><td>MOV</td><td>f,Wnd</td><td>Move f to Wnd</td><td>٢</td><td>٦</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td> </td><td>Ι</td><td> </td><td> </td><td>1</td><td>5-147</td></td></th1<>	Wns,fMore Wins to fMore Wins to fMore Wins to fMore Wins to fIII </td <td>MOV</td> <td>f,Wnd</td> <td>Move f to Wnd</td> <td>٢</td> <td>٦</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td> </td> <td>Ι</td> <td> </td> <td> </td> <td>1</td> <td>5-147</td>	MOV	f,Wnd	Move f to Wnd	٢	٦	Ι	Ι	Ι	Ι	Ι	Ι		Ι			1	5-147
#it8.VndMove B-bit unsigned literal to Wnd111 $   -$ <td>#itB,WindMove B-bit unsigned literal to Wind111<math>   -</math><!--</td--><td>MOV</td><td>Wns,f</td><td>Move Wns to f</td><td>٢</td><td>٦</td><td> </td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td> </td><td> </td><td> </td><td> </td><td>1</td><td>5-148</td></td>	#itB,WindMove B-bit unsigned literal to Wind111 $   -$ </td <td>MOV</td> <td>Wns,f</td> <td>Move Wns to f</td> <td>٢</td> <td>٦</td> <td> </td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td>Ι</td> <td> </td> <td> </td> <td> </td> <td> </td> <td>1</td> <td>5-148</td>	MOV	Wns,f	Move Wns to f	٢	٦		Ι	Ι	Ι	Ι	Ι					1	5-148
#it16,WndMove 16-bit literal to WndIn	#it16,WndMove 16-Dit literal to WndMove 16-Dit literal to WndMove 16-Dit literal to WndI11 $  -$ <	MOV.B	#lit8,Wnd		-	-		Ι	Ι	Ι	I	Ι	Ι				1	5-149
	Wins-Slitt 0j, WindMove (Wins + Slitt 0j to WindMove (Wins + Slitt 0jMove (Wins + Slitt 0jMove (Wins + Slitt 0jMove (Wins + Slitt 0j)Move (Wins +	MOV	#lit16,Wnd	Move 16-bit literal to Wnd	٢	٦	Ι	Ι	Ι	Ι	Ι	Ι		Ι			1	5-150
Wns,[Wnd+Slit10]Move Wns to [Wnd + Slit10]III <th< td=""><td>Wns.[Wnd+Slit10]Move Wns to [Wnd + Slit10]Index (Wnd + Slit10)Index (Wnd + Slit10)Index (Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[</td><td>MOV</td><td>[Wns+Slit10],Wnd</td><td>Move [Wns + Slit10] to Wnd</td><td>-</td><td>-</td><td> </td><td>Ι</td><td>Ι</td><td>I</td><td>I</td><td>I</td><td>I</td><td>Ι</td><td>Ι</td><td> </td><td>1</td><td>5-151</td></th<>	Wns.[Wnd+Slit10]Move Wns to [Wnd + Slit10]Index (Wnd + Slit10)Index (Wnd + Slit10)Index (Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wnd + Slit10)Index (Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[Wns.[	MOV	[Wns+Slit10],Wnd	Move [Wns + Slit10] to Wnd	-	-		Ι	Ι	I	I	I	I	Ι	Ι		1	5-151
WS,WdMove Ws to WdMove Ws to WdMove Ws to WdIII <th< td=""><td>WS,WdMove ws to WdMove ws to WdII</td><td>MOV</td><td>Wns,[Wnd+Slit10]</td><td>Move Wns to [Wnd + Slit10]</td><td>٢</td><td>٦</td><td> </td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td> </td><td> </td><td>I</td><td> </td><td>1</td><td>5-152</td></th<>	WS,WdMove ws to WdMove ws to WdII	MOV	Wns,[Wnd+Slit10]	Move Wns to [Wnd + Slit10]	٢	٦		Ι	Ι	Ι	Ι	Ι			I		1	5-152
WS,Wnd         Move double Ws to Wnd+1         1         2 <th< td=""><td>Ws,Wnd         Move double Ws to Wnd+1         1         2   <th< td=""><td>MOV</td><td>Ws,Wd</td><td>Move Ws to Wd</td><td>-</td><td>-</td><td>I</td><td>Ι</td><td>I</td><td>I</td><td>I</td><td>I</td><td>I</td><td> </td><td> </td><td> </td><td>1</td><td>5-153</td></th<></td></th<>	Ws,Wnd         Move double Ws to Wnd+1         1         2 <th< td=""><td>MOV</td><td>Ws,Wd</td><td>Move Ws to Wd</td><td>-</td><td>-</td><td>I</td><td>Ι</td><td>I</td><td>I</td><td>I</td><td>I</td><td>I</td><td> </td><td> </td><td> </td><td>1</td><td>5-153</td></th<>	MOV	Ws,Wd	Move Ws to Wd	-	-	I	Ι	I	I	I	I	I				1	5-153
Wns,Wd         Move double Wns:Wns+1 to Wd         1         2	Wns,Wd         Move double Wns.Wns+1 to Wd         1         2	MOV.D	Ws,Wnd	Move double Ws to Wnd:Wnd+1	-	2		Ι	I	I	I	I		I		1	1	5-155
SAC         Acc,Wx,WXd,Wy,Wyd,AWB         Move [Wx] to Wxd, and [Wy] to Wyd         1         1         -	SACAcc,Wx,Wxd,Wy,Wyd,AWBMove [Wx] to Wxd, and [Wy] to Wyd11 $t$ </td <td>MOV.D</td> <td>Wns,Wd</td> <td>Move double Wns:Wns+1 to Wd</td> <td>-</td> <td>2</td> <td> </td> <td>I</td> <td>I</td> <td>I</td> <td>I</td> <td>I</td> <td> </td> <td>I</td> <td> </td> <td> </td> <td>1</td> <td>5-157</td>	MOV.D	Wns,Wd	Move double Wns:Wns+1 to Wd	-	2		I	I	I	I	I		I			1	5-157
Wm <sup>-</sup> Wn, Acc, Wx, Wxd, Wy, Wyd         Multiply Wn by Wm to accumulator         1	Wm*Wn, Acc, Wx, Wxd, Wy, WydMultiply Wn by Wm to accumulator11 $\bigcirc$ $\frown$ <th< td=""><td>MOVSAC</td><td>Acc,Wx,Wxd,Wy,Wyd,AWB</td><td>Move [Wx] to Wxd, and [Wy] to Wyd</td><td>-</td><td>-</td><td> </td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td>Ι</td><td> </td><td>Ι</td><td>Ι</td><td> </td><td>1</td><td>5-159</td></th<>	MOVSAC	Acc,Wx,Wxd,Wy,Wyd,AWB	Move [Wx] to Wxd, and [Wy] to Wyd	-	-		Ι	Ι	Ι	Ι	Ι		Ι	Ι		1	5-159
Wm*Wm,Acc,Wx,Wxd,Wy,Wyd         Square to Accumulator         1 <td>Wm*Wm,Acc,Wx,Wxd,Wy,WydSquare to Accumulator11<math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math><math>(1)</math></td> <td>МРҮ</td> <td>Wm*Wn,Acc,Wx,Wxd,Wy,Wyd</td> <td>Multiply Wn by Wm to accumulator</td> <td>-</td> <td>-</td> <td>⇔</td> <td>¢</td> <td>⊅</td> <td>Û</td> <td>⇔</td> <td>¢</td> <td>I</td> <td>I</td> <td> </td> <td>1</td> <td><u>م</u></td> <td>5-161</td>	Wm*Wm,Acc,Wx,Wxd,Wy,WydSquare to Accumulator11 $(1)$	МРҮ	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	Multiply Wn by Wm to accumulator	-	-	⇔	¢	⊅	Û	⇔	¢	I	I		1	<u>م</u>	5-161
N         Wm*Wn,Acc,Wx,Wxd,Wy,Wyd         (Multiply Wn by Wm) to Accumulator         1         1         0         0	N       Wm*Wn,Acc,Wx,Wxd,Wy,Wyd       -(Multiply Wn by Wm) to Accumulator       1       1       0       0       - </td <td>МРҮ</td> <td>Wm*Wm,Acc,Wx,Wxd,Wy,Wyd</td> <td>Square to Accumulator</td> <td>۲</td> <td>٦</td> <td>¢</td> <td>Û</td> <td>Ŷ</td> <td>Ų</td> <td>Ŷ</td> <td>Û</td> <td> </td> <td> </td> <td> </td> <td> </td> <td>22</td> <td>5-163</td>	МРҮ	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square to Accumulator	۲	٦	¢	Û	Ŷ	Ų	Ŷ	Û					22	5-163
Wm*Wn,Acc,Wx,Wxd,Wyd,AWB         Multiply and subtract from Accumulator         1         1         2         1         1         -	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	MPY.N	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	-(Multiply Wn by Wm) to Accumulator	۲	۲	0	0	I	Ι	0	I	I				1	5-165
f         W3:W2 = f * WREG         1         1         -	f         W3:W2 = f * WREG         1         1         -	MSC	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd,AWB	Multiply and subtract from Accumulator	۲	1	€	¢	Û	Û	¢	Û	Ι				1	5-167
Wb,Ws,Wnd {{Wnd+1,Wnd} = sign(Wb) * sign(Ws) 1 1 1	Wb,Ws,Wnd       {Wnd+1,Wnd}       sign(Wb) * sign(Ws)       1       1       -	MUL	f	W3:W2 = f * WREG	-	-		Ι	I	I	I	I	I				1	5-169
	(i) set or cleared; <sup>(1)</sup> may be cleared, but never set; <sup>(1)</sup> may be set, but never cleared; <sup>(1)</sup> always set; <sup>(0)</sup> always cleared; <sup>(1)</sup> always set; <sup>(0)</sup> always cleared; <sup>(1)</sup> always set; <sup>(1)</sup> always se	MUL.SS	Wb,Ws,Wnd	* (dW)	٢	-	I	Ι	Ι				I				1	5-170

Table 6-3: dsPIC30F Instruction Set Summary Table (Continued)

Table 6-3:	3: dsPIC30F Instruction Set Summary Tab	et Summary Table (Continued)														
	Assembly Syntax Mnemonic,Operands	Description	Words	Cycles	OA	ОВ	SA	SB	OAB	SAB	DC	z	٥٧	z	C	Page #
MUL.SU	Wb,#lit5,Wnd	{Wnd+1,Wnd} = sign(Wb) * unsign(lit5)	-	-	1	1	1	1	I	1		1	1	1	- 2	5-172
MUL.SU	Wb,Ws,Wnd	{Wnd+1,Wnd} = sign(Wb) * unsign(Ws)	٢	÷											- 2	5-174
MUL.US	Wb,Ws,Wnd	{Wnd+1,Wnd} = unsign(Wb) * sign(Ws)	1	٢	I		Ι		Ι		-	-			- 5	5-176
Μυμ.υυ	Wb,#lit5,Wnd	{Wnd+1,Wnd} = unsign(Wb) * unsign(lit5)	۰	-		Ι		Ι	Ι						- 2	5-178
Μυμ.υυ	Wb,Ws,Wnd	{Wnd+1,Wnd} = unsign(Wb) * unsign(Ws)	۲	۲	l	I		I	Ι	I			I		- 2	5-179
NEG	f {,WREG}	Destination = $\overline{f} + 1$	٢	٢	I						€	⇔	⇔	Ŷ	€ €	5-181
NEG	Ws,Wd	Wd = $\overline{WS} + 1$	-	-	I	1	1		1		⇔	⇔	¢	€	⊕ 2·	5-182
NEG	Acc	Negate Accumulator	-	-	⇔	⇔	Ŷ	Ŷ	⇔	Ŷ	1	1	1		ن ا	5-183
NOP		No operation	-	-	I	1	1	I	I	1	1	1			ۍ ا	5-184
NOPR		No operation	-	-	1	1	1	I	1	1	1				ن ا	5-185
РОР	÷	Pop TOS to f	-	-	I	I	I	Ι	I	I		1		1	ن ا	5-186
РОР	Md	Pop TOS to Wd	٦	۰	I	I	I	I	I	I		I			- 2	5-187
POP.D	Wnd	Pop double from TOS to Wnd:Wnd+1	٢	2				I							- 2	5-188
POP.S		Pop shadow registers	٢	٢	I						⇔	⇔	€	Ŷ	€ €	5-189
PUSH	÷	Push f to TOS	-	-	I	I	I	Ι	I	I		1		1	ن ا	5-190
PUSH	Ws	Push Ws to TOS	۲	۲	I	I	I	I	Ι	I			I		- 2	5-191
PUSH.D	Wns	Push double Wns:Wns+1 to TOS	٦	2	I	I	I	I	I	I			1		- 2	5-192
PUSH.S		Push shadow registers	۰	-		I		I	Ι						- 2	5-193
PWRSAV	#lit1	Enter Power Saving mode	۲	۲	I	I	I	Ι	I						- 2	5-194
RCALL	Expr	Relative call	۰	2		I	Ι	Ι	Ι	Ι		-			- 5	5-195
RCALL	Wn	Computed call	۰	2		I		Ι	Ι						2 	5-196
REPEAT	#lit14	Repeat next instruction (lit14+1) times	1	٢	I		Ι		Ι		-	-			- 5	5-197
REPEAT	Wn	Repeat next instruction (Wn+1) times	۲	-	I	I	I	I	I	I		I	I		- 2	5-198
RESET		Software device RESET	۲	۲	l	I		I	Ι	I			I		- 2	5-200
RETFIE		Return from interrupt enable	-	3 (2)	I	I	I	I	I	I		⇔	⇔	⇔	€ €	5-201
RETLW	#lit10,Wn	Return with lit10 in Wn	٢	3 (2)											- 2	5-202
RETURN		Return from subroutine	۲	3 (2)		Ι	Ι	Ι	Ι	Ι					- 2	5-203
RLC	f {,WREG}	Destination = rotate left through Carry f	٢	-	I	I		I				⇔		Ŷ	€ €	5-204
RLC	Ws,Wd	Wd = rotate left through Carry Ws	٢	-								⇔		Ŷ	€ €	5-205
RLNC	f {,WREG}	Destination = rotate left (no Carry) f	٦	۲								⇔		¢	- 2	5-207
RLNC	Ws,Wd	Wd = rotate left (no Carry) Ws	-	-	I	I	I	Ι	Ι	1	1	⇔		⇔	ي ا	5-208
RRC	f {,WREG}	Destination = rotate right through Carry f	-	-	I		1		1			⇔		€	€ €	5-210
RRC	Ws,Wd	Wd = rotate right through Carry Ws	۲	-								€		Û	<del>(</del> 5	5-211
RRNC	f {,WREG}	Destination = rotate right (no Carry) f	٢	1								¢		Û	- 2	5-213
RRNC	Ws,Wd	Wd = rotate right (no Carry) Ws	٢	٢	Ι		I	Ι				€		Û	- 2	5-214
Legend: <b>Note:</b>	<ul> <li>the set or cleared; <ul> <li>Image and set only modified if</li> <li>Image and set only modified if</li> </ul> </li> </ul>	$\oplus$ set or cleared; $\oplus$ <i>may</i> be cleared, but never set; $\oplus$ <i>may</i> be set, but never cleared; '1' always set; SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged	<li>'1' always set; ise unchanged.</li>		'0' always cleared;		— unchanged	anged								

# dsPIC30F Programmer's Reference Manual

														_		
	Assembly Syntax Mnemonic,Operands	Description	Words	Cycles	AO	ОВ	SA	SB	OAB	SAB	БС	z	٥	z	ц С	Page #
SAC	Acc,#Slit4,Wd	Store Accumulator	-	-	1	1	I	1	1	1	1	1	1	1	- 5	5-216
SAC.R	Acc,#Slit4,Wd	Store rounded Accumulator	۲	1		Ι	Ι								- 5	5-218
SE	Ws,Wd	Wd = sign-extended Ws	1	1				I		I		⇔		¢	¢ 5	5-220
SETM	ţ	$f = 0 \times FFFF$	-	-	I	I	I	I	1			I	I		- 5	5-221
SETM	WREG	WREG = 0xFFFF	٢	۲		Ι	Ι	Ι				I	I		- 5	5-221
SETM	Ws	WS = 0XFFFF	1	1			I	I				I	I		- 5	5-222
SFTAC	Acc,#Slit6	Arithmetic shift accumulator by Slit6	۲	-	¢	¢	Û	Ŷ	Ŷ	Ŷ	1				- 5	5-223
SFTAC	Acc,Wn	Arithmetic shift accumulator by (Wn)	-	-	⇔	≎	Ŷ	Ŷ	€	Ŷ	1	1	1	1	- 5	5-224
SL	f {,WREG}	Destination = arithmetic left shift f	-	-	I		I		1	1	1	⇔		⇔	÷ 5	5-225
SL	Ws,Wd	Wd = arithmetic left shift Ws	-	-	I		Ι			1	1	⇔		⇔	÷ 5	5-226
SL	Wb,#lit4,Wnd	Wnd = left shift Wb by lit4	-	-		1	I		1	1	1	⇔		⇔	- 5	5-228
SL	Wb,Wns,Wnd	Wnd = left shift Wb by Wns	-	-	I		I		1	1	1	⇔		⇔	- 5	5-229
SUB	f {,WREG}	Destination = f – WREG	٦	٢			I	Ι			Ŷ	⇔	⇔	Ŷ	Û 5	5-230
SUB	#lit10,Wn	Wn = Wn - lit10	-	-	I		I			1	¢	⇔	⇔	⇔	÷ 5	5-231
SUB	Wb,#lit5,Wd	Wd = Wb - lit5	٦	1	I			I			¢	⇔	⇔	€	÷ 5	5-232
SUB	Wb,Ws,Wd	Wd = Wb - Ws	1	1	I			I	I	I	¢	⇔	⇔	Ŷ	<del>(</del> 5	5-233
SUB	Acc	Subtract Accumulators	1	-	ţ	¢	Û	Û	¢	Û					- 5	5-235
SUBB	f {,WREG}	destination = f – WREG – $(\overline{C})$	1	1	Ι		I	Ι			Û	¢	¢	仚	Û 5	5-236
SUBB	#lit10,Wn	$Wn = Wn - lit10 - (\overline{C})$	٢	1	Ι			Ι			ţ	Û	Û	Ŷ	<u>ئ</u> 5	5-237
SUBB	Wb,#lit5,Wd	Wd = Wb - lit5 - $(\overline{C})$	٦	-							Ŷ	⇔	⇔	Ŷ	Û 5	5-238
SUBB	Wb,Ws,Wd	$Wd = Wb - Ws - (\overline{C})$	٦	1	I			I			Ŷ	⇔	⇔	Ŷ	÷ 5	5-239
SUBBR	f {,WREG}	Destination = WREG $- f - (\overline{C})$	٢	٢	Ι			Ι			ţ	Û	Û	Ŷ	<u>ئ</u> 5	5-241
SUBBR	Wb,#lit5,Wd	Wd = lit5 – Wb – $(\overline{C})$	1	1				I			¢	⇔	⇔	Ŷ	Û 5	5-242
SUBBR	Wb,Ws,Wd	$Wd = Ws - Wb - (\overline{C})$	٦	1	I			I			Ŷ	⇔	⇔	Ŷ	÷ 5	5-243
SUBR	f {,WREG}	Destination = WREG – f	٦	1	I						¢	⇔	⇔	€	÷ 5	5-245
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	٦	٢							Ŷ	⇔	⇔	Ŷ	Û 5	5-246
SUBR	Wb,Ws,Wd	Wd = Ws - Wb	1	1	Ι	Ι	Ι	Ι			Û	¢	¢	Û	Û 5	5-247
SWAP	Wn	Wn = byte or nibble swap Wn	۰	٢	1	1	I	1	I	1	1	1	1		- 5	5-249
TBLRDH	Ws,Wd	Read high program word to Wd	-	2	Ι	Ι		Ι				I	I		- 5	5-250
TBLRDL	Ws,Wd	Read low program word to Wd	-	2	Ι	Ι		Ι				I	I		- 5	5-252
TBLWTH	Ws,Wd	Write Ws to high program word	-	2		Ι		Ι				I	I		- 5	5-254
TBLWTL	Ws,Wd	Write Ws to low program word	-	2	Ι	Ι	Ι	Ι	I	I		I	I	I	1	5-256
ULNK		Unlink frame pointer	-	-	Ι										1	5-258
Legend: Note:	(b) set or cleared;	$\oplus$ set or cleared; $\oplus$ <i>may</i> be cleared, but never set; $\oplus$ <i>may</i> be set, but never cleared; '1' always set; $\oplus$ A. SB and SAB are only modified if the corresponding saturation is enabled. otherwise unchanged	'1' always set; se unchanged.		'0' always cleared;		— unchanged	langed								
;																

Table 6-3: dsPIC30F Instruction Set Summary Table (Continued)

Table 6-3:		dsPIC30F Instruction Set Summary Table (Continued)														
	Assembly Syntax Mnemonic,Operands	Description	Words	Words Cycles OA OB	OA	OB	SA	SB	OAB	SB OAB SAB DC		z	٥٧	z	<u>ں</u>	C Page #
XOR	f {,WREG}	Destination = f .XOR. WREG	-	-	I	I	I	I	I	Ι	1	⇔	I	⇔		5-259
XOR	#lit10,Wn	Wn = lit10 .XOR. Wn	1	1				Ι	Ι	I		¢	Ι	Û	-	5-260
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1								Û		ţ	-	5-261
XOR	Wb,Ws,Wd	Wd = Wb .XOR. Ws	1	1			Ι					Û	1	Û	-	5-262
ZE	Ws,Wd	Wd = zero-extended Ws	1	1		I	I	I	I	I	I	0		¢	1	5-264
Legend: Note:	<ul> <li>teat or cleared;          <ul> <li><i>umay</i> be cleared, busiles</li> <li>SA, SB and SAB are only modified if <sup>1</sup></li> </ul> </li> </ul>	Legend: ③ set or cleared; <sup>①</sup> <i>may</i> be cleared, but never set; ① <i>may</i> be set, but never cleared; '1' always set; '0' always cleared; — unchanged <b>Note:</b> SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.	L' always s e unchanç	set; 'o' alw jed.	ays cle	ared;	— unch	langed								

# dsPIC30F Programmer's Reference Manual

# INDEX

### Α

Accumulator A, Accumulator B	2-5
Accumulator Access	
Accumulator Usage	
Addressing Modes for Wd Destination Register	
Addressing Modes for Ws Source Register	5-3
Assigned Working Register Usage	4-27

### в

Byte (	Operations			4-13
--------	------------	--	--	------

### С

Code Examples	
'Z' Status bit Operation for 32-bit Addition	.4-26
Base MAC Syntax	
File Register Addressing	
File Register Addressing and WREG	
Frame Pointer Usage	
Illegal Word Move Operations	.4-18
Immediate Addressing	
Indirect Addressing with Effective Address Update	
Indirect Addressing with Register Offset	
Legal Word Move Operations	
MAC Accumulator WB Syntax	
MAC Pre-Fetch Syntax	
Move with Literal Offset Instructions	4-7
MSC Instruction with Two Pre-Fetches and	
Accumulator Write Back	. 4-36
Normalizing with FBCL	. 4-39
Register Direct Addressing	4-4
Sample Byte Math Operations	.4-15
Sample Byte Move Operations	
Scaling with FBCL	. 4-38
Stack Pointer Usage	.4-21
Unsigned f and WREG Multiply	
(Legacy MULWF Instruction)	. 4-29
Using 10-bit Literals for Byte Operands	
Using the Default Working Register WREG	
Conditional Branch Instructions	
Core Control Register	2-9
Core Special Function Register Map	6-3

### D

Data Addressing Mode Tree	4-10
Data Addressing Modes	
Data Memory Map	6-2
DCOUNT Register	2-6
Default Working Register (WREG)	2-3
Default Working Register WREG	
Development Support	
DOEND Register	2-6
DOSTART Register	2-6
DSP Accumulator Instructions	4-37
DSP Data Formats	
DSP MAC Indirect Addressing Modes	
DSP MAC Instructions	
dsPIC30F Overview	2-2
F	

File Register Addre	ssing	4-2
---------------------	-------	-----

### L

Internet all at a Andreas all and the second s
Immediate Addressing
Operands in the Instruction Set 4-9
Implied DSP Operands
Implied Frame and Stack Pointer 4-27
Instruction Bit Map
Instruction Description Example
Instruction Descriptions
ADD (16-bit Signed Add to Accumulator) 5-12
ADD (Add Accumulators) 5-11
ADD (Add f to WREG) 5-7
ADD (Add Literal to Wn)5-8
ADD (Add Wb to Short Literal)
ADD (Add Wb to Ws)
ADDC (Add f to WREG with Carry) 5-14
ADDC (Add Literal to Wn with Carry) 5-15
ADDC (Add Wb to Short Literal with Carry)5-16
ADDC (Add Wb to Ws with Carry) 5-17
AND (AND f and WREG)5-19
AND (AND Literal and Wd)5-20
AND (AND Wb and Short Literal)
AND (AND Wb and Ws)
ASR (Arithmetic Shift Right by Short Literal)
ASR (Arithmetic Shift Right by Wns) 5-28
ASR (Arithmetic Shift Right f) 5-24
ASR (Arithmetic Shift Right Ws) 5-25
BCLR (Bit Clear in Ws)
BCLR.B (Bit Clear f)
BRA (Branch Unconditionally)
BRA (Computed Branch)5-32
BRA C (Branch if Carry) 5-33
BRA GE (Branch if Signed Greater Than
or Equal)5-35
BRA GEU (Branch if Unsigned Greater Than
or Equal)
BRA GT (Branch if Signed Greater Than)
BRA GTU (Branch if Unsigned Greater Than) 5-38
BRA LE (Branch if Signed Less Than or Equal) 5-39
BRA LEU (Branch if Unsigned Less Than
or Equal)5-40
BRA LT (Branch if Signed Less Than) 5-41
BRA LTU (Branch if Not Carry)
BRA LTU (Branch if Unsigned Less Than)
BRA N (Branch if Negative)
BRA NN (Branch if Not Negative)5-45
BRA NOV (Branch if Not Overflow) 5-46
BRA NZ (Branch if Not Zero)5-47
BRA OA (Branch if Overflow Accumulator A) 5-48
BRA OB (Branch if Overflow Accumulator B) 5-49
BRA OV (Branch if Overflow)
BRA SA (Branch if Saturation Accumulator A)
BRA SB (Branch if Saturation Accumulator B) 5-52
BRA Z (Branch if Zero) 5-53
BSET (Bit Set f) 5-54
BSET (Bit Set in Ws) 5-55
BSW (Bit Write in Ws)5-56
DIU (DILIOUOIEI)
BTG (Bit Toggle f)
BTG (Bit Toggle in Ws)5-59
BTG (Bit Toggle in Ws)5-59 BTSC (Bit Test f, Skip if Clear)5-60
BTG (Bit Toggle in Ws)5-59 BTSC (Bit Test f, Skip if Clear)5-60 BTSC (Bit Test Ws, Skip if Clear)
BTG (Bit Toggle in Ws)
BTG (Bit Toggle in Ws)5-59 BTSC (Bit Test f, Skip if Clear)5-60 BTSC (Bit Test Ws, Skip if Clear)
BTG (Bit Toggle in Ws)
BTG (Bit Toggle in Ws)

	F 74
BTSTS (Bit Test/Set f)	
BTSTS (Bit Test/Set in Ws)	5-72
CALL (Call Indirect Subroutine)	5-74
CALL (Call Subroutine)	
CLR (Clear Accumulator, Pre-Fetch Operands)	
CLR (Clear f or WREG)	
CLR (Clear Wd)	
CLRWDT (Clear Watchdog Timer)	5-79
COM (Complement f)	
COM (Complement Ws)	
CP (Compare f with WREG, Set Status Flags)	
CP (Compare Wb with lit5, Set Status Flags)	
CP (Compare Wb with Ws, Set Status Flags)	5-84
CP0 (Compare f with 0x0, Set Status Flags)	
CP0 (Compare Ws with 0x0, Set Status Flags)	
CPB (Compare f with WREG using Borrow,	
Set Status Flags)	5-87
CPB (Compare Wb with lit5 using Borrow,	
Set Status Flags)	5-88
CPB (Compare Ws with Wb using Borrow,	
Set Status Flags)	
CPSEQ (Compare Wb with Wn, Skip if Equal)	5-91
CPSGT (Signed Compare Wb with Wn,	
Skip if Greater Than)	5-92
CPSLT (Signed Compare Wb with Wn,	
Skip if Less Than)	5-93
CPSNE (Signed Compare Wb with Wn,	
Skip if Not Equal)	5-94
DAW.B (Decimal Adjust Wn)	5-95
	5 00
DEC (Decrement f)	5-96
DEC (Decrement Ws)	5-97
DEC2 (Decrement f by 2)	5-98
DEC2 (Decrement Ws by 2)	
DISI (Disable Interrupts Temporarily)	
DIV.S (Signed Integer Divide)	5-101
DIV.U (Unsigned Integer Divide)	5-103
DIVF (Fractional Divide)	5-105
DO (Initialize Hardware Loop Literal)	
DO (Initialize Hardware Loop Wn)	5-100
ED (Euclidean Distance, No Accumulate)	
EDAC (Euclidean Distance)	5-113
EXCH (Exchange Wns and Wnd)	5-115
FBCL (Find First Bit Change from Left)	5-116
FF1L (Find First One from Left)	
	5 1 1 9
FF1R (Find First One from Right)	5-120
GOTO (Unconditional Indirect Jump)	5-120 5-123
GOTO (Unconditional Indirect Jump)	5-120 5-123
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump)	5-120 5-123 5-122
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f)	5-120 5-123 5-122 5-124
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws)	5-120 5-123 5-122 5-124 5-125
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment f by 2)	5-120 5-123 5-122 5-124 5-125 5-126
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment f by 2) INC2 (Increment Ws by 2)	5-120 5-123 5-122 5-124 5-125 5-126 5-127
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment f by 2) INC2 (Increment Ws by 2)	5-120 5-123 5-122 5-124 5-125 5-126 5-127
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment f by 2) INC2 (Increment Ws by 2) INC2 (Inclusive OR f and WREG)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment f by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Literal and Wn)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Literal and Wn) IOR (Inclusive OR Wb and Short Literal)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-135
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-135 5-139
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-139 5-140
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-135 5-139 5-140 5-136
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right f) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-135 5-139 5-136 5-137
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right f) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-135 5-139 5-136 5-137
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws) MAC (Multiply and Accumulate)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-135 5-139 5-136 5-137 5-137 5-141
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment Ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws) MAC (Multiply and Accumulate) MAC (Square and Accumulate)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-1340 5-136 5-137 5-1341 5-143
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws) MAC (Multiply and Accumulate) MAC (Square and Accumulate) MOV (Move 16-bit Literal to Wn)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-1340 5-136 5-137 5-1341 5-143 5-150
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment fby 2) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws) MAC (Multiply and Accumulate) MAC (Square and Accumulate) MOV (Move 16-bit Literal to Wn) MOV (Move f to Destination)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-134 5-136 5-137 5-141 5-143 5-150 5-145
GOTO (Unconditional Indirect Jump) GOTO (Unconditional Jump) INC (Increment f) INC (Increment ws) INC2 (Increment Ws by 2) INC2 (Increment Ws by 2) IOR (Inclusive OR f and WREG) IOR (Inclusive OR f and WREG) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Short Literal) IOR (Inclusive OR Wb and Ws) LAC (Load Accumulator) LNK (Allocate Stack Frame) LSR (Logical Shift Right by Short Literal) LSR (Logical Shift Right by Wns) LSR (Logical Shift Right f) LSR (Logical Shift Right Ws) MAC (Multiply and Accumulate) MAC (Square and Accumulate) MOV (Move 16-bit Literal to Wn)	5-120 5-123 5-122 5-124 5-125 5-126 5-127 5-128 5-129 5-130 5-131 5-133 5-133 5-134 5-137 5-143 5-143 5-145 5-145 5-147

MOV (Move Wns to f)	5-148
MOV (Move WREG to f)	5-146
MOV (Move Ws to Wd)	
MOV (Move Ws with offset to Wnd)	5-151
MOV.B (Move 8-bit Literal to Wnd)	
MOV.D (Double-Word Move from Source	5-143
	F 4FF
to Wnd)	5-155
MOV.D (Double-Word Move from Wns	
to Destination)	5-157
MOVSAC (Pre-Fetch Operands and	
Store Accumulator)	5-159
MPY (Multiply Wm by Wn to Accumulator)	
MPY (Square to Accumulator)	
MPY.N (Multiply -Wm by Wn to Accumulator)	
MSC (Multiply and Subtract from Accumulator)	
MUL (Integer Unsigned Multiply f and WREG)	5-169
MUL.SS (Integer 16x16-bit Signed Multiply)	5-170
MUL.SU (Integer 16x16-bit	
Signed-Unsigned Multiply)	5-174
MUL.SU (Integer 16x16-bit	- 1-0
Signed-Unsigned Short Literal Multiply)	5-1/2
MUL.US (Integer 16x16-bit	
Unsigned-Signed Multiply)	5-176
MUL.UU (Integer 16x16-bit Unsigned Multiply)	
MUL.UU (Integer 16x16-bit Unsigned	
	E 170
Short Literal Multiply)	
NEG (Negate Accumulator)	
NEG (Negate f)	
NEG (Negate Ws)	5-182
NOP (No Operation)	5-184
NOPR (No Operation)	5-185
POP (Pop TOS to f)	E 100
	5-100
POP (Pop TOS to Wd)	5-187
POP.D (Double Pop TOS to Wnd/	
	5-188
Wnd+1)	
Wnd+1) POP.S (Pop Shadow Registers)	5-189
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS)	5-189 5-190
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS)	5-189 5-190
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/	5-189 5-190 5-191
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS)	5-189 5-190 5-191 5-192
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers)	5-189 5-190 5-191 5-192 5-193
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers)	5-189 5-190 5-191 5-192 5-193
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode)	5-189 5-190 5-191 5-192 5-193 5-194
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call)	5-189 5-190 5-191 5-192 5-193 5-194 5-196
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call)	5-189 5-190 5-191 5-192 5-193 5-194 5-196 5-195
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times).	5-189 5-190 5-191 5-192 5-193 5-194 5-196 5-195 5-197
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-196 5-195 5-197 5-198
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset)	5-189 5-190 5-191 5-192 5-193 5-194 5-196 5-195 5-197 5-198 5-200
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-200 5-201
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-200 5-201
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-200 5-201 5-202
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-200 5-201 5-203
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-200 5-201 5-202 5-203 5-204
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-197 5-198 5-200 5-201 5-203 5-204 5-205
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-201 5-203 5-203 5-204 5-205 5-207
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-198 5-201 5-203 5-203 5-204 5-205 5-207
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry)	5-189 5-190 5-191 5-192 5-193 5-193 5-194 5-195 5-195 5-197 5-197 5-198 5-201 5-203 5-203 5-204 5-205 5-207 5-208
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction 'lit14' Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry) RRC (Rotate Left Ws without Carry) RC (Rotate Left Ws without Carry) RC (Rotate Left Ws without Carry) RC (Rotate Left f through Carry) RC (Rotate Right f through Carry) RC (Rot	5-189 5-190 5-191 5-193 5-193 5-194 5-195 5-195 5-195 5-197 5-198 5-201 5-203 5-203 5-203 5-204 5-205 5-207 5-208 5-210
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction 'lit14' Times) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry) RRC (Rotate Left Ws without Carry) RRC (Rotate Right f through Carry) RRC (Rotate Right f through Carry) RRC (Rotate Right Ws through Carry) RC (Ro	5-189 5-190 5-191 5-193 5-193 5-194 5-195 5-195 5-195 5-197 5-198 5-201 5-203 5-203 5-203 5-204 5-205 5-207 5-208 5-211
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction 'lit14' Times) RESET (Reset) RESET (Reset) RETFIE (Return from Interrupt) RETLW (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry) RC (Rotate Right f through Carry) RC (Rotate Right f without Carry) RENC (Rotate Right f without C	5-189 5-190 5-191 5-193 5-193 5-194 5-196 5-195 5-195 5-197 5-198 5-200 5-202 5-203 5-204 5-205 5-207 5-208 5-211 5-213
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REFEAT (Repeat Next Instruction 'Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETIR (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RNNC (Rotate Right f without Carry) RN	5-189 5-190 5-191 5-193 5-193 5-194 5-196 5-195 5-197 5-197 5-197 5-201 5-202 5-203 5-204 5-205 5-207 5-208 5-211 5-214
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REFEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETIR (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLNC (Rotate Left f without Carry) RC (Rotate Right f through Carry) RC (Rotate Right f without Carry) RC (Rotate Right f without Carry) RNC (Rotate Right f without Carry) RN	5-189 5-190 5-191 5-191 5-193 5-193 5-194 5-195 5-195 5-197 5-197 5-198 5-200 5-201 5-202 5-203 5-204 5-205 5-207 5-208 5-211 5-213 5-214 5-216
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REFEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETIR (Return from Interrupt) RETURN (Return) RLC (Rotate Left f through Carry) RLC (Rotate Left Ws through Carry) RLNC (Rotate Right f through Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RNNC (Rotate Right f	5-189 5-190 5-191 5-193 5-194 5-194 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-203 5-203 5-203 5-203 5-203 5-211 5-213 5-218
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) REFEAT (Repeat Next Instruction Wn Times) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETTIW (Return) RETURN (Return) RLC (Rotate Left f through Carry) RLC (Rotate Left f without Carry) RLC (Rotate Left Ws without Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RRNC (Rotate Right Ws without Carry) RRNC (Rotate Right Ws without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SE (Sign-Extend Ws)	5-189 5-190 5-191 5-193 5-194 5-195 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-204 5-203 5-204 5-207 5-208 5-210 5-211 5-213 5-214 5-218 5-220
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) REFEAT (Repeat Next Instruction Wn Times) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETTIW (Return) RETURN (Return) RLC (Rotate Left f through Carry) RLC (Rotate Left f without Carry) RLC (Rotate Left Ws without Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RRNC (Rotate Right Ws without Carry) RRNC (Rotate Right Ws without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SE (Sign-Extend Ws)	5-189 5-190 5-191 5-193 5-194 5-195 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-204 5-203 5-204 5-207 5-208 5-210 5-211 5-213 5-214 5-218 5-220
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) REFEAT (Repeat Next Instruction Wn Times) REFEAT (Repeat Next Instruction Wn Times) RETFIE (Return from Interrupt) RETFIE (Return with Literal in Wn) RETURN (Return) RLC (Rotate Left f through Carry) RLC (Rotate Left Ws through Carry) RLC (Rotate Left Ws without Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RRNC (Rotate Right f without Carry) RRNC (Rotate Right Ws without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SETM (Set f or WREG)	5-189 5-190 5-191 5-193 5-194 5-195 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-204 5-205 5-207 5-208 5-210 5-213 5-214 5-218 5-221
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) REFEAT (Repeat Next Instruction Wn Times) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETURN (Return) RETURN (Return) RLC (Rotate Left f through Carry) RLC (Rotate Left Ws through Carry) RLC (Rotate Left Ws without Carry) RRC (Rotate Right f through Carry) RRC (Rotate Right f without Carry) RRC (Rotate Right f without Carry) RRNC (Rotate Right Ws through Carry) RRNC (Rotate Right Ws without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SETM (Set f or WREG) SETM (Set Ws)	5-189 5-190 5-191 5-193 5-193 5-194 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-203 5-204 5-207 5-208 5-210 5-213 5-214 5-218 5-221 5-222
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETURN (Return) REC (Rotate Left f through Carry) RLC (Rotate Left f through Carry) RLC (Rotate Left f without Carry) RRDC (Rotate Left Ws without Carry) RRC (Rotate Right f without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SETM (Set Ws) SETM (Set Ws) SETM (Set Ws) SETAC (Arithmetic Shift Accumulator by Slit6)	5-189 5-190 5-191 5-193 5-194 5-195 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-203 5-203 5-207 5-203 5-207 5-203 5-210 5-213 5-214 5-218 5-223 5-223
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETURN (Return) REC (Rotate Left f through Carry) RLC (Rotate Left f without Carry) RLC (Rotate Left Ws without Carry) RRC (Rotate Left Ws without Carry) RRC (Rotate Right f through Carry) RRC (Rotate Right f without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SETM (Set f or WREG) SETM (Set Ws) SFTAC (Arithmetic Shift Accumulator by Slit6) SFTAC (Arithmetic Shift Accumulator by Wb)	5-189 5-190 5-191 5-193 5-193 5-194 5-195 5-195 5-195 5-195 5-197 5-198 5-200 5-201 5-203 5-203 5-207 5-203 5-210 5-213 5-214 5-218 5-221 5-223 5-223 5-224
Wnd+1) POP.S (Pop Shadow Registers) PUSH (Push f to TOS) PUSH (Push Ws to TOS) PUSH.D (Double Push Wns/ Wns+1 to TOS) PUSH.S (Push Shadow Registers) PWRSAV (Enter Power Saving Mode) RCALL (Computed Relative Call) RCALL (Relative Call) REPEAT (Repeat Next Instruction 'lit14' Times) REPEAT (Repeat Next Instruction Wn Times) RESET (Reset) RETFIE (Return from Interrupt) RETFIE (Return from Interrupt) RETURN (Return) REC (Rotate Left f through Carry) RLC (Rotate Left f through Carry) RLC (Rotate Left f without Carry) RRDC (Rotate Left Ws without Carry) RRC (Rotate Right f without Carry) SAC (Store Accumulator) SAC.R (Store Rounded Accumulator) SETM (Set Ws) SETM (Set Ws) SETM (Set Ws) SETAC (Arithmetic Shift Accumulator by Slit6)	5-189 5-190 5-191 5-193 5-193 5-194 5-195 5-195 5-195 5-197 5-198 5-201 5-203 5-204 5-203 5-207 5-208 5-210 5-213 5-214 5-218 5-221 5-223 5-223 5-224 5-228

SL (Shift Left f)		5-225
	s)	
SUB (Subtract A	ccumulators)	5-235
	iteral from Wn)	
	Short Literal from Wb)	
	VREG from f)	
	Vs from Wb)	5-233
SUBB (Subtract	Short Literal from	
Wb with Bo	prrow)	5-238
	Wn from Literal with Borrow)	
	WREG and Carry bit from f)	
	Ws from Wb with Borrow)	
	ct f from WREG with Borrow)	5-241
SUBBR (Subtrac		
	al with Borrow)	
SUBBR (Subtrac	ct Wb from Ws with Borrow)	5-243
	f from WREG)	
	Wb from Short Literal)	
	Wb from Ws)	
	Nibble Swap Wn)	
TBLRDH (Table	Read High)	5-250
TBLRDL (Table	Read Low)	5-252
	Write High)	
	Write Low)	
	ate Stack Frame)	
	OR f and WREG)	
XOR (Exclusive	OR Literal and Wn)	5-260
XOR (Exclusive	OR Wb and Short Literal)	5-261
	OR Wb and Ws)	
	d Wn)	
	Field Descriptors Introduction	
	iew	
Bit Instructions		3-7
Compare/Skip Ir	nstructions	3-8
Compare/Skip Ir Control Instruction	nstructions ons	3-8 3-10
Compare/Skip Ir Control Instruction DSP Instructions	nstructions ons s	3-8 3-10 3-10
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru	nstructions ons s ction Groups	3-8 3-10 3-10 3-2
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction	nstructions ons s ction Groups s	3-8 3-10 3-10 3-2 3-5
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instruction	nstructions ons s ction Groups s s	3-8 3-10 3-10 3-2 3-5 3-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instruction Move Instruction	nstructions ons s ction Groups s s	3-8 3-10 3-10 3-2 3-5 3-4 3-3
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instruction Move Instruction	nstructions ons s ction Groups s s	3-8 3-10 3-10 3-2 3-5 3-4 3-3
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In	nstructions ons s ction Groups s s s nstructions	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst	nstructions ons s ction Groups s s s s nstructions ructions	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir	nstructions ons s s s nstructions nstructions nstructions	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-9 3-6 3-10
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Summ	nstructions ons s s s s nstructions nstructions nstructions nstructions nstructions nstructions	3-8 3-10 3-10 3-5 3-4 3-3 3-9 3-6 3-10 6-9
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Summ Instruction Set Symbol	nstructions ons s s s nstructions nstructions nstructions nstructions nstructions nstructions nstructions nstructions nstructions nstructions nstructions	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text	nstructions ons	3-8 3-10 3-2 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text	nstructions ons	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Summ Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Summ Instruction Set Symbol #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Summ Instruction Set Symbol #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Symbol #text (text) {}	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Instr Shadow/Stack Ir Instruction Set Symbol #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text)	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) [text] { ]	nstructions ons	3-8 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) {label:} Acc AWB bit4 Expr. f. lit1 lit10 lit14	nstructions ons	3-8 3-10 3-10 3-2 3-5 3-4 3-3 3-9 3-6 3-10 6-9 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4 1-4
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) {label:} Acc AWB bit4 Expr f. lit1 lit10 lit14 lit16	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) {label:} Acc AWB bit4 Expr. f. lit1 lit10 lit14 lit16 lit23	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) {label:} Acc AWB bit4 Expr. f. lit1 lit10 lit14 lit16 lit23	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) {label:} Acc AWB bit4 Expr. f. lit1 lit10 lit14 lit16 lit23 lit4	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) {} {label:} Acc. AWB bit4 Expr. f. lit1 lit10 lit14 lit16 lit23 lit4 lit5	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text)	nstructions	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$
Compare/Skip Ir Control Instruction DSP Instructions dsPIC30F Instru Logic Instruction Math Instructions Move Instruction Program Flow In Rotate/Shift Inst Shadow/Stack Ir Instruction Set Symbo #text (text) (text) (text) {} {label:} Acc. AWB bit4 Expr. f. lit1 lit10 lit14 lit16 lit23 lit4 lit5 lit8 Slit10	nstructions ons	$\begin{array}{c} 3-8\\ -3-10\\ -3-2\\ -3-5\\ -3-4\\ -3-3\\ -3-3\\ -3-9\\ -3-6\\ -3-10\\ -6-9\\ -1-4$

Slit41-4
Slit51-4
TOS 1-4
Wb 1-4
Wd 1-4
Wm*Wm 1-4
Wm*Wn 1-4
Wm, Wn 1-4
Wn 1-4
Wnd 1-4
Wns 1-4
WREG 1-4
Ws 1-4
Wx 1-4
Wxd 1-4
Wy 1-4
Wyd 1-4
Instruction Stalls
DO/REPEAT Loops 4-13
Exceptions
Instructions that Change Program Flow 4-13
PSV 4-13
RAW Dependency Detection 4-12
Instruction Symbols
Integer and Fractional Data 4-30
Representation 4-31
Interrupt Priority Level
Introduction 1-2

### М

MAC	
Operations	4-34
Pre-Fetch Register Updates	4-34
Pre-Fetches	4-33
Syntax	4-34
Write Back	4-34
MAC Accumulator Write Back Selection	5-5
MAC or MPY Source Operands	
(Different Working Register)	5-5
MAC or MPY Source Operands	
(Same Working Register)	5-5
Manual Objective	
Microchip Documentation	1-5
Modulo and Bit-Reversed Addressing Modes	
Multi-Cycle Instructions	3-2
Multi-Word Instructions	3-3

### Ν

Normalizing the Accumulator with the	
FBCL Instruction	4-39

Offset Addressing Modes for Wd Destination Register (with Register Offset)	5-3
Offset Addressing Modes for Ws Source Register (with Register Offset)	5-3
Ρ	
PICmicro® Microcontroller Compatibility	-28
PRODL Register Pair 4	-28
Program Addressing Modes 4	-11
Methods of Modifying Flow 4	-11
Program Counter	2-5
Program Memory Map	

# dsPIC30F Programmer's Reference Manual

т

Programmer's Model	2-3
Diagram	
Register Descriptions	2-3
PSVPAG Register	2-5

### R

RCOUNT Register2-6
Register Direct Addressing4-4
Register Indirect Addressing4-5
Modes4-5
Register Indirect Addressing and the Instruction Set4-8
Registers
CORCON (Core Control) Register2-12
SR (Status) Register2-10
Related Documents1-5

### S

Scaling Data with the FBCL Instruction	4-37
Scaling Examples	4-38
Shadow Registers	
Automatic Usage	2-9
Software Stack Frame Pointer	2-3, 4-22
Example	4-23
Overflow	4-24
Underflow	4-24
Software Stack Pointer	
Example	4-21
Stack Pointer Limit Register (SPLIM)	2-5
Status Register	2-7
DSP ALU Status Bits	
Loop Status Bits	2-7
MCU ALU Status Bits	2-7
Style and Symbol Conventions	1-3
Document Conventions	1-3

TBLPAG Register   2-5     Third Party Documentation   1-5
U
Using 10-bit Literal Operands
w
Word Move Operations       4-16         Data Alignment in Memory       4-16         Working Register Array       2-3
x
X Data Space Pre-Fetch Destination
Y
Y Data Space Pre-Fetch Destination
Z
Z Status Bit 4-26

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:\\support.microchip.com Web Address: www.microchip.com

**Atlanta** Alpharetta, GA Tel: 770-640-0034 Fax: 770-640-0307

Boston Westford, MA Tel: 978-692-3848 Fax: 978-692-3821

Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075

**Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924

Detroit Farmington Hills, MI Tel: 248-538-2250 Fax: 248-538-2260

**Kokomo** Kokomo, IN Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608

San Jose Mountain View, CA Tel: 650-215-1444 Fax: 650-961-0286

Toronto Mississauga, Ontario, Canada Tel: 905-673-0699 Fax: 905-673-6509

### ASIA/PACIFIC

Australia - Sydney Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

**China - Beijing** Tel: 86-10-8528-2100 Fax: 86-10-8528-2104

**China - Chengdu** Tel: 86-28-8676-6200 Fax: 86-28-8676-6599

**China - Fuzhou** Tel: 86-591-8750-3506 Fax: 86-591-8750-3521

China - Hong Kong SAR Tel: 852-2401-1200 Fax: 852-2401-3431

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 - Shunde** Tel: 86-757-2839-5507 Fax: 86-757-2839-5571

**China - Qingdao** Tel: 86-532-502-7355 Fax: 86-532-502-7205 ASIA/PACIFIC

India - Bangalore Tel: 91-80-2229-0061 Fax: 91-80-2229-0062

India - New Delhi Tel: 91-11-5160-8632 Fax: 91-11-5160-8632

**Japan - Kanagawa** Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea - Seoul Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934

**Singapore** Tel: 65-6334-8870 Fax: 65-6334-8850

**Taiwan - Kaohsiung** Tel: 886-7-536-4818 Fax: 886-7-536-4803

**Taiwan - Taipei** Tel: 886-2-2500-6610 Fax: 886-2-2508-0102

**Taiwan - Hsinchu** Tel: 886-3-572-9526 Fax: 886-3-572-6459

### EUROPE

Austria - Weis Tel: 43-7242-2244-399 Fax: 43-7242-2244-393 Denmark - Ballerup

Tel: 45-4420-9895 Fax: 45-4420-9910 France - Massy

Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

**Germany - Ismaning** 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

**England - Berkshire** Tel: 44-118-921-5869 Fax: 44-118-921-5820

10/20/04