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

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

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

= ISO/TS 16949 =
# Table of Contents

## SECTION 1. INTRODUCTION
- Introduction .................................................................................................................. 5
- Manual Objective .......................................................................................................... 6
- Development Support ................................................................................................. 6
- Style and Symbol Conventions ............................................................................... 7
- Instruction Set Symbols .......................................................................................... 8

## SECTION 2. PROGRAMMER’S MODEL
- 16-Bit MCU and DSC Core Architecture Overview ...................................................... 9
- Programmer’s Model .................................................................................................. 14
- Working Register Array .............................................................................................. 19
- Default Working Register (WREG) ............................................................................. 20
- Software Stack Frame Pointer .................................................................................... 20
- Software Stack Pointer ............................................................................................. 20
- Stack Pointer Limit Register (SPLIM) ........................................................................ 20
- Accumulator A and Accumulator B (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .................................................. 20
- Program Counter ....................................................................................................... 21
- TBLPAG Register ....................................................................................................... 21
- PSVPAG Register (PIC24F, PIC24H, dsPIC30F and dsPIC33F) ................................. 21
- RCOUNT Register ..................................................................................................... 21
- DCOUNT Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .......... 21
- DOSTART Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) ...... 22
- DOEND Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .......... 22
- STATUS Register ...................................................................................................... 22
- Core Control Register ............................................................................................... 25
- Shadow Registers ..................................................................................................... 25
- Stack (dsPIC33E and dsPIC33C Devices) .................................................................. 26

## SECTION 3. INSTRUCTION SET OVERVIEW
- Introduction .................................................................................................................. 39
- Instruction Set Overview ........................................................................................... 40
- Instruction Set Summary Tables .............................................................................. 42

## SECTION 4. INSTRUCTION SET DETAILS
- Data Addressing Modes ............................................................................................ 53
- Program Addressing Modes ..................................................................................... 63
- Instruction Stalls ....................................................................................................... 64
- Byte Operations ....................................................................................................... 66
- Word Move Operations ............................................................................................. 68
- Using 10-Bit Literal Operands .................................................................................. 71
- Bit Field Insert/Extract Instructions (dsPIC33C Devices Only) .............................. 71
- Software Stack Pointer and Frame Pointer ............................................................. 72
- Conditional Branch Instructions ............................................................................. 78
- Z Status Bit ............................................................................................................... 79
- Assigned Working Register Usage .......................................................................... 80
- DSP Data Formats (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .......... 83
- Accumulator Usage (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .... 85
- Accumulator Access (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) ....... 86
- DSP MAC Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .... 86
DSP Accumulator Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) ........................................... 90
Scaling Data with the FBL Instruction (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) .......................... 90
Data Range Limit Instructions (dsPIC33C Devices Only) .................................................................................. 92
Normalizing the Accumulator with the FBL Instruction (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices) ........................................... 93
Normalizing the Accumulator with the NORM Instruction (dsPIC33C Devices Only) ................................ 93
Extended Precision Arithmetic Using Mixed-Sign Multiplications (dsPIC33E and dsPIC33C Only) ............ 94

SECTION 5. INSTRUCTION DESCRIPTIONS 95
Instruction Symbols ............................................................................................................................................... 96
Instruction Encoding Field Descriptors Introduction ...................................................................................... 96
Instruction Description Example .................................................................................................................. 101
Instruction Descriptions .............................................................................................................................. 102

SECTION 6. BUILT-IN FUNCTIONS 459
Introduction ..................................................................................................................................................... 460
Built-in Function List ................................................................................................................................... 461

SECTION 7. REFERENCE 497
Instruction Bit Map ........................................................................................................................................... 498
Instruction Set Summary Table .................................................................................................................... 501
Revision History ............................................................................................................................................ 511

SECTION 8. INDEX 513

SECTION 9. WORLDWIDE SALES AND SERVICE 520
Section 1. Introduction

HIGHLIGHTS

This section of the manual contains the following major topics:

1.1 Introduction .................................................................................................................... 6
1.2 Manual Objective .......................................................................................................... 6
1.3 Development Support ................................................................................................. 6
1.4 Style and Symbol Conventions ..................................................................................... 7
1.5 Instruction Set Symbols ............................................................................................... 8
1.1 INTRODUCTION
Microchip Technology focuses on products for the embedded control market. Microchip is a leading supplier of the following devices and products:
• 8-Bit General Purpose Microcontrollers (PIC® MCUs)
• 16-Bit Digital Signal Controllers (dsPIC® DSCs)
• 16-Bit and 32-Bit Microcontrollers (MCUs)
• Specialty and Standard Nonvolatile Memory Devices
• Security Devices (KEELoQ® Security ICs)
• Application-Specific Standard Products
Information about these devices and products, with corresponding technical documentation, is available on the Microchip web site (www.microchip.com).

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

This manual does not include detailed information about the core, peripherals, system integration or device-specific information. The user should refer to the specific device family reference manual for information about the core, peripherals and system integration. For device-specific information, the user should refer to the specific device data sheets. The information that can be found in the data sheets includes:
• Device memory map
• Device pinout and packaging details
• Device electrical specifications
• List of peripherals included on the device

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

1.3 DEVELOPMENT SUPPORT
Microchip offers a wide range of development tools that allow users to efficiently develop and debug application code. Microchip’s development tools can be broken down into four categories:
• Code Generation
• Hardware/Software Debug
• Device Programmer
• Product Evaluation Boards

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

Microchip offers other reference tools to speed up the development cycle. These include:
• Application Notes
• Reference Designs
• Microchip Web Site
• Local Sales Offices with Field Application Support
• Corporate Support Line

The Microchip web site also lists other sites that may be useful references.
1.4 STYLE AND SYMBOL CONVENTIONS

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

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

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

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

<table>
<thead>
<tr>
<th>Symbol(t)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>Optional field or operation</td>
</tr>
<tr>
<td>[text]</td>
<td>The location addressed by text</td>
</tr>
<tr>
<td>(text)</td>
<td>The contents of text</td>
</tr>
<tr>
<td>#text</td>
<td>The literal defined by text</td>
</tr>
<tr>
<td>a ∈ [b, c, d]</td>
<td>“a” must be in the set of [b, c, d]</td>
</tr>
<tr>
<td>&lt;n:m&gt;</td>
<td>Register bit field</td>
</tr>
<tr>
<td>(label:)</td>
<td>Optional label name</td>
</tr>
<tr>
<td>Acc</td>
<td>Accumulator A or Accumulator B</td>
</tr>
<tr>
<td>AWB</td>
<td>Accumulator Write-Back</td>
</tr>
<tr>
<td>bit4</td>
<td>4-bit wide bit position (0:7 in Byte mode, 0:15 in Word mode)</td>
</tr>
<tr>
<td>Expr</td>
<td>Absolute address, label or expression (resolved by the linker)</td>
</tr>
<tr>
<td>f</td>
<td>File register address</td>
</tr>
<tr>
<td>lit1</td>
<td>1-bit literal (0:1)</td>
</tr>
<tr>
<td>lit4</td>
<td>4-bit literal (0:15)</td>
</tr>
<tr>
<td>lit5</td>
<td>5-bit literal (0:31)</td>
</tr>
<tr>
<td>lit8</td>
<td>8-bit literal (0:255)</td>
</tr>
<tr>
<td>lit10</td>
<td>10-bit literal (0:255 in Byte mode, 0:1023 in Word mode)</td>
</tr>
<tr>
<td>lit14</td>
<td>14-bit literal (0:16383)</td>
</tr>
<tr>
<td>lit16</td>
<td>16-bit literal (0:65535)</td>
</tr>
<tr>
<td>lit23</td>
<td>23-bit literal (0:8388607)</td>
</tr>
<tr>
<td>Sli4</td>
<td>Signed 4-bit literal (-8:7)</td>
</tr>
<tr>
<td>Sli6</td>
<td>Signed 6-bit literal (-32:31) (range is limited to -16:16)</td>
</tr>
<tr>
<td>Sli10</td>
<td>Signed 10-bit literal (-512:511)</td>
</tr>
<tr>
<td>Sli16</td>
<td>Signed 16-bit literal (-32768:32767)</td>
</tr>
<tr>
<td>TOS</td>
<td>Top-of-Stack</td>
</tr>
<tr>
<td>Wb</td>
<td>Base Working register</td>
</tr>
<tr>
<td>Wd</td>
<td>Destination Working register (Direct and Indirect Addressing)</td>
</tr>
<tr>
<td>Wdo</td>
<td>Destination Working register (Direct and Indirect Addressing, including Indirect Addressing with Offset)</td>
</tr>
<tr>
<td>Wm, Wn</td>
<td>Working register divide pair (dividend, divisor)</td>
</tr>
<tr>
<td>Wm * Wm</td>
<td>Working register multiplier pair (same source register)</td>
</tr>
<tr>
<td>Wm * Wn</td>
<td>Working register multiplier pair (different source registers)</td>
</tr>
<tr>
<td>Wn</td>
<td>Both source and destination Working register (Direct Addressing)</td>
</tr>
<tr>
<td>Wnd</td>
<td>Destination Working register (Direct Addressing)</td>
</tr>
<tr>
<td>Wns</td>
<td>Source Working register (Direct Addressing)</td>
</tr>
<tr>
<td>WREG</td>
<td>Default Working register (assigned to W0)</td>
</tr>
<tr>
<td>Ws</td>
<td>Source Working register (Direct and Indirect Addressing)</td>
</tr>
<tr>
<td>Wso</td>
<td>Source Working register (Direct and Indirect Addressing, including Indirect Addressing with Offset)</td>
</tr>
<tr>
<td>Wx</td>
<td>Source Addressing mode and Working register for X data bus prefetch</td>
</tr>
<tr>
<td>Wxd</td>
<td>Destination Working register for X data bus prefetch</td>
</tr>
<tr>
<td>Wy</td>
<td>Source Addressing mode and Working register for Y data bus prefetch</td>
</tr>
<tr>
<td>Wyd</td>
<td>Destination Working register for Y data bus prefetch</td>
</tr>
</tbody>
</table>

Note 1: The range of each symbol is instruction-dependent. Refer to Section 5. “Instruction Descriptions” for the specific instruction range.
### Section 2. Programmer’s Model

**HIGHLIGHTS**

This section of the manual contains the following major topics:

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>16-Bit MCU and DSC Core Architecture Overview</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Programmer’s Model</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Working Register Array</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Default Working Register (WREG)</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Software Stack Frame Pointer</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>Software Stack Pointer</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>Stack Pointer Limit Register (SPLIM)</td>
<td>20</td>
</tr>
<tr>
<td>2.8</td>
<td>Accumulator A and Accumulator B (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)</td>
<td>20</td>
</tr>
<tr>
<td>2.9</td>
<td>Program Counter</td>
<td>21</td>
</tr>
<tr>
<td>2.10</td>
<td>TBLPAG Register</td>
<td>21</td>
</tr>
<tr>
<td>2.11</td>
<td>PSVPAG Register (PIC24F, PIC24H, dsPIC30F and dsPIC33F)</td>
<td>21</td>
</tr>
<tr>
<td>2.12</td>
<td>RCOUNT Register</td>
<td>21</td>
</tr>
<tr>
<td>2.13</td>
<td>DCOUNT Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)</td>
<td>21</td>
</tr>
<tr>
<td>2.14</td>
<td>DOSTART Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)</td>
<td>22</td>
</tr>
<tr>
<td>2.15</td>
<td>DOEND Register (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)</td>
<td>22</td>
</tr>
<tr>
<td>2.16</td>
<td>STATUS Register</td>
<td>22</td>
</tr>
<tr>
<td>2.17</td>
<td>Core Control Register</td>
<td>25</td>
</tr>
<tr>
<td>2.18</td>
<td>Shadow Registers</td>
<td>25</td>
</tr>
<tr>
<td>2.19</td>
<td>Do Stack (dsPIC33E and dsPIC33C Devices)</td>
<td>26</td>
</tr>
</tbody>
</table>
2.1 16-BIT MCU AND DSC CORE ARCHITECTURE OVERVIEW

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

- **16-Bit Microcontrollers (MCU):**
  - PIC24F
  - PIC24H
  - PIC24E

- **16-Bit Digital Signal Controllers (DSC):**
  - dsPIC30F
  - dsPIC33F
  - dsPIC33E
  - dsPIC33C

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

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

#### 2.1.1.1 REGISTERS

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

#### 2.1.1.2 INSTRUCTION SET

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

#### 2.1.1.3 DATA SPACE ADDRESSING

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

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

#### 2.1.1.4 ADDRESSING MODES

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

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

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

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

2.1.1.6 EXCEPTION PROCESSING

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

2.1.2 PIC24E, dsPIC33E and dsPIC33C Features

In addition to the information provided in Section 2.1.1 “Features Specific to 16-Bit MCU and DSC Core”, this section describes the enhancements that are available in the PIC24E, dsPIC33E and dsPIC33C families of devices.

2.1.2.1 DATA SPACE ADDRESSING

The Base Data Space address is used in conjunction with a Read or Write Page register (DSRPAG or DSWPAG) to form an Extended Data Space (EDS) address, which can also be used for PSV access. The EDS can be addressed as 8M words or 16 Mbytes. Refer to “Data Memory” (DS70595) in the “dsPIC33/PIC24 Family Reference Manual” for more details on EDS, PSV and table accesses.

\begin{center}
\textbf{Note:} Some PIC24F devices also support Extended Data Space. Refer to “CPU with Extended Data Space (EDS)” (DS39732) and “Data Memory with Extended Data Space (EDS)” (DS39733) in the “dsPIC33/PIC24 Family Reference Manual” for details.
\end{center}

2.1.2.2 AUTOMATIC MIXED-SIGN MULTIPLICATION MODE (dsPIC33E AND dsPIC33C ONLY)

In addition to signed and unsigned DSP multiplications, dsPIC33E and dsPIC33C devices support mixed-sign (unsigned-signed and signed-unsigned) multiplications without the need to dynamically reconfigure the Multiplication mode and shift data to account for the difference in operand formats. This mode is particularly beneficial for dsPIC33C executing extended precision (32-bit and 64-bit) algorithms. Besides DSP instructions, MCU multiplication (\texttt{MUL}) instructions can also utilize either accumulator as a result destination, thereby enabling faster extended precision arithmetic. Refer to Section 4.11.1 “Implied DSP Operands (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)” and Section 4.21 “Extended Precision Arithmetic Using Mixed-Sign Multiplications (dsPIC33E and dsPIC33C Only)” for more details on mixed-sign DSP multiplications.

2.1.2.3 MCU MULTIPLICATIONS WITH 16-BIT RESULT

16x16-bit \texttt{MUL} instructions include an option to store the product in a single 16-bit Working register rather than a pair of registers. This feature helps free up a register for other purposes, in cases where the numbers being multiplied are small in magnitude, and therefore, expected to provide a 16-bit result. See the individual \texttt{MUL} instruction descriptions in Section 5.4 “Instruction Descriptions” for more details.
2.1.2.4 HARDWARE STACK FOR DO LOOPS (dsPIC33E AND dsPIC33C ONLY)

The single-level DO Loop Shadow register set has been replaced by a 4-level deep DO loop hardware stack. This provides automatic DO Loop register save/restore for up to 3 levels of DO loop nesting, resulting in more efficient implementation of nested loops. Refer to Section 2.19 “DO Stack (dsPIC33E and dsPIC33C Devices)” for more details on DO loop nesting in dsPIC33E and dsPIC33C devices.

2.1.2.5 DSP CONTEXT SWITCH SUPPORT (dsPIC33E AND dsPIC33C ONLY)

In dsPIC33E and dsPIC33C devices, the DSP Overflow and Saturation Status bits are writable. This allows the state of the DSP engine to be efficiently saved and restored while switching between DSP tasks. See Section 2.16.4 “DSP ALU Status Bits (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)” for more details on DSP Status bits. In addition, dsPIC33C devices have up to four additional sets of DSP Accumulators A and B for fast context switching. Please see the specific device data sheet for details.

2.1.2.6 EXTENDED CALL AND GOTO INSTRUCTIONS (PIC24E, dsPIC33E AND dsPIC33C ONLY)

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

2.1.2.7 COMPARE/BRANCH INSTRUCTIONS (PIC24E, dsPIC33E AND dsPIC33C ONLY)

PIC24E/dsPIC33E/dsPIC33C devices feature conditional Compare/Branch (CPBxx) instructions. These instructions extend the capabilities of the Compare/Skip (CPSxx) instructions by allowing branches, rather than only skipping over a single instruction. See the CPBEQ, CPBNE, CPBGT and CPBLT instruction descriptions in Section 5.4 “Instruction Descriptions” for more details on Compare/Branch instructions.

2.1.3 dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Features

In addition to the information provided in Section 2.1.1 “Features Specific to 16-Bit MCU and DSC Core”, this section describes the DSP enhancements that are available in the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C families of devices.

2.1.3.1 PROGRAMMING LOOP CONSTRUCTS

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

2.1.3.2 DSP INSTRUCTION CLASS

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

2.1.3.3 DATA SPACE ADDRESSING

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

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

2.1.3.5 DSP ENGINE

The DSP engine features a high-speed, 17-bit by 17-bit multiplier, a 40-bit ALU, two 40-bit saturating accumulators and a 40-bit bidirectional barrel shifter. The barrel shifter is capable of shifting a 40-bit value, up to 16 bits right or up to 16 bits left, in a single cycle. The DSP instructions operate seamlessly with all other instructions and have been designed for optimal real-time performance. The MAC instruction and other associated instructions can concurrently fetch two data operands from memory while multiplying two Working registers. This requires that the data space be split for these instructions and linear for all others. This is achieved in a transparent and flexible manner through dedicating certain Working registers to each address space.

2.1.3.6 EXCEPTION PROCESSING

The dsPIC30F devices have a vectored exception scheme with support for up to eight sources of non-maskable traps and up to 54 interrupt sources. The dsPIC33F, dsPIC33E and dsPIC33C have a similar exception scheme, but support up to 118, and up to 246 interrupt sources, respectively. In all three families, each interrupt source can be assigned to one of seven priority levels. Refer to “Interrupts” (DS70000600) of the “dsPIC33/PIC24 Family Reference Manual” for more details on exception processing.
2.2 PROGRAMMER’S MODEL

Figure 2-1 through Figure 2-5 show the programmer’s model diagrams for the 16-bit MCU and DSC families of devices.

Figure 2-1: PIC24F and PIC24H Programmer’s Model Diagram

![Programmer's Model Diagram]

- **Program Counter**
- **Data Table Page Address**
- **Program Space Visibility Page Address**
- **Repeat Loop Counter**
- **CPU Core Control Register**
- **Status Register**
- **Stack Pointer Limit Register**
- **Result Registers**
- **Working Registers**
- **Page Addresses**
- **Shadow Registers**
- **Legend**
Figure 2-2: PIC24E Programmer’s Model Diagram

- **Program Counter**
- **Status Register**
- **Working Registers**
- **Data Table Page Address**
- **Data Space Read Page Address**
- **Data Space Write Page Address**
- **repeat Loop Counter**
- **CPU Core Control Register**
- **Stack Pointer Limit Register**
- **DIV and MUL Result Registers**
- **PUSH, POP, and POP, S Shadow Registers**
- **Legend**

**Legend:**
- **CORCON**: CPU Core Control Register
- **SRH**: Status Register
- **SRL**: Status Register
- **TBLPAG**: Data Table Page Address
- **DSRPAG**: Data Space Read Page Address
- **DSWPAG**: Data Space Write Page Address
- **RCOUNT**: Repeat Loop Counter
- **SPLIM**: Stack Pointer Limit Register
### Figure 2-3: dsPIC30F and dsPIC33F Programmer’s Model Diagram

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0/WREG</td>
<td>W0/WREG</td>
<td>15:0</td>
</tr>
<tr>
<td>W1</td>
<td>W1</td>
<td>14:0</td>
</tr>
<tr>
<td>W2</td>
<td>W2</td>
<td>13:0</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
<td>12:0</td>
</tr>
<tr>
<td>W4</td>
<td>W4</td>
<td>11:0</td>
</tr>
<tr>
<td>W5</td>
<td>W5</td>
<td>10:0</td>
</tr>
<tr>
<td>W6</td>
<td>W6</td>
<td>09:0</td>
</tr>
<tr>
<td>W7</td>
<td>W7</td>
<td>08:0</td>
</tr>
<tr>
<td>W8</td>
<td>W8</td>
<td>07:0</td>
</tr>
<tr>
<td>W9</td>
<td>W9</td>
<td>06:0</td>
</tr>
<tr>
<td>W10</td>
<td>W10</td>
<td>05:0</td>
</tr>
<tr>
<td>W11</td>
<td>W11</td>
<td>04:0</td>
</tr>
<tr>
<td>W12</td>
<td>W12</td>
<td>03:0</td>
</tr>
<tr>
<td>W13</td>
<td>W13</td>
<td>02:0</td>
</tr>
<tr>
<td>W14/Frame Pointer</td>
<td>W14/Frame Pointer</td>
<td>01:0</td>
</tr>
<tr>
<td>W15/Stack Pointer</td>
<td>W15/Stack Pointer</td>
<td>00:0</td>
</tr>
<tr>
<td>SPLIM</td>
<td>SPLIM</td>
<td>39:15</td>
</tr>
<tr>
<td>ACCA</td>
<td>ACCA</td>
<td>31:15</td>
</tr>
<tr>
<td>ACCB</td>
<td>ACCB</td>
<td>30:14</td>
</tr>
<tr>
<td>DSP Accumulators</td>
<td>DSP Accumulators</td>
<td>29:13</td>
</tr>
<tr>
<td>Program Counter</td>
<td>Program Counter</td>
<td>22:0</td>
</tr>
<tr>
<td>Data Table Page Address</td>
<td>Data Table Page Address</td>
<td>21:7</td>
</tr>
<tr>
<td>Program Space Visibility Page Address</td>
<td>Program Space Visibility Page Address</td>
<td>20:7</td>
</tr>
<tr>
<td>REPEAT Loop Counter</td>
<td>REPEAT Loop Counter</td>
<td>19:0</td>
</tr>
<tr>
<td>DO Loop Counter</td>
<td>DO Loop Counter</td>
<td>18:0</td>
</tr>
<tr>
<td>DO Loop Start Address</td>
<td>DO Loop Start Address</td>
<td>17:0</td>
</tr>
<tr>
<td>DO Loop End Address</td>
<td>DO Loop End Address</td>
<td>16:0</td>
</tr>
<tr>
<td>CPU Core Control Register</td>
<td>CPU Core Control Register</td>
<td>15:0</td>
</tr>
<tr>
<td>STATUS Register</td>
<td>STATUS Register</td>
<td>7:0</td>
</tr>
</tbody>
</table>

**Legend:**
- **DIV and MUL Result Registers**
- **MAC Operand Registers**
- **MAC Address Registers**
- **Working Registers**
- **Stack Pointer Limit Register**
Section 2. Programmer’s Model

Figure 2-4: dsPIC33E Programmer’s Model Diagram

Note 1: Some dsPIC33E devices have up to four additional sets of Working registers (W0-W14) for context switching. Please see the specific device data sheet for details.
All registers in the programmer’s model are memory-mapped and can be manipulated directly by the instruction set. A description of each register is provided in Table 2-1.

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

Table 2-1: Programmer’s Model Register Descriptions

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORCON</td>
<td>CPU Core Configuration register</td>
</tr>
<tr>
<td>PC</td>
<td>23-Bit Program Counter</td>
</tr>
<tr>
<td>PSVPAG(1)</td>
<td>Program Space Visibility Page Address register</td>
</tr>
<tr>
<td>DSRPAG(2)</td>
<td>Extended Data Space (EDS) Read Page register</td>
</tr>
<tr>
<td>DSWPAG(2)</td>
<td>Extended Data Space (EDS) Write Page register</td>
</tr>
<tr>
<td>RCOUNT</td>
<td>REPEAT Loop Counter register</td>
</tr>
<tr>
<td>SPLIM</td>
<td>Stack Pointer Limit Value register</td>
</tr>
<tr>
<td>SR</td>
<td>ALU and DSP Engine STATUS Register</td>
</tr>
<tr>
<td>TBLPAG</td>
<td>Table Memory Page Address register</td>
</tr>
<tr>
<td>W0-W15(4)</td>
<td>Working register array</td>
</tr>
<tr>
<td>ACCA, ACCB(3,5)</td>
<td>40-Bit DSP Accumulators</td>
</tr>
<tr>
<td>DCOUNT(3)</td>
<td>DO Loop Counter register</td>
</tr>
<tr>
<td>DOSTART(3)</td>
<td>DO Loop Start Address register</td>
</tr>
<tr>
<td>DOEND(3)</td>
<td>DO Loop End Address register</td>
</tr>
</tbody>
</table>

Note 1: This register is only available on PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
Note 2: This register is only available on PIC24E, dsPIC33E and dsPIC33C devices.
Note 3: This register is only available on dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
Note 4: dsPIC33C devices and some dsPIC33E devices have up to four additional sets of Working registers for context switching. Please see the device data sheet for details.
Note 5: dsPIC33C devices have up to four additional sets of accumulators for context switching. Please see the device data sheet for details.

### 2.3 WORKING REGISTER ARRAY

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

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

Note: dsPIC33C devices and some dsPIC33E devices have up to four additional sets of Working registers for context switching. Please see the device data sheet to find out the exact number of register contexts available on a device. The context switching can be performed quickly using the CTXTSWP instruction.
2.4 DEFAULT WORKING REGISTER (WREG)

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

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

2.5 SOFTWARE STACK FRAME POINTER

A frame is a user-defined section of memory in the stack, used by a function to allocate memory for local variables. W14 has been assigned for use as a Stack Frame Pointer with the link (LNK) and unlink (ULNK) instructions. However, if a Stack Frame Pointer and the LNK and ULNK instructions are not used, W14 can be used by any instruction in the same manner as all other W registers. On dsPIC33E, dsPIC33C and PIC24E devices, a Stack Frame Active (SFA) Status bit is used to support nested stack frames. See Section 4.8.2 “Software Stack Frame Pointer” for detailed information about the Frame Pointer.

2.6 SOFTWARE STACK POINTER

W15 serves as a dedicated Software Stack Pointer, and will be automatically modified by function calls, exception processing and returns. However, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies reading, writing and manipulating the Stack Pointer. Refer to Section 4.8.1 “Software Stack Pointer” for detailed information about the Stack Pointer.

2.7 STACK POINTER LIMIT REGISTER (SPLIM)

The SPLIM is a 16-bit register associated with the Stack Pointer. It is used to prevent the Stack Pointer from overflowing and accessing memory beyond the user allocated region of stack memory. Refer to Section 4.8.3 “Stack Pointer Overflow” for detailed information about the SPLIM.

2.8 ACCUMULATOR A AND ACCUMULATOR B (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

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

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

Refer to Figure 4-13 for details on using ACCA and ACCB.

Note: dsPIC33C devices have up to four additional sets of accumulators for context switching. Please see the device data sheet to find out the exact number of register contexts available on a device. The context switching can be performed quickly using the CTXTSWP instruction.
2.9 PROGRAM COUNTER

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

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

2.10 TBLPAG REGISTER

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

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

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

2.12 RCOUNT REGISTER

The 14-bit RCOUNT register (16-bit for PIC24E, dsPIC33E and dsPIC33C devices) contains the loop counter for the REPEAT instruction. When a REPEAT instruction is executed, RCOUNT is loaded with the repeat count of the instruction, either “lit14” for the “REPEAT #lit14” instruction ("lit15" for the “REPEAT #lit15” instruction for PIC24E, dsPIC33E and dsPIC33C devices) or the 14 LSbs of the Wn register for the “REPEAT Wn” instruction (entire Wn for PIC24E, dsPIC33E and dsPIC33C devices). The REPEAT loop will be executed RCOUNT + 1 time.

Note 1: If a REPEAT loop is executing and gets interrupted, RCOUNT may be cleared by the Interrupt Service Routine (ISR) to break out of the REPEAT loop when the foreground code is re-entered.

2: Refer to the specific device family reference manual for complete details about REPEAT loops.

2.13 DCOUNT REGISTER (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

The 14-bit DCOUNT register (16-bit for dsPIC33E and dsPIC33C devices) contains the loop counter for hardware DO loops. When a DO instruction is executed, DCOUNT is loaded with the loop count of the instruction, either "lit14" for the "DO #lit14, Expr" instruction ("lit15" for the "DO #lit15, Expr" instruction for dsPIC33E devices) or the 14 LSbs of the Ws register for the "DO Ws, Expr" instruction (entire Wn for dsPIC33E devices). The DO loop will be executed DCOUNT + 1 time.

Note 1: In dsPIC30F and dsPIC33F devices, the DCOUNT register contains a shadow register. See Section 2.18 “Shadow Registers” for information on shadow registers.

2: The dsPIC33E devices have a 4-level deep, nested DO stack instead of a shadow register.

3: Refer to the specific device family reference manual for complete details about DO loops.
2.14 DOSTART REGISTER (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

Note 1: For dsPIC30F and dsPIC33F devices, DOSTART has a shadow register. See Section 2.18 “Shadow Registers” for information on shadowing.

2: The dsPIC33E and dsPIC33C devices have a 4-level deep, nested DO stack instead of a shadow register.

3: Refer to the specific device family reference manual for complete details about DO loops.

2.15 DOEND REGISTER (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

Note 1: For dsPIC30F and dsPIC33F devices, DOEND has a shadow register. See Section 2.18 “Shadow Registers” for information on shadow registers.

2: The dsPIC33E and dsPIC33C devices have a 4-level deep, nested DO stack instead of a shadow register.

3: Refer to the specific device family reference manual for complete details about DO loops.

2.16 STATUS REGISTER

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

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

- Register 2-1 for PIC24F, PIC24H and PIC24E devices
- Register 2-2 for dsPIC30F and dsPIC33F devices
- Register 2-3 for dsPIC33E and dsPIC33C devices

2.16.1 MCU ALU Status Bits

The MCU operation Status bits are either affected or used by the majority of instructions in the instruction set. Most of the logic, math, rotate/shift and bit instructions modify the MCU Status bits after execution, and the conditional branch instructions use the state of individual Status bits to determine the flow of program execution. All conditional branch instructions are listed in Section 4.9 “Conditional Branch Instructions”.

The Carry (C), Zero (Z), Overflow (OV), Negative (N) and Digit Carry (DC) bits show the immediate status of the MCU ALU by indicating whether an operation has resulted in a Carry, Zero, Overflow, Negative result or Digit Carry. When a subtract operation is performed, the C flag is used as a Borrow flag.
The Z Status bit is useful for extended precision arithmetic. The Z Status bit functions like a normal Z flag for all instructions except those that use a carry or borrow input (ADDC, CPB, SUBB and SUBBR). See Section 4.10 “Z Status Bit” for more detailed information.

| Note 1: All MCU bits are shadowed during execution of the PUSH.S instruction and they are restored on execution of the POP.S instruction. |
| 2: All MCU bits, except the DC flag (which is not in the SRL), are stacked during exception processing (see Section 4.8.1 “Software Stack Pointer”). |

### 2.16.2 REPEAT Loop Active (RA) Status Bit

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

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

### 2.16.3 DO Loop Active (DA) Status Bit (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)

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

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

### 2.16.4 DSP ALU Status Bits (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)

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

The OA and OB bits are used to indicate when an operation has generated an overflow into the guard bits (bits 32 through 39) of the respective accumulator. This condition can only occur when the processor is in Super Saturation mode or if saturation is disabled. It indicates that the operation has generated a number which cannot be represented with the lower 31 bits of the accumulator. The OA and OB bits are writable in dsPIC33E and dsPIC33C devices.

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

In addition, the SA and SB bits can be set by software in dsPIC33E and dsPIC33C devices, enabling efficient context state switching.

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

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

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

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

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

For all MCU and DSC devices, the 16-bit CPU Core Control register (CORCON) is used to set the configuration of the CPU. This register provides the ability to map program space into data space. In addition to setting CPU modes, the CORCON register contains status information about the I<3> Status bit, which indicates if a trap exception is being processed.

Depending on the MCU and DSC family, one of the following CORCON registers is used:
- Register 2-4 for PIC24F and PIC24H devices
- Register 2-5 for PIC24E devices
- Register 2-6 for dsPIC30F and dsPIC33F devices
- Register 2-7 for dsPIC33E and dsPIC33C devices

2.17.1 dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Specific Bits

In addition to setting CPU modes, the following features are available through the CORCON register:
- Sets the ACCA and ACCB saturation enable
- Sets the Data Space Write Saturation mode
- Sets the Accumulator Saturation and Rounding modes
- Sets the Multiplier mode for DSP operations
- Terminates DO loops prematurely
- Provides status information about the DO loop nesting level (DL<2:0>)
- Selects fixed or variable interrupt latency (dsPIC33E and dsPIC33C only)

2.17.1.1 PIC24E, dsPIC33E AND dsPIC33C SPECIFIC BITS

A Status bit (SFA) is available that indicates whether the stack frame is active.

Note: PIC24E, dsPIC33E and dsPIC33C devices do not have a PSV control bit; it has been replaced by the SFA bit.

2.18 SHADOW REGISTERS

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

Note: The DO instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

Table 2-2: Automatic Shadow Register Usage

<table>
<thead>
<tr>
<th>Location</th>
<th>DO(1)</th>
<th>POP.S/PUSH.S</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCOUNT(1)</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>DOSTART(1)</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>DOEND(1)</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>STATUS Register – DC, N, OV, Z and C bits</td>
<td>—</td>
<td>Yes</td>
</tr>
<tr>
<td>W0-W3</td>
<td>—</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Note: The DO Shadow registers are only available in dsPIC30F and dsPIC33F devices.

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

Note: All shadow registers are one register deep and not directly accessible. Additional shadowing may be performed in software using the software stack.
2.19 **DO STACK (dsPIC33E AND dsPIC33C DEVICES)**

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

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

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

The DOSTART, DOEND, and DCOUNT registers each have an associated hardware stack that allows the DO loop hardware to support up to three levels of nesting. A conceptual representation of the DO stack is shown in Figure 2-6.

**Figure 2-6: DO Stack Conceptual Diagram**

<table>
<thead>
<tr>
<th>DL&lt;2:0&gt;</th>
<th>DOSTART</th>
<th>DOEND</th>
<th>DCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>Level 1 Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Level 2 Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>Level 3 Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** For DO register entries, DL<2:0> bits represent the value before the DO stack is executed.

**Note 2:** For DO instruction buffer entries, DL<2:0> bits represent the value after the DO stack is executed.

**Note 3:** If DL<2:0> = 000, no DO loops are active (DA = 0).
### Register 2-1: SR: CPU STATUS Register (PIC24H, PIC24F and PIC24E Devices)

| Bit 15-9 | Unimplemented: Read as ‘0’ |
| Bit 8 | **DC**: MCU ALU Half Carry/Borrow bit |
| 1 | A carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred |
| 0 | No carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred |
| Bit 7-5 | **IPL<2:0>**: CPU Interrupt Priority Level Status bits$^{(1,2)}$ |
| 111 | CPU Interrupt Priority Level is 7 (15); user interrupts are disabled |
| 110 | CPU Interrupt Priority Level is 6 (14) |
| 101 | CPU Interrupt Priority Level is 5 (13) |
| 100 | CPU Interrupt Priority Level is 4 (12) |
| 011 | CPU Interrupt Priority Level is 3 (11) |
| 010 | CPU Interrupt Priority Level is 2 (10) |
| 001 | CPU Interrupt Priority Level is 1 (9) |
| 000 | CPU Interrupt Priority Level is 0 (8) |
| Bit 4 | **RA**: REPEAT Loop Active bit |
| 1 | REPEAT loop is in progress |
| 0 | REPEAT loop is not in progress |
| Bit 3 | **N**: MCU ALU Negative bit |
| 1 | Result was negative |
| 0 | Result was non-negative (zero or positive) |
| Bit 2 | **OV**: MCU ALU Overflow bit |
| This bit is used for signed arithmetic (2’s complement). It indicates an overflow of the magnitude that causes the sign bit to change state. |
| 1 | Overflow occurred for signed arithmetic (in this arithmetic operation) |
| 0 | No overflow occurred |
| Bit 1 | **Z**: MCU ALU Zero bit |
| 1 | An operation that affects the Z bit has set it at some time in the past |
| 0 | The most recent operation that affects the Z bit has cleared it (i.e., a non-zero result) |
| Bit 0 | **C**: MCU ALU Carry/Borrow bit |
| 1 | A carry-out from the MSb occurred |
| 0 | No carry-out from the MSb occurred |

**Note 1:** The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1. User interrupts are disabled when IPL<3> = 1.

**Note 2:** The IPL<2:0> Status bits are read-only when the NSTDIS bit (INTCON1<15>) = 1. Refer to the family reference manual of the specific device family to see the associated interrupt register.
### Register 2-2: SR: CPU STATUS Register (dsPIC30F and dsPIC33F Devices)

<table>
<thead>
<tr>
<th>R-0</th>
<th>R-0</th>
<th>R/C-0</th>
<th>R/C-0</th>
<th>R-0</th>
<th>R/C-0</th>
<th>R-0</th>
<th>R/W-0</th>
<th>OA</th>
<th>OB</th>
<th>SA(^{(1,2)})</th>
<th>SB(^{(1,2)})</th>
<th>OAB</th>
<th>SAB(^{(1,2,3)})</th>
<th>DA(^{(4)})</th>
<th>DC</th>
</tr>
</thead>
</table>

**Legend:**

- **C** = Clearable bit
- **R** = Readable bit
- **W** = Writable bit
- **U** = Unimplemented bit, read as '0'
- \(\neg n\) = Value at POR
  - '1' = Bit is set
  - '0' = Bit is cleared
  - \(x\) = Bit is unknown

**Note:**

1: This bit may be read or cleared, but not set.
2: Once this bit is set, it must be cleared manually by software.
3: Clearing this bit will clear SA and SB.
4: This bit is read-only.
5: The IPL\(<2:0>\) bits are concatenated with the IPL3 bit (CORCON\(<3>\) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1.
Register 2-2:  SR: CPU STATUS Register (dsPIC30F and dsPIC33F Devices) (Continued)

bit 7-5  IPL<2:0>: Interrupt Priority Level bits
        111 = CPU Interrupt Priority Level is 7 (15); user interrupts are disabled
        110 = CPU Interrupt Priority Level is 6 (14)
        101 = CPU Interrupt Priority Level is 5 (13)
        100 = CPU Interrupt Priority Level is 4 (12)
        011 = CPU Interrupt Priority Level is 3 (11)
        010 = CPU Interrupt Priority Level is 2 (10)
        001 = CPU Interrupt Priority Level is 1 (9)
        000 = CPU Interrupt Priority Level is 0 (8)

bit 4  RA: REPEAT Loop Active bit
        1 = REPEAT loop is in progress
        0 = REPEAT loop is not in progress

bit 3  N: MCU ALU Negative bit
        1 = The result of the operation was negative
        0 = The result of the operation was not negative

bit 2  OV: MCU ALU Overflow bit
        1 = Overflow occurred
        0 = No overflow occurred

bit 1  Z: MCU ALU Zero bit
        1 = The result of the operation was zero
        0 = The result of the operation was not zero

bit 0  C: MCU ALU Carry/Borrow bit
        1 = A carry-out from the MSb occurred
        0 = No carry-out from the MSb occurred

Note 1:  This bit may be read or cleared, but not set.
   2:  Once this bit is set, it must be cleared manually by software.
   3:  Clearing this bit will clear SA and SB.
   4:  This bit is read-only.
   5:  The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1.
Register 2-3: SR: CPU STATUS Register (dsPIC33E and dsPIC33C Devices)

<table>
<thead>
<tr>
<th>Bit</th>
<th>OA</th>
<th>OB</th>
<th>SA(3)</th>
<th>SB(3)</th>
<th>OAB</th>
<th>SAB</th>
<th>DA</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/C-0</td>
<td>R/C-0</td>
<td>R-0</td>
<td>R/W-0</td>
</tr>
</tbody>
</table>

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

bit 15: **OA**: Accumulator A Overflow Status bit
- 1 = Accumulator A has overflowed
- 0 = Accumulator A has not overflowed

bit 14: **OB**: Accumulator B Overflow Status bit
- 1 = Accumulator B has overflowed
- 0 = Accumulator B has not overflowed

bit 13: **SA**: Accumulator A Saturation Status bit(3)
- 1 = Accumulator A is saturated or has been saturated since this bit was last cleared
- 0 = Accumulator A is not saturated

bit 12: **SB**: Accumulator B Saturation Status bit(3)
- 1 = Accumulator B is saturated or has been saturated since this bit was last cleared
- 0 = Accumulator B is not saturated

bit 11: **OAB**: OA || OB Combined Accumulator Overflow Status bit
- 1 = Accumulator A or B has overflowed
- 0 = Neither Accumulator A nor B has overflowed

bit 10: **SAB**: SA || SB Combined Accumulator Status bit
- 1 = Accumulator A or B is saturated or has been saturated since this bit was last cleared
- 0 = Neither Accumulator A nor B is saturated

bit 9: **DA**: DO Loop Active bit
- 1 = DO loop is in progress
- 0 = DO loop is not in progress

bit 8: **DC**: MCU ALU Half Carry/Borrow bit
- 1 = A carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred
- 0 = No carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred

Note 1: The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1. User interrupts are disabled when IPL3 = 1.

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

Note 3: A data write to SR can modify the SA or SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA/SB bit write race condition, the SA and SB bits should not be modified using bit operations.
Register 2-3: SR: CPU STATUS Register (dsPIC33E and dsPIC33C Devices) (Continued)

<table>
<thead>
<tr>
<th>Bit 7-5</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
</table>
| bit 7-5 | IPL<2:0>: CPU Interrupt Priority Level Status bits\(^1,2\) | 111 = CPU Interrupt Priority Level is 7 (15); user interrupts are disabled  
110 = CPU Interrupt Priority Level is 6 (14)  
101 = CPU Interrupt Priority Level is 5 (13)  
100 = CPU Interrupt Priority Level is 4 (12)  
011 = CPU Interrupt Priority Level is 3 (11)  
010 = CPU Interrupt Priority Level is 2 (10)  
001 = CPU Interrupt Priority Level is 1 (9)  
000 = CPU Interrupt Priority Level is 0 (8) |

| Bit 4 | RA: REPEAT Loop Active bit | 1 = REPEAT loop is in progress  
0 = REPEAT loop is not in progress |

| Bit 3 | N: MCU ALU Negative bit | 1 = Result was negative  
0 = Result was non-negative (zero or positive) |

| Bit 2 | OV: MCU ALU Overflow bit | 1 = Overflow occurred for signed arithmetic (in this arithmetic operation)  
0 = No overflow occurred |

| Bit 1 | Z: MCU ALU Zero bit | 1 = The result of the operation was zero  
0 = The result of the operation was not zero |

| Bit 0 | C: MCU ALU Carry/Borrow bit | 1 = A carry-out from the MSb of the result occurred  
0 = No carry-out from the MSb of the result occurred |

**Note 1**: The IPL<2:0> bits are concatenated with the IPL3 bit (CORCON<3>) to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL, if IPL3 = 1. User interrupts are disabled when IPL3 = 1.

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

**3**: A data write to SR can modify the SA or SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA/SB bit write race condition, the SA and SB bits should not be modified using bit operations.
Register 2-4: CORCON: Core Control Register (PIC24F and PIC24H Devices)

<table>
<thead>
<tr>
<th>bit 15</th>
<th>bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>U-0</td>
<td>R/C-0</td>
</tr>
<tr>
<td>R/W-0</td>
<td>U-0</td>
</tr>
</tbody>
</table>

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

| bit 15-4 Unimplemented: Read as ‘0’ |
|-----------------|-----------------|
| bit 3 IPL3: Interrupt Priority Level 3 Status bit<br>
| 1 = CPU Interrupt Priority Level is 8 or greater (trap exception activated)<br>
| 0 = CPU Interrupt Priority Level is 7 or less (no trap exception activated)<br>
| bit 2 PSV: Program Space Visibility in Data Space Enable bit
| 1 = Program space is visible in data space<br>
| 0 = Program space is not visible in data space<br>
| bit 1-0 Unimplemented: Read as ‘0’

Note 1: This bit may be read or cleared, but not set.
2: This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.
### Register 2-5: CORCON: Core Control Register (PIC24E Devices)

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

#### bit 15
VAR: Variable Exception Processing Latency Control bit
- 1 = Variable (bounded deterministic) exception processing latency
- 0 = Fixed (fully deterministic) exception processing latency

#### bit 14-4
Unimplemented: Read as ‘0’

#### bit 3
IPL3: CPU Interrupt Priority Level Status bit 3
- 1 = CPU Interrupt Priority Level is greater than 7
- 0 = CPU Interrupt Priority Level is 7 or less

#### bit 2
SFA: Stack Frame Active Status bit
- 1 = Stack frame is active; W14 and W15 address 0x0000 to 0xFFFF, regardless of DSRPAG and DSWPAG values
- 0 = Stack frame is not active; W14 and W15 address of EDS or Base Data Space

#### bit 1-0
Unimplemented: Read as ‘0’

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

**Note 2:** The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.
## Register 2-6: CORCON: Core Control Register (dsPIC30F and dsPIC33F Devices)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SATA</td>
<td>SATB</td>
<td>SATDW</td>
<td>ACCSAT</td>
<td>IPL3</td>
<td>PSV</td>
<td>RND</td>
<td>IF</td>
<td>C</td>
<td>R</td>
<td>W</td>
<td>U</td>
<td>R</td>
<td>W</td>
<td>U</td>
<td>R</td>
</tr>
<tr>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-1</td>
<td>R/W-0</td>
<td>R/C-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>bit 15</td>
<td>bit 14</td>
<td>bit 13</td>
<td>bit 12</td>
<td>bit 11</td>
<td>bit 10</td>
<td>bit 9</td>
<td>bit 8</td>
</tr>
</tbody>
</table>

**Legend:**
- C = Clearable bit
- R = Readable bit
- W = Writable bit
- U = Unimplemented bit, read as ‘0’
- ‘1’ = Bit is set
- ‘0’ = Bit is cleared
- x = Bit is unknown

### Bit 15-13
- **Unimplemented:** Read as ‘0’

### Bit 12
- **US:** Unsigned or Signed Multiplier Mode Select bit
  - 1 = Unsigned mode enabled for DSP multiply operations
  - 0 = Signed mode enabled for DSP multiply operations

### Bit 11
- **EDT:** Early **DO** Loop Termination Control bit
  - 1 = Terminates executing **DO** loop at the end of current iteration
  - 0 = No effect

### Bit 10-8
- **DL<2:0>: DO** Loop Nesting Level Status bits
  - 111 = **DO** looping is nested at 7 levels
  - 110 = **DO** looping is nested at 6 levels
  - 110 = **DO** looping is nested at 5 levels
  - 110 = **DO** looping is nested at 4 levels
  - 011 = **DO** looping is nested at 3 levels
  - 010 = **DO** looping is nested at 2 levels
  - 001 = **DO** looping is active, but not nested (just 1 level)
  - 000 = **DO** looping is not active

### Bit 7
- **SATA:** ACCA Saturation Enable bit
  - 1 = Accumulator A saturation is enabled
  - 0 = Accumulator A saturation is disabled

### Bit 6
- **SATB:** ACCB Saturation Enable bit
  - 1 = Accumulator B saturation is enabled
  - 0 = Accumulator B saturation is disabled

### Bit 5
- **SATDW:** Data Space Write from DSP Engine Saturation Enable bit
  - 1 = Data space write saturation is enabled
  - 0 = Data space write saturation is disabled

### Bit 4
- **ACCSAT:** Accumulator Saturation Mode Select bit
  - 1 = 9.31 saturation (super saturation)
  - 0 = 1.31 saturation (normal saturation)

### Bit 3
- **IPL3:** Interrupt Priority Level 3 Status bit
  - 1 = CPU Interrupt Priority Level is 8 or greater (trap exception is activated)
  - 0 = CPU Interrupt Priority Level is 7 or less (no trap exception is activated)

**Note 1:** This bit will always read as ‘0’.
**Note 2:** DL<2:1> bits are read-only.
**Note 3:** The first two levels of **DO** loop nesting are handled by hardware.
**Note 4:** This bit may be read or cleared, but not set.
**Note 5:** This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.
Register 2-6: CORCON: Core Control Register (dsPIC30F and dsPIC33F Devices) (Continued)

bit 2  PSV: Program Space Visibility in Data Space Enable bit
       1 = Program space is visible in data space
       0 = Program space is not visible in data space

bit 1  RND: Rounding Mode Select bit
       1 = Biased (conventional) rounding is enabled
       0 = Unbiased (convergent) rounding is enabled

bit 0  IF: Integer or Fractional Multiplier Mode Select bit
       1 = Integer mode enabled for DSP multiply operations
       0 = Fractional mode enabled for DSP multiply operations

Note 1: This bit will always read as ‘0’.
2: DL<2:1> bits are read-only.
3: The first two levels of DO loop nesting are handled by hardware.
4: This bit may be read or cleared, but not set.
5: This bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.
Register 2-7: **CORCON: Core Control Register (dsPIC33E and dsPIC33C Devices)**

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R-0</th>
<th>R-0</th>
<th>R-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td></td>
<td>US1</td>
<td>US0</td>
<td>EDT(1)</td>
<td>DL2</td>
<td>DL1</td>
<td>DL0</td>
</tr>
</tbody>
</table>

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

**bit 15**
- **VAR:** Variable Exception Processing Latency Control bit
  - 1 = Variable (bounded deterministic) exception processing latency
  - 0 = Fixed (fully deterministic) exception processing latency

**bit 14**
- **Unimplemented:** Read as ‘0’

**bit 13-12**
- **US<1:0>:** DSP Multiply Unsigned/Signed Control bits
  - 11 = Reserved
  - 10 = DSP engine multiplies are mixed-sign
  - 01 = DSP engine multiplies are unsigned
  - 00 = DSP engine multiplies are signed

**bit 11**
- **EDT:** Early DO Loop Termination Control bit(1)
  - 1 = Terminates executing DO loop at end of current loop iteration
  - 0 = No effect

**bit 10-8**
- **DL<2:0>:** DO Loop Nesting Level Status bits
  - 111 = 7 DO loops are active
  - 110 = 6 DO loops are active
  - 101 = 5 DO loops are active
  - 100 = 4 DO loops are active
  - 011 = 3 DO loops are active
  - 010 = 2 DO loops are active
  - 001 = 1 DO loop is active
  - 000 = 0 DO loops are active

**bit 7**
- **SATA:** ACCA Saturation Enable bit
  - 1 = Accumulator A saturation is enabled
  - 0 = Accumulator A saturation is disabled

**bit 6**
- **SATB:** ACCB Saturation Enable bit
  - 1 = Accumulator B saturation is enabled
  - 0 = Accumulator B saturation is disabled

**bit 5**
- **SATDW:** Data Space Write from DSP Engine Saturation Enable bit
  - 1 = Data space write saturation is enabled
  - 0 = Data space write saturation is disabled

**bit 4**
- **ACCSAT:** Accumulator Saturation Mode Select bit
  - 1 = 9.31 saturation (super saturation)
  - 0 = 1.31 saturation (normal saturation)

**bit 3**
- **IPL3:** CPU Interrupt Priority Level Status bit 3(2,3)
  - 1 = CPU Interrupt Priority Level is greater than 7
  - 0 = CPU Interrupt Priority Level is 7 or less

**Note 1:** This bit always reads as ‘0’.
**Note 2:** This bit may be read or cleared, but not set.
**Note 3:** The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU interrupt priority level.
Register 2-7: CORCON: Core Control Register (dsPIC33E and dsPIC33C Devices) (Continued)

bit 2  **SFA**: Stack Frame Active Status bit
      1 = Stack frame is active; W14 and W15 address 0x0000 to 0xFFFF, regardless of DSRPAG and DSWPAG values
      0 = Stack frame is not active; W14 and W15 address of EDS or Base Data Space

bit 1  **RND**: Rounding Mode Select bit
      1 = Biased (conventional) rounding is enabled
      0 = Unbiased (convergent) rounding is enabled

bit 0  **IF**: Integer or Fractional Multiplier Mode Select bit
      1 = Integer mode is enabled for DSP multiply
      0 = Fractional mode is enabled for DSP multiply

**Note 1:** This bit always reads as '0'.
**2:** This bit may be read or cleared, but not set.
**3:** The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU interrupt priority level.
Section 3. Instruction Set Overview

HIGHLIGHTS

This section of the manual contains the following major topics:

3.1 Introduction ................................................................................................................ 40
3.2 Instruction Set Overview ............................................................................................. 40
3.3 Instruction Set Summary Tables .................................................................................... 42
3.1 INTRODUCTION

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

3.2 INSTRUCTION SET OVERVIEW

Depending on the device family, the 16-bit MCU and DSC instruction set contains up to 105 instructions, which can be grouped into the functional categories shown in Table 3-1. Table 1-2 defines the symbols used in the instruction summary tables. Table 3-2 through Table 3-11 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.

Table 3-1: Instruction Groups

<table>
<thead>
<tr>
<th>Functional Group</th>
<th>Summary Table</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move Instructions</td>
<td>Table 3-2</td>
<td>42</td>
</tr>
<tr>
<td>Math Instructions</td>
<td>Table 3-3</td>
<td>43</td>
</tr>
<tr>
<td>Logic Instructions</td>
<td>Table 3-4</td>
<td>45</td>
</tr>
<tr>
<td>Rotate/Shift Instructions</td>
<td>Table 3-5</td>
<td>46</td>
</tr>
<tr>
<td>Bit Instructions</td>
<td>Table 3-6</td>
<td>47</td>
</tr>
<tr>
<td>Compare/Skip and Compare/Branch Instructions</td>
<td>Table 3-7</td>
<td>48</td>
</tr>
<tr>
<td>Program Flow Instructions</td>
<td>Table 3-8</td>
<td>49</td>
</tr>
<tr>
<td>Shadow/Stack Instructions</td>
<td>Table 3-9</td>
<td>51</td>
</tr>
<tr>
<td>Control Instructions</td>
<td>Table 3-10</td>
<td>51</td>
</tr>
<tr>
<td>DSP Instructions(1)</td>
<td>Table 3-11</td>
<td>52</td>
</tr>
</tbody>
</table>

Note 1: DSP instructions are only available in the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C device families.

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

As shown in the instruction summary tables, 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 two 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 two cycles to execute, with the extra cycle executed as a **NOP**. Compare/Skip instructions, which skip over a two-word instruction, require three instruction cycles to execute, with two cycles executed as a **NOP**. Compare/Branch instructions (dsPIC33E/dsPIC33F/PIC24E devices only) require five instruction cycles to execute when the branch is taken.
- The **RETFIE**, **RETLW** and **RETURN** are a special case of an instruction that changes the Program Counter. These execute in three cycles, unless an exception is pending, and then they execute in two cycles.

**Note:** The **DO** and **DIVF** instructions are only available in the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C device families.

3.2.2 Multiword Instructions

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

---

**Note:** The **DO** and **DIVF** instructions are only available in the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C device families.
### 3.3 INSTRUCTION SET SUMMARY TABLES

#### Table 3-2: Move Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCH Wns,Wnd</td>
<td>Swap Wns and Wnd</td>
<td>1</td>
<td>1</td>
<td>254</td>
</tr>
<tr>
<td>LDSLV Wns,Wnd,#lit2[5]</td>
<td>Move a single instruction word from Master to Slave PRAM</td>
<td>1</td>
<td>1</td>
<td>279</td>
</tr>
<tr>
<td>MOV f (,WREG)[1]</td>
<td>Move f to destination</td>
<td>1</td>
<td>1</td>
<td>299</td>
</tr>
<tr>
<td>MOV WREG,f</td>
<td>Move WREG to f</td>
<td>1</td>
<td>1</td>
<td>300</td>
</tr>
<tr>
<td>MOV f,Wnd</td>
<td>Move f to Wnd</td>
<td>1</td>
<td>1(4)</td>
<td>301</td>
</tr>
<tr>
<td>MOV Wns,f</td>
<td>Move Wns to f</td>
<td>1</td>
<td>1</td>
<td>302</td>
</tr>
<tr>
<td>MOV.B #lit8,Wnd</td>
<td>Move 8-bit literal to Wnd</td>
<td>1</td>
<td>1</td>
<td>303</td>
</tr>
<tr>
<td>MOV #lit16,Wnd</td>
<td>Move 16-bit literal to Wnd</td>
<td>1</td>
<td>1</td>
<td>304</td>
</tr>
<tr>
<td>MOV [Ws+Slit10],Wnd</td>
<td>Move [Ws with offset] to Wnd</td>
<td>1</td>
<td>1(4)</td>
<td>305</td>
</tr>
<tr>
<td>MOV Wns,[Wd+Slit10]</td>
<td>Move Wns to [Wd with offset]</td>
<td>1</td>
<td>1</td>
<td>306</td>
</tr>
<tr>
<td>MOV Ws,Wd</td>
<td>Move Ws to Wd</td>
<td>1</td>
<td>1(4)</td>
<td>307</td>
</tr>
<tr>
<td>MOVPAG #lit10,DSRPAG[2]</td>
<td>Move 10-bit literal to DSRPAG</td>
<td>1</td>
<td>1</td>
<td>311</td>
</tr>
<tr>
<td>MOVPAG #lit9,DSWPAG[2]</td>
<td>Move 9-bit literal to DSWPAG</td>
<td>1</td>
<td>1</td>
<td>311</td>
</tr>
<tr>
<td>MOVPAG #lit8,TBLPAG[2]</td>
<td>Move 8-bit literal to TBLPAG</td>
<td>1</td>
<td>1</td>
<td>311</td>
</tr>
<tr>
<td>MOVPAG Wn, DSRPAG[2]</td>
<td>Move Wn to DSRPAG</td>
<td>1</td>
<td>1</td>
<td>312</td>
</tr>
<tr>
<td>SWAP Wn</td>
<td>Wn = byte or nibble swap Wn</td>
<td>1</td>
<td>1</td>
<td>439</td>
</tr>
<tr>
<td>TBLRDH [Ws],Wd</td>
<td>Read high program word to Wd</td>
<td>1</td>
<td>2(4)</td>
<td>440</td>
</tr>
<tr>
<td>TBLRDL [Ws],Wd</td>
<td>Read low program word to Wd</td>
<td>1</td>
<td>2(4)</td>
<td>442</td>
</tr>
<tr>
<td>TBLWTH Ws,[Wd]</td>
<td>Write Ws to high program word</td>
<td>1</td>
<td>2(4)</td>
<td>444</td>
</tr>
<tr>
<td>TBLWTL Ws,[Wd]</td>
<td>Write Ws to low program word</td>
<td>1</td>
<td>2(4)</td>
<td>446</td>
</tr>
<tr>
<td>VFSLV Wns,Wnd,#lit2[5]</td>
<td>Verify Slave processor program RAM</td>
<td>1</td>
<td>1</td>
<td>450</td>
</tr>
</tbody>
</table>

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

2: The MOVPAG instruction is only available in PIC24E, dsPIC33E and dsPIC33C devices.

3: In dsPIC33E and PIC24E devices, and in dsPIC33C Master cores, these instructions require three additional cycles – compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices and in dsPIC33C Slave cores.

4: In dsPIC33E, dsPIC33C and PIC24E devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

5: These instructions are only available in dsPIC33C devices.
## Table 3-3: Math Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD f {,WREG}</td>
<td>Destination = f + WREG</td>
<td>1</td>
<td>1(6)</td>
<td>102</td>
</tr>
<tr>
<td>ADD #lit10,Wn</td>
<td>Wn = lit10 + Wn</td>
<td>1</td>
<td>1</td>
<td>103</td>
</tr>
<tr>
<td>ADD Wb,#lit5,Wd</td>
<td>Wd = Wb + lit5</td>
<td>1</td>
<td>1</td>
<td>104</td>
</tr>
<tr>
<td>ADD Wb,ws,Wd</td>
<td>Wd = Wb + Ws</td>
<td>1</td>
<td>1(5)</td>
<td>105</td>
</tr>
<tr>
<td>ADDC f {,WREG}</td>
<td>Destination = f + WREG + (C)</td>
<td>1</td>
<td>1(6)</td>
<td>110</td>
</tr>
<tr>
<td>ADDC #lit10,Wn</td>
<td>Wn = lit10 + Wn + (C)</td>
<td>1</td>
<td>1</td>
<td>111</td>
</tr>
<tr>
<td>ADDC Wb,#lit5,Wd</td>
<td>Wd = Wb + lit5 + (C)</td>
<td>1</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>ADDC Wb,ws,Wd</td>
<td>Wd = Wb + Ws + (C)</td>
<td>1</td>
<td>1(6)</td>
<td>114</td>
</tr>
<tr>
<td>DAW.B Wn</td>
<td>Wn = decimal adjust Wn</td>
<td>1</td>
<td>1</td>
<td>225</td>
</tr>
<tr>
<td>DEC f {,WREG}</td>
<td>Destination = f – 1</td>
<td>1</td>
<td>1(6)</td>
<td>226</td>
</tr>
<tr>
<td>DEC Ws,Wd</td>
<td>Wd = Ws – 1</td>
<td>1</td>
<td>1</td>
<td>227</td>
</tr>
<tr>
<td>DEC2 f {,WREG}</td>
<td>Destination = f – 2</td>
<td>1</td>
<td>1(6)</td>
<td>229</td>
</tr>
<tr>
<td>DEC2 Ws,Wd</td>
<td>Wd = Ws – 2</td>
<td>1</td>
<td>1(6)</td>
<td>230</td>
</tr>
<tr>
<td>DIV.S Wm,Wn</td>
<td>Signed 16/16-bit integer divide, Q → W0, R → W1</td>
<td>1</td>
<td>18/6(2)</td>
<td>233</td>
</tr>
<tr>
<td>DIV.U Wm,Wn</td>
<td>Unsigned 16/16-bit integer divide, Q – W0, R → W1</td>
<td>1</td>
<td>18/6(2)</td>
<td>235</td>
</tr>
<tr>
<td>DIVF Wm,Wn</td>
<td>Signed 16/16-bit fractional divide, Q – W0, R → W1</td>
<td>1</td>
<td>18/6(2)</td>
<td>236</td>
</tr>
<tr>
<td>DIVF2 Wm,Wn(6)</td>
<td>Signed 16/16-bit fractional divide (W1:W0 preserved)</td>
<td>1</td>
<td>6</td>
<td>238</td>
</tr>
<tr>
<td>DIV2.S Wm,Wn(6)</td>
<td>Signed 16/16-bit fractional divide (W1:W0 preserved)</td>
<td>1</td>
<td>6</td>
<td>240</td>
</tr>
<tr>
<td>DIV2.U Wm,Wn(6)</td>
<td>Unsigned 16/16-bit integer divide (W1:W0 preserved)</td>
<td>1</td>
<td>6</td>
<td>241</td>
</tr>
<tr>
<td>FLIM Wb,ws</td>
<td>Force data (upper and lower) range limit without limit excess result</td>
<td>1</td>
<td>1</td>
<td>261</td>
</tr>
<tr>
<td>FLIM.V Wb,ws,Wnd(6)</td>
<td>Force data (upper and lower) range limit with limit excess result</td>
<td>1</td>
<td>1</td>
<td>262</td>
</tr>
<tr>
<td>INC f {,WREG}</td>
<td>Destination = f + 1</td>
<td>1</td>
<td>1(6)</td>
<td>267</td>
</tr>
<tr>
<td>INC Ws,Wd</td>
<td>Wd = Ws + 1</td>
<td>1</td>
<td>1</td>
<td>268</td>
</tr>
<tr>
<td>INC2 f {,WREG}</td>
<td>Destination = f + 2</td>
<td>1</td>
<td>1(6)</td>
<td>269</td>
</tr>
<tr>
<td>INC2 Ws,Wd</td>
<td>Wd = Ws + 2</td>
<td>1</td>
<td>1(6)</td>
<td>270</td>
</tr>
<tr>
<td>MUL f</td>
<td>W3:W2 = f * WREG</td>
<td>1</td>
<td>1(6)</td>
<td>323</td>
</tr>
<tr>
<td>MUL.SS Wb,ws,Wnd</td>
<td>(Wnd + 1,Wnd) = signed(Wb) * signed(Ws)</td>
<td>1</td>
<td>1(6)</td>
<td>325</td>
</tr>
<tr>
<td>MUL.SS Wb,ws,Acc(4)</td>
<td>Accumulator = signed(Wb) * signed(Ws)</td>
<td>1</td>
<td>1(6)</td>
<td>327</td>
</tr>
<tr>
<td>MUL.SU Wb,#lit5,Wnd</td>
<td>(Wnd + 1, Wnd) = signed(Wb) * unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>328</td>
</tr>
</tbody>
</table>

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

**2:** In PIC24F, PIC24H, PIC24E, dsPIC30F, dsPIC33F and dsPIC33E devices, the divide instructions must be preceded with a "REPEAT #17" instruction, such that they are executed 18 consecutive times, thus taking 18 instruction cycles. In dsPIC33C devices, the divide instructions must be preceded with a "REPEAT #5" instruction, such that they are executed six consecutive times, thus taking six instruction cycles.

**3:** These instructions are only available in PIC24E, dsPIC33E and dsPIC33C devices.

**4:** These instructions are only available in dsPIC33E and dsPIC33C devices.

**5:** In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.

**6:** These instructions are only available in dsPIC33C devices.
### Table 3-3: Math Instructions (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL.SU Wb,Ws,Wn</td>
<td>( \text{(Wnd + 1, Wnd)} = \text{signed(Wb)} \times \text{unsigned(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>329</td>
</tr>
<tr>
<td>MUL.SU Wb,Ws,Acc</td>
<td>Accumulator = signed(Wb) \times unsigned(Ws)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>331</td>
</tr>
<tr>
<td>MUL.SU Wb,#lit5,Acc</td>
<td>Accumulator = signed(Wb) \times unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>332</td>
</tr>
<tr>
<td>MUL.US Wb,Ws,Wn</td>
<td>( \text{(Wnd + 1, Wnd)} = \text{unsigned(Wb)} \times \text{signed(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>333</td>
</tr>
<tr>
<td>MUL.US Wb,Ws,Acc</td>
<td>Accumulator = unsigned(Wb) \times signed(Ws)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>335</td>
</tr>
<tr>
<td>MUL.UU Wb,#lit5,Wn</td>
<td>( \text{(Wnd + 1, Wnd)} = \text{unsigned(Wb)} \times \text{unsigned(lit5)} )</td>
<td>1</td>
<td>1</td>
<td>336</td>
</tr>
<tr>
<td>MUL.UU Wb,Ws,Wn</td>
<td>( \text{(Wnd + 1, Wnd)} = \text{unsigned(Wb)} \times \text{unsigned(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>337</td>
</tr>
<tr>
<td>MUL.UU Wb,Ws,Acc</td>
<td>Accumulator = unsigned(Wb) \times unsigned(Ws)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>339</td>
</tr>
<tr>
<td>MUL.W.SS Wb,Ws,Wn</td>
<td>( \text{Wnd} = \text{signed(Wb)} \times \text{signed(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>340</td>
</tr>
<tr>
<td>MULW.SU Wb,Ws,Wn</td>
<td>( \text{Wnd} = \text{signed(Wb)} \times \text{unsigned(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>341</td>
</tr>
<tr>
<td>MULW.SU Wb,#lit5,Wn</td>
<td>( \text{Wnd} = \text{signed(Wb)} \times \text{unsigned(lit5)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>343</td>
</tr>
<tr>
<td>MULW.US Wb,Ws,Wn</td>
<td>( \text{Wnd} = \text{unsigned(Wb)} \times \text{signed(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>345</td>
</tr>
<tr>
<td>MULW.US Wb,#lit5,Wn</td>
<td>( \text{Wnd} = \text{unsigned(Wb)} \times \text{unsigned(lit5)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>346</td>
</tr>
<tr>
<td>MULW.UU Wb,Ws,Wn</td>
<td>( \text{Wnd} = \text{unsigned(Wb)} \times \text{unsigned(Ws)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>348</td>
</tr>
<tr>
<td>MULW.UU Wb,#lit5,Wn</td>
<td>( \text{Wnd} = \text{unsigned(Wb)} \times \text{unsigned(lit5)} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>349</td>
</tr>
<tr>
<td>SE Ws,Wn</td>
<td>( \text{Wnd} = \text{sign-extended Ws} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>406</td>
</tr>
<tr>
<td>SUB f {,WREG}</td>
<td>Destination = ( f - \text{WREG} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>418</td>
</tr>
<tr>
<td>SUB #lit10,Wn</td>
<td>( Wn = Wn - \text{lit10} )</td>
<td>1</td>
<td>1</td>
<td>419</td>
</tr>
<tr>
<td>SUB Wb,#lit5,Wd</td>
<td>( Wd = Wb - \text{lit5} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>420</td>
</tr>
<tr>
<td>SUB Wb,Ws,Wd</td>
<td>( Wd = Wb - Ws )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>421</td>
</tr>
<tr>
<td>SUBB f {,WREG}</td>
<td>Destination = ( f - \text{WREG} ) – ( C )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>424</td>
</tr>
<tr>
<td>SUBB #lit10,Wn</td>
<td>( Wn = Wn - \text{lit10} - (C) )</td>
<td>1</td>
<td>1</td>
<td>425</td>
</tr>
<tr>
<td>SUBB Wb,#lit5,Wd</td>
<td>( Wd = Wb - \text{lit5} - (C) )</td>
<td>1</td>
<td>1</td>
<td>426</td>
</tr>
<tr>
<td>SUBB Wb,Ws,Wd</td>
<td>( Wd = Wb - Ws - (C) )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>428</td>
</tr>
<tr>
<td>SUBBR f {,WREG}</td>
<td>Destination = ( \text{WREG} - f ) – ( C )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>430</td>
</tr>
<tr>
<td>SUBBR Wb,#lit5,Wd</td>
<td>( Wd = \text{lit5} - Wb - (C) )</td>
<td>1</td>
<td>1</td>
<td>431</td>
</tr>
<tr>
<td>SUBBR Wb,Ws,Wd</td>
<td>( Wd = Ws - Wb - (C) )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>433</td>
</tr>
<tr>
<td>SUBR f {,WREG}</td>
<td>Destination = ( \text{WREG} - f )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>435</td>
</tr>
<tr>
<td>SUBR Wb,#lit5,Wd</td>
<td>( Wd = \text{lit5} - Wb )</td>
<td>1</td>
<td>1</td>
<td>436</td>
</tr>
<tr>
<td>ZER Ws,Wd</td>
<td>( Wd = \text{zero-extended Ws} )</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>437</td>
</tr>
</tbody>
</table>

**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.

**Note 2:** In PIC24F, PIC24H, PIC24E, dsPIC30F, dsPIC33F and dsPIC33E devices, the divide instructions must be preceded with a “REPEAT #17” instruction, such that they are executed 18 consecutive times, thus taking 18 instruction cycles. In dsPIC33C devices, the divide instructions must be preceded with a “REPEAT #5” instruction, such that they are executed six consecutive times, thus taking six instruction cycles.

**Note 3:** These instructions are only available in PIC24E, dsPIC33E and dsPIC33C devices.

**Note 4:** These instructions are only available in dsPIC33E and dsPIC33C devices.

**Note 5:** In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.

**Note 6:** These instructions are only available in dsPIC33C devices.
### Table 3-4: Logic Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND f {,WREG}</td>
<td>Destination = f .AND. WREG</td>
<td>1</td>
<td>1(2)</td>
<td>116</td>
</tr>
<tr>
<td>AND #lit10,Wn</td>
<td>Wn = lit10 .AND. Wn</td>
<td>1</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>AND Wb,#lit5,Wd</td>
<td>Wd = Wb .AND. lit5</td>
<td>1</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>AND Wb,Ws,Wd</td>
<td>Wd = Wb .AND. Ws</td>
<td>1</td>
<td>1(2)</td>
<td>119</td>
</tr>
<tr>
<td>CLR f</td>
<td>f = 0x0000</td>
<td>1</td>
<td>1</td>
<td>192</td>
</tr>
<tr>
<td>CLR WREG</td>
<td>WREG = 0x0000</td>
<td>1</td>
<td>1</td>
<td>192</td>
</tr>
<tr>
<td>CLR Wd</td>
<td>Wd = 0x0000</td>
<td>1</td>
<td>1</td>
<td>193</td>
</tr>
<tr>
<td>COM f {,WREG}</td>
<td>Destination = f</td>
<td>1</td>
<td>1(2)</td>
<td>197</td>
</tr>
<tr>
<td>COM Ws,Wd</td>
<td>Wd = Ws</td>
<td>1</td>
<td>1(2)</td>
<td>198</td>
</tr>
<tr>
<td>IOR f {,WREG}</td>
<td>Destination = f .IOR. WREG</td>
<td>1</td>
<td>1(2)</td>
<td>271</td>
</tr>
<tr>
<td>IOR #lit10,Wn</td>
<td>Wn = lit10 .IOR. Wn</td>
<td>1</td>
<td>1</td>
<td>272</td>
</tr>
<tr>
<td>IOR Wb,#lit5,Wd</td>
<td>Wd = Wb .IOR. lit5</td>
<td>1</td>
<td>1</td>
<td>273</td>
</tr>
<tr>
<td>IOR Wb,Ws,Wd</td>
<td>Wd = Wb .IOR. Ws</td>
<td>1</td>
<td>1(2)</td>
<td>274</td>
</tr>
<tr>
<td>NEG f {,WREG}</td>
<td>Destination = f + 1</td>
<td>1</td>
<td>1(2)</td>
<td>350</td>
</tr>
<tr>
<td>NEG Ws,Wd</td>
<td>Wd = Ws + 1</td>
<td>1</td>
<td>1(2)</td>
<td>351</td>
</tr>
<tr>
<td>SETM f</td>
<td>f = 0xFFFF</td>
<td>1</td>
<td>1</td>
<td>408</td>
</tr>
<tr>
<td>SETM WREG</td>
<td>WREG = 0xFFFF</td>
<td>1</td>
<td>1</td>
<td>409</td>
</tr>
<tr>
<td>SETM Wd</td>
<td>Wd = 0xFFFF</td>
<td>1</td>
<td>1</td>
<td>409</td>
</tr>
<tr>
<td>XOR f {,WREG}</td>
<td>Destination = f .XOR. WREG</td>
<td>1</td>
<td>1(2)</td>
<td>451</td>
</tr>
<tr>
<td>XOR #lit10,Wn</td>
<td>Wn = lit10 .XOR. Wn</td>
<td>1</td>
<td>1</td>
<td>452</td>
</tr>
<tr>
<td>XOR Wb,#lit5,Wd</td>
<td>Wd = Wb .XOR. lit5</td>
<td>1</td>
<td>1</td>
<td>453</td>
</tr>
<tr>
<td>XOR Wb,Ws,Wd</td>
<td>Wd = Wb .XOR. Ws</td>
<td>1</td>
<td>1(2)</td>
<td>454</td>
</tr>
</tbody>
</table>

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

**2:** In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
Table 3-5: Rotate/Shift Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR f {,WREG}</td>
<td>Destination = arithmetic right shift f, LSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>121</td>
</tr>
<tr>
<td>ASR Ws,Wd</td>
<td>Wd = arithmetic right shift Ws, LSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>123</td>
</tr>
<tr>
<td>ASR Wb,#lit4,Wnd</td>
<td>Wnd = arithmetic right shift Wb by lit4</td>
<td>1</td>
<td>1</td>
<td>125</td>
</tr>
<tr>
<td>ASR Wb,Wns,Wnd</td>
<td>Wnd = arithmetic right shift Wb by Wns</td>
<td>1</td>
<td>1</td>
<td>126</td>
</tr>
<tr>
<td>LSR f {,WREG}</td>
<td>Destination = logical right shift f, LSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>282</td>
</tr>
<tr>
<td>LSR Ws,Wd</td>
<td>Wd = logical right shift Ws, LSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>284</td>
</tr>
<tr>
<td>LSR Wb,#lit4,Wnd</td>
<td>Wnd = logical right shift Wb by lit4</td>
<td>1</td>
<td>1</td>
<td>286</td>
</tr>
<tr>
<td>LSR Wb,Wns,Wnd</td>
<td>Wnd = logical right shift Wb by Wns</td>
<td>1</td>
<td>1</td>
<td>287</td>
</tr>
<tr>
<td>RLC f {,WREG}</td>
<td>Destination = rotate left through Carry f</td>
<td>1</td>
<td>1(2)</td>
<td>388</td>
</tr>
<tr>
<td>RLC Ws,Wd</td>
<td>Wd = rotate left through Carry Ws</td>
<td>1</td>
<td>1(2)</td>
<td>389</td>
</tr>
<tr>
<td>RLNC f {,WREG}</td>
<td>Destination = rotate left (no Carry) f</td>
<td>1</td>
<td>1(2)</td>
<td>391</td>
</tr>
<tr>
<td>RLNC Ws,Wd</td>
<td>Wd = rotate left (no Carry) Ws</td>
<td>1</td>
<td>1(2)</td>
<td>392</td>
</tr>
<tr>
<td>RRC f {,WREG}</td>
<td>Destination = rotate right through Carry f</td>
<td>1</td>
<td>1(2)</td>
<td>394</td>
</tr>
<tr>
<td>RRC Ws,Wd</td>
<td>Wd = rotate right through Carry Ws</td>
<td>1</td>
<td>1(2)</td>
<td>396</td>
</tr>
<tr>
<td>RRNC f {,WREG}</td>
<td>Destination = rotate right (no Carry) f</td>
<td>1</td>
<td>1(2)</td>
<td>398</td>
</tr>
<tr>
<td>RRNC Ws,Wd</td>
<td>Wd = rotate right (no Carry) Ws</td>
<td>1</td>
<td>1(2)</td>
<td>399</td>
</tr>
<tr>
<td>SL f {,WREG}</td>
<td>Destination = left shift f, MSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>412</td>
</tr>
<tr>
<td>SL Ws,Wd</td>
<td>Wd = left shift Ws, MSb → C</td>
<td>1</td>
<td>1(2)</td>
<td>414</td>
</tr>
<tr>
<td>SL Wb,#lit4,Wnd</td>
<td>Wnd = left shift Wb by lit4</td>
<td>1</td>
<td>1</td>
<td>416</td>
</tr>
<tr>
<td>SL Wb,Wns,Wnd</td>
<td>Wnd = left shift Wb by Wns</td>
<td>1</td>
<td>1</td>
<td>417</td>
</tr>
</tbody>
</table>

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.

Note 2: In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
### Table 3-6: Bit Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles (1)</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCLR f,#bit4</td>
<td>Bit clear in f</td>
<td>1</td>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>BCLR Ws,#bit4</td>
<td>Bit clear in Ws</td>
<td>1</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>BFEXT #bit4,#wid5,Ws,Wb (2)</td>
<td>Bit field extract from Ws to Wb</td>
<td>2</td>
<td>2</td>
<td>130</td>
</tr>
<tr>
<td>BFEXT #bit4,#wid5,f,Wb (2)</td>
<td>Bit field extract from f to Wb</td>
<td>2</td>
<td>2</td>
<td>131</td>
</tr>
<tr>
<td>BFINS #bit4,#wid5,Wb,Ws (2)</td>
<td>Bit field insert from Wb into Ws</td>
<td>2</td>
<td>2</td>
<td>132</td>
</tr>
<tr>
<td>BFINS #bit4,#wid5,Wb,#lit8,Ws (2)</td>
<td>Bit field insert from #lit8 into Ws</td>
<td>2</td>
<td>2</td>
<td>133</td>
</tr>
<tr>
<td>BSET f,#bit4</td>
<td>Bit set in f</td>
<td>1</td>
<td>1</td>
<td>160</td>
</tr>
<tr>
<td>BSET Ws,#bit4</td>
<td>Bit set in Ws</td>
<td>1</td>
<td>1</td>
<td>161</td>
</tr>
<tr>
<td>BSW Ws,Wb</td>
<td>Write C bit to Ws&lt;Wb&gt;</td>
<td>1</td>
<td>1</td>
<td>163</td>
</tr>
<tr>
<td>BTG f,#bit4</td>
<td>Bit toggle in f</td>
<td>1</td>
<td>1</td>
<td>165</td>
</tr>
<tr>
<td>BTG Ws,#bit4</td>
<td>Bit toggle in Ws</td>
<td>1</td>
<td>1</td>
<td>166</td>
</tr>
<tr>
<td>BTST f,#bit4</td>
<td>Bit test in f</td>
<td>1</td>
<td>1</td>
<td>175</td>
</tr>
<tr>
<td>BTST Ws,#bit4</td>
<td>Bit test in Ws</td>
<td>1</td>
<td>1</td>
<td>176</td>
</tr>
<tr>
<td>BTST Ws,Wb</td>
<td>Bit test in Ws</td>
<td>1</td>
<td>1</td>
<td>178</td>
</tr>
<tr>
<td>BTSTS f,#bit4</td>
<td>Bit test f to Z, then set f</td>
<td>1</td>
<td>1</td>
<td>180</td>
</tr>
<tr>
<td>BTSTS Ws,#bit4</td>
<td>Bit test Ws to C, then set Ws</td>
<td>1</td>
<td>1</td>
<td>181</td>
</tr>
<tr>
<td>FBCL Ws,Wd</td>
<td>Find bit change from left (MSb) side</td>
<td>1</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>FF1L Ws,Wd</td>
<td>Find first one from left (MSb) side</td>
<td>1</td>
<td>1</td>
<td>257</td>
</tr>
<tr>
<td>FF1R Ws,Wd</td>
<td>Find first one from right (LSb) side</td>
<td>1</td>
<td>1</td>
<td>259</td>
</tr>
</tbody>
</table>

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.

**Note 2:** These instructions are only available in dsPIC33C devices.
### Table 3-7: Compare/Skip and Compare/Branch Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles(^{(1)})</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTSC f,#bit4</td>
<td>Bit test f, skip if clear</td>
<td>1</td>
<td>1 (2 or 3)(^{(5)})</td>
<td>168</td>
</tr>
<tr>
<td>BTSC Ws,#bit4</td>
<td>Bit test Ws, skip if clear</td>
<td>1</td>
<td>1 (2 or 3)(^{(5)})</td>
<td>170</td>
</tr>
<tr>
<td>BTSS f,#bit4</td>
<td>Bit test f, skip if set</td>
<td>1</td>
<td>1 (2 or 3)(^{(5)})</td>
<td>172</td>
</tr>
<tr>
<td>BTSS Ws,#bit4</td>
<td>Bit test Ws, skip if set</td>
<td>1</td>
<td>1 (2 or 3)(^{(5)})</td>
<td>173</td>
</tr>
<tr>
<td>CP f</td>
<td>Compare (f – WREG)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>200</td>
</tr>
<tr>
<td>CP Wb,#lit5</td>
<td>Compare (Wb – lit5)</td>
<td>1</td>
<td>1</td>
<td>201</td>
</tr>
<tr>
<td>CP Wb,#lit8</td>
<td>Compare (Wb – lit8)</td>
<td>1</td>
<td>1</td>
<td>202</td>
</tr>
<tr>
<td>CP Wb,Ws</td>
<td>Compare (Wb – Ws)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>203</td>
</tr>
<tr>
<td>CP0 f</td>
<td>Compare (f – 0x0000)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>204</td>
</tr>
<tr>
<td>CP0 Ws</td>
<td>Compare (Ws – 0x0000)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>205</td>
</tr>
<tr>
<td>CPB f</td>
<td>Compare with Borrow (f – WREG – C)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>206</td>
</tr>
<tr>
<td>CPB Wb,#lit5</td>
<td>Compare with Borrow (Wb – lit5 – C)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>207</td>
</tr>
<tr>
<td>CPB Wb,#lit8</td>
<td>Compare with Borrow (Wb – lit8 – C)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>208</td>
</tr>
<tr>
<td>CPB Wb,Ws</td>
<td>Compare with Borrow (Wb – Ws – C)</td>
<td>1</td>
<td>1(^{(5)})</td>
<td>209</td>
</tr>
<tr>
<td>CPBEQ Wb,Wn,Expr</td>
<td>Compare Wb with Wn, branch if =</td>
<td>1</td>
<td>1 (5)(^{(4)})</td>
<td>211</td>
</tr>
<tr>
<td>CPBGTE Wb,Wn,Expr</td>
<td>Signed compare Wb with Wn, branch if &gt;</td>
<td>1</td>
<td>1 (5)(^{(4)})</td>
<td>212</td>
</tr>
<tr>
<td>CPBLET Wb,Wn,Expr</td>
<td>Signed compare Wb with Wn, branch if &lt;</td>
<td>1</td>
<td>1 (5)(^{(4)})</td>
<td>213</td>
</tr>
<tr>
<td>CPBN E Wb,Wn,Expr</td>
<td>Compare Wb with Wn, branch if ≠</td>
<td>1</td>
<td>1 (5)(^{(4)})</td>
<td>212</td>
</tr>
<tr>
<td>CPSEQ Wb,Wn</td>
<td>Compare (Wb – Wn), skip if =</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>215</td>
</tr>
<tr>
<td>CPSEQ Wb,Wn</td>
<td>Compare (Wb – Wn), skip if =</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>216</td>
</tr>
<tr>
<td>CPSGT Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if &gt;</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>217</td>
</tr>
<tr>
<td>CPSGT Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if &gt;</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>218</td>
</tr>
<tr>
<td>CPSLTE Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if &lt;</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>219</td>
</tr>
<tr>
<td>CPSLTE Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if &lt;</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>220</td>
</tr>
<tr>
<td>CPSNE Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if ≠</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>221</td>
</tr>
<tr>
<td>CPSNE Wb,Wn</td>
<td>Signed compare (Wb – Wn), skip if ≠</td>
<td>1</td>
<td>1 (2 or 3)(^{(3)})</td>
<td>222</td>
</tr>
</tbody>
</table>

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

1:  This instruction is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
2:  This instruction is only available in PIC24E, dsPIC33E and dsPIC33C devices.
3:  Compare/Branch instructions in PIC24E/dsPIC33E devices and in dsPIC33C Master cores execute in one cycle if the branch is not taken, and five cycles if the branch is taken. Compare/Branch instructions in dsPIC33C Slave cores execute in one cycle if the branch is not taken and two cycles if the branch is taken.
5:  In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
### Table 3-8: Program Flow Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA Expr</td>
<td>Branch unconditionally</td>
<td>1</td>
<td>2&lt;sup&gt;6&lt;/sup&gt;</td>
<td>136</td>
</tr>
<tr>
<td>BRA Wn&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Computed branch</td>
<td>1</td>
<td>2&lt;sup&gt;8&lt;/sup&gt;</td>
<td>137</td>
</tr>
<tr>
<td>BRA Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Computed branch</td>
<td>1</td>
<td>2&lt;sup&gt;8&lt;/sup&gt;</td>
<td>138</td>
</tr>
<tr>
<td>BRA C Expr</td>
<td>Branch if Carry (no Borrow)</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>139</td>
</tr>
<tr>
<td>BRA GE Expr</td>
<td>Branch if signed greater than or equal</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>141</td>
</tr>
<tr>
<td>BRA GEU Expr</td>
<td>Branch if unsigned greater than or equal</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>142</td>
</tr>
<tr>
<td>BRA GT Expr</td>
<td>Branch if signed greater than</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>143</td>
</tr>
<tr>
<td>BRA GTU Expr</td>
<td>Branch if unsigned greater than</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>144</td>
</tr>
<tr>
<td>BRA LE Expr</td>
<td>Branch if signed less than or equal</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>145</td>
</tr>
<tr>
<td>BRA LEU Expr</td>
<td>Branch if unsigned less than or equal</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>146</td>
</tr>
<tr>
<td>BRA LT Expr</td>
<td>Branch if signed less than</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>147</td>
</tr>
<tr>
<td>BRA LTU Expr</td>
<td>Branch if unsigned less than</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>148</td>
</tr>
<tr>
<td>BRA N Expr</td>
<td>Branch if Negative</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>149</td>
</tr>
<tr>
<td>BRA NC Expr</td>
<td>Branch if not Carry (Borrow)</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>150</td>
</tr>
<tr>
<td>BRA NN Expr</td>
<td>Branch if not Negative</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>151</td>
</tr>
<tr>
<td>BRA NOV Expr</td>
<td>Branch if not Overflow</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>152</td>
</tr>
<tr>
<td>BRA NZ Expr</td>
<td>Branch if not Zero</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>153</td>
</tr>
<tr>
<td>BRA OA Expr&lt;sup&gt;(3)&lt;/sup&gt;</td>
<td>Branch if Accumulator A Overflow</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>154</td>
</tr>
<tr>
<td>BRA OB Expr&lt;sup&gt;(3)&lt;/sup&gt;</td>
<td>Branch if Accumulator B Overflow</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>155</td>
</tr>
<tr>
<td>BRA OV Expr</td>
<td>Branch if Overflow</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>156</td>
</tr>
<tr>
<td>BRA SA Expr&lt;sup&gt;(3)&lt;/sup&gt;</td>
<td>Branch if Accumulator A Saturate</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>157</td>
</tr>
<tr>
<td>BRA SB Expr&lt;sup&gt;(3)&lt;/sup&gt;</td>
<td>Branch if Accumulator B Saturate</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>158</td>
</tr>
<tr>
<td>BRA Z Expr</td>
<td>Branch if Zero</td>
<td>1</td>
<td>1 (2)&lt;sup&gt;1,8&lt;/sup&gt;</td>
<td>159</td>
</tr>
<tr>
<td>CALL Expr&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Call subroutine</td>
<td>2</td>
<td>2&lt;sup&gt;8&lt;/sup&gt;</td>
<td>183</td>
</tr>
<tr>
<td>CALL Wn&lt;sup&gt;(6)&lt;/sup&gt;</td>
<td>Call subroutine long (long address)</td>
<td>1</td>
<td>4</td>
<td>191</td>
</tr>
<tr>
<td>DO #lit14,Expr&lt;sup&gt;(6)&lt;/sup&gt;</td>
<td>Do code through PC + Expr, (lit14 + 1) times</td>
<td>2</td>
<td>2</td>
<td>242</td>
</tr>
<tr>
<td>DO #lit15,Expr&lt;sup&gt;(7)&lt;/sup&gt;</td>
<td>Do code through PC + Expr, (lit15 + 1) times</td>
<td>2</td>
<td>2</td>
<td>244</td>
</tr>
<tr>
<td>DO Wn,Expr&lt;sup&gt;(6)&lt;/sup&gt;</td>
<td>Do code through PC + Expr, (Wn + 1) times</td>
<td>2</td>
<td>2</td>
<td>246</td>
</tr>
<tr>
<td>DO Wn,Expr&lt;sup&gt;(7)&lt;/sup&gt;</td>
<td>Do code through PC + Expr, (Wn + 1) times</td>
<td>2</td>
<td>2</td>
<td>248</td>
</tr>
</tbody>
</table>

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

**Note 2:** RETURN instructions execute in three cycles, but if an exception is pending, they execute in two cycles.

**Note 3:** This instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

**Note 4:** This instruction is only available in PIC24E, dsPIC33E and dsPIC33C devices.

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

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

**Note 7:** This instruction is only available in dsPIC33E and dsPIC33C devices.

**Note 8:** In PIC24E and dsPIC33E devices, and in dsPIC33C Master cores, these instructions require two additional cycles (four cycles overall) when the branch is taken compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices, and dsPIC33C Slave cores.

**Note 9:** In dsPIC33E and PIC24E devices, and in dsPIC33C Master cores, these instructions require three additional cycles when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices, and dsPIC33C Slave cores.
Table 3-8: Program Flow Instructions (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTO Expr</td>
<td>Go to address</td>
<td>2</td>
<td>2&lt;sup&gt;(6)&lt;/sup&gt;</td>
<td>263</td>
</tr>
<tr>
<td>GOTO Wn&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Go to address indirectly</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>264</td>
</tr>
<tr>
<td>GOTO.Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Go to address indirectly</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>265</td>
</tr>
<tr>
<td>GOTO.L Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Go to indirect (long address)</td>
<td>1</td>
<td>4</td>
<td>266</td>
</tr>
<tr>
<td>RCALL Expr&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Relative call</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>367</td>
</tr>
<tr>
<td>RCALL Expr&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Relative call</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>369</td>
</tr>
<tr>
<td>RCALL Wn&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Computed call</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>371</td>
</tr>
<tr>
<td>RCALL Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Computed call</td>
<td>1</td>
<td>2&lt;sup&gt;(8)&lt;/sup&gt;</td>
<td>373</td>
</tr>
<tr>
<td>REPEAT #lit14&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Repeat next instruction (lit14 + 1) times</td>
<td>1</td>
<td>1</td>
<td>375</td>
</tr>
<tr>
<td>REPEAT #lit15&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Repeat next instruction (lit15 + 1) times</td>
<td>1</td>
<td>1</td>
<td>376</td>
</tr>
<tr>
<td>REPEAT Wn&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Repeat next instruction (Wn + 1) times</td>
<td>1</td>
<td>1</td>
<td>377</td>
</tr>
<tr>
<td>REPEAT Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Repeat next instruction (Wn + 1) times</td>
<td>1</td>
<td>1</td>
<td>378</td>
</tr>
<tr>
<td>RETFIE&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Return from interrupt enable</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>380</td>
</tr>
<tr>
<td>RETFIE&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Return from interrupt enable</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>381</td>
</tr>
<tr>
<td>RETLW #lit10,Wn&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Return with lit10 in Wn</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>382</td>
</tr>
<tr>
<td>RETLW #lit10,Wn&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Return with lit10 in Wn</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>384</td>
</tr>
<tr>
<td>RETURN&lt;sup&gt;(5)&lt;/sup&gt;</td>
<td>Return from subroutine</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>386</td>
</tr>
<tr>
<td>RETURN&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>Return from subroutine</td>
<td>1</td>
<td>3&lt;sup&gt;(2,9)&lt;/sup&gt;</td>
<td>387</td>
</tr>
</tbody>
</table>

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

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

3: This instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

4: This instruction is only available in PIC24E, dsPIC33E and dsPIC33C devices.

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

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

7: This instruction is only available in dsPIC33E and dsPIC33C devices.

8: In PIC24E and dsPIC33E devices, and in dsPIC33C Master cores, these instructions require two additional cycles (four cycles overall) when the branch is taken when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices, and dsPIC33C Slave cores.

9: In dsPIC33E and PIC24E devices, and in dsPIC33C Master cores, these instructions require three additional cycles when compared to dsPIC30F, dsPIC33F, PIC24F and PIC24H devices, and dsPIC33C Slave cores.
### Table 3-9: Shadow/Stack/Context Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOTSWP(4)</td>
<td>Swap the active and inactive program Flash spaces</td>
<td>1</td>
<td>2</td>
<td>135</td>
</tr>
<tr>
<td>CTXTSWF #lit3(2,3)</td>
<td>Switch CPU register context to context defined by #lit3</td>
<td>1</td>
<td>2</td>
<td>223</td>
</tr>
<tr>
<td>CTXTSWF Wn(4,3)</td>
<td>Switch CPU register context to context defined by Wn</td>
<td>1</td>
<td>2</td>
<td>224</td>
</tr>
<tr>
<td>LNK #lit14(5)</td>
<td>Link Frame Pointer</td>
<td>1</td>
<td>1</td>
<td>280</td>
</tr>
<tr>
<td>LNK #lit14(6)</td>
<td>Link Frame Pointer</td>
<td>1</td>
<td>1</td>
<td>281</td>
</tr>
<tr>
<td>POP f</td>
<td>Pop TOS to f</td>
<td>1</td>
<td>1</td>
<td>357</td>
</tr>
<tr>
<td>POP Wd</td>
<td>Pop TOS to Wd</td>
<td>1</td>
<td>1</td>
<td>358</td>
</tr>
<tr>
<td>POP.D Wn</td>
<td>Double pop from TOS to Wnd:Wnd + 1</td>
<td>1</td>
<td>2</td>
<td>359</td>
</tr>
<tr>
<td>POP.S</td>
<td>POP shadow registers</td>
<td>1</td>
<td>1</td>
<td>360</td>
</tr>
<tr>
<td>PUSH f</td>
<td>Push f to TOS</td>
<td>1</td>
<td>1(1)</td>
<td>361</td>
</tr>
<tr>
<td>PUSH Ws</td>
<td>Push Ws to TOS</td>
<td>1</td>
<td>1(1)</td>
<td>362</td>
</tr>
<tr>
<td>PUSH.D Wns</td>
<td>Push double Wns:Wns + 1 to TOS</td>
<td>1</td>
<td>2</td>
<td>364</td>
</tr>
<tr>
<td>PUSH.S</td>
<td>Push shadow registers</td>
<td>1</td>
<td>1</td>
<td>365</td>
</tr>
<tr>
<td>ULNK(6)</td>
<td>Unlink Frame Pointer</td>
<td>1</td>
<td>1</td>
<td>448</td>
</tr>
</tbody>
</table>

**Note 1:** In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.

2: These instructions are only available in dsPIC33C and some dsPIC33E devices. Please see the specific device data sheet for details.

3: In dsPIC33C devices, these instructions also switch the accumulator context in addition to the CPU register context.

4: These instructions are only available in some PIC24F, dsPIC33E and dsPIC33C devices. Please see the specific device data sheet for details.

5: These instructions are only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices. Please see the specific device data sheet for details.

6: These instructions are only available in PIC24E, dsPIC33E and dsPIC33C devices. Please see the specific device data sheet for details.

### Table 3-10: Control Instructions

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLRWDT</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>196</td>
</tr>
<tr>
<td>DISI #lit14</td>
<td>Disable interrupts for (lit14 + 1) instruction cycles</td>
<td>1</td>
<td>1</td>
<td>232</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
<td>354</td>
</tr>
<tr>
<td>NOPR</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>PWRSAV #lit1</td>
<td>Enter Power-Saving mode lit1</td>
<td>1</td>
<td>1</td>
<td>366</td>
</tr>
<tr>
<td>RESET</td>
<td>Software Device Reset</td>
<td>1</td>
<td>1</td>
<td>379</td>
</tr>
</tbody>
</table>
### Table 3-11: DSP Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Acc</td>
<td>Add accumulators</td>
<td>1</td>
<td>1</td>
<td>107</td>
</tr>
<tr>
<td>ADD Ws,#Slit4,Acc</td>
<td>16-bit signed add to accumulator</td>
<td>1</td>
<td>1(1)</td>
<td>108</td>
</tr>
<tr>
<td>CLR Acc,Wx,Wx,Wy,Wyd,AWB</td>
<td>Clear accumulator, prefetch operands</td>
<td>1</td>
<td>1</td>
<td>194</td>
</tr>
<tr>
<td>ED Wm*Wm,Acc,Wx,Wy,Wxd</td>
<td>Euclidean distance (no accumulate)</td>
<td>1</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>EDAC Wm*Wm,Acc,Wx,Wy,Wxd</td>
<td>Euclidean distance</td>
<td>1</td>
<td>1</td>
<td>252</td>
</tr>
<tr>
<td>LAC Ws,#Slit4,Acc</td>
<td>Load accumulator</td>
<td>1</td>
<td>1(1)</td>
<td>276</td>
</tr>
<tr>
<td>LAC.D Ws,#Slit4,Acc</td>
<td>Load accumulator double word</td>
<td>1</td>
<td>1</td>
<td>278</td>
</tr>
<tr>
<td>MAC Wm*Wn,Acc,Wx,Wy,Wyd,AWB</td>
<td>Multiply and accumulate</td>
<td>1</td>
<td>1</td>
<td>288</td>
</tr>
<tr>
<td>MAC Wm*Wn,Acc,Wx,Wy,Wyd</td>
<td>Square and accumulate</td>
<td>1</td>
<td>1</td>
<td>290</td>
</tr>
<tr>
<td>MAX Acc</td>
<td>Force accumulator maximum data limit</td>
<td>1</td>
<td>1</td>
<td>292</td>
</tr>
<tr>
<td>MAX.V Acc,Wd</td>
<td>Force accumulator maximum data limit and store limit excess result</td>
<td>1</td>
<td>1</td>
<td>293</td>
</tr>
<tr>
<td>MIN Acc</td>
<td>Force accumulator minimum data limit</td>
<td>1</td>
<td>1</td>
<td>294</td>
</tr>
<tr>
<td>MIN.V Acc,Wd</td>
<td>Force accumulator minimum data limit and store limit excess result</td>
<td>1</td>
<td>1</td>
<td>295</td>
</tr>
<tr>
<td>MINZ Acc</td>
<td>Conditionally force accumulator minimum data limit if Z flag is set</td>
<td>1</td>
<td>1</td>
<td>296</td>
</tr>
<tr>
<td>MINZ.V Acc,Wd</td>
<td>Conditionally force accumulator minimum data limit and store limit excess result if Z flag is set</td>
<td>1</td>
<td>1</td>
<td>297</td>
</tr>
<tr>
<td>MOVSAc Acc,Wx,Wy,Wyd,AWB</td>
<td>Move Wx to Wx and Wy to Wyd</td>
<td>1</td>
<td>1</td>
<td>313</td>
</tr>
<tr>
<td>MPY Wm*Wn,Acc,Wx,Wy,Wyd</td>
<td>Multiply Wm by Wn to accumulator</td>
<td>1</td>
<td>1</td>
<td>315</td>
</tr>
<tr>
<td>MPY Wm*Wm,Acc,Wx,Wy,Wyd</td>
<td>Square to accumulator</td>
<td>1</td>
<td>1</td>
<td>317</td>
</tr>
<tr>
<td>MPY.N Wm*Wn,Acc,Wx,Wy,Wyd</td>
<td>(Multiply -Wm by Wn) to accumulator</td>
<td>1</td>
<td>1</td>
<td>319</td>
</tr>
<tr>
<td>MSC Wm*Wn,Acc,Wx,Wy,Wyd,AWB</td>
<td>Multiply and subtract from accumulator</td>
<td>1</td>
<td>1</td>
<td>321</td>
</tr>
<tr>
<td>NEG Acc</td>
<td>Negate accumulator</td>
<td>1</td>
<td>1</td>
<td>350</td>
</tr>
<tr>
<td>NORM Acc,Wd</td>
<td>Normalize accumulator</td>
<td>1</td>
<td>1</td>
<td>356</td>
</tr>
<tr>
<td>SAC Acc,#Slit4,Wd</td>
<td>Store accumulator</td>
<td>1</td>
<td>1</td>
<td>401</td>
</tr>
<tr>
<td>SAC.D Acc,#Slit4,Wnd</td>
<td>Store accumulator double word</td>
<td>1</td>
<td>1</td>
<td>403</td>
</tr>
<tr>
<td>SAC.R Acc,#Slit4,Wd</td>
<td>Store rounded accumulator</td>
<td>1</td>
<td>1</td>
<td>404</td>
</tr>
<tr>
<td>SFTAC Acc,#Slit6</td>
<td>Arithmetic shift accumulator by Slit6</td>
<td>1</td>
<td>1</td>
<td>410</td>
</tr>
<tr>
<td>SFTAC Acc,Wb</td>
<td>Arithmetic shift accumulator by (Wb)</td>
<td>1</td>
<td>1</td>
<td>411</td>
</tr>
<tr>
<td>SUB Acc</td>
<td>Subtract accumulators</td>
<td>1</td>
<td>1</td>
<td>418</td>
</tr>
</tbody>
</table>

**Note 1:** In PIC24E, dsPIC33E and dsPIC33C devices, read and Read-Modify-Write operations on non-CPU Special Function Registers require an additional cycle when compared to PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.

**Note 2:** These instructions are only available in dsPIC33C devices.
Section 4. Instruction Set Details

HIGHLIGHTS

This section of the manual contains the following major topics:

4.1 Data Addressing Modes.................................................................................................. 54
4.2 Program Addressing Modes ........................................................................................... 63
4.3 Instruction Stalls.......................................................................................................... .... 64
4.4 Byte Operations ............................................................................................................. .6 6
4.5 Word Move Operations ................................................................................................ 68
4.6 Using 10-Bit Literal Operands..................................................................................... 71
4.8 Software Stack Pointer and Frame Pointer ................................................................. 72
4.9 Conditional Branch Instructions .................................................................................. 78
4.10 Z Status Bit................................................................................................................... 79
4.11 Assigned Working Register Usage .............................................................................. 80
4.12 DSP Data Formats (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)............ 83
4.13 Accumulator Usage (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)......... 85
4.14 Accumulator Access (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)........ 86
4.15 DSP MAC Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)....... 86
4.16 DSP Accumulator Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)....90
4.17 Scaling Data with the \texttt{FBCL} Instruction (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C
Devices).......................................................................................................................... 90
4.19 Normalizing the Accumulator with the \texttt{FBCL} Instruction (dsPIC30F, dsPIC33F, dsPIC33E
and dsPIC33C Devices).................................................................................................. 93
4.21 Extended Precision Arithmetic Using Mixed-Sign Multiplications (dsPIC33E and dsPIC33C
Only).................................................................................................................................... 94
4.1 DATA ADDRESSING MODES

The 16-bit MCU and DSC devices support three native addressing modes for accessing data memory, along with several forms of Immediate Addressing. Data accesses may be performed using File Register Addressing, Register Direct or Indirect Addressing, and Immediate Addressing, allowing a fixed value to be used by the instruction.

File Register Addressing provides the ability to operate on data stored in the lower 8K of data memory (Near RAM), and also move data between the Working registers and the entire 64K data space. Register Direct Addressing is used to access the 16 memory-mapped Working registers, W0:W15. Register Indirect Addressing is used to efficiently operate on data stored in the entire 64K data space (and also Extended Data Space in the case of dsPIC33E/dsPIC33C/PIC24E and some PIC24F devices), using the contents of the Working registers as an Effective Address (EA). 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: 16-Bit MCU and DSC Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Register</td>
<td>0x0000-0x1FFF(1)</td>
</tr>
<tr>
<td>Register Direct</td>
<td>0x0000-0x001F (Working register array, W0:W15)</td>
</tr>
<tr>
<td>Register Indirect</td>
<td>0x0000-0xFFFF</td>
</tr>
<tr>
<td>Immediate</td>
<td>N/A (constant value)</td>
</tr>
</tbody>
</table>

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

4.1.1 File Register Addressing

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

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

Note: Instructions which support File Register Addressing use ‘f’ as an operand in the instruction summary tables of Section 3. “Instruction Set Overview”.

© 2005-2018 Microchip Technology Inc.
Section 4. Instruction Set Details

Example 4-1: File Register Addressing

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Instruction:</th>
<th>After Instruction:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 0x1000</td>
<td>Data Memory 0x1000 = 0x5555</td>
<td>Data Memory 0x1000 = 0x5554</td>
</tr>
<tr>
<td>MOV 0x27FE, W0</td>
<td>W0 = 0x5555; Data Memory 0x27FE = 0x1234</td>
<td>W0 = 0x1234; Data Memory 0x27FE = 0x1234</td>
</tr>
</tbody>
</table>

Example 4-2: File Register Addressing and WREG

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Instruction:</th>
<th>After Instruction:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 0x1000</td>
<td>W0 (WREG) = 0x332C; Data Memory 0x1000 = 0x5555</td>
<td>W0 (WREG) = 0x1104; Data Memory 0x1000 = 0x1104</td>
</tr>
<tr>
<td>AND 0x1000, WREG</td>
<td>W0 (WREG) = 0x332C; Data Memory 0x1000 = 0x5555</td>
<td>W0 (WREG) = 0x1104; Data Memory 0x1000 = 0x1104</td>
</tr>
</tbody>
</table>

4.1.2 Register Direct Addressing

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

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

Note: Instructions which must use Register Direct Addressing, use the symbols Wb, Wn, Wns and Wnd in the summary tables of Section 3. “Instruction Set Overview”. Commonly, Register Direct Addressing may also be used when Register Indirect Addressing may be used. Instructions which use Register Indirect Addressing, use the symbols Wd and Ws in the summary tables of Section 3. “Instruction Set Overview”.

© 2005-2018 Microchip Technology Inc.
4.1.3 Register Indirect Addressing

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

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

<table>
<thead>
<tr>
<th>Indirect Mode</th>
<th>Syntax</th>
<th>Function (Byte Instruction)</th>
<th>Function (Word Instruction)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Increment</td>
<td>[Wn + 1]</td>
<td>EA = [Wn + 1]</td>
<td>EA = [Wn + 2]</td>
<td>Wn is pre-incremented to form the EA.</td>
</tr>
<tr>
<td>Pre-Decrement</td>
<td>[Wn - 1]</td>
<td>EA = [Wn - 1]</td>
<td>EA = [Wn - 2]</td>
<td>Wn is pre-decremented to form the EA.</td>
</tr>
<tr>
<td>Register Offset</td>
<td>[Wn + Wb]</td>
<td>EA = [Wn + Wb]</td>
<td>EA = [Wn + Wb]</td>
<td>The sum of Wn and Wb forms the EA. Wn and Wb are not modified.</td>
</tr>
</tbody>
</table>
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 two. Namely, in Word mode, pre/post-decrements subtract two from the EA stored in the Working register and pre/post-increments add two to the EA. This feature ensures that after an EA modification is made, the EA will point to the next adjacent word in memory. Example 4-4 shows how Indirect Addressing may be used to update the EA.

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

Note: The MOV with offset instructions (see pages 299 and 300) provides a literal addressing offset ability to be used with Indirect Addressing. In these instructions, the EA is formed by adding the contents of a Working register to a signed 10-bit literal. Example 4-6 shows how these instructions may be used to move data to and from the Working register array.

Example 4-4: Indirect Addressing with Effective Address Update

<table>
<thead>
<tr>
<th>MOV.B [W0++], [W13--]</th>
<th>byte move [W0] to [W13]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>post-inc W0, post-dec W13</td>
</tr>
</tbody>
</table>

Before Instruction:

- W0 = 0x2300
- W13 = 0x2708
- Data Memory 0x2300 = 0x7783
- Data Memory 0x2708 = 0x904E

After Instruction:

- W0 = 0x2301
- W13 = 0x2707
- Data Memory 0x2300 = 0x7783
- Data Memory 0x2708 = 0x9083

ADD W1, [--W5], [++W8] ; pre-dec W5, pre-inc W8 ; add W1 to [W5], store in [W8]

Before Instruction:

- W1 = 0x0800
- W5 = 0x2200
- W8 = 0x2400
- Data Memory 0x21FE = 0x7783
- Data Memory 0x2402 = 0xAACC

After Instruction:

- W1 = 0x0800
- W5 = 0x21FE
- W8 = 0x2402
- Data Memory 0x21FE = 0x7783
- Data Memory 0x2402 = 0x7F83
Example 4-5:  Indirect Addressing with Register Offset

MOV.B  [W0+W1], [W7++] ; byte move [W0+W1] to W7, post-inc W7

Before Instruction:
W0 = 0x2300
W1 = 0x01FE
W7 = 0x1000
Data Memory 0x24FE = 0x7783
Data Memory 0x1000 = 0x11DC

After Instruction:
W0 = 0x2300
W1 = 0x01FE
W7 = 0x1001
Data Memory 0x24FE = 0x7783
Data Memory 0x1000 = 0x1183

LAC  [W0+W8], A ; load ACCA with [W0+W8]
; (sign-extend and zero-backfill)

Before Instruction:
W0 = 0x2344
W8 = 0x0008
ACCA = 0x00 7877 9321
Data Memory 0x234C = 0xE290

After Instruction:
W0 = 0x2344
W8 = 0x0008
ACCA = 0xFF E290 0000
Data Memory 0x234C = 0xE290

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

MOV W4, [W8-0x300] ; 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

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

MOV W4, [W8-0x300] ; 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 16-bit MCU and DSC devices. Due to operation encoding and functional considerations, not every instruction which supports Indirect Addressing supports all modes shown in Table 4-2. The majority of instructions which use Indirect Addressing support the No Modify, Pre-Increment, Pre-Decrement, Post-Increment and Post-Decrement Addressing modes. The MOV instructions, and several accumulator-based DSP instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices only), are also capable of using the Register Offset Addressing mode.

Note: Instructions which use Register Indirect Addressing use the operand symbols, Wd and Ws, in the summary tables of Section 3, “Instruction Set Overview”.

4.1.3.2 DSP MAC INDIRECT ADDRESSING MODES (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

A special class of Indirect Addressing modes is utilized by the DSP MAC instructions. As is described later in Section 4.15 “DSP MAC Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)”, the DSP MAC class of instructions is capable of performing two fetches from memory using Effective Addressing. Since DSP algorithms frequently demand a broader range of address updates, the addressing modes offered by the DSP MAC instructions provide greater range in the size of the Effective Address update which may be made. Table 4-3 shows that both X and Y prefetches support Post-Increment and Post-Decrement Addressing modes, with updates of two, four and six bytes. Since DSP instructions only execute in Word mode, no provisions are made for odd-sized EA updates.

Table 4-3: DSP MAC Indirect Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>X Memory</th>
<th>Y Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect with No Modification</td>
<td>EA = [Wx]</td>
<td>EA = [Wy]</td>
</tr>
<tr>
<td>Indirect with Post-Increment by two</td>
<td>EA = [Wx] + = 2</td>
<td>EA = [Wy] + = 2</td>
</tr>
<tr>
<td>Indirect with Post-Increment by four</td>
<td>EA = [Wx] + = 4</td>
<td>EA = [Wy] + = 4</td>
</tr>
<tr>
<td>Indirect with Post-Increment by six</td>
<td>EA = [Wx] + = 6</td>
<td>EA = [Wy] + = 6</td>
</tr>
<tr>
<td>Indirect with Post-Decrement by four</td>
<td>EA = [Wx] – = 4</td>
<td>EA = [Wy] – = 4</td>
</tr>
</tbody>
</table>

Note: As described in Section 4.15 “DSP MAC Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)”, only W8 and W9 may be used to access X memory, and only W10 and W11 may be used to access Y memory.

4.1.3.3 MODULO AND BIT-REVERSED ADDRESSING MODES (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

Both of these addressing modes are powerful features of the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C architectures, which can be exploited by any instruction that uses Indirect Addressing. Refer to the specific device family reference manual for details on using Modulo and Bit-Reversed Addressing.
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.

<table>
<thead>
<tr>
<th>Operand</th>
<th>Instruction Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#lit1</td>
<td>PWRSAV</td>
</tr>
<tr>
<td>#lit3</td>
<td>CTXTSWP</td>
</tr>
<tr>
<td>#bit4</td>
<td>BCLR, BSET, BTG, BTSC, BTSS, BTST, BTST.C, BTST.Z, BTSTS, BTSTS.C, BTSTS.Z</td>
</tr>
<tr>
<td>#lit4</td>
<td>ASR, LSR, SL</td>
</tr>
<tr>
<td>#Slit4</td>
<td>ADD, LAC, SAC, SAC.R</td>
</tr>
<tr>
<td>#wid4</td>
<td>BFEXT, BFINS</td>
</tr>
<tr>
<td>#lit5</td>
<td>ADD, ADDC, AND, CP[5], CPB[5], IOR, MUL.SU, MUL.UU, SUB, SUBB, SUBBR, SUBR, XOR</td>
</tr>
<tr>
<td>#Slit6</td>
<td>SFTAC</td>
</tr>
<tr>
<td>#lit8</td>
<td>MOV.B, CP[4], CPB[4]</td>
</tr>
<tr>
<td>#lit10</td>
<td>ADD, ADDC, AND, CP, CPB, IOR, RETLN, SUB, SUBB, XOR</td>
</tr>
<tr>
<td>#Slit10</td>
<td>MOV</td>
</tr>
<tr>
<td>#lit14</td>
<td>DISI, DO[2], LNK, REPEAT[5]</td>
</tr>
<tr>
<td>#lit15</td>
<td>DO[3], REPEAT[4]</td>
</tr>
<tr>
<td>#lit16</td>
<td>MOV</td>
</tr>
</tbody>
</table>

**Note:** The 6-bit (#Slit6) operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

### Table 4-4: Immediate Operands in the Instruction Set

1. This operand or instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
2. This operand or instruction is only available in dsPIC30F and dsPIC33F devices.
3. This operand or instruction is only available in dsPIC33E and dsPIC33C devices.
4. This operand or instruction is only available in dsPIC33E, dsPIC33C and PIC24E devices.
5. This operand or instruction is only available in dsPIC30F, dsPIC33F, PIC24F and PIC24H devices.
6. This operand or instruction is only available in dsPIC33C devices.
The syntax for Immediate Addressing requires that the number sign (#) must immediately precede the constant operand value. The "#" symbol indicates to the assembler that the quantity is a constant. If an out-of-range constant is used with an instruction, the assembler will generate an error. Several examples of Immediate Addressing are shown in Example 4-7.

**Example 4-7: Immediate Addressing**

```assembly
PWRSAV #1 ; Enter IDLE mode
ADD.B #0x10, W0 ; Add 0x10 to W0 (byte mode)
```

**Before Instruction:**

W0 = 0x12A9

**After Instruction:**

W0 = 0x12B9

```assembly
XOR W0, #1, [W1++] ; Exclusive-OR W0 and 0x1
; Store the result to [W1] ; Post-increment W1
```

**Before Instruction:**

W0 = 0xFFFF
W1 = 0x0890
Data Memory 0x0890 = 0x0032

**After Instruction:**

W0 = 0xFFFF
W1 = 0x0892
Data Memory 0x0890 = 0xFFFE

### 4.1.5 Data Addressing Mode Tree

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

**Figure 4-1: Data Addressing Mode Tree (PIC24F, PIC24H, PIC24E)**
The Data Addressing modes of the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C are summarized in Figure 4-2.

**Figure 4-2: Data Addressing Mode Tree (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)**

```
Data Addressing Modes
    | Immediate
    | File Register
    | Direct
    | Indirect

Basic
    | No Modification
    | Pre-Increment
    | Pre-Decrement
    | Post-Increment
    | Post-Decrement
    | Literal Offset
    | Register Offset

DSP MAC
    | Direct
    | No Modification
    | Post-Increment (2, 4 and 6)

    | Indirect
    | Post-Decrement (2, 4 and 6)
    | Register Offset
```
4.2 PROGRAM ADDRESSING MODES

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

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

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

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

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). When representing an address offset value, Expr can also be indicated by using a “.” and a sign, “+” or “-”. For example, the expression, “.+2”, means an address offset of +2 (i.e., the next instruction address relative to the current position of the Program Counter). See Section 5. “Instruction Descriptions” for details.

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

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

4: This condition/instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

5: This condition instruction is only available in dsPIC33E, dsPIC33C and PIC24E devices.
4.3 INSTRUCTION STALLS

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

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

4.3.1 RAW Dependency Detection

During the instruction predecode, the core determines if any address register dependency is imminent across an instruction boundary. The Stall detection logic compares the W register (if any) used for the destination EA of the instruction currently being executed with the W register to be used by the source EA (if any) of the prefetched instruction. When a match between the destination and source registers is identified, a set of rules is 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.

<table>
<thead>
<tr>
<th>Destination Addressing Mode Using Wn</th>
<th>Source Addressing Mode Using Wn</th>
<th>Stall Required?</th>
<th>Examples(2) (Wn = W2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Direct</td>
<td>No Stall</td>
<td>ADD.W W0, W1, W2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W W2, W3</td>
</tr>
<tr>
<td>Indirect</td>
<td>Direct</td>
<td>No Stall</td>
<td>ADD.W W0, W1, [W2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W W2, W3</td>
</tr>
<tr>
<td>Indirect</td>
<td>Indirect</td>
<td>No Stall</td>
<td>ADD.W W0, W1, [W2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2], W3</td>
</tr>
<tr>
<td>Indirect</td>
<td>Indirect with Pre/Post-Modification</td>
<td>No Stall</td>
<td>ADD.W W0, W1, [W2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2++], W3</td>
</tr>
<tr>
<td>Indirect with Pre/Post-Modification</td>
<td>Direct</td>
<td>No Stall</td>
<td>ADD.W W0, W1, [W2++]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W W2, W3</td>
</tr>
<tr>
<td>Direct</td>
<td>Indirect</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, W2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2], W3</td>
</tr>
<tr>
<td>Direct</td>
<td>Indirect with Pre/Post-Modification</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, W2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2++], W3</td>
</tr>
<tr>
<td>Indirect</td>
<td>Indirect</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, [W2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2], W3(2)</td>
</tr>
<tr>
<td>Indirect</td>
<td>Indirect with Pre/Post-Modification</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, [W2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2++], W3(2)</td>
</tr>
<tr>
<td>Indirect with Pre/Post-Modification</td>
<td>Indirect</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, [W2++]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2], W3</td>
</tr>
<tr>
<td>Indirect with Pre/Post-Modification</td>
<td>Indirect with Pre/Post-Modification</td>
<td>Stall(1)</td>
<td>ADD.W W0, W1, [W2++]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV.W [W2++], W3</td>
</tr>
</tbody>
</table>

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

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

Note: When Register Indirect with Offset Addressing is used to specify the destination for an instruction, and Ws is the same register as Wd, the old value of Ws is used for Wd (i.e., the address offset is ignored).
Section 4. Instruction Set Details

4.3.2 Instruction Stalls and Exceptions
In order to maintain deterministic operation, instruction Stalls are allowed to happen, even if they occur immediately prior to exception processing.

4.3.3 Instruction Stalls and Instructions that Change Program Flow
CALL and RCALL write to the stack using W15 and may, therefore, be subject to an instruction Stall if the source read of the subsequent instruction uses W15.
GOTO, RETFIE and RETURN instructions are never subject to an instruction Stall because they do not perform write operations to the Working registers.

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

Note: DO loops are only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

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

Note: Refer to the specific device family reference manual for more detailed information about RAW instruction Stalls.
4.4 BYTE OPERATIONS

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

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

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

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

- CLR.b W0
- CLR.B W0

Example 4-8: Sample Byte Move Operations

Before Instruction:

W0 = 0x5555

After Instruction:

MOV.B #0x30, W0 ; move the literal byte 0x30 to W0

Before Instruction:

W0 = 0x5530

Data Memory 0x1000 = 0x1234

After Instruction:

MOV.B 0x1000, W0 ; move the byte at 0x1000 to W0

Before Instruction:

W0 = 0x1234

Data Memory 0x1000 = 0x5555

After Instruction:

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 4-9: Sample Byte Math Operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
</table>
| CLR.B [W6--] ; byte clear [W6], then post-dec W6 | W6 = 0x1001  
Data Memory 0x1000 = 0x5555 | W6 = 0x1000  
Data Memory 0x1000 = 0x0055 |
| SUB.B W0, #0x10, W1 ; byte subtract literal 0x10 from W0 ; and store to W1 | W0 = 0x1234  
W1 = 0xFFFF | W0 = 0x1234  
W1 = 0xFF24 |
| ADD.B W0, W1, [W2++] ; byte add W0 and W1, store to [W2] ; and post-inc W2 | W0 = 0x1234  
W1 = 0x5678  
W2 = 0x1000  
Data Memory 0x1000 = 0x5555 | W0 = 0x1234  
W1 = 0x5678  
W2 = 0x1001  
Data Memory 0x1000 = 0x55AC |
4.5 WORD MOVE OPERATIONS

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

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

Figure 4-3: Data Alignment in Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Byte</th>
<th>Address</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1001</td>
<td>b0</td>
<td>0x1000</td>
<td>b0</td>
</tr>
<tr>
<td>0x1003</td>
<td>b1</td>
<td>0x1002</td>
<td></td>
</tr>
<tr>
<td>0x1005</td>
<td>b3</td>
<td>0x1004</td>
<td>b2</td>
</tr>
<tr>
<td>0x1007</td>
<td>b5</td>
<td>0x1006</td>
<td>b4</td>
</tr>
<tr>
<td>0x1009</td>
<td>b7</td>
<td>0x1008</td>
<td>b6</td>
</tr>
<tr>
<td>0x100B</td>
<td>b8</td>
<td>0x100A</td>
<td></td>
</tr>
</tbody>
</table>

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

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

- CLR W0
- CLR.w W0
- CLR.W W0
Example 4-10: Legal Word Move Operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV #0x30, W0 ; move the literal word 0x30 to W0</td>
<td>W0 = 0x5555</td>
<td>W0 = 0x0030</td>
</tr>
<tr>
<td>MOV 0x1000, W0 ; move the word at 0x1000 to W0</td>
<td>W0 = 0x5555</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x1234</td>
<td>W0 = 0x1234</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x1234</td>
<td>W0 = 0x1234</td>
<td></td>
</tr>
<tr>
<td>MOV [W0], [W1++] ; word move [W0] to [W1], then post-inc W1</td>
<td>W0 = 0x1234</td>
<td></td>
</tr>
<tr>
<td>W1 = 0x1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x5555</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1234 = 0xAAAA</td>
<td>Data Memory 0x1000 = 0xAAAA</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1234 = 0xAAAA</td>
<td>Data Memory 0x1234 = 0xAAAA</td>
<td></td>
</tr>
</tbody>
</table>
## Example 4-11: Illegal Word Move Operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV 0x1001, W0; move the word at 0x1001 to W0</td>
<td>W0 = 0x5555</td>
<td>W0 = 0x1234</td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x1234</td>
<td>Data Memory 0x1000 = 0x1234</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1002 = 0x5678</td>
<td>Data Memory 0x1002 = 0x5678</td>
<td></td>
</tr>
<tr>
<td>ADDRESS ERROR TRAP GENERATED (source address is misaligned, so MOV is performed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV W0, 0x1001; move W0 to the word at 0x1001</td>
<td>W0 = 0x1234</td>
<td>W0 = 0x1234</td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x5555</td>
<td>Data Memory 0x1000 = 0x5555</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1002 = 0x6666</td>
<td>Data Memory 0x1002 = 0x6666</td>
<td></td>
</tr>
<tr>
<td>ADDRESS ERROR TRAP GENERATED (destination address is misaligned, so MOV is not performed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV [W0], [W1++]; word move [W0] to [W1], then post-inc W1</td>
<td>W0 = 0x1235</td>
<td>W0 = 0x1235</td>
</tr>
<tr>
<td>W1 = 0x1000</td>
<td>W1 = 0x1000</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1000 = 0x1234</td>
<td>Data Memory 0x1000 = 0x1234</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1234 = 0xAAAA</td>
<td>Data Memory 0x1234 = 0xAAAA</td>
<td></td>
</tr>
<tr>
<td>Data Memory 0x1236 = 0xBBBB</td>
<td>Data Memory 0x1236 = 0xBBBB</td>
<td></td>
</tr>
<tr>
<td>ADDRESS ERROR TRAP GENERATED (source address is misaligned, so MOV is performed)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.6 USING 10-BIT LITERAL OPERANDS

Several instructions that support Byte and Word mode have 10-bit operands. For byte instructions, a 10-bit literal is too large to use. So when 10-bit literals are used in Byte mode, the range of the operand must be reduced to eight 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.

Table 4-7: 10-Bit Literal Coding

<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Word Mode</th>
<th>Byte Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kk kkkk kkkk</td>
<td>kkkk kkkk</td>
</tr>
<tr>
<td>0</td>
<td>00 0000 0000</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>00 0000 0001</td>
<td>0000 0001</td>
</tr>
<tr>
<td>2</td>
<td>00 0000 0010</td>
<td>0000 0010</td>
</tr>
<tr>
<td>127</td>
<td>00 0111 1111</td>
<td>0111 1111</td>
</tr>
<tr>
<td>128</td>
<td>00 1000 0000</td>
<td>1000 0000</td>
</tr>
<tr>
<td>255</td>
<td>00 1111 1111</td>
<td>1111 1111</td>
</tr>
<tr>
<td>256</td>
<td>01 0000 0000</td>
<td>N/A</td>
</tr>
<tr>
<td>512</td>
<td>10 0000 0000</td>
<td>N/A</td>
</tr>
<tr>
<td>1023</td>
<td>11 1111 1111</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Example 4-12: Using 10-Bit Literals for Byte Operands

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD.B #0x80, W0</td>
<td>add 128 (or -128) to W0</td>
</tr>
<tr>
<td>ADD.B #0x380, W0</td>
<td>ERROR... Illegal syntax for byte mode</td>
</tr>
<tr>
<td>ADD.B #0xFF, W0</td>
<td>add 255 (or -1) to W0</td>
</tr>
<tr>
<td>ADD.B #0x3FF, W0</td>
<td>ERROR... Illegal syntax for byte mode</td>
</tr>
<tr>
<td>ADD.B #0xF, W0</td>
<td>add 15 to W0</td>
</tr>
<tr>
<td>ADD.B #0x7F, W0</td>
<td>add 127 to W0</td>
</tr>
<tr>
<td>ADD.B #0x100, W0</td>
<td>ERROR... Illegal syntax for byte mode</td>
</tr>
</tbody>
</table>

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 BIT FIELD INSERT/EXTRACT INSTRUCTIONS (dsPIC33C DEVICES ONLY)

The dsPIC33C family provides a set of instructions that operate on bit fields within a target word.

4.7.1 BFEXT

This instruction can extract multiple bits from a W register or data memory location into a destination W register.

4.7.2 BFINS

This instruction can insert multiple bits from a source W register, or 8-bit literal value into a W register or data memory location.

In both instructions, the location and width of the bit field within the target word are defined as literal values within the instruction.
4.8 SOFTWARE STACK POINTER AND FRAME POINTER

4.8.1 Software Stack Pointer

The 16-bit MCU and DSC devices feature a software stack which facilitates function calls and exception handling. W15 is the default Stack Pointer (SP) and after any Reset, it is initialized to 0x0800 (0x1000 for PIC24E, dsPIC33E and dsPIC33C devices). This ensures that the SP will point to valid RAM and permits stack availability for exceptions, which may occur before the SP is set by the user software. The user may reprogram the SP during initialization to any location within data space.

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

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

```
PUSH W0
```

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> are PUSHed onto the first available stack word, then PC<22:16> are PUSHed. When PC<22:16> are PUSHed, the Most Significant seven bits of the PC are zero-extended before the PUSH is made, as shown in Figure 4-4. During exception processing, the Most Significant seven bits of the PC are concatenated with the lower byte of the STATUS Register (SRL) and IPL<3> (CORCON<3>). This allows the primary STATUS Register contents and CPU Interrupt Priority Level to be automatically preserved during interrupts.

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

Figure 4-4: Stack Operation for CALL Instruction

![Stack Operation for CALL Instruction](image)
4.8.1.1 STACK POINTER EXAMPLE

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

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

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

**Figure 4-5: Stack Pointer Before the First PUSH**

```
0x0000
0x0800 <TOS>
0xFFFE
```

W0 = 0x5A5A
W1 = 0x3636
W15 = 0x0800

**Figure 4-6: Stack Pointer After “PUSH W0” Instruction**

```
0x0000
0x0800 5A5A
0x0902 <TOS>
0xFFFE
```

W0 = 0x5A5A
W1 = 0x3636
W15 = 0x0802
4.8.2 Software Stack Frame Pointer

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

The Link (LNK) and Unlink (ULNK) instructions provide stack frame functionality. The LNK instruction is used to create a stack frame. It is used during a call sequence to adjust the SP, such that the stack may be used to store temporary variables utilized by the called function. After the function completes execution, the ULNK instruction is used to remove the stack frame created by the LNK instruction. The LNK and ULNK instructions must always be used together to avoid stack overflow.
4.8.2.1 STACK FRAME POINTER EXAMPLE

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

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

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

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

Example 4-14: Frame Pointer Usage

```plaintext
TASKA:
...
  PUSH W0 ; Push parameter 1
  PUSH W1 ; Push parameter 2
  PUSH W2 ; Push parameter 3
  CALL COMPUTE ; Call COMPUTE function
  POP W2 ; Pop parameter 3
  POP W1 ; Pop parameter 2
  POP W0 ; Pop parameter 1
...
COMPUTE:
  LNK #4 ; Stack FP, allocate 4 bytes for local variables
  ...
  ULNK ; Free allocated memory, restore original FP
  RETURN ; Return to TASKA
```

Figure 4-9: Stack at the Beginning of Example 4-14

![Stack Diagram](image-url)
4.8.3 Stack Pointer Overflow

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

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

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

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

---

**Figure 4-10: Stack After "CALL COMPUTE" Executes**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Frame of TASKA</td>
</tr>
<tr>
<td>0x0800</td>
<td>Parameter 1</td>
</tr>
<tr>
<td></td>
<td>Parameter 2</td>
</tr>
<tr>
<td></td>
<td>Parameter 3</td>
</tr>
</tbody>
</table>
|         | PC<15:0>[
|         | 0:PC<22:16> |
| <TOS>   | W14 (FP)    |
| 0xFFFF  | W15 (SP)    |

**Note 1:** In dsPIC33E/PIC24E devices, the SFA bit is stacked instead of PC<0>.

**Figure 4-11: Stack After "LNK #4" Executes**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Frame of TASKA</td>
</tr>
<tr>
<td>0x0800</td>
<td>Parameter 1</td>
</tr>
<tr>
<td></td>
<td>Parameter 2</td>
</tr>
<tr>
<td></td>
<td>Parameter 3</td>
</tr>
</tbody>
</table>
|         | PC<15:0>[
|         | 0:PC<22:16> |
|         | FP of TASKA |
|         | Temp Word 1 |
|         | Temp Word 2 |
| <TOS>   | W14 (FP)    |
| 0xFFFF  | W15 (SP)    |

**Note 1:** In dsPIC33E/PIC24E devices, the SFA bit is stacked instead of PC<0>. 
4.8.4 Stack Pointer Underflow

The stack is initialized to 0x0800 during Reset (0x1000 for PIC24E, dsPIC33E and dsPIC33C devices). A stack error trap will be initiated should the Stack Pointer address ever be less than 0x0800 (0x1000 for PIC24E, dsPIC33E and dsPIC33C devices).

| Note: | Locations in data space, between 0x0000 and 0x07FF (0x0FFF for PIC24E, dsPIC33E and dsPIC33C devices), are in general, reserved for core and peripheral Special Function Registers (SFRs). |

4.8.5 Stack Frame Active (SFA) Control (dsPIC33E, dsPIC33C and PIC24E Devices)

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

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

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

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

| Note: | In dsPIC33E, dsPIC33C and PIC24E devices, the SFA bit is stacked instead of PC<0>. |
4.9 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, dsPIC33F, dsPIC33E and dsPIC33C devices are shown in Table 4-8. This table identifies the condition in the STATUS Register which must be true for the branch to be taken. In some cases, just a single bit is tested (as in $BRA\ C$), while in other cases, a complex logic operation is performed (as in $BRA\ GT$). For dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices, it is worth noting that both signed and unsigned conditional tests are supported, and that support is provided for DSP algorithms with the OA, OB, SA and SB condition mnemonics.

Table 4-8: Conditional Branch Instructions

<table>
<thead>
<tr>
<th>Condition Mnemonic(1)</th>
<th>Description</th>
<th>Status Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Carry (not Borrow)</td>
<td>C</td>
</tr>
<tr>
<td>GE</td>
<td>Signed Greater Than or Equal</td>
<td>$(\neg N &amp; \neg OV)</td>
</tr>
<tr>
<td>GEU(2)</td>
<td>Unsigned Greater Than or Equal</td>
<td>C</td>
</tr>
<tr>
<td>GT</td>
<td>Signed Greater Than</td>
<td>$(\neg Z &amp; \neg N &amp; \neg OV)</td>
</tr>
<tr>
<td>GTU</td>
<td>Unsigned Greater Than</td>
<td>$C &amp; Z$</td>
</tr>
<tr>
<td>LE</td>
<td>Signed Less Than or Equal</td>
<td>$Z</td>
</tr>
<tr>
<td>LEU</td>
<td>Unsigned Less Than or Equal</td>
<td>$C | Z$</td>
</tr>
<tr>
<td>LT</td>
<td>Signed Less Than</td>
<td>$(\neg N &amp; OV)</td>
</tr>
<tr>
<td>LTU(3)</td>
<td>Unsigned Less Than</td>
<td>C</td>
</tr>
<tr>
<td>N</td>
<td>Negative</td>
<td>N</td>
</tr>
<tr>
<td>NC</td>
<td>Not Carry (Borrow)</td>
<td>C</td>
</tr>
<tr>
<td>NN</td>
<td>Not Negative</td>
<td>N</td>
</tr>
<tr>
<td>NOV</td>
<td>Not Overflow</td>
<td>OV</td>
</tr>
<tr>
<td>NZ</td>
<td>Not Zero</td>
<td>Z</td>
</tr>
<tr>
<td>OA(4)</td>
<td>Accumulator A Overflow</td>
<td>OA</td>
</tr>
<tr>
<td>OB(4)</td>
<td>Accumulator B Overflow</td>
<td>OB</td>
</tr>
<tr>
<td>OV</td>
<td>Overflow</td>
<td>OV</td>
</tr>
<tr>
<td>SA(4)</td>
<td>Accumulator A Saturate</td>
<td>SA</td>
</tr>
<tr>
<td>SB(4)</td>
<td>Accumulator B Saturate</td>
<td>SB</td>
</tr>
<tr>
<td>Z</td>
<td>Zero</td>
<td>Z</td>
</tr>
</tbody>
</table>

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

**Note 2:** GEU is identical to C and will reverse assemble to $BRA\ C$, Expr.

**Note 3:** LTU is identical to NC and will reverse assemble to $BRA\ NC$, Expr.

**Note 4:** This condition is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

**Note:** The “Compare and Skip” instructions ($CPBEQ$, $CPBGT$, $CPBLT$, $CPBNE$, $CPSEQ$, $CPSGT$, $CPSLT$ and $CPSNE$) do not modify the STATUS Register.
4.10 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 multiprecision data (starting with an instruction with no Carry/Borrow input) will automatically logically AND the successive results of the zero test. All results must be zero for the Z flag to remain set at the end of the sequence of operations. If the result of the ADDC, CPB, SUBB or SUBBR instruction is non-zero, the Z bit will be cleared and remain cleared for all subsequent ADDC, CPB, SUBB or SUBBR instructions. Example 4-15 shows how the Z bit operates for a 32-bit addition. It shows how the Z bit is affected for a 32-bit addition implemented with an ADD/ADDC instruction sequence. The first example generates a zero result for only the most significant word, and the second example generates a zero result for both the least significant word and most significant word.

Example 4-15: ‘Z’ Status Bit Operation for 32-Bit Addition

; Add two doubles (W0:W1 and W2:W3)
; Store the result in W5:W4
ADD W0, W2, W4 ; Add LSWord and store to W4
ADDC W1, W3, W5 ; Add MSWord and store to W5

Before 32-Bit Addition (zero result for the most significant word):

W0 = 0x2342
W1 = 0xFFFF
W2 = 0x39AA
W3 = 0x0010
W4 = 0x0000
W5 = 0x0000
SR = 0x0000

After 32-Bit Addition:

W0 = 0x2342
W1 = 0xFFFF
W2 = 0x39AA
W3 = 0x0010
W4 = 0x5CEC
W5 = 0x0000
SR = 0x0201 (DC,C=1)

Before 32-Bit Addition (zero result for the least significant word and most significant word):

W0 = 0xB76E
W1 = 0xFB7B
W2 = 0x4892
W3 = 0x0484
W4 = 0x0000
W5 = 0x0000
SR = 0x0000

After 32-Bit Addition:

W0 = 0xB76E
W1 = 0xFB7B
W2 = 0x4892
W3 = 0x0485
W4 = 0x0000
W5 = 0x0000
SR = 0x0103 (DC,Z,C=1)
4.11 ASSIGNED WORKING REGISTER USAGE

The 16 Working registers of the 16-bit MCU and DSC devices provide a large register set for efficient code generation and algorithm implementation. In an effort to maintain an instruction set that provides advanced capability, a stable run-time environment and backwards compatibility with earlier Microchip processor cores, some Working registers have a preassigned usage. Table 4-9 summarizes these Working register assignments. For the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C, additional details are provided in subsections, Section 4.11.1 “Implied DSP Operands (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)” through Section 4.11.3 “PIC® Microcontroller Compatibility”.

Table 4-9: Special Working Register Assignments

<table>
<thead>
<tr>
<th>Register</th>
<th>Special Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>Default WREG, Divide Quotient for DIV instructions</td>
</tr>
<tr>
<td>W1</td>
<td>Divide Remainder for DIV instructions</td>
</tr>
<tr>
<td>W2</td>
<td>&quot;MUL f&quot; Product least significant word</td>
</tr>
<tr>
<td>W3</td>
<td>&quot;MUL f&quot; Product most significant word</td>
</tr>
<tr>
<td>W4</td>
<td>MAC Operand(1)</td>
</tr>
<tr>
<td>W5</td>
<td>MAC Operand(1)</td>
</tr>
<tr>
<td>W6</td>
<td>MAC Operand(1)</td>
</tr>
<tr>
<td>W7</td>
<td>MAC Operand(1)</td>
</tr>
<tr>
<td>W8</td>
<td>MAC Prefetch Address (X Memory)(1)</td>
</tr>
<tr>
<td>W9</td>
<td>MAC Prefetch Address (X Memory)(1)</td>
</tr>
<tr>
<td>W10</td>
<td>MAC Prefetch Address (Y Memory)(1)</td>
</tr>
<tr>
<td>W11</td>
<td>MAC Prefetch Address (Y Memory)(1)</td>
</tr>
<tr>
<td>W12</td>
<td>MAC Prefetch Offset(1)</td>
</tr>
<tr>
<td>W13</td>
<td>MAC Write-Back Destination(1)</td>
</tr>
<tr>
<td>W14</td>
<td>Frame Pointer</td>
</tr>
<tr>
<td>W15</td>
<td>Stack Pointer</td>
</tr>
</tbody>
</table>

Note 1: This assignment is only applicable in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

4.11.1 Implied DSP Operands (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)

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

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

These restrictions only apply to the DSP MAC class of instructions, which utilize Working registers and have prefetch ability (described in Section 4.16 “DSP Accumulator Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)”). These instructions are CLR, ED, EDAC, MAC, MOVSA, MPY, MPY.N and MSC.

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

The DSP accumulator class of instructions (described in Section 4.16 “DSP Accumulator Instructions (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)” is not required to follow the Working register assignments in Table 4-9 and may freely use any Working register when required.
4.11.2 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, RETFILE, 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 4.11.1 “Implied DSP Operands (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C Devices)”. If W14 and W15 must be used for other purposes (it is strongly advised that they remain reserved for the Frame and Stack Pointers), extreme care must be taken such that the run-time environment is not corrupted.

4.11.3 PIC® Microcontroller Compatibility

4.11.3.1 DEFAULT WORKING REGISTER (WREG)

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

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

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD RAM100</td>
<td>add RAM100 and WREG, store in RAM100</td>
</tr>
<tr>
<td>ASR RAM100, WREG</td>
<td>shift RAM100 right, store in WREG</td>
</tr>
<tr>
<td>CLR.B WREG</td>
<td>clear the WREG LS Byte</td>
</tr>
<tr>
<td>DEC RAM100, WREG</td>
<td>decrement RAM100, store in WREG</td>
</tr>
<tr>
<td>MOV WREG, RAM100</td>
<td>move WREG to RAM100</td>
</tr>
<tr>
<td>SETM WREG</td>
<td>set all bits in the WREG</td>
</tr>
<tr>
<td>XOR RAM100</td>
<td>XOR RAM100 and WREG, store in RAM100</td>
</tr>
</tbody>
</table>

4.11.3.2 PRODH:PRODL REGISTER PAIR

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

Despite this architectural difference, the 16-bit MCU and DSC devices still support the legacy file register multiply instruction (MULWF) with the "MUL(.B) f" instruction (described on page 323). 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.
Example 4-17: Unsigned f and WREG Multiply (Legacy \texttt{MULWF} Instruction)

\begin{verbatim}
MUL.B 0x100 ; (0x100)*WREG (byte mode), store to W2

Before Instruction:
\begin{itemize}
\item W0 (WREG) = 0x7705
\item W2 = 0x1235
\item W3 = 0x1000
\item Data Memory 0x0100 = 0x1255
\end{itemize}

After Instruction:
\begin{itemize}
\item W0 (WREG) = 0x7705
\item W2 = 0x01A9
\item W3 = 0x1000
\item Data Memory 0x0100 = 0x1255
\end{itemize}

MUL 0x100 ; (0x100)*WREG (word mode), store to W3:W2

Before Instruction:
\begin{itemize}
\item W0 (WREG) = 0x7705
\item W2 = 0x1235
\item W3 = 0x1000
\item Data Memory 0x0100 = 0x1255
\end{itemize}

After Instruction:
\begin{itemize}
\item W0 (WREG) = 0x7705
\item W2 = 0xDEA9
\item W3 = 0x0885
\item Data Memory 0x0100 = 0x1255
\end{itemize}
\end{verbatim}

4.11.3.3 MOVING DATA WITH WREG

The \texttt{MOV(.B) f{}, WREG}” instruction (described on page 299) and \texttt{MOV(.B) WREG, f} instruction (described on page 300) allow for byte or word data to be moved between file register memory and the WREG (Working register, W0). These instructions provide equivalent functionality to the legacy Microchip PIC MCU \texttt{MOVF} and \texttt{MOVWF} instructions.

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

Note: When moving word data between file register memory and the Working register array, the \texttt{MOV Wns, f}” and \texttt{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

\begin{verbatim}
MOV.B 0x1001, WREG ; move the byte stored at location 0x1001 to W0
MOV 0x1000, WREG ; move the word stored at location 0x1000 to W0
MOV.B WREG, TBLPAG ; move the byte stored at W0 to the TBLPAG register
MOV WREG, 0x804 ; move the word stored at W0 to location 0x804
\end{verbatim}
4.12 DSP DATA FORMATS (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

4.12.1 Integer and Fractional Data

The dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices support both integer and fractional data types. Integer data is inherently represented as a signed two’s complement value, where the Most Significant bit is defined as a sign bit. Generally speaking, the range of an N-bit two’s complement integer is $-2^{N-1}$ to $2^{N-1} - 1$. For a 16-bit integer, the data range is -32768 (0x8000) to 32767 (0x7FFF), including ‘0’. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to 2,147,483,647 (0x7FFF FFFF).

Fractional data is represented as a two’s complement number, where the Most Significant bit is defined as a sign bit and the radix point is implied to lie just after the sign bit. This format is commonly referred to as a 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 $(1 - 2^{-15})$. For a 16-bit fraction, the 1.15 data range is -1.0 (0x8000) to 0.999969482 (0x7FFF), including 0.0 and it has a precision of $3.05176 	imes 10^{-5}$. In Normal Saturation mode, the 32-bit accumulators use a 1.31 format, which enhances the precision to $4.6566 	imes 10^{-10}$.

The dynamic range of the accumulators can be expanded by using the eight bits of the Upper Accumulator register (ACCxU) as guard bits. Guard bits are used if the value stored in the accumulator overflows beyond the 32nd bit and they are useful for implementing DSP algorithms. This mode is enabled when the ACCSAT bit (CORCON<4>) is set to ‘1’ and it expands the accumulators to 40 bits. The guard bits are also used when the accumulator saturation is disabled. The accumulators then support an integer range of $-5.498 	imes 10^{11}$ (0x80 0000 0000) to $5.498 	imes 10^{11}$ (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 – $4.65661 	imes 10^{-10}$) (0x7F FFFF FFFF).

Table 4-10 identifies the range and precision of integers and fractions on the dsPIC30F/33F/33E/33C devices for 16-bit, 32-bit and 40-bit registers.

It should be noted that, with the exception of DSP multiplies, the ALU operates identically on integer and fractional data. Namely, an addition of two integers will yield the same result (binary number) as the addition of two fractional numbers. The only difference is how the result is interpreted by the user. However, multiplies performed by DSP operations are different. In these instructions, data format selection is made by the IF bit (CORCON<0>) and it must be set accordingly (‘0’ for Fractional mode, ‘1’ for Integer mode). This is required because of the implied radix point used by dsPIC30F/33F/33E/33C fractional numbers. In Integer mode, multiplying two 16-bit integers produces a 32-bit integer result. However, multiplying two 1.15 values generates a 2.30 result. Since the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices use a 1.31 format for the accumulators, a DSP multiply in Fractional mode also includes a left shift of one bit to keep the radix point properly aligned. This feature reduces the resolution of the DSP multiplier to $2^{-30}$, but has no other effect on the computation (e.g., $0.5 \times 0.5 = 0.25$).

<table>
<thead>
<tr>
<th>Register Size</th>
<th>Integer Range</th>
<th>Fraction Range</th>
<th>Fraction Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit</td>
<td>-32768 to 32767</td>
<td>-1.0 to $(1.0 - 2^{-15})$</td>
<td>$3.052 \times 10^{-5}$</td>
</tr>
<tr>
<td>32-bit</td>
<td>-2,147,483,648 to 2,147,483,647</td>
<td>-1.0 to $(1.0 - 2^{-31})$</td>
<td>$4.657 \times 10^{-10}$</td>
</tr>
<tr>
<td>40-bit</td>
<td>-549,755,813,888 to 549,755,813,887</td>
<td>-256.0 to $(256.0 - 2^{-31})$</td>
<td>$4.657 \times 10^{-10}$</td>
</tr>
</tbody>
</table>
4.12.2 Integer and Fractional Data Representation

Having a working knowledge of how integer and fractional data is represented on the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C is fundamental to working with the devices. Both integer and fractional data treat the Most Significant bit as a sign bit, and the binary exponent decreases by one as the bit position advances toward the Least Significant bit. The binary exponent for an N-bit integer starts at \((N-1)\) for the Most Significant bit and ends at \(0\) for the Least Significant bit. For an N-bit fraction, the binary exponent starts at \(0\) for the Most Significant bit and ends at \((1-N)\) for the Least Significant bit (as shown in Figure 4-12 for a positive value and in Figure 4-13 for a negative value).

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

**Figure 4-12: Different Representations of 0x4001**

<table>
<thead>
<tr>
<th>Integer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>(-2^{15} \quad 2^{14} \quad 2^{13} \quad 2^{12} \quad \ldots \ldots \quad 2^0)</td>
</tr>
<tr>
<td>0x4001 = (2^{14} + 2^0 = 16384 + 1 = 16385)</td>
</tr>
<tr>
<td>1.15 Fractional:</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>(-2^0 \cdot 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad \ldots \ldots \quad 2^{-15})</td>
</tr>
<tr>
<td>Implied Radix Point</td>
</tr>
<tr>
<td>0x4001 = (-2^{-1} + 2^{-15} = 0.5 + 0.000030518 = 0.500030518)</td>
</tr>
</tbody>
</table>

**Figure 4-13: Different Representations of 0xC002**

<table>
<thead>
<tr>
<th>Integer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>(-2^{15} \quad 2^{14} \quad 2^{13} \quad 2^{12} \quad \ldots \ldots \quad 2^0)</td>
</tr>
<tr>
<td>0xC002 = (-2^{15} + 2^{14} + 2^1 = -32768 + 16384 + 2 = -16382)</td>
</tr>
<tr>
<td>1.15 Fractional:</td>
</tr>
<tr>
<td>1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>(-2^0 \cdot 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad \ldots \ldots \quad 2^{-15})</td>
</tr>
<tr>
<td>Implied Radix Point</td>
</tr>
<tr>
<td>0xC002 = (-2^0 + 2^{-1} + 2^{-14} = -1.0 + 0.5 + 0.0000061035 = -0.499938965)</td>
</tr>
</tbody>
</table>
4.13 ACCUMULATOR USAGE (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

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

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

When super saturation is enabled, or when saturation is disabled, all registers of the accumulator may be used (Item D in Figure 4-14) and the results of DSP operations are stored in ACCxU:ACCxH:ACCxL. The benefit of ACCxU is that it increases the dynamic range of the accumulator, as described in Section 4.12.1 “Integer and Fractional Data”. Refer to Table 4-10 to see the range of values which may be stored in the accumulator when in Normal and Super Saturation modes.

Figure 4-14: Accumulator Alignment and Usage

<table>
<thead>
<tr>
<th>A)</th>
<th>ACCxU</th>
<th>ACCxH</th>
<th>ACCxL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>39</td>
<td>31,30</td>
<td>16</td>
</tr>
</tbody>
</table>

Implied Radix Point (between bits 31 and 30)

B) Load and store operations

C) Operations in Normal Saturation mode

D) Operations in Super Saturation mode or with saturation disabled

Note: dsPIC33C devices provide double-word LAC.D and SAC.D instructions, which allow both ACCxH and ACCxL to be loaded or stored in a single instruction.
4.14 ACCUMULATOR ACCESS (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

**Note 1:** For convenience, ACCAU and ACCBU are sign-extended to 16 bits. This provides the flexibility to access these registers using either Byte or Word mode (when File Register or Indirect Addressing is used).

2: The OA, OB, SA or SB bit cannot be set by writing overflowed values to the memory-mapped accumulators using MOV instructions, as these Status bits are only affected by DSP operations.

3: dsPIC33C devices provide double-word LAC.D and SAC.D instructions, which allow both ACCxH and ACCxL to be loaded or stored in a single instruction.

4.15 DSP MAC INSTRUCTIONS (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

The DSP Multiply and Accumulate (MAC) operations are a special suite of instructions which provide the most efficient use of the dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C architectures. The DSP MAC instructions, shown in Table 4-11, utilize both the X and Y data paths of the CPU core, which enables these instructions to perform the following operations all in one cycle:

- Two reads from data memory using prefetch Working registers (MAC Prefetches)
- Two updates to prefetch Working registers (MAC Prefetch Register Updates)
- One mathematical operation with an accumulator (MAC Operations)

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

**Table 4-11:** DSP MAC Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Accumulator WB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>Clear Accumulator</td>
<td>Yes</td>
</tr>
<tr>
<td>ED</td>
<td>Euclidean Distance (no accumulate)</td>
<td>No</td>
</tr>
<tr>
<td>EDAC</td>
<td>Euclidean Distance</td>
<td>No</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiply and Accumulate</td>
<td>Yes</td>
</tr>
<tr>
<td>MAC</td>
<td>Square and Accumulate</td>
<td>No</td>
</tr>
<tr>
<td>MOVXAC</td>
<td>Move from X and Y Bus</td>
<td>Yes</td>
</tr>
<tr>
<td>MPY</td>
<td>Multiply to Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>MPY</td>
<td>Square to Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>MPYN</td>
<td>Negative Multiply to Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>MSC</td>
<td>Multiply and Subtract</td>
<td>Yes</td>
</tr>
</tbody>
</table>
4.15.1 MAC Prefetches

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

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

4.15.2 MAC Prefetch Register Updates

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

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

4.15.3 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 prefetched from X and Y memory are not actually stored in W4-W7. Instead, only the difference of the prefetched data words is stored in W4-W7.

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

4.15.4 MAC Write-Back

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

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

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

Example 4-19: Base MAC Syntax

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

Example 4-20: MAC Prefetch Syntax
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.

Example 4-21: MAC Accumulator WB Syntax

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

Example 4-22: MSC Instruction with Two Prefetches and Accumulator Write-Back
4.16 DSP ACCUMULATOR INSTRUCTIONS (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

Table 4-12: DSP Accumulator Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Accumulator WB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add Accumulators</td>
<td>No</td>
</tr>
<tr>
<td>ADD</td>
<td>16-Bit Signed Accumulator Add</td>
<td>No</td>
</tr>
<tr>
<td>LAC</td>
<td>Load Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>LAC.D</td>
<td>Load Accumulator Double Word</td>
<td>No</td>
</tr>
<tr>
<td>NEG</td>
<td>Negate Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>SAC</td>
<td>Store Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>SAC.D</td>
<td>Store Accumulator Double Word</td>
<td>No</td>
</tr>
<tr>
<td>SAC.R</td>
<td>Store Rounded Accumulator</td>
<td>No</td>
</tr>
<tr>
<td>SFTAC</td>
<td>Arithmetic Shift Accumulator by Literal</td>
<td>No</td>
</tr>
<tr>
<td>SFTAC</td>
<td>Arithmetic Shift Accumulator by (Wn)</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract Accumulators</td>
<td>No</td>
</tr>
</tbody>
</table>

4.17 SCALING DATA WITH THE FBCL INSTRUCTION (dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

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

The FBCL instruction determines the exponent of a word by detecting the first bit change, starting from the value’s sign bit and working towards the LSB. Since the dsPIC DSC device’s barrel shifter uses negative values to specify a left shift, the FBCL instruction returns the negated exponent of a value. If the value is being scaled up, this allows the ensuing shift to be performed immediately with the value returned by FBCL. Additionally, since the FBCL instruction only operates on signed quantities, FBCL produces results in the range of -15:0. When the FBCL instruction returns 0, it indicates that the value is already at full scale. When the instruction returns -15, it indicates that the value cannot be scaled (as is the case with 0x0 and 0xFFFF).

Table 4-13 shows word data with various dynamic ranges, their exponents and the value after scaling each data to maximize the dynamic range. Example 4-23 shows how the FBCL instruction may be used for block processing.
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 \texttt{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.

\begin{verbatim}
; assume \texttt{W0} contains the largest absolute value of the data block
; assume \texttt{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
\texttt{FBCL \ W0, W2} ; store exponent in \texttt{W2}

; scale the entire data block before processing
\texttt{DO \ (#(BLOCK_SIZE-1), SCALE}
\texttt{LAC [W4], A} ; move the next data sample to ACCA
\texttt{SFTAC A, W2} ; shift ACCA by \texttt{W2} bits
\texttt{SCALE:}
\texttt{SAC A, [W4++]} ; store scaled input (overwrite original)
\texttt{ SCALE:}
\texttt{SAC A, [W4++]} ; store scaled input (overwrite original)
\texttt{ ; now process the data}
\texttt{ ; (processing block goes here)}
\end{verbatim}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Word Value & Exponent & Full-Scale Value (Word Value \texttt{<<} Exponent) \\
\hline
0x0001 & 14 & 0x4000 \\
0x0002 & 13 & 0x4000 \\
0x0004 & 12 & 0x4000 \\
0x0100 & 6 & 0x4000 \\
0x01FF & 6 & 0x7F00 \\
0x0806 & 3 & 0x4030 \\
0x2007 & 1 & 0x400E \\
0x4800 & 0 & 0x4800 \\
0x7000 & 0 & 0x7000 \\
0x8000 & 0 & 0x8000 \\
0x900A & 0 & 0x900A \\
0xE001 & 2 & 0x8004 \\
0xFF07 & 7 & 0x8380 \\
\hline
\end{tabular}
\caption{Scaling Examples}
\end{table}

\textbf{Note:} For the word values, 0x0000 and 0xFFFF, the \texttt{FBCL} instruction returns -15.
4.18 DATA RANGE LIMIT INSTRUCTIONS (dsPIC33C DEVICES ONLY)

The dsPIC33C family provides special instructions that automatically limit the data in a W register or an accumulator to lie within a user-specified numerical range. These include the FLIM, MAX, MIN and MINZ instructions.

4.18.1 FLIM/FLIM.V

The FLIM instruction simultaneously compares a maximum and a minimum data limit value with the specified W register (or data pointed to by the W register), and clamps the target data to the user-specified limit if the limit is exceeded. SR Status bits are set accordingly for subsequent signed branching. In the FLIM.V instruction, an additional W register is specified, in which the limit test result (known as "limit excess") value is loaded.

4.18.2 MAX/MAX.V

The MAX instruction compares a maximum data limit value (stored in a W register or the other accumulator) with the target accumulator and clamps the target accumulator to the user-specified limit if this upper limit is exceeded. SR Status bits are set accordingly for subsequent signed branching. In the MAX.V instruction, an additional W register (or W register in Indirect Addressing mode) is specified, in which the limit excess value is loaded.

4.18.3 MIN/MIN.V/MINZ

The MIN instruction compares a minimum data limit value (stored in a W register or the other accumulator) with the target accumulator and clamps the target accumulator to the user-specified limit if the data is smaller than this minimum limit. SR Status bits are set accordingly for subsequent signed branching. In the MIN.V instruction, an additional W register (or W register in Indirect Addressing mode) is specified, in which the limit excess value is loaded. The MINZ instruction is simply a conditional version of the MIN instruction, which is executed only when Z = 1.
4.19 NORMALIZING THE ACCUMULATOR WITH THE FBCL INSTRUCTION
(dsPIC30F, dsPIC33F, dsPIC33E AND dsPIC33C DEVICES)

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

The two 40-bit accumulators each have eight guard bits from the ACCxU register, which expands the dynamic range of the accumulators from 1.31 to 9.31 when operating in Super Saturation mode (see Section 4.12.1 “Integer and Fractional Data”). However, even in Super Saturation mode, the Store Rounded Accumulator (SAC.R) instruction only stores 16-bit data (in 1.15 format) from ACCxH, as described in Section Figure 4-13: “Different Representations of 0xC002”. Under certain conditions, this may pose a problem.

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

```
; assume an operation in ACCA has just completed (SR intact)
; 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], W0       ; extract exponent for left shift
   BRA SHIFT_ACC       ; branch to the shift
FBCL_GUARD:
   FBCL [++W5], W0     ; extract exponent for right shift
   ADD.B W0, #15, W0   ; adjust the sign for right shift
SHIFT_ACC:
   SFTAC A, W0         ; shift ACCA to normalize
```

4.20 NORMALIZING THE ACCUMULATOR WITH THE NORM INSTRUCTION
(dsPIC33C DEVICES ONLY)

The NORM instruction automatically normalizes the target accumulator to obtain the largest fractional value possible without overflow. The target accumulator must contain signed fractional data for the instruction result to be valid. It will shift the target accumulator right or left by as many bits as required to normalize the data, keeping the sign consistent. The shift value is stored in a destination address. The N Status bit reflects the direction of the accumulator shift.

If the accumulator cannot be normalized, the accumulator contents will not be affected.
4.21 EXTENDED PRECISION ARITHMETIC USING MIXED-SIGN MULTIPLICATIONS (dsPIC33E AND dsPIC33C ONLY)

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

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

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

Example 4-25: 32-Bit Signed Multiplication Using Implicit Mixed-Sign Mode

<table>
<thead>
<tr>
<th>Case A: Mixed-Sign Multiplication Mode Not Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL.SU W5, W6, W0 ; Word1 (signed) x Word2 (unsigned)</td>
</tr>
<tr>
<td>MUL.US W4, W7, W2 ; Word0 (unsigned) x Word3 (signed)</td>
</tr>
<tr>
<td>CLR B ; Clear Accumulator B</td>
</tr>
<tr>
<td>ADD W1, B</td>
</tr>
<tr>
<td>ADD W3, B</td>
</tr>
<tr>
<td>SFTAC B, #15 ; Shift right by 15 bits to align for Q31 format</td>
</tr>
<tr>
<td>MAC W5*W7, B ; Word1 (signed) x Word 3 (signed)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case B: Mixed-Sign Multiplication Mode Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPY W5*W6, B ; Word1 (signed) x Word2 (unsigned)</td>
</tr>
<tr>
<td>MAC W4*W7, B ; Word0 (unsigned) x Word3 (signed)</td>
</tr>
<tr>
<td>SFTAC B, #15 ; Shift right by 15 bits to align for Q31 format</td>
</tr>
<tr>
<td>MAC W5*W7, B ; Word1 (signed) x Word 3 (signed)</td>
</tr>
</tbody>
</table>

Besides DSP instructions, MCU Multiplication (MUL) instructions can also utilize Accumulator A or Accumulator B as a result destination, which enables faster extended precision arithmetic, even when not using DSP multiplication instructions such as MPY or MAC.
Section 5. Instruction Descriptions

HIGHLIGHTS

This section of the manual contains the following major topics:

5.1 Instruction Symbols......................................................................................................... 96
5.2 Instruction Encoding Field Descriptors Introduction........................................................ 96
5.3 Instruction Description Example ................................................................................... 101
5.4 Instruction Descriptions................................................................................................. 102
5.1 INSTRUCTION SYMBOLS

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

5.2 INSTRUCTION ENCODING FIELD DESCRIPTORS INTRODUCTION

All instruction encoding field descriptors used in Section 5.4 “Instruction Descriptions” are shown in Table 5-2 through Table 5-15.

Table 5-1: Instruction Encoding Field Descriptors

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(1)</td>
<td>Accumulator selection bit: 0 = ACCA; 1 = ACCB</td>
</tr>
<tr>
<td>aa(1)</td>
<td>Accumulator Write-Back mode (see Table 5-13)</td>
</tr>
<tr>
<td>B</td>
<td>Byte mode selection bit: 0 = word operation; 1 = byte operation</td>
</tr>
<tr>
<td>bbbb</td>
<td>4-bit bit position select: 0000 = LSB; 1111 = MSB</td>
</tr>
<tr>
<td>D</td>
<td>Destination address bit: 0 = result stored in WREG; 1 = result stored in file register</td>
</tr>
<tr>
<td>dddd</td>
<td>Wd Destination register select: 0000 = W0; 1111 = W15</td>
</tr>
<tr>
<td>f fffe fffe fffe</td>
<td>13-bit register file address (0x0000 to 0x1FFF)</td>
</tr>
<tr>
<td>fffe fffe fffe fffe</td>
<td>15-bit register file word address – implied 0 LSB (0x0000 to 0xFFFF)</td>
</tr>
<tr>
<td>fffe fffe fffe fffe</td>
<td>16-bit register file byte address (0x0000 to 0xFFFFFF)</td>
</tr>
<tr>
<td>gggg</td>
<td>Register Offset Addressing mode for Ws Source register (see Table 5-5)</td>
</tr>
<tr>
<td>hhhh</td>
<td>Register Offset Addressing mode for Wd Destination register (see Table 5-6)</td>
</tr>
<tr>
<td>iiii(1)</td>
<td>Prefetch X operation (see Table 5-7)</td>
</tr>
<tr>
<td>jjjj(1)</td>
<td>Prefetch Y operation (see Table 5-9)</td>
</tr>
<tr>
<td>k</td>
<td>1-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk</td>
<td>4-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk kkkk</td>
<td>6-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk kkkk kkkk</td>
<td>8-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk kkkk kkkk kkkk</td>
<td>10-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk kkkk kkkk kkkk kkkk</td>
<td>14-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>kkkk kkkk kkkk kkkk kkkk kkkk</td>
<td>16-bit literal field, constant data or expression</td>
</tr>
<tr>
<td>mm</td>
<td>Multiplier source select with same Working registers (see Table 5-11)</td>
</tr>
<tr>
<td>mmmm</td>
<td>Multiplier source select with different Working registers (see Table 5-12)</td>
</tr>
<tr>
<td>nnnn nnnn nnnn nnn0 nnn nnnn</td>
<td>23-bit program address for CALL and GOTO instructions</td>
</tr>
<tr>
<td>nnnn nnnn nnnn nnnnn</td>
<td>16-bit program offset field for relative branch/call instructions</td>
</tr>
<tr>
<td>pppp</td>
<td>Addressing mode for Ws Source register (see Table 5-2)</td>
</tr>
<tr>
<td>qqqq</td>
<td>Addressing mode for Wd Destination register (see Table 5-3)</td>
</tr>
<tr>
<td>rrrr</td>
<td>Barrel shift count</td>
</tr>
<tr>
<td>ssss</td>
<td>Ws Source register select: 0000 = W0; 1111 = W15</td>
</tr>
<tr>
<td>tttt</td>
<td>Dividend select, most significant word</td>
</tr>
<tr>
<td>vvvv</td>
<td>Dividend select, least significant word</td>
</tr>
<tr>
<td>w</td>
<td>Double-Word mode selection bit: 0 = word operation; 1 = double-word operation</td>
</tr>
<tr>
<td>wwww</td>
<td>Wb Base register select: 0000 = W0; 1111 = W15</td>
</tr>
<tr>
<td>xxx(1)</td>
<td>Prefetch X destination (see Table 5-8)</td>
</tr>
<tr>
<td>yyyy(1)</td>
<td>16-bit unused field (don’t care)</td>
</tr>
<tr>
<td>z</td>
<td>Bit test destination: 0 = C flag bit; 1 = Z flag bit</td>
</tr>
</tbody>
</table>

Note 1: This field is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
## Section 5. Instruction Descriptions

### Table 5-2: Addressing Modes for Ws Source Register

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Source Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Direct</td>
<td>Ws</td>
</tr>
<tr>
<td>Indirect [Ws]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Post-Decrement [Ws--]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Post-Increment [Ws++]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Pre-Decrement [-Ws]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Pre-Increment [++Ws]</td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Destination Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Direct</td>
<td>Wd</td>
</tr>
<tr>
<td>Indirect [Wd]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Post-Decrement [Wd--]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Post-Increment [Wd++]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Pre-Decrement [-Wd]</td>
<td></td>
</tr>
<tr>
<td>Indirect with Pre-Increment [++Wd]</td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td>(an attempt to use this addressing mode will force a RESET instruction)</td>
</tr>
</tbody>
</table>

### Table 5-4: Destination Addressing Modes for MCU Multiplications

<table>
<thead>
<tr>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1:W0</td>
</tr>
<tr>
<td>W0</td>
</tr>
<tr>
<td>W3:W2</td>
</tr>
<tr>
<td>W2</td>
</tr>
<tr>
<td>W5:W4</td>
</tr>
<tr>
<td>W4</td>
</tr>
<tr>
<td>W7:W6</td>
</tr>
<tr>
<td>W6</td>
</tr>
<tr>
<td>W9:W8</td>
</tr>
<tr>
<td>W8</td>
</tr>
<tr>
<td>W11:W10</td>
</tr>
<tr>
<td>W10</td>
</tr>
<tr>
<td>W13:W12</td>
</tr>
<tr>
<td>W12</td>
</tr>
<tr>
<td>ACCA&lt;39:0&gt;</td>
</tr>
<tr>
<td>ACCB&lt;39:0&gt;</td>
</tr>
</tbody>
</table>
### Table 5-5: Offset Addressing Modes for Ws Source Register (with Register Offset)

<table>
<thead>
<tr>
<th>ggg</th>
<th>Addressing Mode</th>
<th>Source Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Register Direct</td>
<td>Ws</td>
</tr>
<tr>
<td>001</td>
<td>Indirect</td>
<td>[Ws]</td>
</tr>
<tr>
<td>010</td>
<td>Indirect with Post-Decrement</td>
<td>[Ws–]</td>
</tr>
<tr>
<td>011</td>
<td>Indirect with Post-Increment</td>
<td>[Ws++]</td>
</tr>
<tr>
<td>100</td>
<td>Indirect with Pre-Decrement</td>
<td>[–Ws]</td>
</tr>
<tr>
<td>101</td>
<td>Indirect with Pre-Increment</td>
<td>[++Ws]</td>
</tr>
<tr>
<td>11x</td>
<td>Indirect with Register Offset</td>
<td>[Ws+Wb]</td>
</tr>
</tbody>
</table>

### Table 5-6: Offset Addressing Modes for Wd Destination Register (with Register Offset)

<table>
<thead>
<tr>
<th>hhh</th>
<th>Addressing Mode</th>
<th>Source Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Register Direct</td>
<td>Wd</td>
</tr>
<tr>
<td>001</td>
<td>Indirect</td>
<td>[Wd]</td>
</tr>
<tr>
<td>010</td>
<td>Indirect with Post-Decrement</td>
<td>[Wd–]</td>
</tr>
<tr>
<td>011</td>
<td>Indirect with Post-Increment</td>
<td>[Wd++]</td>
</tr>
<tr>
<td>100</td>
<td>Indirect with Pre-Decrement</td>
<td>[–Wd]</td>
</tr>
<tr>
<td>101</td>
<td>Indirect with Pre-Increment</td>
<td>[++Wd]</td>
</tr>
<tr>
<td>11x</td>
<td>Indirect with Register Offset</td>
<td>[Wd+Wb]</td>
</tr>
</tbody>
</table>

### Table 5-7: X Data Space Prefetch Operation (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>iiii</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Wxd = [W8]</td>
</tr>
<tr>
<td>0001</td>
<td>Wxd = [W8], W8 = W8 + 2</td>
</tr>
<tr>
<td>0010</td>
<td>Wxd = [W8], W8 = W8 + 4</td>
</tr>
<tr>
<td>0011</td>
<td>Wxd = [W8], W8 = W8 + 6</td>
</tr>
<tr>
<td>0100</td>
<td>No Prefetch for X Data Space</td>
</tr>
<tr>
<td>0101</td>
<td>Wxd = [W8], W8 = W8 – 6</td>
</tr>
<tr>
<td>0110</td>
<td>Wxd = [W8], W8 = W8 – 4</td>
</tr>
<tr>
<td>0111</td>
<td>Wxd = [W8], W8 = W8 – 2</td>
</tr>
<tr>
<td>1000</td>
<td>Wxd = [W9]</td>
</tr>
<tr>
<td>1001</td>
<td>Wxd = [W9], W9 = W9 + 2</td>
</tr>
<tr>
<td>1010</td>
<td>Wxd = [W9], W9 = W9 + 4</td>
</tr>
<tr>
<td>1011</td>
<td>Wxd = [W9], W9 = W9 + 6</td>
</tr>
<tr>
<td>1100</td>
<td>Wxd = [W9 + W12]</td>
</tr>
<tr>
<td>1101</td>
<td>Wxd = [W9], W9 = W9 – 6</td>
</tr>
<tr>
<td>1110</td>
<td>Wxd = [W9], W9 = W9 – 4</td>
</tr>
<tr>
<td>1111</td>
<td>Wxd = [W9], W9 = W9 – 2</td>
</tr>
</tbody>
</table>

### Table 5-8: X Data Space Prefetch Destination (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>xx</th>
<th>Wxd</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>W4</td>
</tr>
<tr>
<td>01</td>
<td>W5</td>
</tr>
<tr>
<td>10</td>
<td>W6</td>
</tr>
<tr>
<td>11</td>
<td>W7</td>
</tr>
</tbody>
</table>
### Table 5-9: Y Data Space Prefetch Operation (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Wyd = [W10]</td>
</tr>
<tr>
<td>0001</td>
<td>Wyd = [W10], W10 = W10 + 2</td>
</tr>
<tr>
<td>0010</td>
<td>Wyd = [W10], W10 = W10 + 4</td>
</tr>
<tr>
<td>0011</td>
<td>Wyd = [W10], W10 = W10 + 6</td>
</tr>
<tr>
<td>0100</td>
<td>No Prefetch for Y Data Space</td>
</tr>
<tr>
<td>0101</td>
<td>Wyd = [W10], W10 = W10 – 6</td>
</tr>
<tr>
<td>0110</td>
<td>Wyd = [W10], W10 = W10 – 4</td>
</tr>
<tr>
<td>0111</td>
<td>Wyd = [W10], W10 = W10 – 2</td>
</tr>
<tr>
<td>1000</td>
<td>Wyd = [W11]</td>
</tr>
<tr>
<td>1001</td>
<td>Wyd = [W11], W11 = W11 + 2</td>
</tr>
<tr>
<td>1010</td>
<td>Wyd = [W11], W11 = W11 + 4</td>
</tr>
<tr>
<td>1011</td>
<td>Wyd = [W11], W11 = W11 + 6</td>
</tr>
<tr>
<td>1100</td>
<td>Wyd = [W11 + W12]</td>
</tr>
</tbody>
</table>

### Table 5-10: Y Data Space Prefetch Destination (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>yy</th>
<th>Wyd</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>W4</td>
</tr>
<tr>
<td>01</td>
<td>W5</td>
</tr>
<tr>
<td>10</td>
<td>W6</td>
</tr>
<tr>
<td>11</td>
<td>W7</td>
</tr>
</tbody>
</table>

### Table 5-11: MAC or MPY Source Operands – Same Working Register (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>mm</th>
<th>Multiplicands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>W4 * W4</td>
</tr>
<tr>
<td>01</td>
<td>W5 * W5</td>
</tr>
<tr>
<td>10</td>
<td>W6 * W6</td>
</tr>
<tr>
<td>11</td>
<td>W7 * W7</td>
</tr>
</tbody>
</table>

### Table 5-12: MAC or MPY Source Operands – Different Working Register (dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C)

<table>
<thead>
<tr>
<th>mm</th>
<th>Multiplicands</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>W4 * W5</td>
</tr>
<tr>
<td>001</td>
<td>W4 * W6</td>
</tr>
<tr>
<td>010</td>
<td>W4 * W7</td>
</tr>
<tr>
<td>011</td>
<td>Invalid</td>
</tr>
<tr>
<td>100</td>
<td>W5 * W6</td>
</tr>
<tr>
<td>101</td>
<td>W5 * W7</td>
</tr>
<tr>
<td>110</td>
<td>W6 * W7</td>
</tr>
<tr>
<td>111</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
Table 5-13: MAC Accumulator Write-Back Selection (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C)

<table>
<thead>
<tr>
<th>aa</th>
<th>Write-Back Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>W13 = Other Accumulator (Direct Addressing)</td>
</tr>
<tr>
<td>01</td>
<td>[W13] + = 2 = Other Accumulator (Indirect Addressing with Post-Increment)</td>
</tr>
<tr>
<td>10</td>
<td>No Write-Back</td>
</tr>
<tr>
<td>11</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Table 5-14: MOVPAG Destination Selection (dsPIC33E, dsPIC33C and PIC24E)

<table>
<thead>
<tr>
<th>PP</th>
<th>Target Page Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>DSRPAG</td>
</tr>
<tr>
<td>01</td>
<td>DSWPAG</td>
</tr>
<tr>
<td>10</td>
<td>TBLPAG</td>
</tr>
<tr>
<td>11</td>
<td>Invalid (results in Illegal Opcode Reset – do not use)</td>
</tr>
</tbody>
</table>

Table 5-15: Accumulator Selection (dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C)

<table>
<thead>
<tr>
<th>A</th>
<th>Target Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Accumulator A</td>
</tr>
<tr>
<td>1</td>
<td>Accumulator B</td>
</tr>
</tbody>
</table>
5.3 INSTRUCTION DESCRIPTION EXAMPLE

The example description below is for the fictitious instruction, \texttt{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

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Cells marked with an ‘X’ indicate the instruction is implemented for that device family.

Syntax: The Syntax field consists of an optional label, the instruction mnemonic, any optional extensions which exist for the instruction and the operands for the instruction. Most instructions support more than one operand variant to support the various addressing modes. In these circumstances, all possible instruction operands are listed beneath each other and are enclosed in braces.

Operands: 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 that 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 WREG

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} ADD{.B} f {,WREG}

Operands: \( f \in [0 \ldots 8191] \)

Operation: \( (f) + (WREG) \to \text{destination designated by D} \)

Status Affected: DC, N, OV, Z, C

Encoding:

\[
\begin{array}{cccccccc}
1011 & 0100 & 0BDf & ffff & ffff & ffff & ffff \\
\end{array}
\]

Description: Add the contents of the default Working register WREG 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 '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.

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.

Note 2: The WREG is set to Working register, W0.

Words: 1

Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multicycle Instructions".

Example 1: \( \text{ADD.B} \ RAM100 \); Add WREG to RAM100 (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>RAM100</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC80</td>
<td>FFC0</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>RAM100</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC80</td>
<td>FF40</td>
<td>0005</td>
</tr>
</tbody>
</table>

(OV, C = 1)

Example 2: \( \text{ADD} \ RAM200, \text{WREG} \); Add RAM200 to WREG (Word mode)

Before Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>RAM200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC80</td>
<td>FFC0</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>RAM200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC40</td>
<td>FFC0</td>
<td>0001</td>
</tr>
</tbody>
</table>

(C = 1)
ADD

Add Literal to Wn

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} ADD{.B} #lit10, Wn

Operands:

lit10 ∈ [0 ... 255] for byte operation
lit10 ∈ [0 ... 1023] for word operation
Wn ∈ [W0 ... W15]

Operation:

lit10 + (Wn) → Wn

Status Affected:

DC, N, OV, Z, C

Encoding:

| 1011 | 0000 | 0Bkk | kkkk | kkkk | dddd |

Description:

Add the 10-bit unsigned literal operand to the contents of the Working register Wn 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.

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.

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.

Words: 1

Cycles: 1

Example 1:

ADD.B #0xFF, W7 ; Add -1 to W7 (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>W7</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>12C0</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W7</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>12BF</td>
<td>0009</td>
</tr>
</tbody>
</table>

N, C = 1

Example 2:

ADD #0xFF, W1 ; Add 255 to W1 (Word mode)

Before Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>12C0</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13BF</td>
<td>0000</td>
</tr>
</tbody>
</table>
ADD

Add Wb to Short Literal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wb</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>#lit5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Wd]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Wd++]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Wd--]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[++]Wd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[--Wd]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{label:} ADD{.B} Wb, #lit5, Wd

Operands:

Wb ∈ [W0 ... W15]
#lit5 ∈ [0 ... 31]
Wd ∈ [W0 ... W15]

Operation:

(Wb) + #lit5 → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0www</td>
<td>wBqq</td>
<td>qddd</td>
<td>d11k</td>
<td>kkkk</td>
<td></td>
</tr>
</tbody>
</table>

Description:

Add the contents of the base register Wb to the 5-bit unsigned short literal operand and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or Indirect Addressing may be used for 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘k’ bits provide the literal operand, a five-bit integer number.

Note: 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

Example 1:

ADD.B W0, #0x1F, W7 ; Add W0 and 31 (Byte mode)
                     ; Store the result in W7

Before Instruction  After Instruction
W0  2290             W0  2290
W7  12C0             W7  12AF
SR  0000             SR  0008 (N = 1)

Example 2:

ADD W3, #0x6, [--W4] ; Add W3 and 6 (Word mode)
                     ; Store the result in [--W4]

Before Instruction  After Instruction
W3  6006             W3  6006
W4  1000             W4  0FFE
Data 0FFE  DDEE     Data 0FFE  600C
Data 1000  DDEE     Data 1000  DDEE
SR  0000             SR  0000
Section 5. Instruction Descriptions

ADD

Add Wb to Ws

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | X |

Syntax:

\{label:\} \text{ADD\{.B\} Wb, Ws, Wd}

\[Wb, Ws, Wd\]

\[Ws, Wd\]

\[Ws++, Wd++\]

\[Ws--, Wd--\]

\[++Ws, ++Wd\]

\[--Ws, --Wd\]

Operands:

\(Wb \in [W0 \ldots W15]\)

\(Ws \in [W0 \ldots W15]\)

\(Wd \in [W0 \ldots W15]\)

Operation:

\((Wb) + (Ws) \rightarrow Wd\)

Status Affected:

DC, N, OV, Z, C

Encoding:

\[
\begin{array}{cccccc}
0100 & 0www & wBqq & qddd & dppp & ssss \\
\end{array}
\]

Description:

Add the contents of the source register Ws and 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 or 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: The extension \text{.B} in the instruction denotes a byte operation rather than a word operation. You may use a \text{.W} extension to denote a word operation, but it is not required.

Words: 1

Cycles: 1 (1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

ADD.B W5, W6, W7 ; Add W5 to W6, store result in W7 ; (Byte mode)

Before Instruction | After Instruction
---|---
W5 AB00 | W5 AB00
W6 0030 | W6 0030
W7 FFFF | W7 FF30
SR 0000 | SR 0000
Example 2:  

```
ADD W5, W6, W7 ; Add W5 to W6, store result in W7
; (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>AB00</td>
</tr>
<tr>
<td>W6</td>
<td>0030</td>
</tr>
<tr>
<td>W7</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>W5</td>
<td>AB00</td>
</tr>
<tr>
<td>W6</td>
<td>0030</td>
</tr>
<tr>
<td>W7</td>
<td>AB30</td>
</tr>
<tr>
<td>SR</td>
<td>0008 (N = 1)</td>
</tr>
</tbody>
</table>
ADD

Add Accumulators

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label;} ADD Acc

Operands:

Acc ∈ [A, B]

Operation:

If (Acc = A):

(ACCA) + (ACCB) → ACCA

Else:

(ACCA) + (ACCB) → ACCB

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 B and place the result in the selected accumulator. This instruction performs a 40-bit addition.

The ‘A’ bit specifies the destination accumulator.

Words:

1

Cycles:

1

Example 1:

ADD A ; Add ACCB to ACCA

Before Instruction

| ACCA | 00 0022 3300 |
| ACCB | 00 1833 4558 |
| SR   | 0000         |

After Instruction

| ACCA | 00 1855 7858 |
| ACCB | 00 1833 4558 |
| SR   | 0000         |

Example 2:

ADD B ; Add ACCA to ACCB

; Assume Super Saturation mode enabled
; (ACCSAT = 1, SATA = 1, SATB = 1)

Before Instruction

| ACCA | 00 E111 2222 |
| ACCB | 00 7654 3210 |
| SR   | 0000         |

After Instruction

| ACCA | 00 E111 2222 |
| ACCB | 01 5765 4532 |
| SR   | 4800 (OB, OAB = 1) |
ADD

16-Bit Signed Add to Accumulator

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} ADD Ws, {#Slit4,} Acc

[Ws],
[Ws++],
[Ws--],
[--Ws],
[++Ws],
[Ws+Wb],

Operands:

Ws ∈ [W0 ... W15]
Wb ∈ [W0 ... W15]
Slit4 ∈ [-8 ... +7]
Acc ∈ [A,B]

Operation:

\[ \text{Shift}_{\text{S}}(\text{Extend}(Ws)) + (\text{Acc}) \rightarrow \text{Acc} \]

Status Affected:

OA, OB, OAB, SA, SB, SAB

Encoding:

| 1100 | 1001 | Awww | wrrr | rggg | ssss |

Description:

Add a 16-bit value specified by the source Working register to the most significant word of the selected accumulator. The source operand may specify the direct contents of a Working register or an Effective Address. The value specified is added to the most significant word of the accumulator by sign-extending and zero backfilling the source operand prior to the operation. The value added to the accumulator may also be shifted by a 4-bit signed literal before the addition is made.

The ‘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 addressing mode.
The ‘s’ bits specify the source register Ws.

Note: Positive values of operand Slit4 represent an arithmetic shift right and negative values of operand Slit4 represent an arithmetic shift left. The contents of the source register are not affected by Slit4.

Words: 1
Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

ADD W0, #2, A ; Add W0 right-shifted by 2 to ACCA

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 8000</td>
<td>W0 8000</td>
</tr>
<tr>
<td>ACCA 00 7000 0000</td>
<td>ACCA 00 5000 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Example 2: \texttt{ADD \[W5++], A} \quad ; \text{Add the effective value of W5 to ACCA}

\begin{itemize}
  \item \texttt{Post-increment W5}
\end{itemize}

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 0067 2345</td>
</tr>
<tr>
<td>Data 2000</td>
<td>5000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W5</td>
<td>2002</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 5067 2345</td>
</tr>
<tr>
<td>Data 2000</td>
<td>5000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### ADDC

#### Add f to WREG with Carry

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[(\text{label:}) \quad \text{ADDC}\{.B\} \quad f \quad \{.WREG\}\]

**Operands:**

\(f \in [0 \ldots 8191]\)

**Operation:**

\((f) + (WREG) + (C) \rightarrow \text{destination designated by D}\)

**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.

**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.

**Note 2:** The WREG is set to Working register W0.

**Note 3:** The Z flag is “sticky” for ADDC, CFB, SUBB and SUBBR. These instructions can only clear Z.

**Words:**

1

**Cycles:**

1\(^{(1)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.

**Example 1:**

```
ADDC.B RAM100 ; Add WREG and C bit to RAM100
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG CC60</td>
<td>WREG CC60</td>
</tr>
<tr>
<td>RAM100 8006</td>
<td>RAM100 8067</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
ADDC RAM200, WREG ; Add RAM200 and C bit to the WREG
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 5600</td>
<td>WREG 8A01</td>
</tr>
<tr>
<td>RAM200 3400</td>
<td>RAM200 3400</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 000C (N, OV = 1)</td>
</tr>
</tbody>
</table>
### ADDC

**Add Literal to Wn with Carry**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} ADDC{.B} #lit10, Wn
```

**Operands:**

- `lit10` ∈ [0 ... 255] for byte operation
- `lit10` ∈ [0 ... 1023] for word operation
- `Wn` ∈ [W0 ... W15]

**Operation:**

\[
lit10 + (Wn) + (C) \rightarrow Wn
\]

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
1011 0000 lBkk 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.

**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.

**Note 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.

**Note 3:** The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

**Words:** 1

**Cycles:** 1

**Example 1:**

```
ADDC.B  #0xFF, W7 ; Add -1 and C bit to W7 (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 12C0</td>
<td>W7 12BF</td>
</tr>
<tr>
<td>SR 0000 (C = 0)</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

```
ADDC  #0xFF, W1 ; Add 255 and C bit to W1 (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 12C0</td>
<td>W1 13C0</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
ADDC

Add Wb to Short Literal with Carry

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} ADDC{.B} Wb, #lit5, Wd

[Wd]

[Wd++]

[Wd--]

[++Wd]

[--Wd]

Operands:

Wb ∈ [W0 ... W15]

lit5 ∈ [0 ... 31]

Wd ∈ [W0 ... W15]

Operation:

(Wb) + lit5 + (C) → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

0100 1www wBqq qddd d11k kkkk

Description:

Add the contents of the base register Wb, the 5-bit unsigned short literal operand and the Carry bit, and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Register Direct or Indirect Addressing may be used for 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘k’ bits provide the literal operand, a five-bit integer number.

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.

2: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1

Cycles: 1

Example 1: ADDC.B W0, #0x1F, [W7] ; Add W0, 31 and C bit (Byte mode)

; Store the result in [W7]

Before Instruction

<table>
<thead>
<tr>
<th>W0</th>
<th>CC80</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7</td>
<td>12C0</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W0</th>
<th>CC80</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7</td>
<td>12C0</td>
</tr>
</tbody>
</table>

Data 12C0 B000

SR 0000 (C = 0)

Data 12C0 B09F

SR 0008 (N = 1)
Example 2:       ADDC W3, #0x6, [--W4] ; Add W3, 6 and C bit (Word mode)
                ; Store the result in [--W4]

<table>
<thead>
<tr>
<th></th>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>6006</td>
<td>6006</td>
</tr>
<tr>
<td>W4</td>
<td>1000</td>
<td>0FFE</td>
</tr>
<tr>
<td>Data</td>
<td>0FFE</td>
<td>600D</td>
</tr>
<tr>
<td>Data</td>
<td>1000</td>
<td>DDEE</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
<td>0000</td>
</tr>
</tbody>
</table>
**ADDC**

Add Wb to Ws with Carry

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} ADDC{.B} Wb, Ws, Wd

[Wb], [Ws], [Wd]

[Ws++, Wd++]

[Ws--, Wd--]

[++Ws, ++Wd]

[--Ws, --Wd]

**Operands:**

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

**Operation:**

(Wb) + (Ws) + (C) → Wd

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
0100  lwww  wbqq  qddd  dppp  ssss
```

**Description:**

Add the contents of the source register Ws, the contents of the base register Wb and the Carry bit, and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

**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.

**Note 2:** The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

**Words:** 1

**Cycles:** 1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

```
ADDC.B W0,[W1++],[W2++] ; Add W0, [W1] and C bit (Byte mode)
; Store the result in [W2]
; Post-increment W1, W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>CC20</td>
</tr>
<tr>
<td>W1</td>
<td>0800</td>
</tr>
<tr>
<td>W2</td>
<td>1000</td>
</tr>
<tr>
<td>Data 0800</td>
<td>AB25</td>
</tr>
<tr>
<td>Data 1000</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

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

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>1000</td>
</tr>
<tr>
<td>W2</td>
<td>2000</td>
</tr>
<tr>
<td>W3</td>
<td>0180</td>
</tr>
<tr>
<td>Data 1000</td>
<td>8000</td>
</tr>
<tr>
<td>Data 2000</td>
<td>2500</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
AND

AND f and WREG

Implemented in: PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C
X | X | X | X | X | X | X

Syntax: `{label:} AND{.B} f {,WREG}

Operands: f ∈ [0 ... 8191]

Operation: \( (f).\text{AND.}(\text{WREG}) \rightarrow \text{destination designated by D} \)

Status Affected: N, Z

Encoding:

\[
\begin{array}{cccccccc}
1011 & 0110 & 0BDf & ffff & ffff & ffff \\
\end{array}
\]

Description:
Compute the logical AND operation of the contents of the default Working register WREG and 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 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:  
AND.B RAM100 ; AND WREG to RAM100 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG</td>
<td>CC80</td>
</tr>
<tr>
<td>RAM100</td>
<td>FFC0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>WREG</td>
<td>CC80</td>
</tr>
<tr>
<td>RAM100</td>
<td>FF80</td>
</tr>
<tr>
<td>SR</td>
<td>0008</td>
</tr>
</tbody>
</table>

Example 2:  
AND RAM200, WREG ; AND RAM200 to WREG (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG</td>
<td>CC80</td>
</tr>
<tr>
<td>RAM200</td>
<td>12C0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>WREG</td>
<td>0080</td>
</tr>
<tr>
<td>RAM200</td>
<td>12C0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

### AND

**AND Literal and Wn**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} **AND.B** #lit10, Wn

**Operands:**

lit10 ∈ [0 ... 255] for byte operation

Wn ∈ [W0 ... W15]

**Operation:**

lit10.AND.(Wn) → Wn

**Status Affected:**

N, Z

**Encoding:**

```
1011 0010 0Bkk kkkk kkkk dddd
```

**Description:**

Compute the logical AND operation of the 10-bit literal operand and the contents of the Working register Wn, and place the result back into the Working register Wn. Register Direct Addressing must be used for 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.

**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.

**Note 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.

**Words:** 1

**Cycles:** 1

**Example 1:**

AND.B #0x83, W7 ; AND 0x83 to W7 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 12C0</td>
<td>W7 1280</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

AND #0x333, W1 ; AND 0x333 to W1 (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 12D0</td>
<td>W1 0210</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
**AND**

**AND Wb and Short Literal**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} AND{.B} Wb, #lit5, Wd
```

- `{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 destination addressing mode.
- The `.d` bits select the destination register.
- The `.k` bits provide the literal operand, a five-bit integer number.

**Operands:**

- `Wb` ∈ [W0 ... W15]
- `lit5` ∈ [0 ... 31]
- `Wd` ∈ [W0 ... W15]

**Operation:**

`(Wb).AND.lit5 → Wd`

**Status Affected:**

N, Z

**Encoding:**

```
0110 0www wBqq qddd d11k kkkk
```

**Description:**

Compute the logical AND operation of the contents of the base register `Wb` and the 5-bit literal, and place the result in the destination register `Wd`. Register Direct Addressing must be used for `Wb`. Either Register Direct or Indirect Addressing may be used for `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 destination addressing mode.

The `.d` bits select the destination register.

The `.k` bits provide the literal operand, a five-bit integer number.

**Note:** 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

**Example 1:**

```
AND.B W0,#0x3,[W1++] ; AND W0 and 0x3 (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 23A5</td>
<td>W0 23A5</td>
</tr>
<tr>
<td>W1 2211</td>
<td>W1 2212</td>
</tr>
<tr>
<td>Data 2210</td>
<td>Data 2210</td>
</tr>
<tr>
<td>9999</td>
<td>0199</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
AND W0,#0x1F,W1 ; AND W0 and 0x1F (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 6723</td>
<td>W0 6723</td>
</tr>
<tr>
<td>W1 7878</td>
<td>W1 0003</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
AND

AND Wb and Ws

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} AND{.B} Wb, Ws, Wd

[Wb], [Wd]

[Ws], [Wd]

[Ws++], [Wd++]

[Ws-], [Wd-]

[++Ws], [+Wd]

[--Ws], [--Wd]

Operands:

Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:

(Wb).AND.(Ws) → Wd

Status Affected: N, Z

Encoding:

0110 0www wBqq qddd dppp ssss

Description:

Compute the logical AND operation of the contents of the source register Ws and 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 or 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

Note: 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 (1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

AND.B W0, W1 [W2++] ; AND W0 and W1, and
; store to [W2] (Byte mode)
; Post-increment W2

Before Instruction After Instruction

<table>
<thead>
<tr>
<th>W0</th>
<th>AA55</th>
<th>W0</th>
<th>AA55</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>2211</td>
<td>W1</td>
<td>2211</td>
</tr>
<tr>
<td>W2</td>
<td>1001</td>
<td>W2</td>
<td>1002</td>
</tr>
<tr>
<td>Data 1000</td>
<td>FFFF</td>
<td>Data 1000</td>
<td>11FF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:  
```
AND W0, [W1++], W2 ; AND W0 and [W1], and
; store to W2 (Word mode)
; Post-increment W1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 AA55</td>
<td>W0 AA55</td>
</tr>
<tr>
<td>W1 1000</td>
<td>W1 1002</td>
</tr>
<tr>
<td>W2 55AA</td>
<td>W2 2214</td>
</tr>
<tr>
<td>Data 1000 2634</td>
<td>Data 1000 2634</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

ASR
Arithmetic Shift Right f

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} ASR{.B} f {,WREG}
Operands: f ∈ [0 ... 8191]
Operation:

For Byte Operation:
- (f<7>) → Dest<7>
- (f<7>) → Dest<6>
- (f<6:1>) → Dest<5:0>
- (f<0>) → C

For Word Operation:
- (f<15>) → Dest<15>
- (f<15>) → Dest<14>
- (f<14:1>) → Dest<13:0>
- (f<0>) → C

Status Affected: N, Z, C
Encoding: 1101 0101 1BDf ffff ffff ffff
Description: Shift the contents of the file register one bit to the right and place the result in the destination register. The Least Significant bit of the file register is shifted into the Carry bit of the STATUS Register. After the shift is performed, the result is sign-extended. 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 register ('0' for WREG, '1' for file register).
The 'f' bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1
Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1: ASR.B RAM400, WREG ; ASR RAM400 and store to WREG ; (Byte mode)

<table>
<thead>
<tr>
<th></th>
<th>BeforeInstruction</th>
<th>AfterInstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG</td>
<td>0600</td>
<td>0611</td>
</tr>
<tr>
<td>RAM400</td>
<td>0823</td>
<td>0823</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
**Example 2:** ASR RAM200 ; ASR RAM200 (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM200</td>
<td>8009</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>RAM200</td>
<td>C004</td>
</tr>
<tr>
<td>SR</td>
<td>0009 (N, C = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

ASR Arithmetic Shift Right Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} ASR{.B} Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- [--Ws], [--Wd]

Operands:

- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

Operation:

For Byte Operation:

- (Ws<7>) → Wd<7>
- (Ws<7>) → Wd<6>
- (Ws<6:1>) → Wd<5:0>
- (Ws<0>) → C

For Word Operation:

- (Ws<15>) → Wd<15>
- (Ws<15>) → Wd<14>
- (Ws<14:1>) → Wd<13:0>
- (Ws<0>) → C

Status Affected: N, Z, C

Encoding:

```
  1101 0001 1Bqq qddd dppp ssss
```

Description:

Shift 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 shifted into the Carry bit of the STATUS Register. After the shift is performed, the result is sign-extended. Either Register Direct or Indirect Addressing 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 addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: 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 (1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Note 2: 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 (1)

Note 3: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

© 2005-2018 Microchip Technology Inc.
Example 1:   ASR.B [W0++], [W1++] ; ASR [W0] and store to [W1] (Byte mode)
             ; Post-increment W0 and W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0600</td>
</tr>
<tr>
<td>W1</td>
<td>0801</td>
</tr>
<tr>
<td>Data 600</td>
<td>2366</td>
</tr>
<tr>
<td>Data 800</td>
<td>FFC0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W0</td>
<td>0601</td>
</tr>
<tr>
<td>W1</td>
<td>0802</td>
</tr>
<tr>
<td>Data 600</td>
<td>2366</td>
</tr>
<tr>
<td>Data 800</td>
<td>33C0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:   ASR W12, W13 ; ASR W12 and store to W13 (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W12</td>
<td>AB01</td>
</tr>
<tr>
<td>W13</td>
<td>0322</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W12</td>
<td>AB01</td>
</tr>
<tr>
<td>W13</td>
<td>D580</td>
</tr>
</tbody>
</table>
| SR                | 0009              (N, C = 1)
Section 5. Instruction Descriptions

ASR

Arithmetic Shift Right by Short Literal

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} ASR Wb, #lit4, Wnd

Operands:
- Wb ∈ [W0 ... W15]
- lit4 ∈ [0 ... 15]
- Wnd ∈ [W0 ... W15]

Operation:
- lit4<3:0> → Shift_Val
- Wb<15> → Wnd<15:15-Shift_Val + 1>
- Wb<15:Shift_Val> → Wnd<15-Shift_Val:0>

Status Affected: N, Z

Encoding:

```
1101 1110 1www wddd d100 kkkk
```

Description:

Arithmetic shift right the contents of the source register Wb by the 4-bit unsigned literal and store the result in the destination register Wnd. After the shift is performed, the result is sign-extended. Direct Addressing must be used for Wb and Wnd.

The 'w' bits select the address of the base register.
The 'd' bits select the destination register.
The 'k' bits provide the literal operand.

Note: This instruction operates in Word mode only.

Words: 1
Cycles: 1

Example 1:

```
ASR W0, #0x4, W1 ; ASR W0 by 4 and store to W1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 060F</td>
<td>W0 060F</td>
</tr>
<tr>
<td>W1 1234</td>
<td>W1 0060</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

```
ASR W0, #0x6, W1 ; ASR W0 by 6 and store to W1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 80FF</td>
<td>W0 80FF</td>
</tr>
<tr>
<td>W1 0060</td>
<td>W1 FE03</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

Example 3:

```
ASR W0, #0xF, W1 ; ASR W0 by 15 and store to W1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 70FF</td>
<td>W0 70FF</td>
</tr>
<tr>
<td>W1 CC26</td>
<td>W1 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002 (Z = 1)</td>
</tr>
</tbody>
</table>
ASR

Arithmetic Shift Right by Wns

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} ASR Wb, Wns, Wnd

Operands:

Wb ∈ [W0 ... W15]
Wns ∈ [W0 ... W15]
Wnd ∈ [W0 ... W15]

Operation:

Wns<3:0> → Shift_Val
Wb<15> → Wnd<15:15-Shift_Val + 1>
Wb<15:Shift_Val> → Wnd<15-Shift_Val:0>

Status Affected: N, Z

Encoding:

| 1101 | 1110 | lwww | wddd | d000 | ssss |

Description:

Arithmetic shift right the contents of the source register Wb by the 4 Least Significant bits of Wns (up to 15 positions) and store the result in the destination register Wnd.
After the shift is performed, the result is sign-extended. Direct Addressing must be used for Wb, Wns and Wnd.
The ‘w’ bits select the address of the base register.
The ‘d’ bits select the destination register.
The ‘s’ bits select the source register.

Note 1: This instruction operates in Word mode only.
2: If Wns is greater than 15, Wnd = 0x0 if Wb is positive and Wnd = 0xFFFF if Wb is negative.

Words: 1
Cycles: 1

Example 1: ASR W0, W5, W6 ; ASR W0 by W5 and store to W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 80FF</td>
<td>W0 80FF</td>
</tr>
<tr>
<td>W5 0004</td>
<td>W5 0004</td>
</tr>
<tr>
<td>W6 2633</td>
<td>W6 F80F</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2: ASR W0, W5, W6 ; ASR W0 by W5 and store to W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 6688</td>
<td>W0 6688</td>
</tr>
<tr>
<td>W5 000A</td>
<td>W5 000A</td>
</tr>
<tr>
<td>W6 FF00</td>
<td>W6 0019</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 3: ASR W11, W12, W13 ; ASR W11 by W12 and store to W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W11 8765</td>
<td>W11 8765</td>
</tr>
<tr>
<td>W12 88E4</td>
<td>W12 88E4</td>
</tr>
<tr>
<td>W13 A5A5</td>
<td>W13 F876</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
### BCLR

**Bit Clear in f**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} BCLR{.B} f, #bit4
```

**Operands:**
- `f` ∈ [0 ... 8191] for byte operation
- `f` ∈ [0 ... 8190] (even only) for word operation
- `bit4` ∈ [0 ... 7] for byte operation
- `bit4` ∈ [0 ... 15] for byte operation

**Operation:**

\[ 0 \rightarrow f<bit4> \]

**Status Affected:**
None

**Encoding:**

```
1010 1001 bbbf ffff ffff fffb
```

**Description:**
Clear the bit in the file register `f` specified by `bit4`. 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 bit 4 of the bit position to be cleared.

The 'f' bits select the address of the file register.

**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.

**Note 2:** When this instruction operates in Word mode, the file register address must be word-aligned.

**Note 3:** When this instruction operates in Byte mode, `bit4` must be between 0 and 7.

**Words:**
1

**Cycles:**
1(1)

---

**Example 1:**

```
BCLR.B 0x800, #0x7 ; Clear bit 7 in 0x800
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0800</td>
<td>66EF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>Data 0800</td>
<td>66F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
BCLR 0x400, #0x9 ; Clear bit 9 in 0x400
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0400</td>
<td>AA55</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>Data 0400</td>
<td>A855</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in *Section 3.2.1 “Multicycle Instructions”*.  

---

© 2005-2018 Microchip Technology Inc. DS70000157G-page 127
BCLR Bit Clear in Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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 → Ws<bit4>

Status Affected: None

Encoding:

| 1010 | 0001 | bbbb | 0B00 | 0ppp | ssss |

Description:

Clear the bit in register Ws specified by 'bit4'. 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). 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 source/destination register.

The 'p' bits select the source addressing mode.

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.

2: When this instruction operates in Word mode, the source register address must be word-aligned.

3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.

4: In dsPIC33E, dsPIC33C and PIC24E devices, this instruction uses the DSRPAG register for Indirect Addressing generation in Extended Data Space (EDS).

Words: 1

Cycles: 1^{(1)}

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

---

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.

2: When this instruction operates in Word mode, the source register address must be word-aligned.

3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.

4: In dsPIC33E, dsPIC33C and PIC24E devices, this instruction uses the DSRPAG register for Indirect Addressing generation in Extended Data Space (EDS).
Section 5. Instruction Descriptions

**Example 1:**

```
BCLR.B W2, #0x2 ; Clear bit 3 in W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>F234</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W2</td>
<td>F230</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
BCLR [W0++], #0x0 ; Clear bit 0 in [W0] ; Post-increment W0
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>2300</td>
</tr>
<tr>
<td>Data 2300</td>
<td>5607</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W0</td>
<td>2302</td>
</tr>
<tr>
<td>Data 2300</td>
<td>5606</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### BFEXT

**Bit Field Extract from Ws into Wnd**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} BFEXT #bit4, #wid5, Ws, Wnd
[Ws],
[Ws++],
[Ws--],
[++Ws],
[--Ws],
```

**Operands:**

- `bit4` e [0 ... 15]; `wid5` e [1 ... 16];
- `Ws` e [W0 ... W15]; `Wnd` e [W0 ... W15]

**Operation:**

See text

**Status Affected:**

None

**Encoding:**

<table>
<thead>
<tr>
<th>1st word</th>
<th>2nd word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>wwww</td>
<td>0000</td>
</tr>
<tr>
<td>MMMM</td>
<td>0ppp</td>
</tr>
<tr>
<td>LLLL</td>
<td>ssss</td>
</tr>
</tbody>
</table>

**Description:**

A bit field is extracted (copied) from (Ws) and written into Wnd. The bit field data loaded into Wnd starts at Wnd<0>, and all MSbs within Wnd that are beyond the defined bit field width, will be cleared.

The bit location within Ws of the LSb of the bit field to be extracted is defined by operand `bit4`. The width of the bit field may be up to 16 bits and is defined by operand `wid5`.

The ‘w’ bits select the address of the bit field destination register.

The ‘s’ bits select the address of the data source register.

The ‘p’ bits select the source addressing mode.

The ‘MMMM’ bits define the bit field LSb position within the target word.

The ‘LLLL’ bits define the bit field MSb position within the target word.

**Words:** 2

**Cycles:** 2
### BFEXT

**Bit Field Extract from f into Wnd**

*Implemented in:* PIC24F, PIC24H, PIC24E, dsPIC30F, dsPIC33F, dsPIC33E, dsPIC33C

#### Syntax:

```
{label:} BFEXT #bit4, #wid5, f, Wnd
```

#### Operands:

- `bit4` ∈ [0 ... 15]; `wid5` ∈ [1 ... 16];
- `Wnd` ∈ [W0 ... W15]; `f` ∈ [0 ... 65534]

#### Operation:

See text

#### Status Affected:

None

#### Encoding:

<table>
<thead>
<tr>
<th>1st word</th>
<th>2nd word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1010</td>
</tr>
<tr>
<td>1010</td>
<td>wwwwww</td>
</tr>
<tr>
<td>MMMM</td>
<td>LLLL</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>ffff</td>
<td>ffff</td>
</tr>
<tr>
<td>ffff</td>
<td>ffff</td>
</tr>
</tbody>
</table>

#### Description:

A bit field is extracted (copied) from the file register address and written into Wnd. The bit field data loaded into Wnd starts at Wnd<0> and all MSbs within Wnd, that are beyond the defined bit field width, will be cleared.

The bit location within Ws of the LSb of the bit field to be extracted is defined by operand `bit4`. The width of the bit field may be up to 16 bits and is defined by operand `wid5`.

The ‘w’ bits select the address of the bit field destination register.

The ‘f’ bits select the (word) address of the source file register.

The ‘LLLL’ bits define the bit field LSb position within the target word.

The ‘MMMM’ bits define the bit field MSb position within the target word.

#### Words:

2

#### Cycles:

2
**BFINS**

**Bit Field Insert from Wb into Wd**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} BFINS #bit4 #wid5, Wns, Wd

- [Wd]
- [Wd++]
- [Wd--]
- [++Wd]
- [--Wd]

**Operands:**

- bit4 ∈ [0 ... 15]; wid5 ∈ [1 ... 16];
- Wns ∈ [W0 ... W15]; Wd ∈ [W0 ... W15]

**Operation:**

See text

**Status Affected:**

None

**Encoding:**

1st word | 0000 | 1010 | 0000 | www | MMMM | LLLL

2nd word | 0000 | 0000 | 0000 | 0000 | 0ppp | dddd

**Description:**

A bit field is read from (Wb) and inserted (copied) into Wd. The bit field data sourced from Wns starts at Wns<0>. All MSbs within Wns, that are beyond the defined bit field width, are ignored and may be set to any value.

The bit location within Wd of the LSb of the bit field to be inserted is defined by operand bit4. The width of the bit field may be up to 16 bits and is defined by operand wid5. The insert operation overwrites the existing bits within the insert range (i.e., it does not shift the existing bits to accommodate the inserted bits).

The ‘w’ bits select the address of the bit field source register.

The ‘d’ bits select the address of the data destination register.

The ‘p’ bits select Source Addressing Mode 1.

The ‘LLLL’ bits define the bit field LSb position within the target word.

The ‘MMMM’ bits define the bit field MSb position within the target word.

**Words:** 2

**Cycles:** 2
### BFINS

**Bit Field Insert from Wns into f**

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**
```
(label:) BFINS #bit4, #wid5, Wns, f
```

**Operands:**
- `bit4 ∈ [0 ... 15]; wid5 ∈ [1 ... 16];
- `Wns ∈ [W0 ... W15]; f ∈ [0 ... 65534]

**Operation:**
See text

**Status Affected:**
None

**Encoding:**

<table>
<thead>
<tr>
<th>1st word</th>
<th>0000</th>
<th>1010</th>
<th>0010</th>
<th>wwww</th>
<th>MMMM</th>
<th>LLLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>0000</td>
<td>0000</td>
<td>ffff</td>
<td>ffff</td>
<td>ffff</td>
<td>fff0</td>
</tr>
</tbody>
</table>

**Description:**
A bit field is read from (Wns) and inserted (copied) into the file register address. The bit field data sourced from Wns starts at Wns<0>. All MSbs within Wns, that are beyond the defined bit field width, are ignored and may be set to any value.

The bit location within f of the LSb of the bit field to be inserted is defined by operand bit4. The width of the bit field may be up to 16 bits and is defined by operand wid5. The insert operation overwrites the existing bits within the insert range (i.e., it does not shift the existing bits to accommodate the inserted bits).

The ‘f’ bits select the (word) address of the destination file register. The ‘LLLL’ bits define the bit field LSb position within the target word. The ‘MMMM’ bits define the bit field MSb position within the target word.

**Words:**
2

**Cycles:**
2
**BFINS**

**Bit Field Insert Literal into Ws**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} BFINS #bit4, #wid5, #lit8, Ws

-Ws

-Ws++

-Ws--

[Ws]

[Ws++]

[++Ws]

[--Ws]

**Operands:**

- bit4 $\in [0 \ldots 15]$
- wid5 $\in [1 \ldots 16]$
- lit8 $\in [0 \ldots 255]$
- Ws $\in [W0 \ldots W15]$

**Operation:**

See text

**Status Affected:**

None

**Encoding:**

<table>
<thead>
<tr>
<th>1st word</th>
<th>0000</th>
<th>1010</th>
<th>0100</th>
<th>0000</th>
<th>MMMM</th>
<th>LLLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>0000</td>
<td>0000</td>
<td>kkkk</td>
<td>kkkk</td>
<td>0ppp</td>
<td>ssss</td>
</tr>
</tbody>
</table>

**Description:**

A bit field literal value is inserted (copied) into Ws. The bit field data sourced from the literal starts at the LSb of the literal. All MSbs within the literal value, that are beyond the defined bit field width, are ignored and may be set to any value.

The bit location within Ws of the LSb of the bit field to be inserted is defined by operand bit4.

The width of the bit field may be up to 16 bits and is defined by operand wid5.

The ‘k’ bits contain the bit field source value.

The ‘s’ bits select the address of the source/destination register.

The ‘p’ bits select Source Addressing Mode 1.

The ‘LLLL’ bits define the bit field LSb position within the target word.

The ‘MMMM’ bits define the bit field MSb position within the target word.

**Words:**

2

**Cycles:**

2
## Section 5. Instruction Descriptions

### BOOTSWP(1)

**Swap Active and Inactive Flash Address Panel**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} BOOTSWP

**Operands:**

None

**Operation:**

- If (Dual Boot Operating mode and the BOOTSWP instruction are enabled (via device-specific Configuration bits))
  - Then (P2ACTIV (NVMCON<10>) → P2ACTIV
    - 1 → SFTSWP (NVMCON<11>))
  - Else
    - Execute as NOP

**Status Affected:**

None

**Encoding:**

<table>
<thead>
<tr>
<th>1111</th>
<th>1110</th>
<th>0010</th>
<th>0000</th>
<th>0000</th>
<th>0000</th>
</tr>
</thead>
</table>

**Description:**

If the BOOTSWP instruction is enabled (via device-specific Configuration bit) and the device is operating in a Dual Boot mode, and the NVMKEY software interlock sequence has been satisfied, the BOOTSWP instruction will:

1. Toggle the state of the P2ACTIV (NVMCON<10>) status bit, which will swap the Active and Inactive Flash address space within the program space address map.
2. Set SFTSWP (NVMCON<11>), indicating a successful panel swap.

**Words:**

1

**Cycles:**

2

**Note 1:** This instruction is present only in some devices of the device families listed above. Please see the specific device data sheet to ensure that this instruction is supported on a specific device.
BRA         Branch Unconditionally

Implemented in:       PIC24F  PIC24H  PIC24E  dsPIC30F  dsPIC33F  dsPIC33E  dsPIC33C
                      X       X       X       X       X       X       X

Syntax:              {label:} BRA Expr
Operands:            Expr may be a label, absolute address or expression.
                      Expr is resolved by the linker to a Slt16, where Slt16 ∈ [-32768 ... +32767].
Operation:           (PC + 2) + 2 * Slt16 → PC
                      NOP → Instruction Register
Status Affected:     None
Encoding:            0011 0111 nnnn nnnn nnnn nnnn
Description:         The program will branch unconditionally, relative to the next PC. The offset of the branch is the two's complement number, '2 * Slt16', which supports branches of up to 32K instructions, forward or backward. The Slt16 value is resolved by the linker from the supplied label, absolute address or expression. After the branch is taken, the new address will be (PC + 2) + 2 * Slt16, since the PC will have incremented to fetch the next instruction.
                      The 'n' bits are a signed literal that specifies the number of program words offset from (PC + 2).
                     
Words:               1
Cycles:              2 (PIC24F, PIC24H, dsPIC30F, dsPIC33F)
                      4 (PIC24E, dsPIC33E, dsPIC33C)

Example 1:

Before Instruction          After Instruction
PC  00 2000   PC  00 200A
SR  0000      SR  0000

Example 2:

Before Instruction          After Instruction
PC  00 2000   PC  00 200C
SR  0000      SR  0000

Example 3:

Before Instruction          After Instruction
PC  00 2000   PC  00 1366
SR  0000      SR  0000
Section 5. Instruction Descriptions

### BRA

**Computed Branch**

- **Implemented in:**
  - PIC24F
  - PIC24H
  - PIC24E
  - dsPIC30F
  - dsPIC33F
  - dsPIC33E
  - dsPIC33C

- **Syntax:**
  

- **Operands:**
  \( W_n \in [W_0 \ldots W_{15}] \)

- **Operation:**
  \((PC + 2) + (2 \times W_n) \rightarrow PC\)

- **Status Affected:** None

- **Encoding:**
  

- **Description:**

  The program branches unconditionally, relative to the next PC. The offset of the branch is the sign-extended 17-bit value \((2 \times W_n)\), which supports branches up to 32K instructions, forward or backward. After this instruction executes, the new PC will be \((PC + 2) + 2 \times W_n\), since the PC will have incremented to fetch the next instruction.

  The 's' bits select the source register.

- **Words:** 1

- **Cycles:** 2

#### Example 1:

```
002000 HERE:  BRA W7     ; Branch forward (2 + 2 * W7)
002002 . . .
...  . . .
...  . . .
002108 . . .
00210A TABLE7: . .
00210C . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 210A</td>
</tr>
<tr>
<td>W7 0084</td>
<td>W7 0084</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### BRA

**Computed Branch**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{(label:)} BRA Wn

**Operands:**

Wn ∈ [W0 ... W15]

**Operation:**

(PC + 2) + (2 * Wn) → PC

NOP → Instruction Register

**Status Affected:**

None

**Encoding:**

```
0000 001 0000 0110 0000 ssss
```

**Description:**

The program branches unconditionally, relative to the next PC. The offset of the branch is the sign-extended 17-bit value (2 * Wn), which supports branches up to 32K instructions, forward or backward. After this instruction executes, the new PC will be (PC + 2) + 2 * Wn, since the PC will have incremented to fetch the next instruction.

The 's' bits select the source register.

**Words:**

1

**Cycles:**

4

#### Example 1:

```
002000 HERE:  BRA W7 ; Branch forward (2 + 2 * W7)
002002       . .
       . .
       . .
002108       . .
00210A TABLE7: . .
00210C       . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 2000</td>
<td>PC   00 210A</td>
</tr>
<tr>
<td>W7  0084</td>
<td>W7   0084</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR   0000</td>
</tr>
</tbody>
</table>
BRA C Branch if Carry

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: (label:) BRA C, Expr

Operands: Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation: Condition = C
If (Condition)
(PC + 2) + 2 * Slit16 → PC
NOP → Instruction Register

Status Affected: None

Encoding:
0011 0001 nnnn nnnn nnnn nnnn

Description: If the Carry flag bit is '1', then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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.

Words: 1

Cycles: 1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:
002000 HERE:     BRA C, CARRY ; If C is set, branch to CARRY
002002 NO_C: . . . ; Otherwise... continue
002004 . . .
002006 GOTO THERE
002008 CARRY: . . .
00200A . . .
00200C THERE: . . .
00200E . . .

Before Instruction
PC 00 2000
SR 0001 (C = 1)

After Instruction
PC 00 2008
SR 0001 (C = 1)

Example 2:
002000 HERE:     BRA C, CARRY ; If C is set, branch to CARRY
002002 NO_C: . . . ; Otherwise... continue
002004 . . .
002006 GOTO THERE
002008 CARRY: . . .
00200A . . .
00200C THERE: . . .
00200E . . .

Before Instruction
PC 00 2000
SR 0000

After Instruction
PC 00 2002
SR 0000
Example 3:

006230 HERE:    BRA C, CARRY ; If C is set, branch to CARRY
006232 NO_C:    . . ; Otherwise... continue
006234         . .
006236         GOTO THERE
006238 CARRY:   . .
00623A         . .
00623C THERE:   . .
00623E         . .

Before Instruction
PC  00 6230
SR  0001 (C = 1)

After Instruction
PC  00 6238
SR  0001 (C = 1)

Example 4:

006230 START:  . .
006232         . .
006234 CARRY:  . .
006236         . .
006238         . .
00623A         . .
00623C HERE:   BRA C, CARRY ; If C is set, branch to CARRY
00623E         . . ; Otherwise... continue

Before Instruction
PC  00 623C
SR  0001 (C = 1)

After Instruction
PC  00 6234
SR  0001 (C = 1)
BRA GE

Branch if Signed Greater Than or Equal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} BRA GE, Expr

Operands: Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation: Condition = (N&&OV)||(!N&&!OV)
If (Condition)
  (PC + 2) + 2 * Slit16 → PC
  NOP → Instruction Register

Status Affected: 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 two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

If the branch is taken, the new address will be (PC + 2) + 2 * Slit16, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second cycle.

The 'n' bits are a 16-bit signed literal that specify the offset from (PC + 2) in instruction words.

Note: The assembler will convert the specified label into the offset to be used.

Words: 1

Cycles: 1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

007600 LOOP: . . .
007602 . . .
007604 . . .
007606 . . .
007608 HERE: BRA GE, LOOP ; If GE, branch to LOOP
00760A NO_GE: . . . ; Otherwise... continue

Before Instruction

| PC | 00 7608 |
| SR | 0000 |

After Instruction

| PC | 00 7600 |
| 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 Instruction

| PC | 00 7608 | (N = 1) |
| SR | 0008 |

After Instruction

| PC | 00 760A |
| SR | 0008 | (N = 1) |
BRA GEU

Branch if Unsigned Greater Than or Equal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA GEU</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} BRA GEU, Expr

Operands: Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16 offset that supports an offset range of [-32768 ... +32767] program words.

Operation: Condition = C
If (Condition)

(PC + 2) + 2 * Slit16 → PC

NOP → Instruction Register

Status Affected: None

Encoding:

0011 0001 nnnn nnnn nnnn nnnn

Description: If the Carry flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

If the branch is taken, the new address will be (PC + 2) + 2 * Slit16, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a NOP executed in the second cycle.

The 'n' bits are a 16-bit signed literal that specify the offset from (PC + 2) in instruction words.

Note: This instruction is identical to the BRA C, Expr (Branch if Carry) instruction and has the same encoding. It will reverse assemble as BRA C, Slit16.

Words: 1

Cycles: 1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE: BRA GEU, BYPASS  ; If C is set, branch
002002 NO_GEU: ...  ; to BYPASS
002004 ...  ; Otherwise... continue
002006 ...
002008 ...
00200A GOTO THERE
00200C BYPASS: ...
00200E ...

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 200C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

BRA GT

Branch if Signed Greater Than

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BRA GT, Expr

Operands:

Expr may be a label, absolute address or expression.

Expr is resolved by the linker to a Slt16, where

\[ \text{Slt16} \in [-32768 \ldots +32767] \]

Operation:

Condition = (!Z&&N&&OV)||(!Z&&!N&&!OV)

If (Condition)

\[(PC + 2) + 2 \times \text{Slt16} \rightarrow PC\]

NOP \rightarrow Instruction Register

Status Affected:

None

Encoding:

| 0011 | 1100 | nnnn | nnnn | nnnn | nnnn |

Description:

If the logical expression, (IZ&&N&&OV)||(!IZ&&N&&OV), is true, then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, ‘2 * Slt16’, which supports branches up to 32K instructions, forward or backward. The Slt16 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 * Slt16, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a \text{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.

Words:

1

Cycles:

1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

| 002000 HERE: | BRA GT, BYPASS | ; If GT, branch to BYPASS |
| 002002 NO_GT: | \ldots | ; Otherwise... continue |
| 002004 \ldots | \ldots |
| 002006 \ldots | \ldots |
| 002008 \ldots | \ldots |
| 00200A GOTO THERE |
| 00200C BYPASS: | \ldots |
| 00200E \ldots |

Before Instruction

| PC | 00 2000 |
| SR | 0001 (C = 1) |

After Instruction

| PC | 00 200C |
| SR | 0001 (C = 1) |
## BRA GTU

### Branch if Unsigned Greater Than

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
\text{(label:)} \text{BRA GTU, Expr}
\]

**Operands:**

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slt16, where Slt16 \(\in [-32768 \ldots +32767]\).

**Operation:**

Condition = (C\&!Z)

If (Condition)

\[
(\text{PC} + 2) + 2 \times \text{Slt16} \rightarrow \text{PC}
\]

\[
\text{NOP} \rightarrow \text{Instruction Register}
\]

**Status Affected:**

None

**Encoding:**

| 0011 | 1110 | 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 two's complement number, \(2 \times \text{Slt16}\), which supports branches up to 32K instructions, forward or backward. The Slt16 value is resolved by the linker from the supplied label, absolute address or expression.

If the branch is taken, the new address will be \((\text{PC} + 2) + 2 \times \text{Slt16}\), since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a \text{NOP} executed in the second cycle.

The ‘n’ bits are a signed literal that specifies the number of instructions offset from \((\text{PC} + 2)\).

**Words:**

1

**Cycles:**

1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

**Example 1:**

```
002000 HERE:     BRA GTU, BYPASS ; If GTU, branch to BYPASS
002002 NO_GTU: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .
```

### Example 1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 200C</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

**BRA LE**

**Branch if Signed Less Than or Equal**

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} BRA LE, Expr

Operands: Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation: Condition = Z||(N&&!OV)||(!N&&OV)
If (Condition)

(PC + 2) + 2 * Slit16 → PC

NOP → Instruction Register

Status Affected: None

Encoding: 0011 0100 nnnn nnnn nnnn nnnn

Description: If the logical expression, Z||(N&&!OV)||(!N&&OV), is true, then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

**Example 1:**

```
002000 HERE: BRA LE, BYPASS ; If LE, branch to BYPASS
002002 NO_LE: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2002</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>

© 2005-2018 Microchip Technology Inc.
BRA LEU

Branch if Unsigned Less Than or Equal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BRA LEU, Expr

Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a S16, where S16 ∈ [-32768 ... +32767].

Operation:

Condition = !C||Z

If (Condition)

(PC + 2) + 2 * S16 → PC

NOP → Instruction Register

Status Affected:

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 two’s complement number, ‘2 * S16’, which supports branches up to 32K instructions, forward or backward. The S16 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 * S16, 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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE:     BRA LEU, BYPASS ; If LEU, branch to BYPASS
002002 NO_LEU:   . . . ; Otherwise... continue
                  002004
                  002006
                  002008
                  00200A     GOTO THERE
00200C BYPASS:   . . .
00200E

Before Instruction

| PC | 00 2000 |
| SR | 0001 (C = 1) |

After Instruction

| PC | 00 200C |
| SR | 0001 (C = 1) |
BRA LT

Branch if Signed Less Than

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BRA LT, Expr

Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation:

Condition = (N&&!OV)||(N&OV)

If (Condition)

(PC + 2) + 2 * Slit16 → PC

NOP → Instruction Register

Status Affected:

None

Encoding:

| 0011 | 0101 | 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 two’s complement number, ‘2 * Slit16’, which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE: BRA LT, BYPASS ; If LT, branch to BYPASS
002002 NO_LT: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Before Instruction

| PC | 00 2000 |
| SR | 0001 (C = 1) |

After Instruction

| PC | 00 2002 |
| SR | 0001 (C = 1) |
BRA LTU
Branch if Unsigned Less Than

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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 $\in [-32768 \ldots +32767]$.

Operation:  
Condition = !C  
If (Condition)  
(\text{PC} + 2) + 2 \times \text{Slit16} \rightarrow \text{PC}  
\text{NOP} \rightarrow \text{Instruction Register}

Status Affected:  
None

Encoding:  
0011 1001 nnnn nnnn nnnn nnnn

Description:  
If the Carry flag is '0', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 \times \text{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 (\text{PC} + 2) + 2 \times \text{Slit16}, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a \text{NOP} executed in the second cycle.

The 'n' bits are a signed literal that specifies the number of instructions offset from (\text{PC} + 2).

\textbf{Note}:  
This instruction is identical to the BRA NC, Expr (Branch if Not Carry) instruction and has the same encoding. It will reverse assemble as BRA NC, Slit16.

Words:  
1

Cycles:  
1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F  
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:  
002000 HERE: \quad \text{BRA LTU, BYPASS} \quad ; \text{If LTU, branch to BYPASS}  
002002 NO_LTU: \quad \ldots \quad ; \text{Otherwise... continue}  
002004 \quad \ldots  
002006 \quad \ldots  
002008 \quad \ldots  
00200A \quad \text{GOTO THERE}  
00200C BYPASS: \quad \ldots  
00200E \quad \ldots

Before Instruction  
\begin{align*}  
\text{PC} & : & 002000  
\text{SR} & : & 0001 \quad (C = 1)  
\end{align*}

After Instruction  
\begin{align*}  
\text{PC} & : & 002002  
\text{SR} & : & 0001 \quad (C = 1)  
\end{align*}
Section 5. Instruction Descriptions

BRA N
Branch if Negative

Implemented in: |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC24F</td>
<td>PIC24H</td>
<td>PIC24E</td>
<td>dsPIC30F</td>
<td>dsPIC33F</td>
<td>dsPIC33E</td>
<td>dsPIC33C</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Syntax:
{label:} BRA N, Expr

Operands:
Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation:
Condition = N
If (Condition)
(PC + 2) + 2 * Slit16 → PC
NOP → Instruction Register.

Status Affected:
None

Encoding:

| 0011 | 0011 | nnnn | nnnn | nnnn | nnnn |

Description:
If the Negative flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, ‘2 * Slit16’, which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:
002000 HERE: BRA N, BYPASS ; If N, branch to BYPASS
002002 NO_N: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Before Instruction
| PC | 00 2000 |
| SR | 0008 (N = 1) |

After Instruction
| PC | 00 200C |
| SR | 0008 (N = 1) |
### BRA NC

**Branch if Not Carry**

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```plaintext
{label:} BRA NC, Expr
```

#### Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

#### Operation:

Condition = !C

- If (Condition)
  - (PC + 2) + 2 * Slit16 → PC
  - NOP → Instruction Register

#### Status Affected:

None

#### Encoding:

```
0011 1001 nnnn nnnn nnnn nnnn
```

#### Description:

If the Carry flag is '0', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

- 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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
- 1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

#### Example 1:

```assembly
002000 HERE:     BRA NC, BYPASS ; If NC, branch to BYPASS
002002 NO_NC: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2002</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
### BRA NN

**Branch if Not Negative**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label;} BRA NN, Expr

**Operands:**

Expr may be a label, absolute address or expression.

Expr is resolved by the linker to a Slt16, where Slt16 ∈ [-32768 ... +32767].

**Operation:**

Condition = !N

If (Condition)

(PC + 2) + 2 * Slt16 → PC

NOP → Instruction Register

**Status Affected:**

None

**Encoding:**

|       | 0011 | 1011 | nnnn | nnnn | nnnn | nnnn |

**Description:**

If the Negative flag is '0', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slt16', which supports branches up to 32K instructions, forward or backward. The Slt16 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 * Slt16, 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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

**Example 1:**

```
002000 HERE:     BRA NN, BYPASS     ; If NN, branch to BYPASS
002002 NO_NN: . . .                       ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 200C</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
BRA NOV

Branch if Not Overflow

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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 ∈ [-32768 ... +32767].

Operation:  
Condition = !OV
If (Condition)

(PC + 2) + 2 * Slit16 → PC
NOP → Instruction Register

Status Affected:  
None

Encoding:  
0011 1000 nnnn nnnn nnnn nnnn

Description:
If the Overflow flag is ‘0’, then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, ‘2 * Slit16’, which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE: BRA NOV, BYPASS ; If NOV, branch to BYPASS
002002 NO_NOV: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Before Instruction

| PC | 00 2000 |
| SR | 0008 (N = 1) |

After Instruction

| PC | 00 200C |
| SR | 0008 (N = 1) |
BRA NZ

Branch if Not Zero

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:}  BRA  NZ, Expr

Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation:

Condition = !Z
If (Condition)

(PC + 2) + 2 * Slit16 → PC

NOP → Instruction Register

Status Affected:

None

Encoding:

0011  1010  nnnn  nnnn  nnnn  nnnn

Description:

If the Z flag is '0', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F
1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE:     BRA NZ, BYPASS ; If NZ, branch to BYPASS
002002 NO_NZ: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Before Instruction

| PC   | 00 2000 |
| SR   | 0002 (Z = 1) |

After Instruction

| PC   | 00 2002 |
| SR   | 0002 (Z = 1) |
BRA OA

Branch if Overflow Accumulator A

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\[
\text{\{label\}} \quad \text{BRA} \quad \text{OA}, \quad \text{Expr}
\]

Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slt16, where Slt16 \( \in [-32768 ... +32767] \).

Operation:

\[
\text{Condition} = \text{OA}
\]

If (Condition)

\[
(\text{PC} + 2) + 2 \times \text{Slt16} \rightarrow \text{PC}
\]

\[
\text{NOP} \rightarrow \text{Instruction Register}
\]

Status Affected: 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 two's complement number, '2 \times \text{Slt16}', which supports branches up to 32K instructions, forward or backward. The Slt16 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 \times \text{Slt16}, since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a \text{NOP} executed in the second cycle.

The 'n' bits are a signed literal that specifies the number of instructions offset from (PC + 2).

Note: The assembler will convert the specified label into the offset to be used.

Words: 1

Cycles: 1 (2 if branch taken) – dsPIC30F, dsPIC33F

1 (4 if branch taken) – dsPIC33E, dsPIC33C

Example 1:

```
002000 HERE:    BRA OA, BYPASS    ; If OA, branch to BYPASS
002002 NO_OA:   . . .                ; Otherwise... continue
002004          . . .
002006          . . .
002008          . . .
00200A          GOTO THERE
00200C BYPASS:  . . .
00200E          . . .
```

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>8800 (OA, OAB = 1)</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 200C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>8800 (OA, OAB = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

BRA OB

Branch if Overflow Accumulator B

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{labeled;} BRA OB, Expr

Operands:

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation:

Condition = OB
If (Condition)

(PC + 2) + 2 * Slit16 → PC
NOP → Instruction Register

Status Affected:

None

Encoding:

0000 1101 nnnn nnnn nnnn nnnn

Description:

If the Overflow Accumulator B flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – dsPIC30F, dsPIC33F
1 (4 if branch taken) – dsPIC33E, dsPIC33C

Example 1:

002000 HERE:    BRA OB, BYPASS    ; If OB, branch to BYPASS
002002 NO_OB:   . . .             ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS:  . . .
00200E . . .

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>8800</td>
</tr>
<tr>
<td>OA</td>
<td>1</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>8800</td>
</tr>
<tr>
<td>OA</td>
<td>1</td>
</tr>
</tbody>
</table>
**BRA OV**

**Branch if Overflow**

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

\{label:\} BRA OV, Expr

Operands:

Expr may be a label, absolute address or expression.

Expr is resolved by the linker to a Slit16, where Slit16 \(\in [-32768 \ldots +32767]\).

Operation:

Condition = OV

If (Condition)

\((PC + 2) + 2 \times \text{Slit16} \rightarrow PC\)

\[\text{NOP} \rightarrow \text{Instruction Register}\]

Status Affected:

None

Encoding:

| 0011 | 0000 | nnnn | nnnn | nnnn | nnnn |

Description:

If the Overflow flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

If the branch is taken, the new address will be \((PC + 2) + 2 \times \text{Slit16}\), since the PC will have incremented to fetch the next instruction. The instruction then becomes a two-cycle instruction, with a \[\text{NOP}\] 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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

**Example 1:**

002000 HERE: BRA OV, BYPASS ; If OV, branch to BYPASS
002002 NO_OV . . ; Otherwise... continue
002004 . .
002006 . .
002008 . .
00200A GOTO THERE
00200C BYPASS: . .
00200E . .

Before Instruction

| PC | 00 2000 |
| SR | 0002 (Z = 1) |

After Instruction

| PC | 00 2002 |
| SR | 0002 (Z = 1) |
BRA SA  

Branch if Saturation Accumulator A

Implemented in: 

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: 

{label:} BRA SA, Expr

Operands: 

Expr may be a label, absolute address or expression. Expr is resolved by the linker to a Slt16, where Slt16 \( \in \{-32768 ... +32767\} \).

Operation: 

Condition = SA

If (Condition)

\( (PC + 2) + 2 \times \text{Slt16} \rightarrow \text{PC} \)

NOP \rightarrow \text{Instruction Register}

Status Affected: 

None

Encoding: 

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>1110</th>
<th>nnnn</th>
<th>nnnn</th>
<th>nnnn</th>
<th>nnnn</th>
</tr>
</thead>
</table>

Description: 

If the Saturation Accumulator A flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two’s complement number, \(2 \times \text{Slt16}\), which supports branches up to 32K instructions, forward or backward. The Slt16 value is resolved by the linker from the supplied label, absolute address or expression.

If the branch is taken, the new address will be \((PC + 2) + 2 \times \text{Slt16}\), 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) – dsPIC30F, dsPIC33F

1 (4 if branch taken) – dsPIC33E, dsPIC33C

Example 1: 

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002000</td>
<td>HERE: BRA SA, BYPASS</td>
<td>; If SA, branch to BYPASS</td>
</tr>
<tr>
<td>002002</td>
<td>NO_SA: ...</td>
<td>; Otherwise... continue</td>
</tr>
<tr>
<td>002004</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>002006</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>002008</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00200A</td>
<td>GOTO THERE</td>
<td></td>
</tr>
<tr>
<td>00200C</td>
<td>BYPASS</td>
<td></td>
</tr>
<tr>
<td>00200E</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>2400 (SA, SAB = 1)</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 200C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>2400 (SA, SAB = 1)</td>
</tr>
</tbody>
</table>
BRA SB

Branch if Saturation Accumulator B

Implemented in:

<table>
<thead>
<tr>
<th>Implementor</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label;} BRA SB, Expr

Operands:  

Expr may be a label, absolute address or expression.

Expr is resolved by the linker to a Slt16, where 

Slt16 ∈ [-32768 ... +32767].

Operation:  

Condition = SB  

if (Condition)  

(PC + 2) + 2 * Slt16 → PC  

NOP → Instruction Register

Status Affected:  

None

Encoding:  

| 0000 | 1111 | nnnn | 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 two’s complement number, ‘2 * Slt16’, which supports branches up to 32K instructions, forward or backward. The Slt16 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 * Slt16, 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) – dsPIC30F, dsPIC33F

1 (4 if branch taken) – dsPIC33E, dsPIC33C

Example 1:  

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>002000</td>
<td>BRA SB, BYPASS</td>
<td>; If SB, branch to BYPASS</td>
</tr>
<tr>
<td>002002</td>
<td>NO_SB</td>
<td>; Otherwise... continue</td>
</tr>
<tr>
<td>002004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00200A</td>
<td>GOTO THERE</td>
<td></td>
</tr>
<tr>
<td>00200C</td>
<td>BYPASS</td>
<td></td>
</tr>
<tr>
<td>00200E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before Instruction

| PC | 00 2000 |
| SR | 0000    |

After Instruction

| PC | 00 2002 |
| SR | 0000    |
BRA Z

Branch if Zero

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

X X X X X X X

Syntax: {label:} BRA Z, Expr

Operands: Expr may be a label, absolute address or expression.

Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation: Condition = Z
if (Condition)
    (PC + 2) + 2 * Slit16 → PC

NOP → Instruction Register

Status Affected: None

Encoding:

| 0011 | 0010 | nnnn | nnnn | nnnn | nnnn |

Description: If the Zero flag is '1', then the program will branch relative to the next PC. The offset of the branch is the two's complement number, '2 * Slit16', which supports branches up to 32K instructions, forward or backward. The Slit16 value is resolved by the linker from the supplied label, absolute address or expression.

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) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE:     BRA Z, BYPASS ; If Z, branch to BYPASS
002002 NO_Z: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Words: 1

Cycles: 1 (2 if branch taken) – PIC24F, PIC24H, dsPIC30F, dsPIC33F

1 (4 if branch taken) – PIC24E, dsPIC33E, dsPIC33C

Example 1:

002000 HERE:     BRA Z, BYPASS ; If Z, branch to BYPASS
002002 NO_Z: . . . ; Otherwise... continue
002004 . . .
002006 . . .
002008 . . .
00200A GOTO THERE
00200C BYPASS: . . .
00200E . . .

Before Instruction

| PC 00 2000 |
| SR 0002 (Z = 1) |

After Instruction

| PC 00 200C |
| SR 0002 (Z = 1) |
BSET

Bit Set in f

Implemented in:
<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:
{label:} BSET.{B} f, #bit4

Operands:
- f ∈ [0 ... 8191] for byte operation
- f ∈ [0 ... 8190] (even only) for word operation
- bit4 ∈ [0 ... 7] for byte operation
- bit4 ∈ [0 ... 15] for word operation

Operation:
1 → f<bit4>

Status Affected: None

Encoding:
```
1010 1000 bbbf ffff ffff fffb
```

Description:
Set the bit in the file register ‘f’ specified by ‘bit4’. 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 of the bit position to be set.
The ‘f’ bits select the address of the file register.

**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.

2: When this instruction operates in Word mode, the file register address must be word-aligned.

3: When this instruction operates in Byte mode, ‘bit4’ must be between 0 and 7.

Words: 1

Cycles: 1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:
```
BSET.B 0x601, #0x3 ; Set bit 3 in 0x601
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0600 F234</td>
<td>Data 0600 FA34</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:
```
BSET 0x444, #0xF ; Set bit 15 in 0x444
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0444 5604</td>
<td>Data 0444 D604</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

BSET

Bit Set in Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BSET{.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:

1 → Ws<bit4>

Status Affected: None

Encoding:

| 1010 | 0000 | bbbb | 0B00 | 0ppp | ssss |

Description:

Set the bit in register Ws specified by ‘bit4’. 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). 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 ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source/destination register.

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.

2: When this instruction operates in Word mode, the source register address must be word-aligned.

3: When this instruction operates in Byte mode, ‘bit4’ must be between 0 and 7.

4: In dsPIC33E, dsPIC33C and PIC24E devices, this instruction uses the DSRPAG register for indirect address generation in Extended Data Space.

Words: 1

Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

© 2005-2018 Microchip Technology Inc. DS70000157G-page 161
Example 1:  
\[ \text{BSET.B W3, #0x7} \quad ; \text{Set bit 7 in W3} \]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3 0026</td>
<td>W3 00A6</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:  
\[ \text{BSET [W4++], #0x0} \quad ; \text{Set bit 0 in [W4]} \quad ; \text{Post-increment W4} \]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 6700</td>
<td>W4 6702</td>
</tr>
<tr>
<td>Data 6700 1734</td>
<td>Data 6700 1735</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
BSW

Bit Write in Ws

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BSW.C Ws, Wb

BSW.Z [Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands:

Ws ∈ [W0 ... W15]

Wb ∈ [W0 ... W15]

Operation:

For "C" Operation:

C → Ws<(Wb)>

For "Z" Operation (default):

Z → Ws<(Wb)>

Status Affected:

None

Encoding:

| 1010 | 1101 | Zwww | w000 | 0ppp | ssss |

Description:

The (Wb) bit in register Ws is written with the value of the C or Z flag from the STATUS Register. Bit numbering begins with the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 15) of the Working register. Only the four Least Significant bits of Wb are used to determine the destination bit number. Register Direct Addressing must be used for Wb, and either Register Direct or Indirect Addressing may be used for Ws.

The 'Z' bit selects the C or Z flag as source.

The 'w' bits select the address of the bit select register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

Note: This instruction only operates in Word mode. If no extension is provided, the "Z" operation is assumed.

Words: 1

Cycles: 1⁰

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

BSW.C W2, W3 ; Set bit W3 in W2 to the value of the C bit

Before Instruction | After Instruction

W2 | F234 | W2 | 7234 |

W3 | 111F | W3 | 111F |

SR | 0002 | SR | 0002 |

(Z = 1, C = 0) | (Z = 1, C = 0)
### Example 2: BSW.Z W2, W3

; Set bit W3 in W2 to the complement of the Z bit

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 E235</td>
<td>W2 E234</td>
</tr>
<tr>
<td>W3 0550</td>
<td>W3 0550</td>
</tr>
<tr>
<td>SR 0002 (Z = 1, C = 0)</td>
<td>SR 0002 (Z = 1, C = 0)</td>
</tr>
</tbody>
</table>

### Example 3: BSW.C [++W0], W6

; Set bit W6 in [W0++] to the value of the C bit

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 1000</td>
<td>W0 1002</td>
</tr>
<tr>
<td>W6 34A3</td>
<td>W6 34A3</td>
</tr>
<tr>
<td>Data 1002 2380</td>
<td>Data 1002 2388</td>
</tr>
<tr>
<td>SR 0001 (Z = 0, C = 1)</td>
<td>SR 0001 (Z = 0, C = 1)</td>
</tr>
</tbody>
</table>

### Example 4: BSW.Z [W1--], W5

; Set bit W5 in [W1] to the complement of the Z bit
; Post-decrement W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 1000</td>
<td>W1 0FFE</td>
</tr>
<tr>
<td>W5 888B</td>
<td>W5 888B</td>
</tr>
<tr>
<td>Data 1000 C4DD</td>
<td>Data 1000 CCDD</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

BTG

Bit Toggle in f

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: \{label:} BTG{.B} f, #bit4

Operands:

- f ∈ [0 ... 8191] for byte operation
- f ∈ [0 ... 8190] (even only) for word operation
- bit4 ∈ [0 ... 7] for byte operation
- bit4 ∈ [0 ... 15] for word operation

Operation:

(f)<bit4> → (f)<bit4>

Status Affected: None

Encoding:

| 1010 | 1010 | bbbf | ffff | ffff | fffb |

Description:

Bit 'bit4' in file register 'f' is toggled (complemented). 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 operation, bit 15 for word operation) of the byte.

The 'b' bits select value bit4, the bit position to toggle.

The 'f' bits select the address of the file register.

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.

2: When this instruction operates in Word mode, the file register address must be word-aligned.

3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.

Words: 1

Cycles: 1

Example 1: BTG.B 0x1001, #0x4 ; Toggle bit 4 in 0x1001

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1000</td>
<td>Data 1000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
<tr>
<td>F234</td>
<td>E234</td>
</tr>
</tbody>
</table>

Example 2: BTG 0x1660, #0x8 ; Toggle bit 8 in RAM660

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1660</td>
<td>Data 1660</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
<tr>
<td>5606</td>
<td>5706</td>
</tr>
</tbody>
</table>

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

© 2005-2018 Microchip Technology Inc.

DS70000157G-page 165
### BTG

Bit Toggle in Ws

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
\{\text{label:}\} \ BTG\{.B\} \ Ws, \ #\text{bit4} \\
\text{[Ws]}, \\
\text{[Ws++]}, \\
\text{[Ws--]}, \\
\text{[++Ws]}, \\
\text{[--Ws]} \\
\]

**Operands:**

\[Ws \in [W0 \ldots W15]\]
\[\text{bit4} \in [0 \ldots 7] \text{ for byte operation}\]
\[\text{bit4} \in [0 \ldots 15] \text{ for word operation}\]

**Operation:**

\[(Ws)<\text{bit4}> \rightarrow Ws<\text{bit4}>\]

**Status Affected:**

None

**Encoding:**

| 1010 | 0010 | bbbb | 0B00 | 0ppp | ssss |

**Description:**

Bit 'bit4' in register Ws is toggled (complemented). 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). Register Direct or Indirect Addressing may be used for Ws.

The 'b' bits select value bit4, the bit position to test.

The 'B' bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘s’ bits select the source/destination register.

The ‘p’ bits select the source addressing mode.

**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.

**Note 2:** When this instruction operates in Word mode, the source register address must be word-aligned.

**Note 3:** When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.

**Note 4:** In dsPIC33E, dsPIC33C and PIC24E devices, this instruction uses the DSRPAG register for indirect address generation in Extended Data Space.

**Words:**

1

**Cycles:**

1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

**BTG W2, #0x0**  ; Toggle bit 0 in W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2: F234</td>
<td>W2: F235</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>

### Example 2:

**BTG [W0++], #0x0**  ; Toggle bit 0 in [W0]
; Post-increment W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0: 2300</td>
<td>W0: 2302</td>
</tr>
<tr>
<td>Data 2300: 5606</td>
<td>Data 2300: 5607</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>
BTSC

Bit Test f, Skip if Clear

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
--- | --- | --- | --- | --- | --- | --- | --- |
| X | X | X | X | X | X | X |

Syntax: {label:} BTSC{.B} f, #bit4

Operands:

- f ∈ [0 ... 8191] for byte operation
- f ∈ [0 ... 8190] (even only) for word operation
- bit4 ∈ [0 ... 7] for byte operation
- bit4 ∈ [0 ... 15] for word operation

Operation: Test (f)<bit4>, skip if clear

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.

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.

2: When this instruction operates in Word mode, the file register address must be word-aligned.

3: When this instruction operates in Byte mode, 'bit4' must be between 0 and 7.

Words: 1

Cycles: 1 (2 or 3)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multicycle Instructions".

Example 1:

```
002000 HERE:   BTSC.B 0x1201, #2 ; If bit 2 of 0x1201 is 0,
002002       GOTO   BYPASS    ; skip the GOTO
002004       . .
002006       . .
002008       BYPASS:  . .
00200A       . .
```

Before Instruction After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 1200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>264F</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 1200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>264F</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:

002000 HERE: BTSC 0x804, #14 ; If bit 14 of 0x804 is 0,
002002 GOTO BYPASS ; skip the GOTO
002004 . . .
002006 . . .
002008 BYPASS: . . .
00200A . . .

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2006</td>
</tr>
<tr>
<td>Data 0804 2647</td>
<td>Data 0804 2647</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
BTSC

Bit Test Ws, Skip if Clear

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: 

{(label:) BTSC Ws, #bit4

[Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands:

Ws ∈ [W0 ... W15]

bit4 ∈ [0 ... 15]

Operation:

Test (Ws)<bit4>, skip if clear

Status Affected:

None

Encoding:

| 1010 | 0111 | bbbb | 0000 | 0ppp | ssss |

Description:

Bit 'bit4' in Ws 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 Ws 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 15) of the word. Either Register Direct or Indirect Addressing may be used for Ws.

The 'b' bits select value bit4, the bit position to test.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

Note: This instruction operates in Word mode only.

Words:

1

Cycles: 1 (2 or 3 if the next instruction is skipped)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

002000 HERE: BTSC W0, #0x0 ; If bit 0 of W0 is 0, 002002 GOTO BYPASS ; skip the GOTO 002004 . . . 002006 . . . 002008 BYPASS: . . . 00200A . . .

Before Instruction | After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
<th>PC</th>
<th>00 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>264F</td>
<td>W0</td>
<td>264F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### Example 2:

```
002000 HERE:   BTSC  W6, #0xF    ; If bit 15 of W6 is 0,
002002          GOTO    BYPASS    ; skip the GOTO
002004          . . .
002006          . . .
002008 BYPASS:  . . .
00200A          . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2006</td>
</tr>
<tr>
<td>W6 264F</td>
<td>W6 264F</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

### Example 3:

```
003400 HERE:   BTSC  [W6++], #0xC  ; If bit 12 of [W6] is 0,
003402          GOTO    BYPASS    ; skip the GOTO
003404          . . .
003406          . . .
003408 BYPASS:  . . .
00340A          . . .
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 3400</td>
<td>PC 00 3402</td>
</tr>
<tr>
<td>W6 1800</td>
<td>W6 1802</td>
</tr>
<tr>
<td>Data 1800</td>
<td>Data 1800</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
BTSS

Bit Test f, Skip if Set

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:
{label:} BTSS{.B} f, #bit4

Operands:
f ∈ [0 ... 8191] for byte operation
f ∈ [0 ... 8190] (even only) for word operation
bit4 ∈ [0 ... 7] for byte operation
bit4 ∈ [0 ... 15] for word operation

Operation: Test (f)<bit4>, skip if set

Status Affected: None

Encoding:

```
1010 1110 bbbf ffff ffff fffb
```

Description:

Bit ‘bit4’ in file register ‘f’ is tested. If the tested bit is ‘1’, 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 ‘0’, 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 operation, bit 15 for word operation).

The ‘b’ bits select value bit4, the bit position to test.
The ‘f’ bits select the address of the file register.

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.

2: When this instruction operates in Word mode, the file register address must be word-aligned.

3: When this instruction operates in Byte mode, ‘bit4’ must be between 0 and 7.

Words: 1

Cycles: 1 (2 or 3 if the next instruction is skipped)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

```
007100 HERE:     BTSS.B 0x1401, #0x1 ; If bit 1 of 0x1401 is 1,
007102       CLR WREG  ; don’t clear WREG
007104       . . .
```

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 1400</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>007100</td>
<td>0280</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 1400</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>007104</td>
<td>0280</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

```
007100 HERE:   BTSS 0x890, #0x9 ; If bit 9 of 0x890 is 1,
007102      GOTO BYPASS  ; skip the GOTO
007104      . . .
007106 BYPASS: . . .
```

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 0890</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>007100</td>
<td>00FE</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>Data 0890</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>007102</td>
<td>00FE</td>
<td>0000</td>
</tr>
</tbody>
</table>
### BTSS

**Bit Test Ws, Skip if Set**

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

{label;} BTSS Ws, #bit4

- [Ws],
- [Ws++],
- [Ws--],
- [++Ws],
- [--Ws].

#### Operands:

- Ws \( \in \{W0 \ldots W15\}\)
- bit4 \( \in \{0 \ldots 15\}\)

#### Operation:

Test (Ws)<bit4>, skip if set.

#### Status Affected:

None

#### Encoding:

```text
1010 0110 bbbb 0000 0ppp ssss
```

#### Description:

Bit 'bit4' in Ws is tested. If the tested bit is '1', 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 '0', the next instruction is executed as normal. In either case, the contents of Ws 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 15) of the word. Either Register Direct or Indirect Addressing may be used for Ws.

- The 'b' bits select the value bit4, the bit position to test.
- The 's' bits select the source register.
- The 'p' bits select the source addressing mode.

**Note:** This instruction operates in Word mode only.

#### Words:

1

#### Cycles:

1 (2 or 3 if the next instruction is skipped)\(^{(1)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

#### Example 1:

```
002000 HERE:    BTSS    W0, #0x0    ; If bit 0 of W0 is 1,
002002          GOTO    BYPASS         ; skip the GOTO
002004          . . .
002006          . . .
002008 BYPASS: . . .
00200A          . . .
```

#### Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>264F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

#### After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>264F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:

002000 HERE:   BTSS     W6, #0xF ; If bit 15 of W6 is 1,
002002         GOTO     BYPASS ; skip the GOTO
002004 . . .
002006 . . .
002008 BYPASS: . . .
00200A . . .

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>264F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>264F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 3:

003400 HERE:   BTSS  [W6++], 0xC ; If bit 12 of [W6] is 1,
003402         GOTO     BYPASS ; skip the GOTO
003404 . . .
003406 . . .
003408 BYPASS: . . .
00340A . . .

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 3400</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>1800</td>
</tr>
<tr>
<td>Data 1800</td>
<td>1000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 3406</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>1802</td>
</tr>
<tr>
<td>Data 1800</td>
<td>1000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### BTST Bit Test in f

#### Implemented in:
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:
(label:) BTST{.B} f, #bit4

#### Operands:
- f ∈ [0 ... 8191] for byte operation
- f ∈ [0 ... 8190] (even only) for word operation
- bit4 ∈ [0 ... 7] for byte operation
- bit4 ∈ [0 ... 15] for word operation

#### Operation:
(f)<bit4> → Z

#### Status Affected:
Z

#### Encoding:
```
1010 1011 bbbf ffff ffff fffb
```

#### Description:
Bit ‘bit4’ in file register ‘f’ is tested and the complement of the tested bit is stored to the Z flag in the STATUS Register. 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 operation, bit 15 for word operation).

The ‘b’ bits select value bit4, the bit position to be tested. The ‘f’ bits select the address of the file register.

**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.

**Note 2:** When this instruction operates in Word mode, the file register address must be word-aligned.

**Note 3:** When this instruction operates in Byte mode, ‘bit4’ must be between 0 and 7.

#### Words:
1

#### Cycles:
1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

**Example 1:**

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1200</td>
<td>Data 1200</td>
</tr>
<tr>
<td>F7FF</td>
<td>F7FF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002</td>
</tr>
</tbody>
</table>

BTST.B 0x1201, #0x3 ; Set Z = complement of bit 3 in 0x1201

**Example 2:**

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1302</td>
<td>Data 1302</td>
</tr>
<tr>
<td>F7FF</td>
<td>F7FF</td>
</tr>
<tr>
<td>SR 0002</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

BTST 0x1302, #0x7 ; Set Z = complement of bit 7 in 0x1302
BTST

Bit Test in Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BTST.C Ws, #bit4

BTST.Z [Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands:

Ws ∈ [W0 ... W15]

bit4 ∈ [0 ... 15]

Operation:

For " .C" Operation:

(Ws)<bit4> → C

For " .Z" Operation (default):

(Ws)<bit4> → Z

Status Affected:

Z or C

Encoding:

| 1010  | 0011 | bbbb | 2000 | 0ppp | ssse |

Description:

Bit 'bit4' in register Ws is tested. If the " .Z" option of the instruction is specified, the complement of the tested bit is stored to the Zero flag in the STATUS Register. If the " .C" option of the instruction is specified, the value of the tested bit is stored to the Carry flag in the STATUS Register. In either case, the contents of Ws are not changed.

For the bit4 operand, bit numbering begins with the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 15) of the word. Either Register Direct or Indirect Addressing may be used for Ws.

The 'b' bits select value bit4, the bit position to test.

The 'Z' bit selects the C or Z flag as destination.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

Note: This instruction only operates in Word mode. If no extension is provided, the " .Z" operation is assumed.

Words:

1

Cycles:

1(t)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

BTST.C [W0++], #0x3  ; Set C - bit 3 in (W0)
                     ; Post-increment W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 1200</td>
<td>W0 1202</td>
</tr>
<tr>
<td>Data 1200</td>
<td>Data 1200</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

### Example 2:

BTST.Z W0, #0x7     ; Set Z - complement of bit 7 in W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 F234</td>
<td>W0 F234</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002 (Z = 1)</td>
</tr>
</tbody>
</table>
BTST

Bit Test in Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC24F</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} BTST.C Ws, Wb

BTST.Z

Ws,

Wb

[Ws],

[Ws++],

[Ws--],

[++Ws],

[--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 | w000 | 0ppp | ssss |

Description:

The (Wb) bit in register Ws is tested. If the "C" option of the instruction is specified, the value of the tested bit is stored to the Carry flag in the STATUS Register. If the "Z" option of the instruction is specified, the complement of the tested bit is stored to the Zero flag in the STATUS Register. In either case, the contents of Ws are not changed.

Only the four Least Significant bits of Wb are used to determine the bit number. Bit numbering begins with the Least Significant bit (bit 0) and advances to the Most Significant bit (bit 15) of the Working register. Register Direct or Indirect Addressing may be used for Ws.

The ‘Z’ bit selects the C or Z flag as destination.

The ‘w’ bits select the address of the bit select register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note:

This instruction only operates in Word mode. If no extension is provided, the "Z" operation is assumed.

Words:

1

Cycles:

1(1)

Note 1:

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Notice:

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

1(1)
### Example 1: BTST.C W2, W3
; Set C = bit W3 of W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>F234</td>
</tr>
<tr>
<td>W3</td>
<td>2368</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>F234</td>
</tr>
<tr>
<td>W3</td>
<td>2368</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2: BTST.Z [W0++], W1
; Set Z = complement of bit W1 in [W0], Post-increment W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>1200</td>
</tr>
<tr>
<td>W1</td>
<td>CCC0</td>
</tr>
<tr>
<td>Data</td>
<td>6243</td>
</tr>
<tr>
<td>SR</td>
<td>0002 (Z = 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>1202</td>
</tr>
<tr>
<td>W1</td>
<td>CCC0</td>
</tr>
<tr>
<td>Data</td>
<td>6243</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**BTSTS**

**Bit Test/Set in f**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} BTSTS(.B) f, #bit4
```

**Operands:**

- `f` ∈ [0 ... 8191] for byte operation
- `f` ∈ [0 ... 8190] (even only) for word operation
- `bit4` ∈ [0 ... 7] for byte operation
- `bit4` ∈ [0 ... 15] for word operation

**Operation:**

```
(f)<bit4> → Z
1 → (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.

- **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.
- **Note 2:** When this instruction operates in Word mode, the file register address must be word-aligned.
- **Note 3:** When this instruction operates in Byte mode, `bit4` must be between 0 and 7.
- **Note 4:** The file register 'f' must not be the CPU STATUS Register (SR).

**Words:** 1

**Cycles:** 1

**Example 1:**

```
BTSTS.B 0x1201, #0x3 ; Set Z = complement of bit 3 in 0x1201, then set bit 3 of 0x1201 = 1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1200</td>
<td>Data 1200</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002</td>
</tr>
<tr>
<td>F7FF</td>
<td>FFFF</td>
</tr>
<tr>
<td></td>
<td>(Z = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

```
BTSTS 0x808, #15 ; Set Z = complement of bit 15 in 0x808, then set bit 15 of 0x808 = 1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM300</td>
<td>RAM300</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002</td>
</tr>
<tr>
<td>8050</td>
<td>8050</td>
</tr>
<tr>
<td></td>
<td>(Z = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

BTSTS

Bit Test/Set in Ws

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} BTSTS.C Ws, #bit4
BTSTS.Z [Ws],
[Ws++],
[Ws--],
[++Ws],
[--Ws],

Operands:

Ws ∈ [W0 ... W15]
bit4 ∈ [0 ... 15]

Operation:

For " .C" Operation:

(Ws)<bit4> → C
1 → Ws<bit4>

For " .Z" Operation (default):

(Ws)<bit4> → Z
1 → Ws<bit4>

Status Affected:

Z or C

Encoding:

1010 0100 bbbb 2000 0ppp ssss

Description:

Bit 'bit4' in register Ws is tested. If the " .Z" option of the instruction is specified, the complement of the tested bit is stored to the Zero flag in the STATUS Register. If the " .C" option of the instruction is specified, the value of the tested bit is stored to the Carry flag in the STATUS Register. In both cases, the tested bit in Ws is set to '1'.

The 'b' bits select the value bit4, the bit position to test/set.
The 'Z' bit selects the C or Z flag as destination.
The 'p' bits select the source addressing mode.
The 's' bits select the source register.

Note 1: This instruction only operates in Word mode. If no extension is provided, the " .Z" operation is assumed.

2: If Ws is used as a pointer, it must not contain the address of the CPU STATUS Register (SR).

3: In dsPIC33E, dsPIC33C and PIC24E devices, this instruction uses the DSRPAG register for indirect address generation in Extended Data Space.

Words: 1
Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
Example 1: \texttt{BTSTS.C [W0++], #0x3} \quad ; \text{Set C = bit 3 in [W0]}
\quad ; \text{Set bit 3 in [W0] = 1}
\quad ; \text{Post-increment W0}

Before Instruction & After Instruction \\
\textbf{W0} & 1200 & 1202 \\
\textbf{Data} & FFF7 & FFFF \\
\textbf{SR} & 0001 (C = 1) & 0000 \\

Example 2: \texttt{BTSTS.Z W0, #0x7} \quad ; \text{Set Z = complement of bit 7}
\quad ; \text{in W0, and set bit 7 in W0 = 1}

Before Instruction & After Instruction \\
\textbf{W0} & F234 & F2BC \\
\textbf{SR} & 0000 & 0002 (Z = 1)
CALL

Call Subroutine

Implemented in:  
- PIC24F  
- PIC24H  
- PIC24E  
- dsPIC30F  
- dsPIC33F  
- dsPIC33E  
- dsPIC33C

Syntax:  
{label:} CALL Expr

Operands:  
Expr may be a label or expression (but not a literal). Expr is resolved by the linker to a lit23, where lit23 ∈ [0 ... 8388606].

Operation:  
- (PC) + 4 → PC
- (PC<15:0>) → (TOS)
- (W15) + 2 → W15
- (PC<23:16>) → (TOS)
- (W15) + 2 → W15
- lit23 → PC

Status Affected:  
None

Encoding:  
1st word  
| 0000 | 0010 | nnnn | nnnn | nnnn | nnnn | nnn0 |

2nd word  
| 0000 | 0000 | 0000 | 0000 | 0nnn | nnnn |

Description:  
Direct subroutine call over the entire 4-Mbyte instruction program memory range. Before the CALL is made, the 24-bit return address (PC + 4) is PUSHed onto the stack. After the return address is stacked, the 23-bit value, ‘lit23’, is loaded into the PC.

The ‘n’ bits form the target address.

Note:  
The linker will resolve the specified expression into the lit23 to be used.

Words:  
2

Cycles:  
2

Example 1:  
026000 CALL _FIR ; Call _FIR subroutine
026004 MOV W0, W1
...  
026844 _FIR: MOV #0x400, W2 ; _FIR subroutine start
026846 ...
Example 2:

```
072000 CALL _G66 ; call routine _G66
072004 MOV W0, W1
... 
077A28 _G66: INC W6, [W7++] ; routine start
077A2A ...
077A2C
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 07 2000</td>
<td>PC 07 7A28</td>
</tr>
<tr>
<td>W15 9004</td>
<td>W15 9008</td>
</tr>
<tr>
<td>Data 9004</td>
<td>Data 9004</td>
</tr>
<tr>
<td>Data 9006</td>
<td>Data 9006</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### CALL

**Call Subroutine**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} CALL Expr

**Operands:**

Expr may be a label or expression (but not a literal).

**Operation:**

\[(PC) + 4 \rightarrow PC\]
\[(PC<15:1>) \rightarrow TOS<15:1>, SFA Status bit \rightarrow TOS<0>\]
\[(W15) + 2 \rightarrow W15\]
\[(PC<23:16>) \rightarrow TOS\]
\[(W15) + 2 \rightarrow W15\]

0 → SFA Status bit

lit23 → PC

NOP → Instruction Register

**Status Affected:** SFA

**Encoding:**

<table>
<thead>
<tr>
<th>1st word</th>
<th>0000</th>
<th>0010</th>
<th>nnnn</th>
<th>nnnn</th>
<th>nnnn</th>
<th>nnnn</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0nnn</td>
<td>nnnn</td>
</tr>
</tbody>
</table>

**Description:**

Direct subroutine call over the entire 4-Mbyte instruction program memory range. Before the CALL is made, the 24-bit return address \((PC + 4)\) is PUSHed onto the stack. After the return address is stacked, the 23-bit value, \(\text{lit23}\), is loaded into the PC.

The 'n' bits form the target address.

**Note:** The linker will resolve the specified expression into the \(\text{lit23}\) to be used.

**Words:**

2

**Cycles:**

4

**Example 1:**

```assembly
026000 CALL _FIR ; Call _FIR subroutine
026004 MOV W0, W1
... ...
026844 _FIR: MOV #0x400, W2 ; _FIR subroutine start
026846 ...
```

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>02 6000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>A268</td>
</tr>
<tr>
<td>Data A268</td>
<td>FFFF</td>
</tr>
<tr>
<td>Data A26A</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>02 6844</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>A26C</td>
</tr>
<tr>
<td>Data A268</td>
<td>6004</td>
</tr>
<tr>
<td>Data A26A</td>
<td>0002</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:

072000    CALL      _G66    ; call routine _G66
072004    MOV       W0, W1
             ...         
077A28    _G66:    INC       W6, [W7++]  ; routine start
077A2A          ...               
077A2C

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>07 2000</td>
</tr>
<tr>
<td>W15</td>
<td>9004</td>
</tr>
<tr>
<td>Data 9004</td>
<td>FFFF</td>
</tr>
<tr>
<td>Data 9006</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
CALL

Call Indirect Subroutine

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} CALL Wn

Operands: Wn ∈ [W0 ... W15]

Operation:

- (PC) + 2 → PC
- (PC<15:0>) → TOS
- (W15) + 2 → W15
- (PC<23:16>) → TOS
- (W15) + 2 → W15
- 0 → PC<22:16>
- (Wn<15:1>) → PC<15:1>
- NOP → Instruction Register

Status Affected: None

Encoding:

0000 0001 0000 0000 0000 ssss

Description: Indirect subroutine call over the first 32K instructions of program memory. Before the CALL is made, the 24-bit return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, Wn<15:1> is loaded into PC<15:1> and PC<22:16> is cleared. Since PC<0> is always '0', Wn<0> is ignored.

The 's' bits select the source register.

Words: 1

Cycles: 2

Example 1:

001002 CALL W0 ; Call BOOT subroutine indirectly
001004 ... ; using W0
... ...
001600 _BOOT: MOV #0x400, W2 ; _BOOT starts here
001602 MOV #0x300, W6
... ...

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001002</td>
<td>1600</td>
<td>6F00</td>
<td>6F00</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001600</td>
<td>1600</td>
<td>6F04</td>
<td>1004</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:

```
004200 CALL W7 ; Call TEST subroutine indirectly
004202 ... ; using W7

005500 _TEST: INC W1, W2 ; _TEST starts here
005502 DEC W1, W3 ;
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 4200</td>
<td>PC 00 5500</td>
</tr>
<tr>
<td>W7 5500</td>
<td>W7 5500</td>
</tr>
<tr>
<td>W15 6F00</td>
<td>W15 6F04</td>
</tr>
<tr>
<td>Data 6F00 FFFF</td>
<td>Data 6F00 4202</td>
</tr>
<tr>
<td>Data 6F02 FFFF</td>
<td>Data 6F02 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
CALL | Call Indirect Subroutine

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} CALL Wn
Operands: Wn ∈ [W0 ... W15]
Operation:

- (PC) + 2 → PC
- (PC<15:1>) → TOS, SFA Status bit → TOS<0>
- (W15) + 2 → W15
- (PC<23:16>) → TOS
- (W15) + 2 → W15
- 0 → SFA Status bit
- 0 → PC<22:16>
- (Wn<15:1>) → PC<15:1>

NOP → Instruction Register

Status Affected: SFA

Encoding:

| 0000 | 0001 | 0000 | 0000 | 0000 | ss || ss |

Description:

Indirect subroutine call over the first 32K instructions of program memory. Before the CALL is made, the 24-bit return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, Wn<15:1> is loaded into PC<15:1> and PC<22:16> is cleared. Since PC<0> is always '0', Wn<0> is ignored.

The 's' bits select the source register.

Words: 1
Cycles: 4

Example 1:

```assembly
001002 CALL W0 ; Call BOOT subroutine indirectly
001004 ... ; using W0
... 001600 _BOOT: MOV #0x400, W2 ; _BOOT starts here
001602 MOV #0x300, W6
... 001600 _BOOT: MOV #0x400, W2
```

Before Instruction | After Instruction
---|---
PC | 00 1002 | PC | 00 1600
W0 | 1600 | W0 | 1600
W15 | 6F00 | W15 | 6F00
Data 6F00 | FFFF | Data 6F00 | 1004
Data 6F02 | FFFF | Data 6F02 | 0000
SR | 0000 | SR | 0000
Example 2:

```
004200   CALL W7       ; Call TEST subroutine indirectly
004202   ...           ; using W7
005500   _TEST: INC W1, W2 ; _TEST starts here
005502   DEC W1, W3    
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 4200</td>
<td>PC 00 5500</td>
</tr>
<tr>
<td>W7 5500</td>
<td>W7 5500</td>
</tr>
<tr>
<td>W15 6F00</td>
<td>W15 6F04</td>
</tr>
<tr>
<td>Data 6F00 FFFF</td>
<td>Data 6F00 4202</td>
</tr>
<tr>
<td>Data 6F02 FFFF</td>
<td>Data 6F02 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### CALL.L

**Call Indirect Subroutine Long**

**Implemented in:**

- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**

```
{label:} CALL.L Wn
```

**Operands:**

\[ Wn \in [W0, W2, W4, W6, W8, W10, W12] \]

**Operation:**

- \((PC) + 2 \rightarrow PC\)
- \((PC<15:1>) \rightarrow TOS<15:1>, \text{ SFA Status bit} \rightarrow TOS<0>\)
- \((W15) + 2 \rightarrow W15\)
- \((PC<23:16>) \rightarrow TOS\)
- \((W15) + 2 \rightarrow W15\)
- \(0 \rightarrow \text{SFA Status bit}\)
- \(PC<23> \rightarrow PC<23> (\text{see text}); (Wn+1)<6:0> \rightarrow PC<22:16>\)
- \((Wn) \rightarrow PC<15:0>\)
- \(\text{NOP} \rightarrow \text{Instruction Register}\)

**Status Affected:**

- SFA

**Encoding:**

| 0000 | 0001 | lwww | w000 | 0000 | ssss |

**Description:**

Indirect subroutine call to any user program memory address. First, the return address (PC+2) and the state of the Stack Frame Active bit (SFA) are pushed onto the system stack, after which, the SFA bit is cleared.

Then, the Least Significant 7 bits of \((Wn+1)\) are loaded in \(PC<22:16>\) and the 16-bit value \((Wn)\) is loaded into \(PC<15:0>\).

\(PC<23>\) is not modified by this instruction.

The contents of \((Wn+1)<15:7>\) are ignored.

The value of \(Wn<0>\) is also ignored and \(PC<0>\) is always set to ‘0’.

The ‘s’ bits specify the address of the \(Wn\) source register.

The ‘w’ bits specify the address of the \(Wn+1\) source register.

**Words:**

1

**Cycles:**

4

#### Example 1:

```
026000 CALL.L W4       ; Call _FIR subroutine
026004 MOV W0, W1
  .
  .
  .
026844 _FIR: MOV #0x400, W2 ; _FIR subroutine start
026846
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 02 6000</td>
<td>PC 02 6844</td>
</tr>
<tr>
<td>W4 6844</td>
<td>W4 6844</td>
</tr>
<tr>
<td>W5 0002</td>
<td>W5 0002</td>
</tr>
<tr>
<td>W15 A268</td>
<td>W15 A26C</td>
</tr>
<tr>
<td>Data A268 FFFF</td>
<td>Data A268 6004</td>
</tr>
<tr>
<td>Data A26A FFFF</td>
<td>Data A26A 0002</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
CLR

Clear f or WREG

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CLR{.B} f

WREG

Operands:

f ∈ [0 ... 8191]

Operation:

0 → destination designated by D

Status Affected:

None

Encoding:

`1110 1111 0BDf ffff ffff ffff`

Description:

Clear the contents of a file register or the default Working register WREG. If WREG is specified, the WREG is cleared. Otherwise, the specified file register ‘f’ is cleared.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: 1

Example 1:

CLR.B RAM200 ; Clear RAM200 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM200 8009</td>
<td>RAM200 8000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

CLR WREG ; Clear WREG (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 0600</td>
<td>WREG 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

CLR

Clear Wd

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} CLR{.B} Wd

- [Wd]
- [Wd++]
- [Wd--]
- [++Wd]
- [--Wd]

Operands:  

Wd ∈ [W0 ... W15]

Operation:  

0 → Wd

Status Affected:  

None

Encoding:  

| 1110 | 1011 | 0Bqq | qddd | d000 | 0000 |

Description:  

Clear the contents of register Wd. Either Register Direct or Indirect Addressing may be used for Wd.

The ‘B’ bit select byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

Note: 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

Example 1: 

CLR.B W2 ; Clear W2 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 3333</td>
<td>W2 3300</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2: 

CLR [W0++] ; Clear [W0]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 2300</td>
<td>W0 2302</td>
</tr>
<tr>
<td>Data 2300 5607</td>
<td>Data 2300 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
CLR

Clear Accumulator, Prefetch Operands

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CLR Acc {[Wx],Wxd} {[Wy],Wyd} {[AWB]}

{[Wx] = kx,Wxd} {[Wy] = ky,Wyd}

{[W9 + W12],Wxd} {[W11 + W12],Wyd}

Operands:

Acc ∈ [A,B]

Wx ∈ [W8, W9]; kx ∈ [-6, -4, -2, 2, 4, 6]; Wxd ∈ [W4 ... W7]

Wy ∈ [W10, W11]; ky ∈ [-6, -4, -2, 2, 4, 6]; Wyd ∈ [W4 ... W7]

AWB ∈ [W13, [W13] + = 2]

Operation:

0 → Acc(A or B)

([Wx]) → Wxd; ([Wx] +/- kx → Wx

([Wy]) → Wyd; ([Wy] +/- ky → Wy

(Acc(B or A)) rounded → AWB

Status Affected:

OA, OB, SA, SB

Encoding:

| 1100 | 0011 | A0xx | yyii | iijj | jjaa |

Description:

Clear all 40 bits of the specified accumulator. Optionally prefetch operands in preparation for a MAC type instruction and optionally store the non-specified accumulator results. This instruction clears the respective overflow and saturate flags (either OA, SA or OB, SB).

Operands, Wx, Wxd, Wy and Wyd, specify optional prefetch operations, which support Indirect and Register Offset Addressing, as described in Section 4.15.1 “MAC Prefetches.”

Operand AWB specifies the optional register direct or indirect store of the convergently rounded contents of the “other” accumulator, as described in Section 4.15.4 “MAC Write-Back”.

The ‘A’ bit selects the other accumulator used for Write-Back.

The ‘x’ bits select the prefetch Wxd destination.

The ‘y’ bits select the prefetch Wyd destination.

The ‘i’ bits select the Wx prefetch operation.

The ‘j’ bits select the Wy prefetch operation.

The ‘a’ bits select the accumulator Write-Back destination.

Words: 1

Cycles: 1
Section 5. Instruction Descriptions

Example 1:  CLR A, [W8]+2, W4, W13 ; Clear ACCA
             ; Load W4 with [W8], post-inc W8
             ; Store ACCB to W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>W8</td>
<td>W8</td>
</tr>
<tr>
<td>W13</td>
<td>W13</td>
</tr>
<tr>
<td>ACCA</td>
<td>ACCA</td>
</tr>
<tr>
<td>ACCB</td>
<td>ACCB</td>
</tr>
<tr>
<td>Data 2000</td>
<td>Data 2000</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>

             ; Load W6 with [W8]
             ; Load W7 with [W10]
             ; Save ACCA to [W13]
             ; Post-inc W8,W10,W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>W6</td>
</tr>
<tr>
<td>W7</td>
<td>W7</td>
</tr>
<tr>
<td>W8</td>
<td>W8</td>
</tr>
<tr>
<td>W10</td>
<td>W10</td>
</tr>
<tr>
<td>W13</td>
<td>W13</td>
</tr>
<tr>
<td>ACCA</td>
<td>ACCA</td>
</tr>
<tr>
<td>ACCB</td>
<td>ACCB</td>
</tr>
<tr>
<td>Data 2000</td>
<td>Data 2000</td>
</tr>
<tr>
<td>Data 3000</td>
<td>Data 3000</td>
</tr>
<tr>
<td>Data 4000</td>
<td>Data 4000</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>
CLRWDT Clear Watchdog Timer

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

Syntax: {label;} CLRWDT

Operands: None

Operation:

0 → WDT Count register
0 → WDT Prescaler A count
0 → WDT Prescaler B count

Status Affected: None

Encoding:

1111 1110 0110 0000 0000 0000

Description: Clear the contents of the Watchdog Timer Count register and the Prescaler Count
registers. The Watchdog Prescaler A and Prescaler B settings, set by Configuration
fuses in the FWDT, are not changed.

Words: 1

Cycles: 1

Example 1: CLRWDT ; Clear Watchdog Timer

<table>
<thead>
<tr>
<th></th>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
### COM

**Complement f**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**  
{label:} COM{.B} f {,WREG}

**Operands:**  
f ∈ [0 ... 8191]

**Operation:**  
(f) → destination designated by D

**Status Affected:**  
N, Z

**Encoding:**  
```
1110 1110 1BDf ffff ffff ffff
```

**Description:**  
Compute the 1's complement of 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 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

**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.

**Note 2:**  
The WREG is set to Working register W0.

**Words:**  
1

**Cycles:**  
1

---

**Example 1:**  
COM.b RAM200 ; COM RAM200 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM200 80FF</td>
<td>RAM200 8000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002 (Z)</td>
</tr>
</tbody>
</table>

**Example 2:**  
COM RAM400, WREG ; COM RAM400 and store to WREG ; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 1211</td>
<td>WREG F7DC</td>
</tr>
<tr>
<td>RAM400 0823</td>
<td>RAM400 0823</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

---

**Note 1:**  
In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.

---

© 2005-2018 Microchip Technology Inc.  
DS70000157G-page 197
COM

Complement Ws

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label;\} COM{.B} Ws, Wd

\([Ws], \) [Wd]

\([Ws++], \) [Wd++]

\([Ws--], \) [Wd--]

\([++Ws], \) [++Wd]

\([-Ws], \) [-Wd]

Operands:

Ws \in [W0 ... W15]

Wd \in [W0 ... W15]

Operation:

\((Ws) \rightarrow Wd\)

Status Affected:

N, Z

Encoding:

```
1110 1010 1Bqq qddd dppp ssss
```

Description:

Compute the 1's complement of the contents of the source register Ws and place the result in the destination register Wd. Either Register Direct or Indirect Addressing may be used for both Ws and Wd.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'q' bits select the destination addressing mode.

The 'd' bits select the destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

Note: 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(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multicycle Instructions".

Example 1: COM.B [W0++], [W1++] ; COM [W0] and store to [W1] (Byte mode)

; Post-increment W0, W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 2301</td>
<td>W0 2302</td>
</tr>
<tr>
<td>W1 2400</td>
<td>W1 2401</td>
</tr>
<tr>
<td>Data 2300</td>
<td>Data 2300</td>
</tr>
<tr>
<td>Data 2400 ABCD</td>
<td>Data 2400 ABA9</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
### Example 2

COM W0, [W1++] ; COM W0 and store to [W1] (Word mode)
; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 D004</td>
<td>W0 D004</td>
</tr>
<tr>
<td>W1 1000</td>
<td>W1 1002</td>
</tr>
<tr>
<td>Data 1000 ABA9</td>
<td>Data 1000 2FFB</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
CP

Compare f with WREG, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} CP{.B}  f`

Operands:  

\( f \in [0 \ldots 8191] \)

Operation:  

\( f – \text{(WREG)} \)

Status Affected:  

DC, N, OV, Z, C

Encoding:

| 1110 | 0011 | 0B0f | ffff | ffff | ffff |

Description: Compute \( f – \text{(WREG)} \) and update the STATUS Register. This instruction is equivalent to the \text{SUBWF} instruction, but the result of the subtraction is not stored.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘f’ bits select the address of the file register.

\textbf{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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: \( 1^{(1)} \)

\textbf{Note 1:} In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see \textbf{Note 3} in \textit{Section 3.2.1 “Multicycle Instructions”}.

\textbf{Example 1:} \( \text{CP.B} \ \text{RAM400} \); Compare RAM400 with WREG (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 8823</td>
<td>WREG 8823</td>
</tr>
<tr>
<td>RAM400 0823</td>
<td>RAM400 0823</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0003           ( (C = 1) )</td>
</tr>
</tbody>
</table>

\textbf{Example 2:} \( \text{CP} \ \text{0x1200} \); Compare (0x1200) with WREG (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 2377</td>
<td>WREG 2377</td>
</tr>
<tr>
<td>Data 1200 2277</td>
<td>Data 1200 2277</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008           ( (N = 1) )</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

CP

Compare Wb with lit5, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} CP{.B} Wb, #lit5

Operands:

Wb ∈ [W0 ... W15]

lit5 ∈ [0 ... 31]

Operation:

(Wb) – lit5

Status Affected:

DC, N, OV, Z, C

Encoding:

1110 0001 0www wB00 011k kkkk

Description:

Compute (Wb) – lit5 and update the STATUS Register. This instruction is equivalent to the SUB instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb.

The ‘w’ bits select the address of the Wb Base register.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘k’ bits provide the literal operand, a five-bit integer number.

Note: 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

Example 1:

CP.B W4, #0x12 ; Compare W4 with 0x12 (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>W4</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7711</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W4</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7711</td>
<td>0008</td>
</tr>
</tbody>
</table>

Example 2:

CP W4, #0x12 ; Compare W4 with 0x12 (Word mode)

Before Instruction

<table>
<thead>
<tr>
<th>W4</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7713</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W4</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7713</td>
<td>0001</td>
</tr>
</tbody>
</table>

Words: 1

Cycles: 1
### CP

**Compare Wb with lit8, Set Status Flags**

**Implemented in:**

<table>
<thead>
<tr>
<th>16-Bit MCU</th>
<th>DSC Programmer’s Reference Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC24F</td>
<td>X</td>
</tr>
<tr>
<td>PIC24H</td>
<td>X</td>
</tr>
<tr>
<td>PIC24E</td>
<td>X</td>
</tr>
<tr>
<td>dsPIC30F</td>
<td>X</td>
</tr>
<tr>
<td>dsPIC33F</td>
<td>X</td>
</tr>
<tr>
<td>dsPIC33E</td>
<td>X</td>
</tr>
<tr>
<td>dsPIC33C</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:}   CP{.B}  Wb,   #lit8
```

**Operands:**

- `Wb ∈ [W0 ... W15]`
- `lit8 ∈ [0 ... 255]`

**Operation:**

\[(Wb) - \text{lit8}\]

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
1110 0001 0www  wBkk  k11k  kkkk
```

**Description:**

Compute \((Wb) - \text{lit8}\) and update the STATUS Register. This instruction is equivalent to the \(\text{SUB}\) instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb.

The 'w' bits select the address of the Wb Base register.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'k' bits provide the literal operand, a five-bit integer number.

**Note:** 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

**Example 1:**

```
CP.B  W4, #0x12       ; Compare W4 with 0x12 (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>7711</td>
<td>7711</td>
</tr>
</tbody>
</table>
| 0000               | 0009              (N, C = 1)

**Example 2:**

```
CP    W4, #0x12       ; Compare W4 with 0x12 (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>7713</td>
<td>7713</td>
</tr>
</tbody>
</table>
| 0000               | 0001              (C = 1)
### CP

#### Compare Wb with Ws, Set Status Flags

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} CP{.B} Wb, Ws

- [Ws]
- [Ws++]
- [Ws--]
- [++Ws]
- [--Ws]

**Operands:**

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]

**Operation:**

(Wb) – (Ws)

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
1110 0001 0www wB00 0ppp ssss
```

**Description:**

Compute (Wb) – (Ws) and update the STATUS Register. This instruction is equivalent to the SUB instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb. Register Direct or Indirect Addressing may be used for Ws.

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 'p' bits select the source addressing mode.

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 to denote a word operation, but it is not required.

**Words:**

1

**Cycles:**

1(1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

**Example 1:**

CP.B W0, [W1++] ; Compare [W1] with W0 (Byte mode) ; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 ABA9</td>
<td>W0 ABA9</td>
</tr>
<tr>
<td>W1 2000</td>
<td>W1 2001</td>
</tr>
<tr>
<td>Data 2000 D004</td>
<td>Data 2000 D004</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

CP W5, W6 ; Compare W6 with W5 (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5 2334</td>
<td>W5 2334</td>
</tr>
<tr>
<td>W6 8001</td>
<td>W6 8001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 000C (N, OV = 1)</td>
</tr>
</tbody>
</table>

© 2005-2018 Microchip Technology Inc. DS70000157G-page 203
CP0

Compare f with 0x0, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label: \} CP0{.B} f

Operands:

f ∈ [0 ... 8191]

Operation:

(f) – 0x0

Status Affected:

DC, N, OV, Z, C

Encoding:

|          | 1110 | 0010 | 0B0f | ffff | ffff | ffff |

Description:

Compute (f) – 0x0 and update the STATUS Register. The result of the subtraction is not stored.

The 'B' bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The 'f' bits select the address of the file register.

Note: 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

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

CP0.B RAM100 ; Compare RAM100 with 0x0 (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>RAM100</th>
<th>44C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>RAM100</th>
<th>44C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0009</td>
</tr>
</tbody>
</table>

(N, C = 1)

Example 2:

CP0 0x1FFE ; Compare (0x1FFE) with 0x0 (Word mode)

Before Instruction

<table>
<thead>
<tr>
<th>Data 1FFE</th>
<th>0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>Data 1FFE</th>
<th>0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0001</td>
</tr>
</tbody>
</table>

(C = 1)
### CP0

**Compare Ws with 0x0, Set Status Flags**

#### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```
{label;} CP0{.B} Ws

[Ws]
[Ws++]
[Ws--]
[++Ws]
[--Ws]
```

#### Operands:

| Ws ∈ [W0 ... W15] |

#### Operation:

(Ws) – 0x0000

#### Status Affected:

DC, N, OV, Z, C

#### Encoding:

```
1110 0000 0000 0B00 0ppp ssss
```

#### Description:

Compute (Ws) – 0x0000 and update the STATUS Register. The result of the subtraction is not stored. Register Direct or Indirect Addressing may be used for Ws.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘p’ bits select the source addressing mode.

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 to denote a word operation, but it is not required.

#### Words:

1

#### Cycles:

1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

**Example 1:**

```
CP0.B [W4--] ; Compare [W4] with 0 (Byte mode)

; Post-decrement W4
```

### Before Instruction

- W4: 1001
- Data: 1000
- SR: 0000

### After Instruction

- W4: 1000
- Data: 0034
- SR: 0001 (C = 1)

**Example 2:**

```
CP0 [--W5] ; Compare [--W5] with 0 (Word mode)
```

### Before Instruction

- W5: 2400
- Data: 23FE
- SR: 0000

### After Instruction

- W5: 23FE
- Data: 9000
- SR: 0009 (N, C = 1)
CPB

Compare f with WREG Using Borrow, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CPB{.B} f

Operands:

f ∈ [0 ... 8191]

Operation:

(f) – (WREG) – (C)

Status Affected:

DC, N, OV, Z, C

Encoding:

| 1110 0011 1B0f ffff ffff ffff |

Description:

Compute (f) – (WREG) – (C) and update the STATUS Register. This instruction is equivalent to the SUBB instruction, but the result of the subtraction is not stored.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

3: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1

Cycles: 1\(^{(1)}\)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

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

Before Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>8823</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM400</td>
<td>0823</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>8823</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM400</td>
<td>0823</td>
</tr>
</tbody>
</table>
| SR    | 0008 | (N = 1)

Example 2:

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

Before Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>2377</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1200</td>
<td>2377</td>
</tr>
<tr>
<td>SR</td>
<td>0001</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG</th>
<th>2377</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1200</td>
<td>2377</td>
</tr>
<tr>
<td>SR</td>
<td>0001</td>
</tr>
</tbody>
</table>
CPB

Compare Wb with lit5 Using Borrow, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wb</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CPB{.B} Wb, #lit5

Operands:

Wb ∈ [W0 ... W15]

lit5 ∈ [0 ... 31]

Operation:

(Wb) – lit5 – (C)

Status Affected:

DC, N, OV, Z, C

Encoding:

1110 0001 lwww wb00 011k kkkk

Description:

Compute (Wb) – lit5 – (C) and update the STATUS Register. This instruction is equivalent to the SUBB instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb.

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 ‘k’ bits provide the literal operand, a five-bit integer number.

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.

2: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1

Cycles: 1

Example 1:

CPB.B W4, #0x12    ; Compare W4 with 0x12 using C (Byte mode)

Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>7711</td>
<td>7711</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0001 (C = 1)</td>
<td>0008  (N = 1)</td>
</tr>
</tbody>
</table>

Example 2:

CPB.B W4, #0x12    ; Compare W4 with 0x12 using C (Byte mode)

Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>7711</td>
<td>7711</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0008  (N = 1)</td>
</tr>
</tbody>
</table>

Example 3:

CPB W12, #0x1F    ; Compare W12 with 0x1F using C (Word mode)

Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W12</td>
<td>W12</td>
</tr>
<tr>
<td>0020</td>
<td>0020</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0002 (Z = 1)</td>
<td>0003  (Z, C = 1)</td>
</tr>
</tbody>
</table>

Example 4:

CPB W12, #0x1F    ; Compare W12 with 0x1F using C (Word mode)

Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W12</td>
<td>W12</td>
</tr>
<tr>
<td>0020</td>
<td>0020</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0003 (Z, C = 1)</td>
<td>0001  (C = 1)</td>
</tr>
</tbody>
</table>
CPB

Compare Wb with lit8 Using Borrow, Set Status Flags

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CPB.{B} Wb, #lit8

Operands:

Wb ∈ [W0 ... W15]
lit8 ∈ [0 ... 255]

Operation:

(Wb) – lit8 – (C)

Status Affected:

DC, N, OV, Z, C

Encoding:

1110 0001 1WWW WWb kkK kkkk

Description:

Compute (Wb) – lit8 – (C) and update the STATUS Register. This instruction is equivalent to the SUBB instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb.

The 'w' bits select the address of the Wb register. The 'B' bit selects byte or word operation ('0' for word, '1' for byte). The 'k' bits provide the literal operand, a five-bit integer number.

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.

2:

The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1
Cycles: 1

Example 1:

CPB.B W4, #0x12  ; Compare W4 with 0x12 using C (Byte mode)

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 7711</td>
<td>W4 7711</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

Example 2:

CPB.B W4, #0x12  ; Compare W4 with 0x12 using C (Byte mode)

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 7711</td>
<td>W4 7711</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

Example 3:

CPB W12, #0x1F  ; Compare W12 with 0x1F using C (Word mode)

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W12 0020</td>
<td>W12 0020</td>
</tr>
<tr>
<td>SR 0002 (Z = 1)</td>
<td>SR 0003 (Z, C = 1)</td>
</tr>
</tbody>
</table>

Example 4:

CPB W12, #0x1F  ; Compare W12 with 0x1F using C (Word mode)

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W12 0020</td>
<td>W12 0020</td>
</tr>
<tr>
<td>SR 0003 (Z, C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
### CPB

**Compare Ws with Wb Using Borrow, Set Status Flags**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} CPB{.B} Wb, Ws
```

- [Ws]
- [Ws++]
- [Ws--]
- [++Ws]
- [--Ws]

**Operands:**

- Wb $\in$ [W0 ... W15]
- Ws $\in$ [W0 ... W15]

**Operation:**

$(Wb) - (Ws) - (C_0)$

**Status Affected:**

- DC, N, OV, Z, C

**Encoding:**

```
1110 0001 1www wB00 0ppp ssss
```

**Description:**

Compute $(Wb) - (Ws) - (C_0)$ and update the STATUS Register. This instruction is equivalent to the `SUBB` instruction, but the result of the subtraction is not stored. Register Direct Addressing must be used for Wb. Register Direct or Indirect Addressing may be used for Ws.

- 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 'p' bits select the source addressing mode.
- The 's' bits select the address of the Ws source register.

**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.

**Note 2:**

- The Z flag is “sticky” for `ADDC`, `CPB`, `SUBB` and `SUBBR`. These instructions can only clear Z.

**Words:**

1

**Cycles:**

1(1)

**Example 1:**

```
CPB.B W0, [W1++] ; Compare [W1] with W0 using c (Byte mode) ; Post-increment W1
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 ABA9</td>
<td>W0 ABA9</td>
</tr>
<tr>
<td>W1 1000</td>
<td>W1 1001</td>
</tr>
<tr>
<td>Data 1000 D0A9</td>
<td>Data 1000 D0A9</td>
</tr>
<tr>
<td>SR 0002 (Z = 1)</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
**Example 2:**  \( \text{CPB.B W0, [W1++]} \) ; Compare [W1] with W0 using \( \overline{C} \) (Byte mode)  

; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{W0} )</td>
<td>(\text{ABA9})</td>
</tr>
<tr>
<td>(\text{W1} )</td>
<td>(\text{1000})</td>
</tr>
<tr>
<td>Data 1000</td>
<td>(\text{D0A9})</td>
</tr>
<tr>
<td>(\text{SR} )</td>
<td>(0001) (C = 1)</td>
</tr>
</tbody>
</table>

**Example 3:**  \( \text{CPB W4, W5} \) ; Compare W5 with W4 using \( \overline{C} \) (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{W4} )</td>
<td>(\text{4000})</td>
</tr>
<tr>
<td>(\text{W5} )</td>
<td>(\text{3000})</td>
</tr>
<tr>
<td>(\text{SR} )</td>
<td>(0001) (C = 1)</td>
</tr>
</tbody>
</table>
## CPBEQ

### Compare Wb with Wn, Branch if Equal (Wb = Wn)

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

\{label:\} CPBEQ{.B} {Wb, Wn, Expr}

**Operands:**

- Wb ∈ [W0 ... W15]
- Wn ∈ [W0 ... W15]

**Operation:**

(Wb) – (Wn)

If (Wb) = (Wn), [(PC+2) + 2 * Expr] → PC and NOP → Instruction Register

**Status Affected:** None

**Encoding:**

| 1110 | 0111 | 1www | wBnn | nnnn | 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, the PC is recalculated based on the 6-bit signed offset specified by Expr, and on the next cycle, a NOP is executed instead. If (Wb) ≠ (Wn), the next instruction is executed as normal (branch is not taken).

The 'w' bits select the address of the Wb source register.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 's' bits select the address of the Wn source register.

The 'n' bits select the offset of the branch destination.

**Note:** 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 (5 if branch taken)

---

### Example 1:

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>002000</td>
<td>HERE: CPBEQ.B W0, W1, BYPASS</td>
</tr>
<tr>
<td>002002</td>
<td>ADD W2, W3, W4</td>
</tr>
<tr>
<td>002004</td>
<td></td>
</tr>
<tr>
<td>002006</td>
<td></td>
</tr>
<tr>
<td>002008</td>
<td>BYPASS:</td>
</tr>
<tr>
<td>00200A</td>
<td></td>
</tr>
</tbody>
</table>

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1000</td>
<td>1000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1000</td>
<td>1000</td>
<td>0002</td>
</tr>
</tbody>
</table>

(Z = 1)
### CPBGT

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

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**

```
{label:} CPBGT{.B} Wb, Wn, Expr
```

**Operands:**

- `Wb` ∈ [W0 ... W15]
- `Wn` ∈ [W0 ... W15]

**Operation:**

```
(Wb) – (Wn)
```

If `(Wb) = (Wn)`, `[(PC+2) + 2 * Expr] → PC and NOP → Instruction Register`

**Status Affected:** None

**Encoding:**

```
1110 0110 0www wBnn nnnn 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, the PC is recalculated based on the 6-bit signed offset specified by `Expr`, and on the next cycle, a NOP is executed instead. If `(Wb) ≠ (Wn)`, the next instruction is executed as normal (branch is not taken).

The ‘w’ bits select the address of the `Wb` source register.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘s’ bits select the address of the `Wn` source register.

The ‘n’ bits select the offset of the branch destination.

**Note:** 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 (5 if branch taken)

#### Example 1:

```
002000 HERE:   CPBGT.B W0, W1, BYPASS ; If W0 > W1 (Byte mode),
002002        ADD W2, W3, W4 ; Perform branch to BYPASS
002004        ...
002006        ...
002008 BYPASS ...
00200A ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2008</td>
</tr>
<tr>
<td>W0 30FF</td>
<td>W0 00FF</td>
</tr>
<tr>
<td>W1 26FE</td>
<td>W1 26FE</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000 (N, C = 0)</td>
</tr>
</tbody>
</table>

---

DS70000157G-page 212 © 2005-2018 Microchip Technology Inc.
### CPBLT

**Signed Compare Wb with Wn, Branch if Less Than (Wb < Wn)**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\{label: \} CPBLT{.B} Wb, Wn, Expr

**Operands:**

Wb ∈ [W0 ... W15]

Wn ∈ [W0 ... W15]

**Operation:**

(Wb) – (Wn)

If (Wb) = (Wn), [(PC+2) + 2 * Expr] → PC and NOP → Instruction Register

**Status Affected:**

None

**Encoding:**

```
1110 0110 1www wBnn nnnn 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, the PC is recalculated based on the 6-bit signed offset specified by Expr, and on the next cycle, a NOP is executed instead. If (Wb) ≠ (Wn), the next instruction is executed as normal (branch is not taken).

The 'w' bits select the address of the Wb source register.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 's' bits select the address of the Wn source register.

The 'n' bits select the offset of the branch destination.

**Note:** 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 (5 if branch taken)

---

**Example 1:**

```
002000 HERE:   CPBLT.B W8, W9, BYPASS ; If W8 < W9 (Byte mode),
002002         ADD W2, W3, W4 ; Perform branch to BYPASS
002004 ...
002006 ...
002008 BYPASS: ...
00200A ...
```

---

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W8</th>
<th>W9</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00200</td>
<td>00FF</td>
<td>26FE</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W8</th>
<th>W9</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00208</td>
<td>00FF</td>
<td>26FE</td>
<td>0008</td>
</tr>
</tbody>
</table>

(N = 1)
CPBNE

Compare Wb with Wn, Branch if Not Equal (Wb ≠ Wn)

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CPBNE{.B} Wb, Wn, Expr

Operands:

Wb ∈ [W0 ... W15]
Wn ∈ [W0 ... W15]

Operation:

(Wb) – (Wn)
If (Wb) = (Wn), [(PC+2) + 2 * Expr] → PC and NOP → Instruction Register

Status Affected:
None

Encoding:

1110 0111 0www wBnn nnnn ss

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, the PC is recalculated based on the 6-bit signed offset specified by Expr, and on the next cycle, a NOP is executed instead. If (Wb) ≠ (Wn), the next instruction is executed as normal (branch is not taken).

The 'w' bits select the address of the Wb source register.
The 'B' bit selects byte or word operation ('0' for word, '1' for byte).
The 's' bits select the address of the Wn source register.
The 'n' bits select the offset of the branch destination.

Note: 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 (5 if branch taken)

Example 1:

002000 HERE: CPBNE.B W2, W3, BYPASS ; If W2 != W3 (Byte mode),
002002 ADD W2, W3, W4 ; Perform branch to BYPASS
002004 ... 002006 ...
002008 BYPASS: ...
00200A ...

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 2000</td>
<td>PC  00 200A</td>
</tr>
<tr>
<td>W2  00FF</td>
<td>W2  00FF</td>
</tr>
<tr>
<td>W3  26FE</td>
<td>W3  26FE</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0001 (C = 1)</td>
</tr>
</tbody>
</table>
## CPSEQ

**Compare Wb with Wn, Skip if Equal (Wb = Wn)**

### Implemented in:
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

### 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 Wn source register.

**Note:** 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)

**Example 1:**
```
002000 HERE:  CPSEQ.B  W0, W1 ; If W0 = W1 (Byte mode),
002002 GOTO BYPASS ; skip the GOTO
002004 ...
002006 ...
002008 BYPASS: ...
00200A ...
```

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>002000</td>
<td>1001</td>
<td>1000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W0</th>
<th>W1</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>002002</td>
<td>1001</td>
<td>1000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**
```
018000 HERE:  CPSEQ  W4, W8 ; If W4 = W8 (Word mode),
018002 CALL _FIR ; skip the subroutine call
018006 ...
018008 ...
```

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W4</th>
<th>W8</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>018000</td>
<td>3344</td>
<td>3344</td>
<td>0002 (Z = 1)</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W4</th>
<th>W8</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>018006</td>
<td>3344</td>
<td>3344</td>
<td>0002 (Z = 1)</td>
</tr>
</tbody>
</table>
CPSEQ

Compare Wb with Wn, Skip if Equal (Wb = Wn)

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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 0001 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) ≠ (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 Wn source register.

Note: 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)

Example 1:

002000 HERE: CPSEQ.B W0, W1 ; If W0 = W1 (Byte mode),
002002 GOTO BYPASS ; skip the GOTO
002004 ...
002006 ...
002008 BYPASS: ...
00200A ...

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2002</td>
</tr>
<tr>
<td>W0 1001</td>
<td>W0 1001</td>
</tr>
<tr>
<td>W1 1000</td>
<td>W1 1000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

018000 HERE: CPSEQ W4, W8 ; If W4 = W8 (Word mode),
018002 CALL _FIR ; skip the subroutine call
018006 ...
018008 ...

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 01 8000</td>
<td>PC 01 8006</td>
</tr>
<tr>
<td>W4 3344</td>
<td>W4 3344</td>
</tr>
<tr>
<td>W8 3344</td>
<td>W8 3344</td>
</tr>
<tr>
<td>SR 0002 (Z = 1)</td>
<td>SR 0002 (Z = 1)</td>
</tr>
</tbody>
</table>
## CPSGT

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

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

{label:} CPSGT{.B} Wb, Wn

### Operands:

- Wb ∈ [W0 ... W15]
- Wn ∈ [W0 ... W15]

### Operation:

(Wb) – (Wn)

Skip if (Wb) > (Wn)

### Status Affected:

None

### Encoding:

| 1110 | 0110 | 0www | 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 Wn source register.

**Note:** 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)

### Example 1:

```
002000 HERE: CPSGT.B W0, W1 ; If W0 > W1 (Byte mode),
002002 GOTO BYPASS ; skip the GOTO
002006 ...
002008 ...
00200A BYPASS ...
00200C ...
```

### Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>00FF</td>
</tr>
<tr>
<td>W1</td>
<td>26FE</td>
</tr>
<tr>
<td>SR</td>
<td>0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

### After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>00FF</td>
</tr>
<tr>
<td>W1</td>
<td>26FE</td>
</tr>
<tr>
<td>SR</td>
<td>0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

```
018000 HERE: CPSGT W4, W5 ; If W4 > W5 (Word mode),
018002 CALL _FIR ; skip the subroutine call
018006 ...
018008 ...
```

### Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>01 8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>2600</td>
</tr>
<tr>
<td>W5</td>
<td>2600</td>
</tr>
<tr>
<td>SR</td>
<td>0004 (OV = 1)</td>
</tr>
</tbody>
</table>

### After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>01 8002</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>2600</td>
</tr>
<tr>
<td>W5</td>
<td>2600</td>
</tr>
<tr>
<td>SR</td>
<td>0004 (OV = 1)</td>
</tr>
</tbody>
</table>
CPSGT

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

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} CPSGT{.B} Wb, Wn

Operands:

Wb ∈ [W0 ... W15]
Wn ∈ [W0 ... W15]

Operation:

(Wb) – (Wn)
Skip if (Wb) > (Wn)

Status Affected:
None

Encoding:

1110 0110 0www wB00 0001 ss

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 Wn source register.

Note: 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)

Example 1:

002000 HERE: CPSGT.B W0, W1 ; If W0 > W1 (Byte mode),
002002 GOTO BYPASS ; skip the GOTO
002006 ...
002008 ...
00200A BYPASS ...
00200C ...

Example 2:

018000 HERE: CPSGT W4, W5 ; If W4 > W5 (Word mode),
018002 CALL _FIR ; skip the subroutine call
018006 ...
018008 ...

Before Instruction | After Instruction
PC 00 2000           | PC 00 2006
W0 00FF              | W0 00FF
W1 26FE              | W1 26FE
SR 0009 (N, C = 1)   | SR 0009 (N, C = 1)

Before Instruction | After Instruction
PC 01 8000           | PC 01 8002
W4 2600              | W4 2600
W5 2600              | W5 2600
SR 0004 (OV = 1)     | SR 0004 (OV = 1)
CPSLT  
Signed Compare Wb with Wn, Skip if Less Than (Wb < Wn)

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: 
{label:} CPSLT{.B} Wb, Wn

Operands: 
Wb ∈ [W0 ... W15]  
Wn ∈ [W0 ... W15]

Operation: 
(Wb) – (Wn)  
Skip if (Wb) < (Wn)

Status Affected: None

Encoding:

\[
\begin{array}{cccccc}
1110 & 0110 & 1\text{www} & w\text{B00} & 0000 & s\text{sss}
\end{array}
\]

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 \textit{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 Wn source register.

\textbf{Note:} The extension \texttt{.B} in the instruction denotes a byte operation rather than a word operation. You may use a \texttt{.W} extension to 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 Instruction | After Instruction  
PC | 00 2000 | PC | 00 2002  
W8 | 00FF | W8 | 00FF  
W9 | 26FE | W9 | 26FE  
SR | 0008 (N = 1) | SR | 0008 (N = 1)  

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 | PC | 01 8006  
W3 | 2600 | W3 | 2600  
W6 | 3000 | W6 | 3000  
SR | 0000 | SR | 0000  
CPSLT
Signed Compare Wb with Wn, Skip if Less Than (Wb < Wn)

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} CPSLT{.B} Wb, Wn

Operands:

Wb ∈ [W0 ... W15]
Wn ∈ [W0 ... W15]

Operation:

(Wb) – (Wn)
Skip if (Wb) < (Wn)

Status Affected:
None

Encoding:

1110 0110 1www wB00 0001 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 Wn source register.

Note: 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)

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 Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8</td>
<td>00FF</td>
</tr>
<tr>
<td>W9</td>
<td>26FE</td>
</tr>
<tr>
<td>SR</td>
<td>0008(N = 1)</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>00 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8</td>
<td>00FF</td>
</tr>
<tr>
<td>W9</td>
<td>26FE</td>
</tr>
<tr>
<td>SR</td>
<td>0008(N = 1)</td>
</tr>
</tbody>
</table>

Example 2:

018000 HERE: CPSLT W3, W6 ; If W3 < W6 (Word mode),
018002 CALL _FIR ; skip the subroutine call
018006 ...
018008 ...

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>01 8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>2600</td>
</tr>
<tr>
<td>W6</td>
<td>3000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>01 8006</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>2600</td>
</tr>
<tr>
<td>W6</td>
<td>3000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

CPSNE  Signed Compare Wb with Wn, Skip if Not Equal (Wb ≠ Wn)

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
--- | --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | x | x |

Syntax:  

{label:} CPSNE{.B} Wb, Wn

Operands:  

Wb ∈ [W0 ... W15]  
Wn ∈ [W0 ... W15]

Operation:  

(Wb) – (Wn)  
Skip if (Wb) ≠ (Wn)

Status Affected: None

Encoding:

```
1110 0111 0www 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 Wn source register.

Note:  
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)

**Example 1:**

```
002000 HERE: PSNE.B W2, W3  ; If W2 != W3 (Byte mode),
002002      GOTO BYPASS  ; skip the GOTO
002006      ...  
002008      ...  
00200A BYPASS: ...  
00200C      ...
```

**Example 2:**

```
018000 HERE: CPSNE W0, W8  ; If W0 ≠ W8 (Word mode),
018002      CALL _FIR     ; skip the subroutine call
018006      ...  
018008      ...
```

Before Instruction | After Instruction
--- | ---
PC | 00 2000 | 00 2006
W2 | 00FF | 00FF
W3 | 26FE | 26FE
SR | 0001 (C = 1) | 0001 (C = 1)

Before Instruction | After Instruction
--- | ---
PC | 01 8000 | 01 8002
W0 | 3000 | 3000
W8 | 3000 | 3000
SR | 0000 | 0000
CPSNE  Signed Compare Wb with Wn, Skip if Not Equal (Wb ≠ Wn)

Implemented in:  
<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
\{label:\}  CPSNE{.B}  Wb,  Wn

Operands:  
Wb ∈ [W0 ... W15]  
Wn ∈ [W0 ... W15]

Operation:  
(Wb) – (Wn)  
Skip if (Wb) ≠ (Wn)

Status Affected:  
None

Encoding:  
1110 0111 0www wB00 0001 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 Wn source register.

Note:  
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)

Example 1:  
002000  HERE:  CPSNE.B  W2,  W3 ; If W2 ≠ W3 (Byte mode),
002002  GOTO  BYPASS ; skip the GOTO
002006  ...
002008  ...
00200A  BYPASS:  ...
00200C  ...

Before Instruction  |  After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 2000</td>
<td>PC  00 2006</td>
</tr>
<tr>
<td>W2  00FF</td>
<td>W2  00FF</td>
</tr>
<tr>
<td>W3  26FE</td>
<td>W3  26FE</td>
</tr>
<tr>
<td>SR  0001 (C = 1)</td>
<td>SR  0001 (C = 1)</td>
</tr>
</tbody>
</table>

Example 2:  
018000  HERE:  CPSNE  W0,  W8 ; If W0 ≠ W8 (Word mode),
018002  CALL  _FIR ; skip the subroutine call
018006  ...
018008  ...

Before Instruction  |  After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  01 8000</td>
<td>PC  01 8002</td>
</tr>
<tr>
<td>W0  3000</td>
<td>W0  3000</td>
</tr>
<tr>
<td>W8  3000</td>
<td>W8  3000</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
CTXTSWP(1)

CPU Register Context Swap Literal

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C
---|---|---|---|---|---|---|---

Syntax: {label:} CTXTSWP #lit3

Operands: lit3 ∈ [0 ... 4]

Operation: If context defined by lit3 is valid,
Then
Switch CPU register context to context defined by lit3
Else
Execute as 2-cycle NOP

Status Affected: None

Encoding:

| 1111 | 1110 | 1110 | 0000 | 0000 | 0kkk |

Description: This instruction will force a CPU register context switch (W0 through W14, and Accumulators A and B) from the current context to the target context defined by the value defined by #lit3. If the specified context is not implemented on the device, this instruction will execute as a 2-cycle NOP.

A successful context switch will update the current context identifier and the manual context identifier (held in CCTXI<2:0> (CTXSTAT<10:8>) and MCTXI<2:0> (CTXSTAT<2:0>), respectively) to reflect the new active CPU register context.

Words: 1
Cycles: 2

Note 1: This instruction is present only in some devices of the device families. Please see the specific device data sheet to ensure that this instruction is supported on a specific device.
### CTXTSWP\(^{(1)}\)

**CPU Register Context Swap Wn**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

\{label;\} CTXTSWP Wn

**Operands:**

Wn ∈ [W0 ... W15]

**Operation:**

If context defined by the contents of Wn<2:0> is valid,

Then

Switch CPU register context to context defined by the contents of Wn<2:0>

Else

Execute as 2-cycle NOP

**Status Affected:**

None

**Encoding:**

| 1111 | 1110 | 1111 | 0000 | 0000 | ssss |

**Description:**

This instruction will force a CPU register context switch (W0 through W14, and Accumulators A and B) from the current context to the target context defined by the value in the three Least Significant bits of Wn. If the specified context is not implemented on the device, this instruction will execute as a 2-cycle NOP.

A successful context switch will update the current context identifier and the manual context identifier (held in CCTXI<2:0> (CTXSTAT<10:8>) and MCTXI<2:0> (CTXSTAT<2:0>), respectively) to reflect the new active CPU register context.

**Words:** 1

**Cycles:** 2

**Note 1:** This instruction is present only in some devices of the device families. Please see the specific device data sheet to ensure that this instruction is supported on a specific device.
## DAW.B

### Decimal Adjust Wn

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
\text{Syntax: } \{\text{label:}\} \text{ DAW.B Wn}
\]

**Operands:**

\[
\text{Wn} \in [W0 \ldots W15]
\]

**Operation:**

\[
\begin{align*}
\text{If (Wn<3:0> > 9) or (DC = 1)} \\
(Wn<3:0>) + 6 & \rightarrow Wn<3:0> \\
\text{Else} \\
(Wn<3:0>) & \rightarrow Wn<3:0>
\end{align*}
\]

\[
\begin{align*}
\text{If (Wn<7:4> > 9) or (C = 1)} \\
(Wn<7:4>) + 6 & \rightarrow Wn<7:4> \\
\text{Else} \\
(Wn<7:4>) & \rightarrow Wn<7:4>
\end{align*}
\]

**Status Affected:**

C

**Encoding:**

| 1111 | 1101 | 0100 | 0000 | 0000 | ssss |

**Description:**

Adjust the Least Significant Byte in Wn to produce a Binary Coded Decimal (BCD) result. The Most Significant Byte of Wn is not changed and the Carry flag is used to indicate any decimal rollover. Register Direct Addressing must be used for Wn.

The ‘s’ bits select the source/destination register.

**Note 1:** This instruction is used to correct the data format after two packed BCD bytes have been added.

**Note 2:** This instruction operates in Byte mode only and the .B extension must be included with the opcode.

**Words:**

1

**Cycles:**

1

---

**Example 1:**

DAW.B W0 ; Decimal adjust W0

**Before Instruction**

<table>
<thead>
<tr>
<th>W0</th>
<th>771A</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0002 (DC = 1)</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>W0</th>
<th>7720</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0002 (DC = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

DAW.B W3 ; Decimal adjust W3

**Before Instruction**

<table>
<thead>
<tr>
<th>W3</th>
<th>77AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>W3</th>
<th>7710</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
DEC Decrement \( f \)

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\[
\text{\{label:}\ DEC{.B} f \{,WREG\}
\]

Operands:

\( f \in [0 \ldots 8191] \)

Operation:

\( (f) - 1 \rightarrow \text{destination designated by D} \)

Status Affected:

DC, N, OV, Z, C

Encoding:

| 1110 1101 0BDf ffff ffff ffff |

Description:

Subtract one from 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 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

**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.

**Note 2:** The WREG is set to Working register W0.

Words: 1

Cycles: 1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 "Multicycle Instructions".

**Example 1:**

```
DEC.B 0x200 ; Decrement (0x200) (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 200 80FF</td>
<td>Data 200 80FE</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

```
DEC RAM400, WREG ; Decrement RAM400 and store to WREG
                  ; (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 1211</td>
<td>WREG 0822</td>
</tr>
<tr>
<td>RAM400 0823</td>
<td>RAM400 0823</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

DEC
Decrement Ws

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} DEC{B} Ws, Wd

[Ws]., [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [++Wd]

[--Ws], [--Wd]

Operands:

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

Operation:

(Ws) – 1 → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

| 1110 | 1001 | OBqq | qddd | dppp | ssss |

Description:

Subtract one from the contents of the source register Ws and place the result in the destination register Wd. Either Register Direct or Indirect Addressing may be used by Ws and Wd.

The ’B’ bit selects byte or word operation (’0’ for word, ’1’ for byte).

The ’q’ bits select the destination addressing mode.

The ’d’ bits select the destination register.

The ’p’ bits select the source addressing mode.

The ’s’ bits select the source register.

Note:

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(f)

Note 1:

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

DEC.B [W7++], [W8++] ; DEC [W7] and store to [W8] (Byte mode)

; Post-increment W7, W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 2301</td>
<td>W7 2302</td>
</tr>
<tr>
<td>W8 2400</td>
<td>W8 2401</td>
</tr>
<tr>
<td>Data 2300</td>
<td>Data 2300</td>
</tr>
<tr>
<td>Data 2400</td>
<td>Data 2400</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example 2:**

```plaintext
DEC W5, [W6++]  ; Decrement W5 and store to [W6] (Word mode)
; Post-increment W6
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5: D004</td>
<td>W5: D004</td>
</tr>
<tr>
<td>Data 2000: ABA9</td>
<td>Data 2000: D003</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0009</td>
</tr>
</tbody>
</table>

(N, C = 1)
### DEC2

**Decrement f by 2**

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

{label;} DEC2{,B} f {,WREG}

#### Operands:

f ∈ [0 ... 8191]

#### Operation:

(f) – 2 → destination designated by D

#### Status Affected:

DC, 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 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 register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

**Note:** 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(1)

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.

---

**Example 1:**

DEC2.B 0x200 ; Decrement (0x200) by 2 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 200</td>
<td>Data 200</td>
</tr>
<tr>
<td>80FF</td>
<td>80FD</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0009</td>
</tr>
</tbody>
</table>

(N, C = 1)

**Example 2:**

DEC2 RAM400, WREG ; Decrement RAM400 by 2 and store to WREG (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG</td>
<td>WREG</td>
</tr>
<tr>
<td>1211</td>
<td>0821</td>
</tr>
<tr>
<td>RAM400</td>
<td>RAM400</td>
</tr>
<tr>
<td>0823</td>
<td>0823</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
### DEC2

**Decrement Ws by 2**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} DEC2{.B} Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [+Wd]

[--Ws], [--Wd]

**Operands:**

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

**Operation:**

(Ws) – 2 → Wd

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
1110 1001 1Bqq qddd dppp ssss
```

**Description:** Subtract two from the contents of the source register Ws and place the result in the destination register Wd. Either Register Direct or Indirect Addressing may be used by Ws and Wd.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note:** 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

**Example 1:**

DEC2.B [W7--], [W8--] ; DEC [W7] by 2, store to [W8] (Byte mode)

; Post-decrement W7, W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 2301 W8 2400</td>
<td>W7 2300 W8 23FF</td>
</tr>
<tr>
<td>Data 2300 0107</td>
<td>Data 2300 0107</td>
</tr>
<tr>
<td>Data 2400 ABCD</td>
<td>Data 2400 ABFF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3 in Section 3.2.1 “Multicycle Instructions”**.
**Example 2:**  
DEC2 W5, [W6++] ; DEC W5 by 2, store to [W6] (Word mode)  
; Post-increment W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>D004</td>
</tr>
<tr>
<td>W6</td>
<td>1000</td>
</tr>
<tr>
<td>Data 1000</td>
<td>ABA9</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**DISI**

**Disable Interrupts Temporarily**

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

```
{label:} DISI #lit14
```

### Operands:

lit14 ∈ [0 ... 16383]

### Operation:

- lit14 → DISICNT
- 1 → DISI

Disable interrupts for (lit14 + 1) cycles

### Status Affected:

None

### Encoding:

```
1111 1100 00kk kkkk kkkk kkkk
```

### Description:

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<14>) is set to ‘1’. This instruction can be used before executing time-critical code to limit the effects of interrupts.

**Note 1:** This instruction does not prevent Priority 7 interrupts and traps from running. See the specific device family reference manual for details.

**Note 2:** This instruction does not prevent any interrupts when the device is in Sleep mode.

### Words:

1

### Cycles:

1

### Example 1:

```
002000 HERE: DISI #100 ; Disable interrupts for 101 cycles
002002          ; next 100 cycles protected by DISI
002004          ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2002</td>
</tr>
<tr>
<td>DISICNT 0000</td>
<td>DISICNT 0100</td>
</tr>
<tr>
<td>INTCON2 0000</td>
<td>INTCON2 4000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
DIV.S

Signed Integer Divide

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} DIV.S(W)  Wm, Wn

DIV.SD  Wm, Wn

Operands:

Wm  ∈ [W0 ... W15] for word operation
Wm  ∈ [W0, W2, W4 ... W14] for double operation
Wn  ∈ [W2 ... W15]

Operation:

For Word Operation (default):

Wm  → W0
If (Wm<15> = 1):
    0xFFFF → W1
Else:
    0x0 → W1
W1:W0/Wn  → W0
Remainder  → W1

For Double Operation (DIV.SD):

Wm + 1:Wm  → W1:W0
W1:W0/Wn  → W0
Remainder  → W1

Status Affected:

N, OV, Z, C

Encoding:

| 1101 | 1000 | 0ttt | tvvv | vW00 | ssss |

Description:

Iterative, signed integer divide, where the dividend is stored in Wm (for a 16-bit by 16-bit divide) or Wm + 1:Wm (for a 32-bit by 16-bit divide) and the divisor is stored in Wn. In the default word operation, Wm is first copied to W0 and sign-extended through W1 to perform the operation. In the double operation, Wm + 1:Wm is first copied to W1:W0. The 16-bit quotient of the divide operation is stored in W0 and the 16-bit remainder is stored in W1.

This instruction must be executed 18 times using the REPEAT instruction (with an iteration count of 17) to generate the correct quotient and remainder. The N flag will be set if the remainder is negative and cleared otherwise. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is ‘0’ and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used.

The ‘t’ bits select the most significant word of the dividend for the double operation. These bits are clear for the word operation.

The ‘v’ bits select the least significant word of the dividend.

The ‘W’ bit selects the dividend size (‘0’ for 16-bit, ‘1’ for 32-bit).

Note 1: The ‘s’ bits select the Divisor register. The extension .D in the instruction denotes a double-word (32-bit) dividend rather than a word dividend. You may use a .W extension to denote a word operation, but it is not required.

2: Unexpected results will occur if the quotient can not be represented in 16 bits. When this occurs for the double operation (DIV.SD), the OV Status bit will be set and the quotient and remainder should not be used. For the word operation (DIV.S), only one type of overflow may occur (0x8000/0xFFFF = +32768 or 0x00008000), which allows the OV Status bit to interpret the result.

3: Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.

4: This instruction is interruptible on each instruction cycle boundary.

Words: 1

Cycles:

18 (plus 1 for REPEAT execution) for PIC24F, PIC24H, PIC24E, dsPIC30F, dsPIC33F, dsPIC33E
6 (plus 1 for REPEAT execution) for dsPIC33C
Example 1:

REPEAT #17 ; Execute DIV.S 18 times
DIV.S W3, W4 ; Divide W3 by W4
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 5555</td>
<td>W0 013B</td>
</tr>
<tr>
<td>W1 1234</td>
<td>W1 0003</td>
</tr>
<tr>
<td>W3 3000</td>
<td>W3 3000</td>
</tr>
<tr>
<td>W4 0027</td>
<td>W4 0027</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

REPEAT #17 ; Execute DIV.SD 18 times
DIV.SD W0, W12 ; Divide W1:W0 by W12
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 2500</td>
<td>W0 FA6B</td>
</tr>
<tr>
<td>W1 FF42</td>
<td>W1 EF00</td>
</tr>
<tr>
<td>W12 2200</td>
<td>W12 2200</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
### DIV.U

**Unsigned Integer Divide**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label;} DIV.U(W) Wm, Wn

DIV_UD Wm, Wn

**Operands:**

Wm ∈ [W0 ... W15] for word operation

Wm ∈ [W0, W2, W4 ... W14] for double operation

Wn ∈ [W2 ... W15]

**Operation:**

**For Word Operation (default):**

Wm → W0
0x0 → W1
W1:W0/Wn → W0
Remainder → W1

**For Double Operation (DIV.UD):**

Wm + 1:Wm → W1:W0
W1:W0/Wns → W0
Remainder → W1

**Status Affected:**

N, OV, Z, C

**Encoding:**

| 1101 | 1000 | lttt | tvvv | vW00 | ssss |

**Description:**

Iterative, unsigned integer divide, where the dividend is stored in Wm (for a 16-bit by 16-bit divide) or Wm + 1:Wm (for a 32-bit by 16-bit divide) and the divisor is stored in Wn. In the word operation, Wm is first copied to W0 and W1 is cleared to perform the divide. In the double operation, Wm + 1:Wm is first copied to W1:W0. The 16-bit quotient of the divide operation is stored in W0 and the 16-bit remainder is stored in W1.

This instruction must be executed 18 times using the REPEAT instruction (with an iteration count of 17) to generate the correct quotient and remainder. The N flag will always be cleared. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is '0' and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used.

The 't' bits select the most significant word of the dividend for the double operation. These bits are clear for the word operation.

The 'v' bits select the least significant word of the dividend.

The 'W' bit selects the dividend size ('0' for 16-bit, '1' for 32-bit).

The 's' bits select the Divisor register.

**Note 1:** The extension .D in the instruction denotes a double-word (32-bit) dividend rather than a word dividend. You may use a .W extension to denote a word operation, but it is not required.

**Note 2:** Unexpected results will occur if the quotient can not be represented in 16 bits. This may only occur for the double operation (DIV.UD). When an overflow occurs, the OV Status bit will be set, and the quotient and remainder should not be used.

**Note 3:** Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.

**Note 4:** This instruction is interruptible on each instruction cycle boundary.

**Words:** 1

**Cycles:**

18 (plus 1 for REPEAT execution) for PIC24F, PIC24H, PIC24E, dsPIC30F, dsPIC33F, dsPIC33E

6 (plus 1 for REPEAT execution) for dsPIC33C
DIVF Fractional Divide

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

\{label:\} DIVF Wm, Wn

Operands:

Wm ∈ [W0 ... W15]  
Wn ∈ [W2 ... W15]

Operation:

0x0 → W0  
Wm → W1  
W1:W0/Wn → W0  
Remainder → W1

Status Affected:  
N, OV, Z, C

Encoding:

| 1101 | 1001 | 0ttt | t000 | 0000 | ssss |

Description:

Iterative, signed fractional 16-bit by 16-bit divide, where the dividend is stored in Wm and the divisor is stored in Wn. To perform the operation, W0 is first cleared and Wm is copied to W1. The 16-bit quotient of the divide operation is stored in W0 and the 16-bit remainder is stored in W1. The sign of the remainder will be the same as the sign of the dividend.

This instruction must be executed 18 times using the REPEAT instruction (with an iteration count of 17) to generate the correct quotient and remainder. The N flag will be set if the remainder is negative and cleared otherwise. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is '0' and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used.

The ‘t’ bits select the Dividend register.  
The ‘s’ bits select the Divisor register.

**Note 1:**  
For the fractional divide to be effective, Wm must be less than Wn. If Wm is greater than or equal to Wn, unexpected results will occur because the fractional result will be greater than or equal to 1.0. When this occurs, the OV Status bit will be set, and the quotient and remainder should not be used.

**Note 2:**  
Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.

**Note 3:**  
This instruction is interruptible on each instruction cycle boundary.

Words:  
1

Cycles:  
18 (plus 1 for REPEAT execution) for dsPIC30F, dsPIC33F, dsPIC33E  
6 (plus 1 for REPEAT execution) for dsPIC33C
Example 1:
REPEAT #17 ; Execute DIVF 18 times
DIVF W8, W9 ; Divide W8 by W9
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 8000</td>
<td>W0 2000</td>
</tr>
<tr>
<td>W1 1234</td>
<td>W1 0000</td>
</tr>
<tr>
<td>W8 1000</td>
<td>W8 1000</td>
</tr>
<tr>
<td>W9 4000</td>
<td>W9 4000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002 (Z = 1)</td>
</tr>
</tbody>
</table>

Example 2:
REPEAT #17 ; Execute DIVF 18 times
DIVF W8, W9 ; Divide W8 by W9
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 8000</td>
<td>W0 F000</td>
</tr>
<tr>
<td>W1 1234</td>
<td>W1 0000</td>
</tr>
<tr>
<td>W8 1000</td>
<td>W8 1000</td>
</tr>
<tr>
<td>W9 8000</td>
<td>W9 8000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0002 (Z = 1)</td>
</tr>
</tbody>
</table>

Example 3:
REPEAT #17 ; Execute DIVF 18 times
DIVF W0, W1 ; Divide W0 by W1
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 8002</td>
<td>W0 7FFE</td>
</tr>
<tr>
<td>W1 8001</td>
<td>W1 8002</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
DIVF2
Signed Fractional Divide, 16/16 (W1:W0 Preserved)

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:  
\{label:\}  DIVF2   Wm, Wn

Operands:  
Wn ∈ [W2 ... W15]; Wm ∈ [W2 ... W15]

Operation:  
Wm = Dividend, Wn = Divisor:
0x0000 → W(m–1)
Wm:W(m–1)/Wn → W(m–1); Remainder → Wm

Status Affected:  
C, N, OV, Z

Encoding:  

```
1101 1001 0ttt t000 0010 ssss
```

Description:  
Iterative, signed fractional 16-bit by 16-bit divide, producing a 16-bit quotient and a 16-bit remainder. The sign of the remainder will be the same as that of the dividend.

This instruction must be executed 6 times to generate the correct quotient and remainder. This may only be achieved by executing a REPEAT with an iteration count of 5 (i.e., 5+1 iterations in all) and the DIVF instruction as its target.

The N flag will be set if the remainder is negative and cleared otherwise. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is ‘0’ and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used.

The ‘s’ bits select the address of the source (divisor) register.
The ‘t’ bits select the address of the source (dividend) register.

Note 1:  
For the fractional divide to be effective, Wm must be less than Wn. If Wm is greater than or equal to Wn, unexpected results will occur because the fractional result will be greater than or equal to 1.0. When this occurs, the OV Status bit will be set, and the quotient and remainder should not be used.

2:  
Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.

3:  
This instruction is interruptible on each instruction cycle boundary.

Words:  
1

Cycles:  
6 (plus 1 for REPEAT instruction execution)

Example 1:  
REPEAT #17 ; Execute DIV.U 18 times
DIV.U  W2, W4 ; Divide W2 by W4
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  5555</td>
<td>W0  0040</td>
</tr>
<tr>
<td>W1  1234</td>
<td>W1  0000</td>
</tr>
<tr>
<td>W2  8000</td>
<td>W2  8000</td>
</tr>
<tr>
<td>W4  0200</td>
<td>W4  0200</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0002 (Z = 1)</td>
</tr>
</tbody>
</table>
Example 2: 
REPEAT  #17       ; Execute DIV.UD 18 times
DIV.UD  W10, W12  ; Divide W11:W10 by W12
; Store quotient to W0, remainder to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 5555</td>
<td>W0 01F2</td>
</tr>
<tr>
<td>W1 1234</td>
<td>W1 0100</td>
</tr>
<tr>
<td>W10 2500</td>
<td>W10 2500</td>
</tr>
<tr>
<td>W11 0042</td>
<td>W11 0042</td>
</tr>
<tr>
<td>W12 2200</td>
<td>W12 2200</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
DIV2.S  Signed Integer Divide (W1:W0 Preserved)

Implemented in:  
<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{label:} DIV2.S[W]  Wm, Wn
DIV2.SD  Wm, Wn

Operands:

Wm ∈ [W0 ... W15] for word operation
Wm ∈ [W0, W2, W4 ... W14] for double operation
Wn ∈ [W2 ... W15]

Operation:

For Word Operation (default):
W(m+1):Wm/Wn → Wm; Remainder → W(m+1)

For Double Operation (DIV2.SD):
W(m+1):Wm/Wn → Wm; Remainder → W(m+1)

Status Affected:  C, N, OV, Z

Encoding:

| 1101 | 1000 | 0ttt | tvvv | v110 | ssss |

Description:

Iterative, signed integer 32-bit by 16-bit divide to a 16-bit quotient and a 16-bit remainder. The sign of the remainder will be the same as that of the dividend. Wm must be an even number and holds the least significant word of the dividend. The most significant word of the dividend is held in W(m+1).

This instruction must be executed 6 times to generate the correct quotient and remainder. This may only be achieved by executing a REPEAT with an iteration count of 5 (i.e., 5+1 iterations in all) and the DIV2.S instruction as its target.

The N flag will be set if the remainder is negative and cleared otherwise. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is '0' and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used.

The 's' bits select the address of the source (divisor) register.
The 't' bits select the address of the source (dividend, most significant word) register.
The 'v' bits select the address of the source (dividend, least significant word) register.

Note 1:  The extension .D in the instruction denotes a double-word (32-bit) dividend rather than a word dividend. You may use a .W extension to denote a word operation, but it is not required.

2:  Unexpected results will occur if the quotient cannot be represented in 16 bits. When this occurs for the double operation (DIV2.SD), the OV Status bit will be set, and the quotient and remainder should not be used. For the word operation (DIV2.S), only one type of overflow may occur (0x8000/0xFFFF = +32768 or 0x00008000), which allows the OV Status bit to interpret the result.

3:  Dividing by zero will initiate an arithmetic error trap during the first cycle of execution.

4:  This instruction is interruptible on each instruction cycle boundary.

Words:  1
Cycles:  6 (plus 1 for REPEAT instruction execution)
**DIV2.U**

**Unsigned Integer Divide (W1:W0 Preserved)**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>{label:} DIV2.U{W} Wm, Wn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIV2.UD Wm, Wn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operands:</td>
<td>Wm ∈ [W0 ... W15] for word operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wm ∈ [W0, W2, W4 ... W14] for double operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wn ∈ [W2 ... W15]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>W(m+1):Wm = Dividend, Wn = Divisor:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(m+1):Wm/Wn → Wm; Remainder → W(m+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For Word Operation (default):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 → W(m+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(m+1):Wm/Wn → Wm; Remainder → W(m+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For Double Operation (DIV2.SD):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(m+1):Wm/Wn → Wm; Remainder → W(m+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Affected:</td>
<td>C, N, OV, Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding:</td>
<td>1101</td>
<td>1000</td>
<td>lttt</td>
<td>tvvv</td>
<td>vl10</td>
<td>ssss</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Iterative, unsigned integer 16-bit by 16-bit or 32-bit by 16-bit divide, producing a 16-bit quotient and a 16-bit remainder. Wm must be an even number and holds the least significant word of the dividend. The most significant word of the dividend is held in W(m+1). This instruction must be executed 6 times to generate the correct quotient and remainder. This may only be achieved by executing a REPEAT with an iteration count of 5 (i.e., 5+1 iterations in all) and the DIV.UD instruction as its target. The N flag is always cleared. The OV flag will be set if the divide operation resulted in an overflow and cleared otherwise. The Z flag will be set if the remainder is '0' and cleared otherwise. The C flag is used to implement the divide algorithm and its final value should not be used. The 's' bits select the address of the source (divisor) register. The 't' bits select the address of the source (dividend, most significant word) register.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note 1:</td>
<td>The extension .D in the instruction denotes a double-word (32-bit) dividend rather than a word dividend. You may use a .W extension to denote a word operation, but it is not required.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Words:** 1

**Cycles:** 6 (plus 1 for REPEAT instruction execution)
DO

Initialize Hardware Loop Literal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

\{label;\} DO #lit14, Expr

Operands:

lit14 e [0 ... 16383]

Expr may be an absolute address, label or expression.

Expr is resolved by the linker to a Slit16, where Slit16 e [-32768 ... +32767].

Operation:

PUSH DO shadows (DCOUNT, DOEND, DOSTART)

(lit14) → DCOUNT

(PC) + 4 → PC

(PC) → DOSTART

(PC) + (2 * Slit16) → DOEND

Increment DL<2:0> (CORCON<10:8>)

Status Affected:

DA

Encoding:

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>1000</th>
<th>00kk</th>
<th>kkkk</th>
<th>kkkk</th>
<th>kkkk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000</td>
<td>0000</td>
<td>nnnn</td>
<td>nnnn</td>
<td>nnnn</td>
<td>nnnn</td>
</tr>
</tbody>
</table>

Description:

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<10:8>), is then incremented. After the DO loop completes execution, the PUSHed DCOUNT, DOSTART and DOEND registers are restored, and DL<2:0> are decremented.

The ‘k’ bits specify the loop count.

The ‘n’ bits are a signed literal that specifies the number of instructions that are offset from the PC to the last instruction executed in the loop.

Special Features, Restrictions:

The following features and restrictions apply to the DO instruction.

1. Using a loop count of 0 will result in the loop being executed one time.
2. Using a loop size of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.
3. The very last two instructions of the DO loop cannot be:
   • an instruction which changes program control flow
   • a DO or REPEAT instruction
   Unexpected results may occur if any of these instructions are used.
4. If a hard trap occurs in the second to last instruction or third to last instruction of a DO loop, the loop will not function properly. The hard trap includes exceptions of Priority Level 13 through Level 15, inclusive.

Note 1: The DO instruction is interruptible and supports 1 level of hardware nesting. Nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual for details.

2: The linker will convert the specified expression into the offset to be used.

Words: 2

Cycles: 2
### Example 1:

```c
002000 LOOP6: DO $5, END6 ; Initiate DO loop (6 reps)
002004 ADD W1, W2, W3 ; First instruction in loop
002006 ...
002008 ...
00200A END6: SUB W2, W3, W4 ; Last instruction in loop
00200C ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2004</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 0005</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 00 2004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 00 200A</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100</td>
</tr>
<tr>
<td>SR 0000 (C = 1)</td>
<td>SR 0201 (DA, C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

```c
01C000 LOOP12: DO $0x160, END12 ; Init DO loop (353 reps)
01C004 DEC W1, W2 ; First instruction in loop
01C006 ...
01C008 ...
01C00A ...
01C00C ...
01C00E CALL _FIR88 ; Call the FIR88 subroutine
01C012 NOP
01C014 END12: NOP ; Last instruction in loop ; (Required NOP filler)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 01 C000</td>
<td>PC 01 C004</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 0160</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 01 C004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 01 C014</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100</td>
</tr>
<tr>
<td>SR 0008 (N = 1)</td>
<td>SR 0208 (DA, N = 1)</td>
</tr>
</tbody>
</table>
DO Initialize Hardware Loop Literal

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} DO #lit15, Expr
```

**Operands:**

- `lit15` ∈ [0 ... 32767]
  - Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

**Operation:**

- PUSH DO shadows (DCOUNT, DOEND, DOSTART)
- (lit15) → DCOUNT
- (PC) + 4 → PC
- (PC) → DOSTART
- (PC) + (2 * Slit16) → DOEND
- Increment DL<2:0> (CORCON<10:8>)

**Status Affected:**

- DA

**Encoding:**

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>1000</th>
<th>0kkk</th>
<th>kkkk</th>
<th>kkkk</th>
<th>kkkk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>nnnn</td>
<td>nnnn</td>
<td>nnnn</td>
<td>nnnn</td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

Initiate a no overhead hardware `DO` loop, which is executed (`lit15 + 1`) times. The `DO` loop begins at the address following the `DO` instruction and ends at the address `2 * Slit16` instruction words away. The 15-bit count value (`lit15`) supports a maximum loop count value of 32768 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> bits (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> are decremented.

The 'k' bits specify the loop count.

The 'n' bits are a signed literal that specifies the number of instructions that are offset from the PC to the last instruction executed in the loop.

**Special Features, Restrictions:**

The following features and restrictions apply to the `DO` instruction.

1. Using a loop count of 0 will result in the loop being executed one time.
2. Using a loop size of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.
3. The very last two instructions of the `DO` loop cannot be:
   - an instruction which changes program control flow
   - a `DO` or `REPEAT` instruction
   Unexpected results may occur if any of these instructions are used.
4. If a hard trap occurs in the second to last instruction or third to last instruction of a `DO` loop, the loop will not function properly. The hard trap includes exceptions of Priority Level 13 through Level 15, inclusive.
5. The first and last instructions of the `DO` loop should not be a PSV read, table read or table write.

**Note 1:** The `DO` instruction is interruptible and supports 1 level of hardware nesting. Nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual for details.

**Note 2:** The linker will convert the specified expression into the offset to be used.

**Words:** 2

**Cycles:** 2
### Example 1:

```
002000 LOOP6: DO #5, END6 ; Initiate DO loop (6 reps)
002004 ADD W1, W2, W3 ; First instruction in loop
002006 ...
002008 ...
00200A END6: SUB W2, W3, W4 ; Last instruction in loop
00200C ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 002000</td>
<td>PC 002004</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 0005</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 0004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 0020A</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100 (DL = 1)</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0201 (DA, C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

```
01C000 LOOP12: DO #0x160, END12 ; Init DO loop (353 reps)
01C004 DEC W1, W2 ; First instruction in loop
01C006 ...
01C008 ...
01C00A ...
01C00C ...
01C00E CALL _FIR88 ; Call the FIR88 subroutine
01C012 NOP
01C014 END12: NOP ; Last instruction in loop
     ; (Required NOP filler)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 01C000</td>
<td>PC 01C004</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 0160</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 01C004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 01C014</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100 (DL = 1)</td>
</tr>
<tr>
<td>SR 0008 (N = 1)</td>
<td>SR 0208 (DA, N = 1)</td>
</tr>
</tbody>
</table>
DO Initialize Hardware Loop Wn

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wn</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax: `{label:} DO Wn, Expr

Operands: Wn ∈ [W0 ... W15]
Expr may be an absolute address, label or expression.
Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation:
PUSH Shadows (DCOUNT, DOEND, DOSTART)
(Wn<13:0>) → DCOUNT
(PC) + 4 → PC
(PC) → DOSTART
(PC) + (2 * Slit16) → DOEND
Increment DL<2:0> (CORCON<10:8>)

Status Affected: DA

Encoding:

```
0000 0000 1111 1111 1111 1111 ssss
```

Description:
Initiate a no overhead hardware DO loop, which is executed (Wn + 1) times. The DO loop begins at the address following the DO instruction and ends at the address 2 * Slit16 instruction words away. The lower 14 bits of Wn 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> are 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 that are offset from (PC + 4), which is the last instruction executed in the loop.

Special Features, Restrictions:
The following features and restrictions apply to the DO instruction.
1. Using a loop count of 0 will result in the loop being executed one time.
2. Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.
3. The very last two instructions of the DO loop cannot be:
   - an instruction which changes program control flow
   - a DO or REPEAT instruction
   Unexpected results may occur if these last instructions are used.

Note 1: The DO instruction is interruptible and supports 1 level of nesting. Nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual for details.

Note 2: The linker will convert the specified expression into the offset to be used.

Words: 2
Cycles: 2
### Example 1:

```
002000 LOOP6:   DO     W0, END6   ; Initiate DO loop (W0 reps)
002004          ADD    W1, W2, W3 ; First instruction in loop
002006          ...
002008          ...
00200A          ...
00200C          REPEAT #6
00200E          SUB    W2, W3, W4
002010 END6:    NOP               ; Last instruction in loop
                ; (Required NOP filler)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2004</td>
</tr>
<tr>
<td>W0 0012</td>
<td>W0 0012</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 0012</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 00 2004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 00 2010</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0080</td>
</tr>
</tbody>
</table>

### Example 2:

```
002000 LOOPA:   DO     W7, ENDA   ; Initiate DO loop (W7 reps)
002004          SWAP   W0         ; First instruction in loop
002006          ...
002008          ...
00200A          ...
002010 ENDA:    MOV    W1, [W2++] ; Last instruction in loop
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 2000</td>
<td>PC 00 2004</td>
</tr>
<tr>
<td>W7 E00F</td>
<td>W7 E00F</td>
</tr>
<tr>
<td>DCOUNT 0000</td>
<td>DCOUNT 200F</td>
</tr>
<tr>
<td>DOSTART FF FFFF</td>
<td>DOSTART 00 2004</td>
</tr>
<tr>
<td>DOEND FF FFFF</td>
<td>DOEND 00 2010</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0100</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0080</td>
</tr>
</tbody>
</table>
DO Initialize Hardware Loop Wn

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: \{label:} DO Wn, Expr

Operands: Wn ∈ [W0 ... W15]
Expr may be an absolute address, label or expression.
Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... +32767].

Operation: PUSH Shadows (DCOUNT, DOEND, DOSTART)
(Wn) → DCOUNT
(PC) + 4 → PC
(PC) → DOSTART
(PC) + (2 * Slit16) → DOEND
Increment DL<2:0> (CORCON<10:8>)

Status Affected: DA

Encoding:

| 0000 | 0000 | nnnn | nnnn | nnnn | nnnn | ssss |

Description: Initiate a no overhead hardware DO loop, which is executed (Wn + 1) times. The DO loop begins at the address following the DO instruction and ends at the address 2 * Slit16 instruction words away. The 16 bits of Wn support a maximum count value of 65536 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> are 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 that are offset from (PC + 4), which is the last instruction executed in the loop.

Special Features, Restrictions:
The following features and restrictions apply to the DO instruction.
1. Using a loop count of 0 will result in the loop being executed one time.
2. Using an offset of -2, -1 or 0 is invalid. Unexpected results may occur if these offsets are used.
3. The very last two instructions of the DO loop cannot be:
   - an instruction which changes program control flow
   - a DO or REPEAT instruction
   Unexpected results may occur if these last instructions are used.
4. The first and last instructions of the DO loop should not be a PSV read, table read or table write.

Note 1: The DO instruction is interruptible and supports 1 level of nesting. Nesting up to an additional 5 levels may be provided in software by the user. See the specific device family reference manual for details.

Note 2: The linker will convert the specified expression into the offset to be used.

Words: 2
Cycles: 2
Section 5. Instruction Descriptions

Example 1:
002000 LOOP6: DO W0, END6 ; Initiate DO loop (W0 reps)
002004 ADD W1, W2, W3 ; First instruction in loop
002006 ...
002008 ...
00200A ...
00200C REPEAT #6
00200E SUB W2, W3, W4
002010 END6: NOP ; Last instruction in loop
                   ; (Required NOP filler)

Before Instruction
PC  00 2000
W0  0012
DCOUNT 0000
DOSTART FF FFFF
DOEND FF FFFF
CORCON 0000
SR  0000

After Instruction
PC  00 2004
W0  0012
DCOUNT 0012
DOSTART 00 2004
DOEND 00 2010
CORCON 0100 (DL = 1)
SR  0080 (DA = 1)

Example 2:
002000 LOOPA: DO W7, ENDA ; Initiate DO loop (W7 reps)
002004 SWAP W0 ; First instruction in loop
002006 ...
002008 ...
00200A ...
002010 ENDA: MOV W1, [W2++] ; Last instruction in loop

Before Instruction
PC  00 2000
W7  E00F
DCOUNT 0000
DOSTART FF FFFF
DOEND FF FFFF
CORCON 0000
SR  0000

After Instruction
PC  00 2004
W7  E00F
DCOUNT 200F
DOSTART 00 2004
DOEND 00 2010
CORCON 0100 (DL = 1)
SR  0080 (DA = 1)
## ED

**Euclidean Distance (No Accumulate)**

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

```
{label:} ED Wm * Wm, Acc, [Wx], [Wy], Wxd

[Wx] += kx, [Wy] += ky,

[Wx] -= kx, [Wy] -= ky,

[W9 + W12], [W11 + W12],
```

### Operands:

- **Acc** ∈ [A,B]
- **Wm * Wm** ∈ [W4 * W4, W5 * W5, W6 * W6, W7 * W7]
- **Wx** ∈ [W8, W9]; **kx** ∈ [-6, -4, -2, 2, 4, 6]
- **Wy** ∈ [W10, W11]; **ky** ∈ [-6, -4, -2, 2, 4, 6]
- **Wxd** ∈ [W4 ... W7]

### Operation:

- **(Wm) * (Wm) → Acc(A or B)**
- **(Wx) − [Wy] → Wxd**
- **(Wx) * kx → Wx**
- **(Wy) + ky → Wy**

### Status Affected:

- OA, OB, OAB, SA, SB, SAB

### Encoding:

```
1111 00mm A1xx 00ii ijjj jj11
```

### Description:

Compute the square of Wm, and compute the difference of the prefetch values specified by [Wx] and [Wy]. The results of Wm * Wm are sign-extended to 40 bits and stored in the specified accumulator. The results of [Wx] – [Wy] are stored in Wxd, which may be the same as Wm.

Operands, Wx, Wxd and Wyd, specify the prefetch operations which support Indirect and Register Offset Addressing, as described in Section 4.15.1 “MAC Prefetches”.

The ‘m’ bits select the operand register Wm for the square.

The ‘A’ bit selects the accumulator for the result.

The ‘x’ bits select the prefetch difference Wxd destination.

The ‘i’ bits select the Wx prefetch operation.

The ‘j’ bits select the Wy prefetch operation.

### Words:

1

### Cycles:

1

### Example 1:

```
ED W4*W4, A, [W8]+2, [W10]=2, W4 ; Square W4 to ACCA
 ; [W8]−[W10] to W4
 ; Post-increment W8
 ; Post-decrement W10
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>009A</td>
</tr>
<tr>
<td>W8</td>
<td>1100</td>
</tr>
<tr>
<td>W10</td>
<td>2300</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 3D0A 0000</td>
</tr>
<tr>
<td>Data 1100</td>
<td>007F</td>
</tr>
<tr>
<td>Data 2300</td>
<td>0028</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### Example 2:

ED W5+W5, B, [W9]+2, [W11+W12], W5 ; Square W5 to ACCB  
; [W9]-[W11+W12] to W5  
; Post-increment W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>43C2</td>
</tr>
<tr>
<td>W9</td>
<td>1200</td>
</tr>
<tr>
<td>W11</td>
<td>2500</td>
</tr>
<tr>
<td>W12</td>
<td>0008</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 28E3 F14C</td>
</tr>
<tr>
<td>Data 1200</td>
<td>6A7C</td>
</tr>
<tr>
<td>Data 2508</td>
<td>2B3D</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
EDAC

Euclidean Distance

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

```
{label:} EDAC Wm * Wm, Acc, [Wx], [Wy], Wxd

[Wx] += kx, [Wy] += ky,
[Wx] -= kx, [Wy] -= ky,
[W9 + W12], [W11 + W12],
```

Operands:

- **Acc**: \( A, B \)
- **Wm**: \( W4 \times W4, W5 \times W5, W6 \times W6, W7 \times W7 \)
- **Wx**: \( W8, W9 \); \( kx \in [-6, -4, -2, 2, 4, 6] \)
- **Wy**: \( W10, W11 \); \( ky \in [-6, -4, -2, 2, 4, 6] \)
- **Wxd**: \( W4 \ldots W7 \)

Operation:

```
(\text{Acc}(A \text{ or } B)) + (Wm) \times (Wm) \rightarrow \text{Acc}(A \text{ or } B)
([Wx] - [Wy]) \rightarrow Wxd
(Wx) + kx \rightarrow Wx
(Wy) + ky \rightarrow Wy
```

Status Affected:

- OA, OB, OAB, SA, SB, SAB

Encoding:

```
1111 00mm A1xx 00ii ijjj j10
```

Description:

Compute the square of \( Wm \), and also the difference of the prefetch values specified by [Wx] and [Wy]. The results of \( Wm \times Wm \) are sign-extended to 40 bits and added to the specified accumulator. The results of [Wx] – [Wy] are stored in Wxd, which may be the same as Wm.

Operands, Wx, Wxd and Wyd, specify the prefetch operations which support Indirect and Register Offset Addressing, as described in Section 4.15.1 “MAC Prefetches”.

The ‘m’ bits select the operand register Wm for the square.
The ‘A’ bit selects the accumulator for the result.
The ‘x’ bits select the prefetch difference Wxd destination.
The ‘i’ bits select the Wx prefetch operation.
The ‘j’ bits select the Wy prefetch operation.

Words: 1
Cycles: 1
### Example 1:
EDAC W4*W4, A, [W8]++2, [W10]--2, W4

; Square W4 and
; add to ACCA
; [W8]-[W10] to W4
; Post-increment W8
; Post-decrement W10

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>009A</td>
</tr>
<tr>
<td>W8</td>
<td>1100</td>
</tr>
<tr>
<td>W10</td>
<td>2300</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 3D0A 3D0A</td>
</tr>
<tr>
<td>Data 1100</td>
<td>007F</td>
</tr>
<tr>
<td>Data 2300</td>
<td>0028</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:
EDAC W5*W5, B, [W9]+2, [W11+W12], W5

; Square W5 and
; add to ACCB
; [W9]-(W11+W12) to W5
; Post-increment W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>43C2</td>
</tr>
<tr>
<td>W9</td>
<td>1200</td>
</tr>
<tr>
<td>W11</td>
<td>2500</td>
</tr>
<tr>
<td>W12</td>
<td>0008</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 28E3 F14C</td>
</tr>
<tr>
<td>Data 1200</td>
<td>6A7C</td>
</tr>
<tr>
<td>Data 2508</td>
<td>2B3D</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
EXCH

Exchange Wns and Wnd

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label;} EXCH Wns, Wnd

Operands: Wns ∈ [W0 ... W15]
Wnd ∈ [W0 ... W15]

Operation: (Wns) ↔ (Wnd)

Status Affected: None

Encoding: 1111 1101 0000 0ddd d000 ssss

Description: Exchange the word contents of two Working registers. Register Direct Addressing must be used for Wns and Wnd.

The 'd' bits select the address of the first register.
The 's' bits select the address of the second register.

Note: This instruction only executes in Word mode.

Words: 1
Cycles: 1

Example 1: EXCH W1, W9 ; Exchange the contents of W1 and W9

Before Instruction

| W1 | 55FF |
| W9 | A3A3 |
| SR | 0000 |

After Instruction

| W1 | A3A3 |
| W9 | 55FF |
| SR | 0000 |

Example 2: EXCH W4, W5 ; Exchange the contents of W4 and W5

Before Instruction

| W4 | ABCD |
| W5 | 4321 |
| SR | 0000 |

After Instruction

| W4 | 4321 |
| W5 | ABCD |
| SR | 0000 |
### FBCL

**Find First Bit Change from Left**

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ws</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Wnd</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**
```
{label:} FBCL Ws, Wnd
[Ws],
[Ws++],
[Ws--],
[++Ws],
[--Ws],
```

**Operands:**
- Ws ∈ [W0 ... W15]
- Wnd ∈ [W0 ... W15]

**Operation:**
- `Max_Shift = 15`
- `Sign = (Ws) & 0x8000`
- `Temp = (Ws) << 1`
- `Shift = 0`
- `While ( (Shift < Max_Shift) && (Temp & 0x8000) == Sign) )`
  - `Temp = Temp << 1`
  - `Shift = Shift + 1`
  - `-Shift → (Wnd)`

**Status Affected:**
- C

**Encoding:**
```
1101 1111 0000 0ddd dppp ssss
```

**Description:**
Find the first occurrence of a one (for a positive value) or zero (for a negative value), starting from the Most Significant bit after the sign bit of Ws and working towards the Least Significant bit of the word operand. The bit number result is sign-extended to 16 bits and placed in Wnd.

The next Most Significant bit after the sign bit is allocated bit number 0 and the Least Significant bit is allocated bit number -14. This bit ordering allows for the immediate use of Wd with the SFTAC instruction for scaling values up. If a bit change is not found, a result of -15 is returned and the C flag is set. When a bit change is found, the C flag is cleared.

The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

**Note:** This instruction operates in Word mode only.

**Words:**
1

**Cycles:**
1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

© 2005-2018 Microchip Technology Inc.  DS70000157G-page 255
Example 1: FBCL W1, W9  ; Find 1st bit change from left in W1  
; and store result to W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1  55FF</td>
<td>W1  55FF</td>
</tr>
<tr>
<td>W9  FFFF</td>
<td>W9  0000</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>

Example 2: FBCL W1, W9  ; Find 1st bit change from left in W1  
; and store result to W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1  FFFF</td>
<td>W1  FFFF</td>
</tr>
<tr>
<td>W9  BBBB</td>
<td>W9  FFF1 (C = 1)</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0001</td>
</tr>
</tbody>
</table>

Example 3: FBCL [W1++], W9  ; Find 1st bit change from left in [W1]  
; and store result to W9  
; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1  2000</td>
<td>W1  2002</td>
</tr>
<tr>
<td>W9  BBBB</td>
<td>W9  FFF9</td>
</tr>
<tr>
<td>Data 2000  FF0A</td>
<td>Data 2000  FF0A</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

### FF1L

**Find First One from Left**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} FF1L Ws, Wnd

[Ws],
[Ws++],
[Ws--],
[++Ws],
[--Ws],
```

**Operands:**

- `Ws` ∈ [W0 ... W15]
- `Wnd` ∈ [W0 ... W15]

**Operation:**

- `Max_Shift = 17`
- `Temp = (Ws)`
- `Shift = 1`
- While `(Shift < Max_Shift) && !(Temp & 0x8000)`
  - `Temp = Temp << 1`
  - `Shift = Shift + 1`
- If `(Shift == Max_Shift)`
  - `0 → (Wnd)`
- Else
  - `Shift → (Wnd)`

**Status Affected:**

- C

**Encoding:**

| 1110 | 1111 | 1000 | 0ddd | dppp | ssss |

**Description:**

Finds the first occurrence of a one 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 result 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 destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

**Note:** This instruction operates in Word mode only.

**Words:**

- 1

**Cycles:**

- 1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”. 
Example 1:  
\text{FF1L W2, W5} ; \text{Find the 1st one from the left in W2}  
\text{; and store result to W5}  

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 000A</td>
<td>W2 000A</td>
</tr>
<tr>
<td>W5 BBBB</td>
<td>W5 000D</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:  
\text{FF1L [W2++], W5} ; \text{Find the 1st one from the left in [W2]}  
\text{; and store the result to W5}  
\text{; Post-increment W2}  

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 2000</td>
<td>W2 2002</td>
</tr>
<tr>
<td>W5 BBBB</td>
<td>W5 0000</td>
</tr>
<tr>
<td>Data 2000 0000</td>
<td>Data 2000 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
### FF1R
- **Find First One from Right**

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

{\textit{label:}} FF1R \ Ws, \ Wnd

\begin{itemize}
  \item \[Ws,\]
  \item \[Ws++\],
  \item \[Ws--\],
  \item \[++Ws\],
  \item \[--Ws\],
\end{itemize}

#### Operands:

- \(Ws \in [W0 \ldots W15]\)
- \(Wnd \in [W0 \ldots W15]\)

#### Operation:

1. \(\text{Max\_Shift} = 17\)
2. \(\text{Temp} = (Ws)\)
3. \(\text{Shift} = 1\)
4. While (\(\text{Shift} < \text{Max\_Shift}\) && !(\(\text{Temp} \& 0x1\) )
   - \(\text{Temp} = \text{Temp} >> 1\)
   - \(\text{Shift} = \text{Shift} + 1\)
5. If (\(\text{Shift} == \text{Max\_Shift}\))
   - \(0 \rightarrow (Wnd)\)
6. Else
   - \(\text{Shift} \rightarrow (Wnd)\)

#### Status Affected:

- \(C\)

#### Encoding:

<table>
<thead>
<tr>
<th></th>
<th>1100</th>
<th>1111</th>
<th>0000</th>
<th>0ddd</th>
<th>dppp</th>
<th>ssss</th>
</tr>
</thead>
</table>

#### Description:

Finds the first occurrence of a one 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 \(Wnd\).

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 found, the \(C\) flag is cleared.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note:** This instruction operates in Word mode only.

#### Words:

1

#### Cycles:

1\(^{(1)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.
Example 1:    FF1R  W1, W9 ; Find the 1st one from the right in W1
              ; and store the result to W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1  000A</td>
<td>W1  000A</td>
</tr>
<tr>
<td>W9  BBBB</td>
<td>W9  0002</td>
</tr>
<tr>
<td>SR   0000</td>
<td>SR   0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1  2000</td>
<td>W1  2002</td>
</tr>
<tr>
<td>W9  BBBB</td>
<td>W9  0010</td>
</tr>
<tr>
<td>Data 2000  8000</td>
<td>Data 2000  8000</td>
</tr>
<tr>
<td>SR   0000</td>
<td>SR   0000</td>
</tr>
</tbody>
</table>
**FLIM**

**Force (Signed) Data Range Limit**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
</table>

**Syntax:**

{label:} FLIM Wb, Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],

**Operands:**

Ws ∈ [W0 ... W15];
Wb ∈ [W0, W2, W4, W6, W8, W10, W12, W14];

**Operation:**

If (Ws) > (Wb)
Then
(Wb) → (Ws);
0 → Z; 0 → N; 0 → OV;

If (Ws) < (Wb+1)
Then
(Wb+1) → Ws;
0 → Z; 1 → N; 0 → OV;

Else
1 → Z; 0 → N; 0 → OV;

N, Z, OV

**Encoding:**

| 1110 | 0100 | 0www | w000 | 0ppp | ssss |

**Description:**

Simultaneously compare a 16-bit signed data value in Ws to a maximum signed limit value held in Wb and a minimum signed limit value held in W(b+1).

If Ws is greater than Wb, set Ws to the limit value held in Wb. The Z, N and OV Status bits are set such that a subsequent BRA GT instruction will take a branch.

If Ws is less than W(b+1), set Ws to the limit value held in W(b+1). The Z, N and OV Status bits are set such that a subsequent BRA LT instruction will take a branch.

If Ws is less than or equal to the maximum limit in Wb, and greater than or equal to the minimum limit in W(b+1), Ws is not modified (i.e., data is within range and limits are not applied). The Z Status bit is set such that a subsequent BRA Z instruction will take a branch.

The OV Status bit is always cleared by this instruction.

The ‘s’ bits select the address of the source (data value) register.
The ‘w’ bits select the address of the base (data limit) register.
The ‘p’ bits select the source addressing mode.

**Note 1:** Although the instruction assumes signed values for all operands, both upper and lower limit values may be of the same sign.

**Note 2:** The Status bits are set based upon the value loaded into Wnd.

**Note 3:** If the operand is greater than the maximum limit value in Wb, the CPU will write back the Wb value, regardless of whether the operand is less than the minimum value held in W(b+1) or not.

**Words:**

1

**Cycles:**

1
### FLIM.V

**Force (Signed) Data Range Limit with Limit Excess Result**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>{label:} FLIM.V Wb, Ws, Wnd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Ws],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Ws++],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Ws--],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[+Ws],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[−Ws],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operands:</td>
<td>Ws ∈ [W0 ... W15];</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wb ∈ [W0, W2, W4, W6, W8, W10, W12, W14];</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wnd ∈ [W0 ... W15]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>If (Ws) &gt; (Wb)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Then</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0x0001 → Wnd OR (Ws–Wb)) → Wnd;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Wb) → (Ws);</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 → Z; 0 → N; 0 → OV;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If (Ws) &lt; (Wb+1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Then</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0xFFFF → Wnd OR (Ws–W(b+1))) → Wnd;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(b+1) → Ws;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 → Z; 1 → N; 0 → OV;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Else</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0 → Wnd;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 → Z; 0 → N; 0 → OV;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N, Z, OV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding:</td>
<td>[1110 0101 xxx wddd dppp ssss]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Simultaneously compare a 16-bit signed data value in Ws to a maximum signed limit value held in Wb and a minimum signed limit value held in W(b+1). Write the limit excess value into Wnd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If Ws is greater than Wb, either write the (signed) value by which the limit is exceeded to Wnd (FLIM.V, where instruction bit x = 1) or set Wnd to +1 (FLIM, where instruction bit x = 0). In both cases, set Ws to the limit value held in Wb. Whenever Ws is greater than Wb, Wnd will always be a positive value. The Z, N and OV Status bits are set such that a subsequent BRA GT instruction will take a branch.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If Ws is less than W(b+1), either write the (signed) value by which the limit is exceeded to Wnd (FLIM.V, where instruction bit x = 1) or set Wnd to -1 (FLIM, where instruction bit x = 0). In both cases, set Ws to the limit value held in W(b+1). Whenever Ws is less than W(b+1), Wnd will always be a negative value. The Z, N and OV Status bits are set such that a subsequent BRA LT instruction will take a branch.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If Ws is less than or equal to the maximum limit in Wb, and greater than or equal to the minimum limit in W(b+1), Ws is not modified (i.e., data is within range and limits are not applied). Wnd is cleared and the Z Status bit is set such that a subsequent BRA Z instruction will take a branch.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The OV Status bit is always cleared by this instruction.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The ‘s’ bits select the address of the source (data value) register.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The ‘w’ bits select the address of the base (data limit) register.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The ‘d’ bits select the address of the destination (limit test result) register.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The ‘p’ bits select the source addressing mode.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The ‘x’ bit defines the presence and result format for Wnd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note 1:</td>
<td>Although the instruction assumes signed values for all operands, both upper and lower limit values may be of the same sign.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: The Status bits are set based upon the value loaded into Wnd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: If the operand is greater than the maximum limit value in Wb, the CPU will write back the Wb value, regardless of whether the operand is less than the minimum value held in W(b+1) or not.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Words:</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles:</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## GOTO

### Unconditional Jump

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

{label:} GOTO Expr

### Operands:

Expr may be label or expression (but not a literal).

Expr is resolved by the linker to a lit23, where lit23 ∈ [0 ... 8388606].

### Operation:

lit23 → PC

NOP → Instruction Register

### Status Affected:

None

### Encoding:

<table>
<thead>
<tr>
<th>1st word</th>
<th>2nd word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0100</td>
<td>0000</td>
</tr>
<tr>
<td>nnnn</td>
<td>0000</td>
</tr>
<tr>
<td>nnnn</td>
<td>0000</td>
</tr>
<tr>
<td>nnnn</td>
<td>0nnn</td>
</tr>
</tbody>
</table>

### Description:

Unconditional jump to anywhere within the 4M instruction word program memory range.

The PC is loaded with the 23-bit literal specified in the instruction. Since the PC must always reside on an even address boundary, lit23<0> is ignored.

The 'n' bits form the target address.

**Note:** The linker will resolve the specified expression into the lit23 to be used.

### Words:

2

### Cycles:

2 (PIC24F, PIC24H, dsPIC30F, dsPIC33F)

4 (PIC24E, dsPIC33E, dsPIC33C)

### Example 1:

```
026000 GOTO _THERE         ; Jump to _THERE
026004 MOV W0, W1
   ...
027844 _THERE: MOV #0x400, W2  ; Code execution
027846 ...

Before Instruction
PC  026000
SR  0000

After Instruction
PC  027844
SR  0000
```

### Example 2:

```
000100 _code: ...
   ...
026000 GOTO _code+2         ; Jump to _code+2
026004 ...

Before Instruction
PC  026000
SR  0000

After Instruction
PC  000102
SR  0000
```
GOTO

Unconditional Indirect Jump

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:          \{\text{label:}\} \text{GOTO} \ Wn
Operand:         Wn \in \{W0 \ldots W15\}
Operation:       0 \rightarrow \text{PC}<22:16>
                 (Wn<15:1>) \rightarrow \text{PC}<15:1>
                 0 \rightarrow \text{PC}<0>
                 \text{NOP} \rightarrow \text{Instruction Register}

Status Affected: None

Encoding:        0000 0001 0100 0000 0000 ssss

Description: Unconditional indirect jump within the first 32K words of program memory. Zero is loaded into \text{PC}<22:16> and the value specified in (Wn) is loaded into \text{PC}<15:1>. Since the PC must always reside on an even address boundary, Wn<0> is ignored.

The 's' bits select the source register.

Words:           1
Cycles:          2

Example 1:       006000 \ GOTO \ W4 \ ; \text{Jump unconditionally}
                 006002 \ MOV \ W0, W1 \ ; \text{to 16-bit value in W4}
                 . \ldots
                 . \ldots
                 007844 _THERE: \ MOV \ #0x400, W2 \ ; \text{Code execution}
                 007846 \ldots \ ; \text{resumes here}

\begin{tabular}{|c|c|}
\hline
\textbf{Before Instruction} & \textbf{After Instruction} \\
\hline
W4  & 7844 \\
PC  & 00 6000 \\
SR  & 0000 \\
\hline
\end{tabular}
### GOTO

#### Unconditional Indirect Jump

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} GOTO Wn

**Operands:**

Wn ∈ [W0 ... W15]

**Operation:**

0 → PC<22:16>

(Wn<15:1>) → PC<15:1>

0 → PC<0>

NOP → Instruction Register

**Status Affected:** None

**Encoding:**

0000 0001 0000 0100 0000 ssss

**Description:**

Unconditional indirect jump within the first 32K words of program memory. Zero is loaded into PC<22:16> and the value specified in (Wn) is loaded into PC<15:1>. Since the PC must always reside on an even address boundary, Wn<0> is ignored.

The 's' bits select the source register.

**Words:** 1

**Cycles:** 4

**Example 1:**

006000 GOTO W4 ; Jump unconditionally
006002 MOV W0, W1 ; to 16-bit value in W4
... ...
007844 THERE: MOV #0x400, W2 ; Code execution
007846 ... ; resumes here

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 7844</td>
<td>W4 7844</td>
</tr>
<tr>
<td>PC 00 6000</td>
<td>PC 00 7844</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
## GOTO.L

### Unconditional Indirect Jump Long

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

`{label:} GOTO.L Wn`

**Operands:**

Wn ∈ [W0, W2, W4, W6, W8, W10, W12]

**Operation:**

PC<23> → PC<23> (see text); (Wn+1)<6:0> → PC<22:16>; (Wn) → PC<15:0>

**Status Affected:**

None

**Encoding:**

```
0000 0001 1www w100 0000 ssss
```

**Description:**

Unconditional indirect jump to any user program memory address.

The Least Significant 7 bits of (Wn+1) are loaded in PC<22:16> and the 16-bit value (Wn) is loaded into PC<15:0>.

PC<23> is not modified by this instruction.

The contents of (Wn+1)<15:7> are ignored.

The value of Wn<0> is also ignored and PC<0> is always set to '0'.

**GOTO** is a two-cycle instruction.

The ‘s’ bits select the address of the Wn source register.

The ‘w’ bits specify the address of the Wn+1 source register.

**Words:**

1

**Cycles:**

4

**Example 1:**

```
026000    GOTO.L W4   ; Call _FIR subroutine
026004    MOV   W0, W1
  ...     ...
026844 _FIR:   MOV   #0x400, W2    ; _FIR subroutine start
026846 ...
```

**Before Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W4</th>
<th>W5</th>
<th>W15</th>
<th>Data A268</th>
<th>Data A26A</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>026000</td>
<td>6844</td>
<td>0002</td>
<td>A268</td>
<td>FFFF</td>
<td>FFFF</td>
<td>0000</td>
</tr>
</tbody>
</table>

**After Instruction**

<table>
<thead>
<tr>
<th>PC</th>
<th>W4</th>
<th>W5</th>
<th>W15</th>
<th>Data A268</th>
<th>Data A26A</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>026844</td>
<td>6844</td>
<td>0002</td>
<td>A26C</td>
<td>6004</td>
<td>0002</td>
<td>0000</td>
</tr>
</tbody>
</table>
INC

Increment f

- Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C
  - X X X X X X X

- Syntax: {label:} INC(B) f {,WREG}
- Operands: f ∈ [0 ... 8191]
- Operation: (f) + 1 → destination designated by D
- Status Affected: DC, N, OV, Z, C
- Encoding:
  - 1110 1100 0BDf ffff ffff ffff
- Description: 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 ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).
  - The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).
  - The ‘f’ bits select the address of the file register.

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.

Note 2: The WREG is set to Working register W0.

- Words: 1
- Cycles: 1(t)

Example 1: INC.B 0x1000 ; Increment 0x1000 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1000</td>
<td>8FFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>Data 1000</td>
<td>8F00</td>
</tr>
<tr>
<td>SR</td>
<td>0108 (DC, C = 1)</td>
</tr>
</tbody>
</table>

Example 2: INC 0x1000, WREG ; Increment 0x1000 and store to WREG ; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG</td>
<td>ABCD</td>
</tr>
<tr>
<td>Data 1000</td>
<td>8FFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>WREG</td>
<td>9000</td>
</tr>
<tr>
<td>Data 1000</td>
<td>8FFF</td>
</tr>
<tr>
<td>SR</td>
<td>0108 (DC, N = 1)</td>
</tr>
</tbody>
</table>
INC

Increment Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} INC.{B} Ws, Wd

Ws, [Ws], [Ws++], [Ws--], [Wd], [Wd++], [Wd--]

Operands:

Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:

(Ws) + 1 → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

1110 1000 0Bqq qddd dppp ssss

Description:

Add one to the contents of the source register Ws and place the result in the destination register Wd. Register Direct or Indirect Addressing 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 addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: 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

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

INC.B W1, [++W2] ; Pre-increment W2
; Increment W1 and store to W2
; (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>FF7F</td>
</tr>
<tr>
<td>W2</td>
<td>2000</td>
</tr>
<tr>
<td>Data</td>
<td>ABCD</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

INC W1, W2
; Increment W1 and store to W2
; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>FF7F</td>
</tr>
<tr>
<td>W2</td>
<td>2000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### INC2

**Increment f by 2**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} INC2.{B} {f} {,WREG}
```

**Operands:**

- `{f} \in \{0 ... 8191\}`

**Operation:**

- `(f) + 2 \rightarrow \text{destination designated by D}`

**Status Affected:**

- DC, N, OV, Z, C

**Encoding:**

```
1110 1100 1BDf ffff ffff ffff
```

**Description:**

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.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

**Note:** 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 (1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.

**Example 1:**

```
INC2.B 0x1000   ; Increment 0x1000 by 2
                ; (Byte mode)
```

**Before Instruction**

- Data 1000 = 08FF
- SR = 0000

**After Instruction**

- Data 1000 = 08F01
- SR = 0101 (DC, C = 1)

**Example 2:**

```
INC2 0x1000, WREG ; Increment 0x1000 by 2 and store to WREG
                   ; (Word mode)
```

**Before Instruction**

- WREG = ABCD
- Data 1000 = 08FF
- SR = 0000

**After Instruction**

- WREG = 9001
- Data 1000 = 08FF
- SR = 0108 (DC, N = 1)
INC2

Increment Ws by 2

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} INC2{.B} Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [++Wd]

[--Ws], [--Wd]

Operands:

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

Operation:

(Ws) + 2 → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

```
1110 1000 1Bqq qddd dppp ssss
```

Description:

Add two to the contents of the source register Ws and place the result in the destination register Wd. Register Direct or Indirect Addressing 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 addressing mode.

The 'd' bits select the destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

Note: 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(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

INC2.B W1, [++W2] ; Pre-increment W2

; Increment by 2 and store to W1

; (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>W2</th>
<th>Data 2000</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF7F</td>
<td>2000</td>
<td>ABCD</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>W2</th>
<th>Data 2000</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF7F</td>
<td>2001</td>
<td>81CD</td>
<td>010C</td>
</tr>
</tbody>
</table>

(1) DC, N, OV = 1

Example 2:

INC2 W1, W2 ; Increment W1 by 2 and store to W2

; (word mode)

Before Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>W2</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF7F</td>
<td>2000</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W1</th>
<th>W2</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF7F</td>
<td>FF81</td>
<td>0108</td>
</tr>
</tbody>
</table>

(1) DC, N = 1
Section 5. Instruction Descriptions

IOR
Inclusive OR f and WREG

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
\{label:} IOR{.B} f \{,WREG\}

Operands:
\( f \in [0 \ldots 8191] \)

Operation:
\( f \).IOR.(WREG) \rightarrow \text{destination designated by D} \)

Status Affected:
N, Z

Encoding:

```
1011 0111 0BDF \text{fff} \text{fff} \text{fff} \text{fff}
```

Description:
Compute the logical inclusive OR operation of the contents of the Working register WREG and 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 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1
Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:
```
IOR.B 0x1000       ; IOR WREG to (0x1000) (Byte mode)
            ; (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 1234</td>
<td>WREG 1234</td>
</tr>
<tr>
<td>Data 1000 FF00</td>
<td>Data 1000 FF34</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:
```
IOR 0x1000, WREG    ; IOR (0x1000) to WREG
            ; (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG 1234</td>
<td>WREG 1FBF</td>
</tr>
<tr>
<td>Data 1000 0FAB</td>
<td>Data 1000 0FAB</td>
</tr>
<tr>
<td>SR 0008 (N = 1)</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
**IOR**

**Inclusive OR Literal and Wn**

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wn</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} IOR{.B} #lit10, Wn

Operands:

- lit10 ∈ [0 ... 255] for byte operation
- lit10 ∈ [0 ... 1023] for word operation
- Wn ∈ [W0 ... W15]

Operation:

lit10.IOR.(Wn) → Wn

Status Affected:

N, Z

Encoding:

```
1011 0011 0Bkk kkkk kkkk dddd
```

Description:

Compute the logical inclusive OR operation of the 10-bit literal operand and the contents of the Working register Wn, 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.

**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.

**Note 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.

**Words:**

1

**Cycles:**

1

**Example 1:**

IOR.B #0xAA, W9 ; IOR 0xAA to W9

; (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W9: 1234</td>
<td>W9: 12BE</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0008          (N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

IOR #0x2AA, W4      ; IOR 0x2AA to W4

; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4: A34D</td>
<td>W4: A3EF</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0008          (N = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

IOR

Inclusive OR Wb and Short Literal

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wd</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} IOR{.B} Wb, #lit5, Wd

[Wd]
[Wd++]
[Wd--]
[++Wd]
[--Wd]

Operands:
Wb ∈ [W0 ... W15]
lit5 ∈ [0 ... 31]
Wd ∈ [W0 ... W15]

Operation: (Wb).IOR.lit5 → Wd

Status Affected: N, Z

Encoding:

| 0111 | 0www | wBqq | qddd | d11k | kkkk |

Description: Compute the logical inclusive OR operation of the contents of the base register Wb and the 5-bit literal operand, and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or Indirect Addressing may be used for 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘k’ bits provide the literal operand, a five-bit integer number.

Note: 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

Example 1:

IOR.B W1, #0x5, [W9++] ; IOR W1 and 0x5 (Byte mode)
; Store to [W9]
; Post-increment W9

Before Instruction

| W1  | AAAA |
| W9  | 2000 |

Data 2000

SR 0000

After Instruction

| W1  | AAAA |
| W9  | 2001 |

Data 2000

SR 0008 (N = 1)

Example 2:

IOR W1, #0x0, W9 ; IOR W1 with 0x0 (Word mode)
; Store to W9

Before Instruction

| W1  | 0000 |
| W9  | A34D |
| SR  | 0000 |

After Instruction

| W1  | 0000 |
| W9  | 0000 |
| SR  | 0002 (Z = 1) |
IOR

Inclusive OR Wb and Ws

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: 

\{label\} IOR{.B} Wb, Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [+Wd]

[--Ws], [--Wd]

Operands:

Wb ∈ [W0 ... W15]

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

Operation:

(Wb).IOR.(Ws) → Wd

Status Affected:

N, Z

Encoding:

0111 0www wBqq qddd dppp ssss

Description:

Compute the logical inclusive OR operation of the contents of the source register Ws and 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 or 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: 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

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

**IOR.B W1, [W5++], [W9++]**  
; IOR W1 and [W5] (Byte mode)  
; Store result to [W9]  
; Post-increment W5 and W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>AAAA</td>
</tr>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>W9</td>
<td>2400</td>
</tr>
<tr>
<td>Data 2000</td>
<td>1155</td>
</tr>
<tr>
<td>Data 2400</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:

**IOR W1, W5, W9**  
; IOR W1 and W5 (Word mode)  
; Store the result to W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>AAAA</td>
</tr>
<tr>
<td>W5</td>
<td>5555</td>
</tr>
<tr>
<td>W9</td>
<td>A34D</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
LAC

Load Accumulator

Implemented in:

\[
\begin{array}{cccccccc}
\text{PIC24F} & \text{PIC24H} & \text{PIC24E} & \text{dsPIC30F} & \text{dsPIC33F} & \text{dsPIC33E} & \text{dsPIC33C} \\
\text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\
\end{array}
\]

Syntax:

\[
\{\text{label:}\} \text{ LAC } \text{Ws}, \{\#\text{Slit4,}\} \text{ Acc}
\]

[Ws],
[Ws++],
[Ws--],
[--Ws],
[++Ws],
[Ws+Wb],

Operands:

Ws \in [W0 ... W15]
Wb \in [W0 ... W15]
Slit4 \in [-8 ... +7]
Acc \in \{A,B\}

Operation:

\text{Shift_{slit4}(Extend(Ws))} \rightarrow \text{Acc(A or B)}

Status Affected:

OA, OB, OAB, SA, SB, SAB

Encoding:

\[
\begin{array}{cccccccc}
1100 & 1010 & Awww & wrrr & rggg & ssss \\
\end{array}
\]

Description:

Read the contents of the source register. Optionally perform a signed 4-bit shift and store
the result in the specified accumulator. The shift range is -8:7, where a negative operand
indicates an arithmetic left shift and a positive operand indicates an arithmetic right shift.
The data stored in the source register is assumed to be 1.15 fractional data, and is
automatically sign-extended (through bit 39) and zero-backfilled (bits<15:0>) prior to
shifting.

The ‘A’ bit specifies the destination accumulator.
The ‘w’ bits specify the offset register Wb.
The ‘r’ bits encode the accumulator preshift.
The ‘g’ bits select the source addressing mode.
The ‘s’ bits specify the source register Ws.

\textbf{Note:} If the operation moves more than sign-extension data into the Accumulator Upper
register (ACCxU), or causes a saturation, the appropriate overflow and saturation
bits will be set.

Words: 1
Cycles: 1

\textbf{Note 1:} In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and
Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see \textbf{Note 3} in
\textit{Section 3.2.1 "Multicycle Instructions"}. 
### Example 1:

**LAC [W4++], #-3, B**  
; Load ACCB with [W4] << 3  
; Contents of [W4] do not change  
; Post increment W4  
; Assume saturation disabled  
; (SATB = 0)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>2000</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 5125 ABCD</td>
</tr>
<tr>
<td>Data 2000</td>
<td>1221</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

| W4  | 2002 |
| ACCB  | FF 9108 0000 |
| Data 2000  | 1221 |
| SR  | 4800 |

(Ob, OAB = 1)

### Example 2:

**LAC [--W2], #7, A**  
; Pre-decrement W2  
; Load ACCA with [W2] >> 7  
; Contents of [W2] do not change  
; Assume saturation disabled  
; (SATA = 0)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>4002</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 5125 ABCD</td>
</tr>
<tr>
<td>Data 4000</td>
<td>9108</td>
</tr>
<tr>
<td>Data 4002</td>
<td>1221</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

| W2  | 4000 |
| ACCA  | FF FF22 1000 |
| Data 4000  | 9108 |
| Data 4002  | 1221 |
| SR  | 0000 |
## LAC.D

**Load Accumulator Double**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
</table>

### Syntax:

```
{label:} LAC.D Ws, [, #Slit4], Acc
```

### Operands:

- **Register Direct**: `Ws` ∈ [W0, W2, W4, W6, W8, W10, W12, W14];
- **Register Indirect**: `Ws` ∈ [W0 ... W15];
- **Slit4**: ∈ [-8 ... +7]
- **Acc**: ∈ [A,B]

### Operation:

\( \text{Shift}_{\text{Slit4}}(\text{Extend}(Ws)) \rightarrow \text{ACC (A,B)} \)

### Status Affected:

- OA, SA or OB, SB

### Encoding:

```
1101 1011 A000 0rrr rppp ssss
```

### Description:

Read the contents of the source register. Optionally perform a signed 4-bit shift and store the result in the specified accumulator. The shift range is -8:7, where a negative operand indicates an arithmetic left shift and a positive operand indicates an arithmetic right shift. The data stored in the source register is assumed to be 1.31 fractional data, and is automatically sign-extended (through bit 39) and zero-backfilled (bits<15:0>) prior to shifting.

The 'A' bit specifies the destination accumulator.

- The 's' bits specify the source register `Ws`.
- The 'p' bits select the source addressing mode.
- The 'r' bits encode the optional operand `Slit4`, which determines the amount of the accumulator preshift; if the operand `Slit4` is absent, a '0' is encoded.

See Table 5-7 for modifier addressing information.

### Note 1:

Unlike the LAC instruction, the LAC.D instruction does not support Indirect with Register Offset Addressing mode.

- **Positive values of operand Slit4 represent arithmetic shift right. Negative values of operand Slit4 represent shift left.**
- **The LAC.D instruction cannot be executed within a REPEAT loop.**

### Words:

1

### Cycles:

2

### Note 2:

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### LDSLV

**Load Slave Processor Program RAM**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} LDSLV [Wns] [Wnd++] #lit2

**Operands:**

Wns ∈ [W0 ... W15];
Wnd ∈ [W0 ... W15];
lit2 ∈ [0 ... 3]

**Operation:**

Master (EAs) → Slave EAd

**Status Affected:**

None

**Encoding:**

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>0011</th>
<th>00kk</th>
<th>0ddd</th>
<th>d0p1</th>
<th>ssss</th>
</tr>
</thead>
</table>

**Description:**

This instruction moves a single instruction word from the target Slave PRAM image (held in the Master program space Flash) into the Slave PRAM. The source address must be located within PSV address space (i.e., DSRPAG ≥ 0x200). The destination address uses DSWPAG and the destination EA to create a 24-bit Slave PS PRAM write address.

Starting with an aligned double instruction word destination address (see note), move the contents of the source Effective Address (in Master program space) to the destination Effective Address (in the Slave PRAM address space).

If the (single instruction word) destination address is even, capture the data in the Slave PRAM wrapper. If the (single instruction word) destination address is odd, the ECC parity bits are calculated from the current and captured source data (48-bits), then stored together with the data into the PRAM double instruction word destination Effective Address.

The target Slave processor is selected by the value defined by lit2.

The instruction may be regarded as a PSV operation, and hence, may be executed within a REPEAT loop to accelerate data processing.

The ‘s’ bits select the address of the source register.

The ‘d’ bits select the address of the destination register.

The ‘k’ bits select the target Slave processor.

The ‘p’ bit selects the destination addressing mode (see note).

**Note 1:** This instruction supports a subset of addressing modes. The Source Addressing mode bit field is constrained to 2 options and the Destination Addressing mode bit field is not required.

**Note 2:** An aligned double instruction word destination address is an even address that addresses the least significant word of a double instruction word.

**Note 3:** This instruction only supports Word mode.

**Words:**

1

**Cycles:**

1
**LNK**

Allocate Stack Frame

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

{label:} LNK #lit14

### Operands:

lit14 ∈ [0 ... 16382]

### Operation:

(W14) → (TOS)

(W15) + 2 → W15

(W15) → W14

(W15) + lit14 → W15

### Status Affected:

None

### Encoding:

```
1111 1010 00kk kkkk kkkk kkk0
```

### Description:

This instruction allocates a stack frame of size lit14 bytes for a subroutine calling sequence. The stack frame is allocated by PUSHing the contents of the Frame Pointer (W14) onto the stack, storing the updated Stack Pointer (W15) to the Frame Pointer and then incrementing the Stack Pointer by the unsigned 14-bit literal operand. This instruction supports a maximum stack frame of 16382 bytes.

The ‘k’ bits specify the size of the stack frame.

**Note:** Since the Stack Pointer can only reside on a word boundary, lit14 must be even.

### Example 1:

```
LNK #0xA0 ; Allocate a stack frame of 160 bytes
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W14 2000</td>
<td>W14 2002</td>
</tr>
<tr>
<td>W15 2000</td>
<td>W15 20A2</td>
</tr>
<tr>
<td>Data 2000 0000</td>
<td>Data 2000 2000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### LNK Allocate Stack Frame

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} LNK #lit14

**Operands:**

lit14 ∈ [0 ... 16382]

**Operation:**

(W14) → (TOS)
(W15) + 2 → W15
(W15) → W14
1 → SFA Status bit
(W15) + lit14 → W15

**Status Affected:** SFA

**Encoding:**

```
0000 1010 00kk kkkk kkkk kkk0
```

**Description:**

This instruction allocates a stack frame of size lit14 bytes for a subroutine calling sequence. The stack frame is allocated by PUSHing the contents of the Frame Pointer (W14) onto the stack, storing the updated Stack Pointer (W15) to the Frame Pointer and then incrementing the Stack Pointer by the unsigned 14-bit literal operand. This instruction supports a maximum stack frame of 16382 bytes.

The ‘k’ bits specify the size of the stack frame.

**Note:** Since the Stack Pointer can only reside on a word boundary, lit14 must be even.

**Words:** 1

**Cycles:** 1

**Example 1:**

LNK #0xA0 ; Allocate a stack frame of 160 bytes

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W14 2000</td>
<td>W14 2002</td>
</tr>
<tr>
<td>W15 2000</td>
<td>W15 20A2</td>
</tr>
<tr>
<td>Data 2000</td>
<td>Data 2000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0004</td>
</tr>
</tbody>
</table>
LSR
Logical Shift Right f

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:
{label:} LSR{.B} f {,WREG}

Operands:
f ∈ [0 ... 8191]

Operation:
For Byte Operation:
0 → Dest<7>
(f<7:1>) → Dest<6:0>
(f<0>) → C

For Word Operation:
0 → Dest<15>
(f<15:1>) → Dest<14:0>
(f<0>) → C

Status Affected: N, Z, C

Encoding:

<table>
<thead>
<tr>
<th></th>
<th>1101</th>
<th>0101</th>
<th>0BDF</th>
<th>ffff</th>
<th>ffff</th>
<th>ffff</th>
</tr>
</thead>
</table>

Description:
Shift the contents of the file register one bit to the right and place the result in the destination register. The Least Significant bit of the file register is shifted into the Carry bit of the STATUS Register. Zero is shifted into the Most Significant bit of 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 register (‘0’ for WREG, ‘1’ for file register).
The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1
Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Note 2: The WREG is set to Working register W0.

Note 3: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

**Instruction:** LSR.B 0x600

; Logically shift right (0x600) by one
; (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 600 55FF</td>
<td>Data 600 557F</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

**Instruction:** LSR 0x600, WREG

; Logically shift right (0x600) by one
; Store to WREG
; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 600 55FF</td>
<td>Data 600 55FF</td>
</tr>
<tr>
<td>WREG 0000</td>
<td>WREG 2AFF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>
LSR Logical Shift Right Ws

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} LSR{.B} Ws, Wd

[Ws], [Wd]
[Ws++], [Wd++]
[Ws--], [Wd--]
[++Ws], [++Wd]
[--Ws], [--Wd]

Operands: Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:
For Byte Operation:
0 → Wd<7>
(Ws<7:1>) → Wd<6:0>
(Ws<0>) → C

For Word Operation:
0 → Wd<15>
(Ws<15:1>) → Wd<14:0>
(Ws<0>) → C

Status Affected: N, Z, C

Encoding:
| 1101 | 0001 | 0Bqq | qddd | dppp | ssss |

Description:
Shift 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 shifted into the Carry bit of the STATUS Register. Zero is shifted into the Most Significant bit of Wd. Either Register Direct or Indirect Addressing 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 addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

Note: 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

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Words: 1
Cycles: 1
Example 1:  LSR.B W0, W1  ; LSR W0 (Byte mode)
            ; Store result to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 FF03</td>
<td>W0 FF03</td>
</tr>
<tr>
<td>W1 2378</td>
<td>W1 2301</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0001           (C = 1)</td>
</tr>
</tbody>
</table>

Example 2:  LSR W0, W1  ; LSR W0 (Word mode)
            ; Store the result to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 8000</td>
<td>W0 8000</td>
</tr>
<tr>
<td>W1 2378</td>
<td>W1 4000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
LSR

Logical Shift Right by Short Literal

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} LSR Wb, #lit4, Wnd

Operands:

Wb ∈ [W0 ... W15]

lit4 ∈ [0 ... 15]

Wnd ∈ [W0 ... W15]

Operation:

lit4<3:0> → Shift_Val

0 → Wnd<15:15-Shift_Val + 1>

Wb<15:Shift_Val> → Wnd<15-Shift_Val:0>

Status Affected:

N, Z

Encoding:

1101 1110 0www wddd d100 kkkk

Description:

Logical shift right the contents of the source register Wb by the 4-bit unsigned literal and store the result in the destination register Wnd. Direct Addressing must be used for Wb and Wnd.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the destination register.

The ‘k’ bits provide the literal operand.

Note: This instruction operates in Word mode only.

Words: 1

Cycles: 1

Example 1:

LSR W4, #14, W5 ; LSR W4 by 14

; Store result to W5

Before Instruction | After Instruction
---|---
W4 800 | W4 800
W5 1200 | W5 0003
SR 0000 | SR 0000

Example 2:

LSR W4, #1, W5 ; LSR W4 by 1

; Store result to W5

Before Instruction | After Instruction
---|---
W4 0505 | W4 0505
W5 F000 | W5 0282
SR 0000 | SR 0000
LSR

Logical Shift Right by Wns

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} LSR Wb, Wns, Wnd

Operands:

Wb ∈ [W0 ... W15]
Wns ∈ [W0 ... W15]
Wnd ∈ [W0 ... W15]

Operation:

Wns<4:0> → Shift_Val
0 → Wnd<15:15-Shift_Val + 1>
Wb<15:Shift_Val> → Wnd<15 – Shift_Val:0>

Status Affected:

N, Z

Encoding:

| 1101 | 1110 | 0www | wddd | d000 | ssss |

Description:

Logical shift right the contents of the source register Wb by the 5 Least Significant bits of Wns (only up to 15 positions) and store the result in the destination register Wnd. Direct Addressing must be used for Wb and Wnd.

The ‘w’ bits select the address of the base register.
The ‘d’ bits select the destination register.
The ‘s’ bits select the source register.

Note 1: This instruction operates in Word mode only.

2: If Wns is greater than 15, Wnd will be loaded with 0x0.

Words: 1

Cycles: 1

Example 1:

```assembly
LSR W0, W1, W2     ; LSR W0 by W1
                  ; Store result to W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0: C00C</td>
<td>W0: C00C</td>
</tr>
<tr>
<td>W1: 0001</td>
<td>W1: 0001</td>
</tr>
<tr>
<td>W2: 2390</td>
<td>W2: 0006</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>

Example 2:

```assembly
LSR W5, W4, W3     ; LSR W5 by W4
                  ; Store result to W3
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3: DD43</td>
<td>W3: 0000</td>
</tr>
<tr>
<td>W4: 000C</td>
<td>W4: 000C</td>
</tr>
<tr>
<td>W5: 0800</td>
<td>W5: 0800</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0002          (Z = 1)</td>
</tr>
</tbody>
</table>
**MAC Multiply and Accumulate**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MAC

Wm*Wn, Acc

{,[Wx], Wxd}  {,[Wy], Wyd}  {,AWB}

{,[Wx] += kx, Wxd}  {,[Wy] += ky, Wyd}

{,[Wx] -= kx, Wxd}  {,[Wy] -= ky, Wyd}

{,[W9 + W12], Wxd}  {,[W11 + W12], Wyd}

**Operands:**


Acc ∈ [A,B]

Wx ∈ [W8, W9]; kx ∈ [-6, -4, -2, 2, 4, 6]; Wxd ∈ [W4 ... W7]

Wy ∈ [W10, W11]; ky ∈ [-6, -4, -2, 2, 4, 6]; Wyd ∈ [W4 ... W7]

AWB ∈ [W13, [W13] += 2]

**Operation:**

(Acc(A or B)) + (Wm) * (Wn) → Acc(A or B)

([Wx]) → Wxd; (Wx) + kx → Wx

([Wy]) → Wyd; (Wy) + ky → Wy

(Acc(B or A)) rounded → AWB

**Status Affected:**

OA, OB, OAB, SA, SB, SAB

**Encoding:**

| 1100 | 0mmm | A0xx | yyii | iijj | jjaa |

**Description:** Multiply the contents of two Working registers. Optionally prefetch operands in preparation for another MAC type instruction and optionally store the unspecified accumulator results. The 32-bit result of the signed multiply is sign-extended to 40 bits and added to the specified accumulator.

Operands, Wx, Wxd, Wy and Wyd, specify optional prefetch operations, which support Indirect and Register Offset Addressing, as described in Section 4.14.1 “MAC Prefetches”. Operand AWB specifies the optional store of the “other” accumulator, as described in Section 4.15.4 “MAC Write-Back”.

The ‘m’ bits select the operand registers Wm and Wn for the multiply.

The ‘A’ bit selects the accumulator for the result.

The ‘x’ bits select the prefetch Wxd destination.

The ‘y’ bits select the prefetch Wyd destination.

The ‘i’ bits select the Wx prefetch operation.

The ‘j’ bits select the Wy prefetch operation.

The ‘a’ bits select the accumulator Write-Back destination.

**Note 1:** The IF bit (CORCON<0>) determines if the multiply is fractional or an integer.

**Note 2:** The US<1:0> bits (CORCON<13:12> in dsPIC33E/dsPIC33C, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed or mixed-sign. Only dsPIC33E/dsPIC33C devices support mixed-sign multiplication.

**Words:**

1

**Cycles:**

1
; 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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>A022</td>
</tr>
<tr>
<td>W5</td>
<td>B900</td>
</tr>
<tr>
<td>W8</td>
<td>0A00</td>
</tr>
<tr>
<td>W10</td>
<td>1800</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 1200 0000</td>
</tr>
<tr>
<td>Data 0A00</td>
<td>2567</td>
</tr>
<tr>
<td>Data 1800</td>
<td>909C</td>
</tr>
<tr>
<td>CORCON</td>
<td>00C0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

; 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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>1000</td>
</tr>
<tr>
<td>W5</td>
<td>3000</td>
</tr>
<tr>
<td>W8</td>
<td>0A00</td>
</tr>
<tr>
<td>W10</td>
<td>1800</td>
</tr>
<tr>
<td>W13</td>
<td>2000</td>
</tr>
<tr>
<td>ACCA</td>
<td>23 5000 2000</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 0000 8F4C</td>
</tr>
<tr>
<td>Data 0A00</td>
<td>5BBE</td>
</tr>
<tr>
<td>Data 1800</td>
<td>C967</td>
</tr>
<tr>
<td>CORCON</td>
<td>00D0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
MAC Square and Accumulate

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: 

{label:} MAC Wm * Wm, Acc {,[Wx], Wxd} {,[Wy], Wyd} 
{,[Wx] + = kx, Wxd} {,[Wy] + = ky, Wyd} 
{,[Wx] – = kx, Wxd} {,[Wy] – = ky, Wyd} 
{,[W9 + W12], Wxd} {,[W11 + W12], Wyd} 

Operands: 

Wm * Wm ∈ [W4 * W4, W5 * W5, W6 * W6, W7 * W7] 
Acc ∈ [A, B] 
Wx ∈ [W8, W9]; kx ∈ [-6, -4, -2, 2, 4, 6]; Wxd ∈ [W4 ... W7] 
Wy ∈ [W10, W11]; ky ∈ [-6, -4, -2, 2, 4, 6]; Wyd ∈ [W4 ... W7] 

Operation: 

(Acc(A or B)) + (Wm) * (Wm) → Acc(A or B) 
([Wx]) → Wxd; (Wx) + kx → Wx 
([Wy]) → Wyd; (Wy) + ky → Wy 

Status Affected: 

OA, OB, OAB, SA, SB, SAB 

Encoding: 

| 1111 | 00 | A0 | yy | ii | jj | 00 |

Description: 

Square the contents of a Working register. Optionally prefetch operands in preparation for another MAC type instruction. The 32-bit result of the signed multiply is sign-extended to 40 bits and added to the specified accumulator. 

Operands, Wx, Wxd, Wy and Wyd, specify optional prefetch operations, which support Indirect and Register Offset Addressing, as described in Section 4.14.1 “MAC Prefetches”. 

The ‘m’ bits select the operand register Wm for the square. 
The ‘A’ bit selects the accumulator for the result. 
The ‘x’ bits select the prefetch Wxd destination. 
The ‘y’ bits select the prefetch Wyd destination. 
The ‘i’ bits select the Wx prefetch operation. 
The ‘j’ bits select the Wy prefetch operation. 

Note 1: The IF bit (CORCON<0>) determines if the multiply is fractional or an integer. 
Note 2: The US<1:0> bits (CORCON<13:12> in dsPIC33E/dsPIC33C, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed or mixed-sign. Only dsPIC33E/dsPIC33C devices support mixed-sign multiplication.

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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>A022</td>
</tr>
<tr>
<td>W5</td>
<td>650B</td>
</tr>
<tr>
<td>W9</td>
<td>0C00</td>
</tr>
<tr>
<td>W10</td>
<td>18FE</td>
</tr>
<tr>
<td>W12</td>
<td>0020</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 2000 0000</td>
</tr>
<tr>
<td>Data 0C20</td>
<td>A230</td>
</tr>
<tr>
<td>Data 1900</td>
<td>650B</td>
</tr>
<tr>
<td>CORCON</td>
<td>00C0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

---

**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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7</td>
<td>76AE</td>
</tr>
<tr>
<td>W11</td>
<td>23FF</td>
</tr>
<tr>
<td>ACCA</td>
<td>2000</td>
</tr>
<tr>
<td>Data 2000</td>
<td>FF 063E 01B8</td>
</tr>
<tr>
<td>CORCON</td>
<td>00D0</td>
</tr>
<tr>
<td>SR</td>
<td>8800 (OA, OAB = 1)</td>
</tr>
</tbody>
</table>
## MAX

### Accumulator Force Maximum Data Range Limit

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
<th>X</th>
</tr>
</thead>
</table>

**Syntax:**

\[
\text{\{label:} \quad \text{MAX} \quad \text{Acc}
\]

**Operands:**

\[\text{Acc} \in [A,B]\]

**Operation:**

- If (MAX A) Then
  - If ACCA – ACCB > 0 Then
    - \((ACCB \rightarrow ACCA;\)
    - \(0 \rightarrow Z; 0 \rightarrow N; 0 \rightarrow OV;\)
  - Else
    - \((1 \rightarrow Z; 0 \rightarrow N; 0 \rightarrow OV;\)
- If (MAX B) Then
  - If ACCB – ACCA > 0 Then
    - \((ACCA \rightarrow ACCB;\)
    - \(0 \rightarrow Z; 0 \rightarrow N; 0 \rightarrow OV;\)
  - Else
    - \((1 \rightarrow Z; 0 \rightarrow N; 0 \rightarrow OV;\)

**Status Affected:**

\[N, \ OV, \ Z\]

**Encoding:**

<table>
<thead>
<tr>
<th>1100</th>
<th>1110</th>
<th>A00x</th>
<th>x000</th>
<th>0000</th>
<th>0000</th>
</tr>
</thead>
</table>

**Description:**

The target accumulator (defined in the instruction) is clamped to the maximum limit value previously loaded into the other accumulator (sign-extended 32-bit value). The comparison examines the full 40-bit value of the target accumulator, and will therefore, clamp an overflowed accumulator.

If the target accumulator is greater than the limit accumulator, load the target accumulator with the contents of the limit accumulator. The Z and N Status bits are set such that a subsequent BRA GT instruction will take a branch. In addition, Z is set such that a subsequent MIN instruction will execute as a NOP if the limit is exceeded. If the limit is not exceeded \((Z = 1)\), the MIN instruction will execute as normal.

If the target accumulator is not greater than the limit accumulator, the target accumulator is unaffected. The Z Status bit is set such that a subsequent BRA Z instruction will take a branch.

The OV Status bit is always cleared by this instruction.

The 'A' bit specifies the destination accumulator.

The 'x' bits define the presence and result format for Wd.

**Note:**

OA and SA or OB and SB Status bits are not modified by this instruction. Execute SFTAC <AccX>, #0 after MAX operation to update DSP status to reflect contents of AccX.

**Words:**

1

**Cycles:**

1
Section 5. Instruction Descriptions

MAX.V

Accumulator Force Maximum Data Range Limit with Limit Excess Result

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{label:} MAX.V Acc Wd

[Wd]

[Wd++]

[Wd--]

[++Wd]

[--Wd]

Operands:

Wd ∈ [W0 ... W15]

A ∈ [A,B]

Operation:

If (MAX A) Then

If ACCA – ACCB > 0 Then

(0x0001 → Wd or ACCA – ACCB → Wd (see text);
ACCB → ACCA;
0 → Z; 0 → N; 0 → OV;)

Else

(0 → Wd;
1 → Z; 0 → N; 0 → OV;)

If (MAX B) Then

If ACCB – ACCA > 0 Then

(0x0001 → Wd or ACCB – ACCA → Wd (see text);
ACCA → ACCB;
1 → Z; 0 → N; 0 → OV;)

Else

(0 → Wd;
1 → Z; 0 → N; 0 → OV;)

Status Affected:

N, OV, Z

Encoding:

1100 1110 A00x x000 0qqq dddd

Description:

The target accumulator (defined in the instruction) is clamped to the maximum limit value previously loaded into the other accumulator. The comparison examines the full 40-bit value of the target accumulator, and will therefore, clamp an overflowed accumulator.

If the target accumulator is greater than the limit accumulator, load the target accumulator with the contents of the limit accumulator. For MAX (instruction bit field xx = 2'b10), set Wd to +1. For MAX.V (instruction bit field xx = 2'b11), write the (signed) value by which the limit is exceeded to Wd. This is sourced from the Least Significant 16 bits of the 40-bit result. If the limit is exceeded by a value greater than that which can be represented by a signed 16-bit number, saturate the Wd write to the maximum positive value (i.e., set Wd to 0x7FFF).

The Z and N Status bits are set such that a subsequent BRA GT instruction will take a branch if the limit is exceeded. In addition, Z is set such that a subsequent MIN{.V} instruction will execute as a NOP if the limit is exceeded. If the limit is not exceeded (Z = 1), the MIN{.V} instruction will execute as normal.

If the target accumulator is not greater than the limit accumulator, the target accumulator is unaffected and Wd is cleared. The Z Status bit is set such that a subsequent BRA Z instruction will take a branch.

The OV Status bit is always cleared by this instruction.

The 'A' bit specifies the destination accumulator.
The 'd' bits select the address of the destination register.
The 'q' bits select the destination addressing mode.
The 'x' bits define the presence and result format for Wd.

Note: OA and SA or OB and SB Status bits are not modified by this instruction. Execute SFTAC <AccX>, #0 after MAX.V operation to update DSP status to reflect contents of AccX.

Words: 1

Cycles: 1
### MIN

Accumulator Force Minimum Data Range Limit (Unconditional Execution)

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Syntax:

{label:} MIN Acc

#### Operands:

A ∈ [A,B]

#### Operation:

If (MIN A) Then
  If ACCA – ACCB < 0 Then
    (ACCB → ACCA;
     0 → Z; 1 → N; 0 → OV;)
  Else
    (1 → Z; 0 → N; 0 → OV;)
If (MIN B) Then
  If ACCB – ACCA < 0 Then
    (ACCA → ACCB;
     0 → Z; 1 → N; 0 → OV;)
  Else
    (1 → Z; 0 → N; 0 → OV;)

#### Status Affected:

N, OV, Z

#### Encoding:

<table>
<thead>
<tr>
<th></th>
<th>1100</th>
<th>1110</th>
<th>A0lx</th>
<th>x000</th>
<th>0000</th>
<th>0000</th>
</tr>
</thead>
</table>

#### Description:

The target accumulator (defined in the instruction) is clamped to the minimum limit value previously loaded into the other accumulator. The comparison examines the full 40-bit value of the target accumulator, and will therefore, clamp an overflowed accumulator.

If the target accumulator is greater than the limit accumulator, load the target accumulator with the contents of the limit accumulator. The Z and N Status bits are set such that a subsequent BRA LT instruction will take a branch.

If the target accumulator is not less than the limit accumulator, the target accumulator is unaffected. The Z Status bit is set (Z = 1) such that a subsequent BRA Z instruction will take a branch.

The OV Status bit is always cleared by this instruction.

The ‘A’ bit specifies the destination accumulator.

The ‘x’ bits define the presence and result format for Wd.

**Note:** OA and SA or OB and SB Status bits are not modified by this instruction. Execute SFTAC <AccX>, #0 after MIN execution to update DSP status to reflect contents of AccX.

#### Words:

1

#### Cycles:

1
## Section 5. Instruction Descriptions

### MIN.V

**Accumulator Force Minimum Data Range Limit with Limit Excess Result**

*(Unconditional Execution)*

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>{label:} MIN.V Acc Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>If (MIN A) Then If ACCA − ACCB &lt; 0 Then (0xFFFF → Wd or ACCA − ACCB → Wd (see text); ACCA → ACCA; 0 → Z; 1 → N; 0 → OV;) Else (0 → Wd; 1 → Z; 0 → N; 0 → OV;) If (MIN B) Then If ACCB − ACCA &lt; 0 Then (0xFFFF → Wd or ACCB − ACCA → Wd (see text); ACCA → ACCB; 0 → Z; 1 → N; 0 → OV;) Else (0 → Wd; 1 → Z; 0 → N; 0 → OV;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Affected:</td>
<td>N, OV, Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding:</td>
<td>1100 1110 A01x x000 0qqq dddd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>The target accumulator (defined in the instruction) is clamped to the minimum limit value previously loaded into the other accumulator. The comparison examines the full 40-bit value of the target accumulator. If the target accumulator is greater than the limit accumulator, load the target accumulator with the contents of the limit accumulator. For MIN (instruction bit field xx = 2’b10), set Wd to -1. For MIN.V (instruction bit field xx = 2’b11), write the (signed) value by which the limit is exceeded to Wd. This is sourced from the Least Significant 16 bits of the 40-bit result. If the limit is exceeded by a value greater than that which can be represented by a signed 16-bit number, saturate the Wd write to the maximum negative value (i.e., set Wd to 0x8000). The Z and N Status bits are set such that a subsequent BRA LT instruction will take a branch if the limit is exceeded. If the target accumulator is not less than the limit accumulator, the target accumulator is unaffected and Wd is cleared. The Z Status bit is set such that a subsequent BRA Z instruction will take a branch. The OV Status bit is always cleared by this instruction. The ‘A’ bit specifies the destination accumulator. The ‘d’ bits select the address of the destination register. The ‘q’ bits select the destination addressing mode. The ‘x’ bits define the presence and result format for Wd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>OA and SA or OB and SB Status bits are not modified by this instruction. Execute SFTAC &lt;AccX&gt;, #0 after MIN.V execution to update DSP status to reflect contents of AccX.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Words:</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles:</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MINZ

**Accumulator Force Minimum Data Range Limit (Conditional Execution)**

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C X

Syntax: {label:} MINZ Acc

Operands: Acc ∈ [A,B]

Operation:

If \( Z = 0 \) execute as NOP

Else

If (MINZ A) Then

If ACCA – ACCB < 0 Then

\[ \text{ACCB} \rightarrow \text{ACCA}; \]

\[ 0 \rightarrow Z; 1 \rightarrow \text{N}; 0 \rightarrow \text{OV}; \]

Else

\[ 1 \rightarrow Z; 0 \rightarrow \text{N}; 0 \rightarrow \text{OV}; \]

If (MINZ B) Then

If ACCB – ACCA < 0 Then

\[ \text{ACCA} \rightarrow \text{ACCB}; \]

\[ 0 \rightarrow Z; 1 \rightarrow \text{N}; 0 \rightarrow \text{OV}; \]

Else

\[ (0 \rightarrow \text{Wd}; \]

\[ 1 \rightarrow Z; 0 \rightarrow \text{N}; 0 \rightarrow \text{OV}; \]

Status Affected: N, OV, Z

Encoding:

<table>
<thead>
<tr>
<th>1100</th>
<th>1110</th>
<th>A0lx</th>
<th>x100</th>
<th>0000</th>
</tr>
</thead>
</table>

Description:

If MINZ is executed when Z = 1 (see note), the target accumulator (defined in the instruction) is clamped to the minimum limit value previously loaded into the other accumulator. If MINZ is executed when Z = 0, the instruction is skipped (executed as a NOP).

The comparison examines the full 40-bit value of the target accumulator, and will therefore, clamp an overflowed accumulator.

If the target accumulator is less than the limit accumulator, load the target accumulator with the contents of the limit accumulator. The Z and N Status bits are set such that a subsequent BRA LT instruction will take a branch.

If the target accumulator is not less than the limit accumulator, the target accumulator is unaffected. The Z Status bit is set (Z = 1) such that a subsequent BRA Z instruction will take a branch.

The OV Status bit is always cleared by this instruction.

The ‘A’ bit specifies the destination accumulator.

The ‘x’ bits define the presence and result format for Wd.

**Note 1:** Execution of the accumulator maximum clamp instruction (MAX) is expected to be immediately followed by execution of the conditionally executed accumulator minimum clamp instruction (MINZ). If MAX resulted in a clamp condition (Z = 0), MINZ will be skipped. Use the unconditionally executed MIN instruction if it is required to be executed in isolation.

**Note 2:** OA and SA or OB and SB Status bits are not modified by this instruction.

Execute SFTAC <AccX>, #0 after MINZ execution to update DSP status to reflect the contents of AccX.

Words: 1

Cycles: 1
MINZ.V

Accumulator Force Minimum Data Range Limit with Limit Excess Result
(Conditional Execution)

Implemented in: PIC24F  PIC24H  PIC24E  dsPIC30F  dsPIC33F  dsPIC33E  dsPIC33C

Syntax: {label:} MINZ.V Acc Wd

[Wd]

[Wd++]

[Wd--]

[++Wd]

[--Wd]

Operands: Wd ∈ [W0 ... W15]

A ∈ [A,B]

Operation:

If (Z = 0) execute as a NOP

Else

If (MINZ A) Then

If (ACCA – ACCB < 0 Then

(0xFFFF → Wd or ACCA – ACCB → Wd (see text);

ACCB → ACCA;

0 → Z; 1 → N; 0 → OV;)

Else

(0 → Wd;

1 → Z; 0 → N; 0 → OV;)

If (MINZ B) Then

If ACCB – ACCA < 0 Then

(0xFFFF → Wd or ACCB – ACCA → Wd (see text);

ACCA → ACCB;

0 → Z; 1 → N; 0 → OV;)

Else

(0 → Wd;

1 → Z; 0 → N; 0 → OV;)

Status Affected: N, OV, Z

Encoding:

| 1100 | 1110 | A01x | x100 | 0qqq | dddd |
MINZ.V

Accumulator Force Minimum Data Range Limit with Limit Excess Result
(Conditional Execution)

Description:
If MINZ is executed when Z = 1 (see note), the target accumulator (defined in the instruction) is clamped to the minimum limit value previously loaded into the other accumulator. If MINZ is executed when Z = 0, the instruction is skipped (executed as a NOP).
The comparison examines the full 40-bit value of the target accumulator, and will therefore, clamp an overflowed accumulator.
If the target accumulator is less than the limit accumulator, load the target accumulator with the contents of the limit accumulator. For MINZ (instruction bit field xx = 2'b10), set Wd to -1. For MINZ.V (instruction bit field xx = 2'b11), write the (signed) value by which the limit is exceeded to Wd. This is sourced from the Least Significant 16 bits of the 40-bit result. If the limit is exceeded by a value greater than that which can be represented by a signed 16-bit number, saturate the Wd write to the maximum negative value (i.e., set Wd to 0x8000).
The Z and N Status bits are set such that a subsequent BRA LT instruction will take a branch if the limit is exceeded.
If the target accumulator is not less than the limit accumulator, the target accumulator is unaffected and Wd is cleared. The Z Status bit is set such that a subsequent BRA Z instruction will take a branch.
The OV Status bit is always cleared by this instruction.
The 'A' bit specifies the destination accumulator.
The 'd' bits select the address of the destination register.
The 'q' bits select the destination addressing mode.
The 'x' bits define the presence and result format for Wd.

Note 1: Execution of the MINZ.V instruction is intended to immediately follow execution of a MAX instruction. If MAX resulted in a clamp condition (Z = 0), the MINZ.V instruction will be skipped.
2: OA and SA or OB and SB Status bits are not modified by this instruction.
Execute SFTAC <AccX>, #0 after MINZ.V execution to update DSP status to reflect the contents of AccX.

Words: 1
Cycles: 1
## Section 5. Instruction Descriptions

### MOV

Move f to Destination

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```{label:} MOV{.B} f {,WREG}
```

**Operands:**

- `f` ∈ [0 ... 8191]

**Operation:**

`(f) → destination designated by D`

**Status Affected:**

- N, Z

**Encoding:**

```txt
1011 1111 1Bdf ffff ffff ffff
```

**Description:**

Move the contents of the specified file register to 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 back to the file register and the only effect is to modify the STATUS Register.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

**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.

**Note 2:**

The WREG is set to Working register W0.

**Note 3:**

When moving word data from file register memory, the "MOV f to Wnd" (page 301) instruction allows any Working register (W0:W15) to be the destination register.

**Words:**

1

**Cycles:**

1

**Note 1:**

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

### Example 1

MOV.B TMR0, WREG ; move (TMR0) to WREG (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>9080</td>
<td>9055</td>
</tr>
<tr>
<td>TMR0</td>
<td>TMR0</td>
</tr>
<tr>
<td>2355</td>
<td>2355</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2

MOV  0x800 ; update SR based on (0x800) (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0800</td>
<td>Data 0800</td>
</tr>
<tr>
<td>B29F</td>
<td>B29F</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0008</td>
</tr>
</tbody>
</table>

(N = 1)
MOV

Move WREG to f

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
---|---|---|---|---|---|---|---|
X | X | X | X | X | X | X | X |

Syntax: 

{label:} MOV{.B} WREG, f

Operands: 

f ∈ [0 ... 8191]

Operation: 

(WREG) → f

Status Affected: None

Encoding: 

| 1011 | 0111 | 1B1f | ffff | ffff | ffff |

Description: Move the contents of the default Working register WREG into the specified file register. The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte). The ‘f’ bits select the address of the file register.

**Note 1:** The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

**Note 2:** The WREG is set to Working register W0.

**Note 3:** When moving word data from the Working register array to file register memory, the "MOV Wns to f" (page 302) instruction allows any Working register (W0:W15) to be the source register.

Words: 1

Cycles: 1

**Example 1:** MOV.B WREG, 0x801 ; move WREG to 0x801 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>Data 0800</td>
<td>98F3</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0800</td>
<td>4509</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:** MOV WREG, DISICNT ; move WREG to DISICNT

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>DISICNT</td>
<td>00A0</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISICNT</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

MOV

Move f to Wnd

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{\text{label:}} MOV f, Wnd

Operands:  

f $\in \{0 \ldots 65534\}$

Wnd $\in \{W0 \ldots W15\}$

Operation:  

(f) \rightarrow Wnd

Status Affected: None

Encoding:

| 1000 | 0fff | ffff | ffff | ffff | dddd |

Description:

Move the word contents of the specified file register to Wnd. The file register may reside anywhere in the 32K words of data memory, but must be word-aligned. Register Direct Addressing must be used for Wnd.

The ‘f’ bits select the address of the file register.

The ‘d’ bits select the destination register.

Note 1: This instruction operates on word operands only.

2: Since the file register address must be word-aligned, only the upper 15 bits of the file register address are encoded (bit 0 is assumed to be ‘0’).

3: To move a byte of data from file register memory, the "MOV f to Destination" instruction (page 299) may be used.

Words: 1

Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

Example 1:

MOV CORCON, W12  ; move CORCON to W12

Before Instruction

<table>
<thead>
<tr>
<th>W12</th>
<th>CORCON</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>78FA</td>
<td>00F0</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W12</th>
<th>CORCON</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00F0</td>
<td>00F0</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

MOV 0x27FE, W3  ; move (0x27FE) to W3

Before Instruction

<table>
<thead>
<tr>
<th>W3</th>
<th>Data 27FE</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0035</td>
<td>ABCD</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>W3</th>
<th>Data 27FE</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>ABCD</td>
<td>0000</td>
</tr>
</tbody>
</table>
MOV

Move Wns to f

Implemented in:  

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} MOV Wns, f

Operands:  

f \in [0 ... 65534]  
Wns \in [W0 ... W15]

Operation:  

(Wns) \rightarrow f

Status Affected:  
None

Encoding:  

```
1000 1fff ffff ffff ffff ssss
```

Description:  

Move the word contents of the Working register Wns to the specified file register. The file register may reside anywhere in the 32K words of data memory, but must be word-aligned. Register Direct Addressing must be used for Wn. The 'f' bits select the address of the file register. The 's' bits select the source register.

**Note 1:** This instruction operates on word operands only.

2: Since the file register address must be word-aligned, only the upper 15 bits of the file register address are encoded (bit 0 is assumed to be '0').

3: To move a byte of data to file register memory, the "MOV WREG to f" instruction (page 300) may be used.

Words:  
1

Cycles:  
1

**Example 1:** MOV W4, XMODSRT ; move W4 to XMODSRT

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 1200</td>
<td>W4 1200</td>
</tr>
<tr>
<td>XMODSRT 1340</td>
<td>XMODSRT 1200</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:** MOV W8, 0x1222 ; move W8 to data address 0x1222

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8 F200</td>
<td>W8 F200</td>
</tr>
<tr>
<td>Data 1222</td>
<td>Data 1222</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### MOV.B

**Move 8-Bit Literal to Wnd**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MOV.B #lit8, Wnd

**Operands:**

lit8 ∈ [0 ... 255]

Wnd ∈ [W0 ... W15]

**Operation:**

lit8 → Wnd

**Status Affected:**

None

**Encoding:**

| 1011 | 0011 | 1100 | kkkk | kkkk | dddd |

**Description:**

The unsigned 8-bit literal 'k' is loaded into the lower byte of Wnd. The upper byte of Wnd is not changed. Register Direct Addressing must be used for Wnd.

The 'k' bits specify the value of the literal.

The 'd' bits select the address of the Working register.

**Note:** This instruction operates in Byte mode and the .B extension must be provided.

**Words:** 1

**Cycles:** 1

**Example 1:**

MOV.B #0x17, W5 ; load W5 with #0x17 (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5 7899</td>
<td>W5 7817</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

MOV.B #0xFE, W9 ; load W9 with #0xFE (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W9 AB23</td>
<td>W9 ABFE</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
MOV

Move 16-Bit Literal to Wnd


<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
[label:] MOV #lit16, Wnd

Operands:  
lit16 ∈ [-32768 ... 65535]  
Wnd ∈ [W0 ... W15]

Operation:  
lit16 → Wnd

Status Affected: None

Encoding:  
0010 kkkk kkkk kkkk dddd

Description: The 16-bit literal ‘k’ is loaded into Wnd. Register Direct Addressing must be used for Wnd.  
The ‘k’ bits specify the value of the literal.  
The ‘d’ bits select the address of the Working register.

Note 1: This instruction operates only in Word mode.  
2: The literal may be specified as a signed value [-32768:32767] or unsigned value [0:65535].

Words: 1

Cycles: 1

Example 1: MOV #0x4231, W13 ; load W13 with #0x4231

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W13 091B</td>
<td>W13 4231</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2: MOV #0x4, W2 ; load W2 with #0x4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 B004</td>
<td>W2 0004</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 3: MOV #-1000, W8 ; load W8 with #-1000

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8 23FF</td>
<td>W8 FC18</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
## MOV

**Move [Ws with Offset] to Wnd**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} MOV{.B} [Ws + Slit10], Wnd
```

**Operands:**

- `Ws ∈ [W0 ... W15]`
- `Slit10 ∈ [-512 ... 511]` for byte operation
- `Slit10 ∈ [-1024 ... 1022]` (even only) for word operation
- `Wnd ∈ [W0 ... W15]`

**Operation:**

```
[Ws + Slit10] → Wnd
```

**Status Affected:**

None

**Encoding:**

```
1001 0kkk kBkk kddd dkkk ssss
```

**Description:**

The contents of `[Ws + Slit10]` are loaded into `Wnd`. In Word mode, the range of `Slit10` is increased to `[-1024 ... 1022]` and `Slit10` must be even to maintain word address alignment. Register Indirect Addressing must be used for the source and Direct Addressing must be used for `Wnd`.

The 'k' bits specify the value of the literal.

The 'B' bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘d’ bits select the destination register.

The ‘s’ bits select the source register.

**Note 1:** The extension `.B` in the instruction denotes a byte move rather than a word move. You may use a `.W` extension to denote a word move, but it is not required.

**Note 2:** In Byte mode, the range of `Slit10` is not reduced as specified in Section 4.6 “Using 10-bit Literal Operands”, since the literal represents an address offset from `Ws`.

**Words:**

1

**Cycles:**

1

---

**Example 1:**

```
MOV.B [W8+0x13], W10 ; load W10 with [W8+0x13]
                   ; (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8 1008</td>
<td>W8 1008</td>
</tr>
<tr>
<td>W10 4009</td>
<td>W10 4033</td>
</tr>
<tr>
<td>Data 101A</td>
<td>Data 101A</td>
</tr>
<tr>
<td>sr 0000</td>
<td>sr 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
MOV    [W4+0x3E8], W2 ; load W2 with [W4+0x3E8]
                   ; (Word mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 9088</td>
<td>W2 5634</td>
</tr>
<tr>
<td>W4 0800</td>
<td>W4 0800</td>
</tr>
<tr>
<td>Data 0BE8</td>
<td>Data 0BE8</td>
</tr>
<tr>
<td>sr 0000</td>
<td>sr 0000</td>
</tr>
</tbody>
</table>

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

---

© 2005-2018 Microchip Technology Inc.
MOV

Move Wns to [Wd with Offset]

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} MOV{.B} Wns, [Wd + Slit10]

Operands:

Wns ∈ [W0 ... W15]
Slit10 ∈ [-512 ... 511] in Byte mode
Slit10 ∈ [-1024 ... 1022] (even only) in Word mode
Wd ∈ [W0 ... W15]

Operation:  

(Wns) → [Wd + Slit10]

Status Affected:  
None

Encoding:

| 1001 | lkkk | kBkk | kddd | dkkk | ssss |

Description:

The contents of Wns are stored to [Wd + Slit10]. In Word mode, the range of Slit10 is increased to [-1024 ... 1022] and Slit10 must be even to maintain word address alignment.

Register Direct Addressing must be used for Wns and Indirect Addressing must be used for the destination.

The 'k' bits specify the value of the literal.
The 'B' bit selects byte or word operation ('0' for word, '1' for byte).
The 'd' bits select the destination register.
The 's' bits select the source register.

Note 1:  
The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

2:  
In Byte mode, the range of Slit10 is not reduced as specified in Section 4.6 "Using 10-bit Literal Operands", since the literal represents an address offset from Wd.

Words:  
1

Cycles:  
1

Example 1:  

MOV.B W0, [W1+0x7] ; store W0 to [W1+0x7]  
; (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 9015</td>
<td>W0 9015</td>
</tr>
<tr>
<td>W1 1800</td>
<td>W1 1800</td>
</tr>
<tr>
<td>Data 1806 2345</td>
<td>Data 1806 1545</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:  

MOV W11, [W1-0x400] ; store W11 to [W1-0x400]  
; (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 1000</td>
<td>W1 1000</td>
</tr>
<tr>
<td>W11 8813</td>
<td>W11 8813</td>
</tr>
<tr>
<td>Data 0C00 FFEA</td>
<td>Data 0C00 8813</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

### MOV

**Move Ws to Wd**

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**

{label:} MOV{.B} Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- [++Ws], [++Wd]
- [Ws + Wb], [Wd + Wb]

**Operands:**
- Ws ∈ [W0 ... W15]
- Wb ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

**Operation:**

(Ws) → Wd

**Status Affected:** None

**Encoding:**

| 0111 | 1www | wBhh | hddd | dggg | sss |

**Description:**

Move the contents of the source register into the destination register. Either Register Direct or Indirect Addressing may be used for Ws and Wd.

The 'w' bits define the offset register Wb.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'h' bits select the destination addressing mode.

The 'd' bits select the destination register.

The 'g' bits select the source addressing mode.

The 's' bits select the source register.

**Note 1:** The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

**Note 2:** When Register Offset Addressing mode is used for both the source and destination, the offset must be the same because the ‘w’ encoding bits are shared by Ws and Wd.

**Note 3:** The instruction, "PUSH Ws", translates to "MOV Ws, [W15++]".

**Note 4:** The instruction, "POP Wd", translates to "MOV [--W15], Wd".

**Words:**

1

**Cycles:**

1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.
**Example 1:**  MOV.B \([W0--]\), W4  ; Move \([W0]\) to W4 (Byte mode)  
; Post-decrement W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 0A01</td>
<td>W0 0A00</td>
</tr>
<tr>
<td>W4</td>
<td>W4 2989</td>
</tr>
<tr>
<td>Data 0A00</td>
<td>Data 8988</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**  MOV \([W6++]\), \([W2+W3]\)  ; Move \([W6]\) to \([W2+W3]\) (Word mode)  
; Post-increment W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 0800</td>
<td>W2 0800</td>
</tr>
<tr>
<td>W3 0040</td>
<td>W3 0040</td>
</tr>
<tr>
<td>W6 1228</td>
<td>W6 122A</td>
</tr>
<tr>
<td>Data 0840</td>
<td>Data 0840</td>
</tr>
<tr>
<td>Data 1228</td>
<td>Data 1228</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

MOV.D

Double-Word Move from Source to Wnd

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} MOV.D Wns, Wnd

[Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands:

Wns ∈ [W0, W2, W4 ... W14]
Ws ∈ [W0 ... W15]
Wnd ∈ [W0, W2, W4 ... W14]

Operation:

For Direct Addressing of Source:

Wns → Wnd

Wns + 1 → Wnd + 1

For Indirect Addressing of Source:

See Description

Status Affected: None

Encoding:

| 1011 | 1110 | 0000 | 0ddd | 0ppp | ssss |

Description:

Move the double word specified by the source to a destination Working register pair (Wnd:Wnd + 1). If Register Direct Addressing is used for the source, the contents of two successive Working registers (Wns:Wns + 1) are moved to Wnd:Wnd + 1. If Indirect Addressing is used for the source, Ws specifies the Effective Address for the least significant word of the double word. Any pre/post-increment or pre/post-decrement will adjust Ws by 4 bytes to accommodate for the double word.

The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the address of the first source register.

Note 1: This instruction only operates on double words. See Figure 4-3 for information on how double words are aligned in memory.

2: Wnd must be an even numbered Working register.

3: The instruction, "POP.D Wnd", translates to "MOV.D [--W15], Wnd".

Words: 1
Cycles: 2(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions".
Example 1: MOV.D W2, W6 ; Move W2 to W6 (Double mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 12FB</td>
<td>W2 12FB</td>
</tr>
<tr>
<td>W3 9877</td>
<td>W3 9877</td>
</tr>
<tr>
<td>W6 9833</td>
<td>W6 12FB</td>
</tr>
<tr>
<td>W7 FCC6</td>
<td>W7 9877</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2: MOV.D [W7--], W4 ; Move [W7] to W4 (Double mode) ; Post-decrement W7

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 B012</td>
<td>W4 A319</td>
</tr>
<tr>
<td>W5 FD89</td>
<td>W5 9927</td>
</tr>
<tr>
<td>W7 0900</td>
<td>W7 08FC</td>
</tr>
<tr>
<td>Data 0900 A319</td>
<td>Data 0900 A319</td>
</tr>
<tr>
<td>Data 0902 9927</td>
<td>Data 0902 9927</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
**Section 5. Instruction Descriptions**

---

**MOVPAG**  
Move Literal to Page Register

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:  
{label:} MOVPAG #lit10, DSRPAG  
#lit9, DSWPAG  
#lit8, TBLPAG

Operands:  
lit10 ∈ [0 ... 1023], lit9 ∈ [0 ... 511], lit8 ∈ [0 ... 255]

Operation:  
lit10 → DSRPAG or lit9 → DSWPAG or lit8 → TBLPAG

Status Affected:  
None

Encoding:  
```
1111 1110 1100 PPkk kkkk kkkk
```

Description:  
The appropriate number of bits from the unsigned literal 'k' is loaded into the DSRPAG, DSWPAG or TBLPAG register. The assembler restricts the literal to a 9-bit unsigned value when the destination is DSWPAG and an 8-bit unsigned value when the destination is TBLPAG.

The 'P' bits select the destination register.
The 'k' bits specify the value of the literal.

**Note:**  
This instruction operates in Word mode only.

Words:  
1

Cycles:  
1

**Example 1:**  
MOVPAG #0x02, DSRPAG

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRPAG 0000</td>
<td>DSRPAG 0002</td>
</tr>
</tbody>
</table>
# MOVPAG

## Move Wn to Page Register

### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

```assembly
{label:} MOVPAG Wn, DSRPAG
  DSWPAG
  TBLPAG
```

### Operands:

Wn ∈ [W0 ... W15]

### Operation:

Wn<9:0> → DSRPAG or Wn<8:0> → DSWPAG or Wn<7:0> → TBLPAG

### Status Affected:

None

### Encoding:

| 1111 | 1110 | 1101 | PP00 | 0000 | ssss |

### Description:

The appropriate number of bits from the register Ws is loaded into the DSRPAG, DSWPAG or TBLPAG register. The assembler restricts the literal to a 9-bit unsigned value when the destination is DSWPAG and an 8-bit unsigned value when the destination is TBLPAG.

The ‘P’ bits select the destination register.

The ‘s’ bits specify the source register.

**Note:** This instruction operates in word mode only.

### Words:

1

### Cycles:

1

### Example 1:

```assembly
MOVPAG W2, DSRPAG
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRPAG 0000</td>
<td>DSRPAG 0002</td>
</tr>
<tr>
<td>W2 0002</td>
<td>W2 0002</td>
</tr>
</tbody>
</table>

---

DS70000157G-page 312 © 2005-2018 Microchip Technology Inc.
Section 5. Instruction Descriptions

**MOVSAC**

Prefetch Operands and Store Accumulator

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MOVSAC Acc {,[Wx], Wxd} {,[Wy], Wyd} {,AWB}

{,[Wx] + = kx, Wxd} {,[Wy] + = ky, Wyd}

{,[Wx] − = kx, Wxd} {,[Wy] − = ky, Wyd}

{,[W9 + W12], Wxd} {,[W11 + W12], Wyd}

**Operands:**

Acc ∈ [A,B]
Wx ∈ [W8, W9]; kx ∈ [-6, -4, -2, 2, 4, 6]; Wxd ∈ [W4 ... W7]
Wy ∈ [W10, W11]; ky ∈ [-6, -4, -2, 2, 4, 6]; Wyd ∈ [W4 ... W7]
AWB ∈ [W13, [W13] + = 2]

**Operation:**

([Wx]) → Wxd; (Wx) + kx → Wx

([Wy]) → Wyd; (Wy) + ky → Wy

(Acc(B or A)) rounded → AWB

**Status Affected:**

None

**Encoding:**

```
1100 0111 A0xx yyii iijj jjaa
```

**Description:**

Optionally fetch operands in preparation for another MAC type instruction and optionally store the unspecified accumulator results. Even though an accumualtor operation is not performed in this instruction, an accumulator must be specified to designate which accumulator to Write-Back.

Operands, Wx, Wxd, Wy and Wyd, specify optional fetch operations, which support Indirect and Register Offset Addressing, as described in Section 4.15.1 “MAC Prefetches”. Operand AWB specifies the optional store of the “other” accumulator, as described in Section 4.15.4 “MAC Write-Back”.

The ‘A’ bit selects the other accumulator used for Write-Back.
The ‘x’ bits select the prefetch Wxd destination.
The ‘y’ bits select the prefetch Wyd destination.
The ‘i’ bits select the Wx prefetch operation.
The ‘j’ bits select the Wy prefetch operation.
The ‘a’ bits select the accumulator Write-Back destination.

**Words:**

1

**Cycles:**

1
Example 1:  
; Fetch [W9] to W6  
; Fetch [W11] to W7, Post-increment W11 by 4  
; Store ACCA to W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>A022</td>
</tr>
<tr>
<td>W7</td>
<td>B200</td>
</tr>
<tr>
<td>W9</td>
<td>0800</td>
</tr>
<tr>
<td>W11</td>
<td>1900</td>
</tr>
<tr>
<td>W13</td>
<td>0020</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 3290 5968</td>
</tr>
<tr>
<td>Data 0800</td>
<td>7811</td>
</tr>
<tr>
<td>Data 1900</td>
<td>B2AF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>Data 0800</td>
<td>7811</td>
</tr>
<tr>
<td>Data 1900</td>
<td>B2AF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:  
; Fetch [W9] to W4, Post-decrement W9 by 2  
; Fetch [W11+W12] to W6  
; Store ACCB to [W13], Post-increment W13 by 2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>76AE</td>
</tr>
<tr>
<td>W6</td>
<td>2000</td>
</tr>
<tr>
<td>W9</td>
<td>1200</td>
</tr>
<tr>
<td>W11</td>
<td>2000</td>
</tr>
<tr>
<td>W12</td>
<td>0024</td>
</tr>
<tr>
<td>W13</td>
<td>2300</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 9834 4500</td>
</tr>
<tr>
<td>Data 1200</td>
<td>BB00</td>
</tr>
<tr>
<td>Data 2024</td>
<td>52CE</td>
</tr>
<tr>
<td>Data 2300</td>
<td>B3FF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### MPY

**Multiply Wm by Wn to Accumulator**

**Implemented in:**
- PIC24F: X
- PIC24H: X
- PIC24E: X
- dsPIC30F: X
- dsPIC33F: X
- dsPIC33E: X
- dsPIC33C: X

**Syntax:**
```
{[label:]} MPY  Wm * Wn,  Acc  {,[Wx], Wxd}  {,[Wy], Wyd}  
{,[Wx] + = kx, Wxd}  {,[Wy] + = ky, Wyd}  
{,[Wx] – = kx, Wxd}  {,[Wy] – = ky, Wyd}  
{,[W9 + W12], Wxd}  {,[W11 + W12], Wyd}  
```

**Operands:**
- Acc ∈ [A,B]
- Wx ∈ [W8, W9]; kx ∈ [-6, -4, -2, 2, 4, 6]; Wxd ∈ [W4 ... W7]
- Wy ∈ [W10, W11]; ky ∈ [-6, -4, -2, 2, 4, 6]; Wyd ∈ [W4 ... W7]
- AWB ∈ [W13], [W13] + = 2

**Operation:**
- (Wm) * (Wn) → Acc(A or B)
- ([Wx]) → Wxd; (Wx) + kx → Wx
- ([Wy]) → Wyd; (Wy) + ky → Wy

**Status Affected:** OA, OB, OAB, SA, SB, SAB

**Encoding:**
```
<table>
<thead>
<tr>
<th>Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>A0xx</td>
<td>yyii</td>
</tr>
<tr>
<td>jjjj</td>
<td>jj11</td>
</tr>
</tbody>
</table>
```

**Description:**
Multiply the contents of two Working registers and optionally prefetch operands in preparation for another MAC type instruction. The 32-bit result of the signed multiply is sign-extended to 40 bits and stored to the specified accumulator.

Operands, Wx, Wxd, Wy and Wyd, specify optional prefetch operations which support Indirect and Register Offset Addressing, as described in Section 4.15.1 “MAC Prefetches”.

The 'm' bits select the operand registers, Wm and Wn, for the multiply.

The 'A' bit selects the accumulator for the result.

The 'x' bits select the prefetch Wxd destination.

The 'y' bits select the prefetch Wyd destination.

The 'i' bits select the Wx prefetch operation.

The 'j' bits select the Wy prefetch operation.

**Note 1:** The IF bit (CORCON<0>) determines if the multiply is fractional or an integer.

**Note 2:** The US<1:0> bits (CORCON<13:12>) in dsPIC33E/dsPIC33C, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed or mixed-sign. Only dsPIC33E/dsPIC33C devices support mixed-sign multiplication.

**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) 

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>C000</td>
</tr>
<tr>
<td>W5</td>
<td>9000</td>
</tr>
<tr>
<td>W6</td>
<td>0800</td>
</tr>
<tr>
<td>W7</td>
<td>B200</td>
</tr>
<tr>
<td>W8</td>
<td>1780</td>
</tr>
<tr>
<td>W10</td>
<td>2400</td>
</tr>
<tr>
<td>ACCA</td>
<td>FF 780 2087</td>
</tr>
<tr>
<td>Data 1780</td>
<td>671F</td>
</tr>
<tr>
<td>Data 2400</td>
<td>E3DC</td>
</tr>
<tr>
<td>CORCON</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

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) 

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>C000</td>
</tr>
<tr>
<td>W5</td>
<td>9000</td>
</tr>
<tr>
<td>W6</td>
<td>671F</td>
</tr>
<tr>
<td>W7</td>
<td>E3DC</td>
</tr>
<tr>
<td>W8</td>
<td>1782</td>
</tr>
<tr>
<td>W10</td>
<td>23FE</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 9834 4500</td>
</tr>
<tr>
<td>Data 1782</td>
<td>8FDC</td>
</tr>
<tr>
<td>Data 23FE</td>
<td>0078</td>
</tr>
<tr>
<td>CORCON</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### MPY

**Square to Accumulator**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} MPY Wm * Wm, Acc {,
                     [Wx], Wxd}
                     {, [Wy], Wyd}
                     {, [Wx] + = kx, Wxd}
                     {, [Wy] + = ky, Wyd}
                     {, [Wx] – = kx, Wxd}
                     {, [Wy] – = ky, Wyd}
                     {, [W9 + W12], Wxd}
                     {, [W11 + W12], Wyd}
```

**Operands:**
- `Wm * Wm` in `[W4 * W4, W5 * W5, W6 * W6, W7 * W7]`
- `Acc` in `[A, B]`
- `Wx` in `[W8, W9]; kx in [-6, -4, -2, 2, 4, 6]; Wxd in [W4 ... W7]`
- `Wy` in `[W10, W11]; ky in [-6, -4, -2, 2, 4, 6]; Wyd in [W4 ... W7]`

**Operation:**
- `(Wm) * (Wm) → Acc(A or B)`
- `([Wx]) → Wxd; (Wx) + kx → Wx`
- `([Wy]) → Wyd; (Wy) + ky → Wy`

**Status Affected:**
- OA, OB, OAB, SA, SB, SAB

**Encoding:**

```
1111 00mm A0xx yyii ijjj jj01
```

**Description:**
Square the contents of a Working register and optionally prefetch operands in preparation for another MAC type instruction. The 32-bit result of the signed multiply is sign-extended to 40 bits and stored in the specified accumulator.

Operands, `Wx`, `Wxd`, `Wy` and `Wyd`, specify optional prefetch operations, which support Indirect and Register Offset Addressing, as described in Section 4.15.1 "MAC Prefetches".

The ‘m’ bits select the operand register `Wm` for the square.
The ‘A’ bit selects the accumulator for the result.
The ‘x’ bits select the prefetch `Wxd` destination.
The ‘y’ bits select the prefetch `Wyd` destination.
The ‘i’ bits select the `Wx` prefetch operation.
The ‘j’ bits select the `Wy` prefetch operation.

**Note 1:** The IF bit (CORCON<0>) determines if the multiply is fractional or an integer.

**Note 2:** The US<1:0> bits (CORCON<13:12> in dsPIC33E/dsPIC33C, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed or mixed-sign. Only dsPIC33E/dsPIC33C devices support mixed-sign multiplication.

**Words:**
1

**Cycles:**
1
### Example 1:

```plaintext
MPY W6* W6, A, [W9]+2, W6
; Square W6 and store to ACCA
; Fetch [W9] to W6, Post-increment W9 by 2
; CORCON = 0x0000 (fractional multiply, no saturation)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>6500</td>
</tr>
<tr>
<td>W9</td>
<td>0900</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 7C80 0908</td>
</tr>
<tr>
<td>Data 0900</td>
<td>B865</td>
</tr>
<tr>
<td>CORCON</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:

```plaintext
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)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>E228</td>
</tr>
<tr>
<td>W5</td>
<td>9000</td>
</tr>
<tr>
<td>W9</td>
<td>1700</td>
</tr>
<tr>
<td>W10</td>
<td>1B00</td>
</tr>
<tr>
<td>W12</td>
<td>FF00</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 9834 4500</td>
</tr>
<tr>
<td>Data 1600</td>
<td>8911</td>
</tr>
<tr>
<td>Data 1B00</td>
<td>F678</td>
</tr>
<tr>
<td>CORCON</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**Section 5. Instruction Descriptions**

### MPY.N

**Multiply -Wm by Wn to Accumulator**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implemented in</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\{label: \} MPY.N  Wm * Wn,  Acc  \{,[Wx], Wxd\}  \{,[Wy], Wyd\}

\{,[Wx] + = kx, Wxd\}  \{,[Wy] + = ky, Wyd\}


\{,[W9 + W12], Wxd\}  \{,[W11 + W12], Wyd\}

**Operands:**

- \(Wm \times Wn \in \{W4 \times W5; W4 \times W6; W4 \times W7; W5 \times W6; W5 \times W7; W6 \times W7\}\)
- \(Acc \in \{A, B\}\)
- \(Wx \in \{W8, W9\}; kx \in [-6, -4, -2, 2, 4, 6]; Wxd \in \{W4 \ldots W7\}\)
- \(Wy \in \{W10, W11\}; ky \in [-6, -4, -2, 2, 4, 6]; Wyd \in \{W4 \ldots W7\}\)

**Operation:**

\(- (Wm) \times (Wn) \to Acc(A \text{ or } B)\)

\(([Wx]) \to Wxd; (Wx) + kx \to Wx\)

\(([Wy]) \to Wyd; (Wy) + ky \to Wy\)

**Status Affected:**

OA, OB, OAB

**Encoding:**

```
1100 0mmm A1xx yyii iijj jj11
```

**Description:**

Multiply the contents of a Working register by the negative of the contents of another Working register. Optionally prefetch operands in preparation for another MAC type instruction and optionally store the unspecified accumulator results. The 32-bit result of the signed multiply is sign-extended to 40 bits and stored to the specified accumulator.

The ‘m’ bits select the operand registers, Wm and Wn, for the multiply.

The ‘A’ bit selects the accumulator for the result.

The ‘x’ bits select the prefetch Wxd destination.

The ‘y’ bits select the prefetch Wyd destination.

The ‘i’ bits select the Wx prefetch operation.

The ‘j’ bits select the Wy prefetch operation.

**Note 1:** The IF bit (CORCON<0>) determines if the multiply is fractional or an integer.

**Note 2:** The US<1:0> bits (CORCON<13:12> in dsPIC33E/dsPIC33C, CORCON<12> in dsPIC30F/dsPIC33F) determine if the multiply is unsigned, signed or mixed-sign. Only dsPIC33E/dsPIC33C devices support mixed-sign multiplication.

**Words:**

1

**Cycles:**

1
### Example 1:

```
; Multiply W4*W5, negate the result and store to ACCA
; Fetch [W8] to W4, Post-increment W8 by 2
; Fetch [W10] to W5, Post-increment W10 by 2
; CORCON = 0x0001 (integer multiply, no saturation)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 3023</td>
<td>W4 0054</td>
</tr>
<tr>
<td>W5 1290</td>
<td>W5 660A</td>
</tr>
<tr>
<td>W8 0B00</td>
<td>W8 0B02</td>
</tr>
<tr>
<td>W10 2000</td>
<td>W10 2002</td>
</tr>
<tr>
<td>ACCA 00 0000 2387</td>
<td>ACCA FF FC82 7650</td>
</tr>
<tr>
<td>Data 0B00 0054</td>
<td>Data 0B00 0054</td>
</tr>
<tr>
<td>Data 2000 660A</td>
<td>Data 2000 660A</td>
</tr>
<tr>
<td>CORCON 0001</td>
<td>CORCON 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

### Example 2:

```
; 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)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 3023</td>
<td>W4 0054</td>
</tr>
<tr>
<td>W5 1290</td>
<td>W5 660A</td>
</tr>
<tr>
<td>W8 0B00</td>
<td>W8 0B02</td>
</tr>
<tr>
<td>W10 2000</td>
<td>W10 2002</td>
</tr>
<tr>
<td>ACCA 00 0000 2387</td>
<td>ACCA FF F904 ECA0</td>
</tr>
<tr>
<td>Data 0B00 0054</td>
<td>Data 0B00 0054</td>
</tr>
<tr>
<td>Data 2000 660A</td>
<td>Data 2000 660A</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

MSC
Multiply and Subtract from Accumulator

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} MSC  Wm * Wn, Acc  {,[Wx], Wxd}  {,[Wy], Wyd}  {,.AWB}
{,[Wx] + = kx, Wxd}  {,[Wy] + = ky, Wyd}
{,[Wx] – = kx, Wxd}  {,[Wy] – = ky, Wyd}
{,[W9 + W12], Wxd}  {,[W11 + W12], Wyd}

Operands:

Wm * Wn  \in [W4 * W5, W4 * W6, W5 * W6, W5 * W7, W6 * W7, W6 * W7]
Acc  \in [A, B]
Wx  \in [W8, W9]; kx  \in [-6, -4, -2, 2, 4, 6]; Wxd  \in [W4 ... W7]
Wy  \in [W10, W11]; ky  \in [-6, -4, -2, 2, 4, 6]; Wyd  \in [W4 ... W7]
AWB  \in [W13, [W13] + = 2]

Operation:

(\text{Acc(A or B)}) – (Wm) * (Wn)  \to  \text{Acc(A or B)}
((Wx))  \to  Wxd; (Wx) + kx  \to  Wx
((Wy))  \to  Wyd; (Wy) + ky  \to  Wy
(\text{Acc(B or A)}) \text{ rounded}  \to  \text{AWB}

Status Affected:

OA, OB, OAB, SA, SB, SAB

Encoding:

| 1100 | Ommm | Alxx | yyii | iijj | jjaa |

Description:

Multiply the contents of two Working registers. Optionally prefetch operands in preparation for another MAC type instruction and optionally store the unspecified accumulator results.

The 32-bit result of the signed multiply is sign-extended to 40 bits and subtracted from the specified accumulator.

Operands, Wx, Wxd, Wy and Wyd, specify optional prefetch operations, which support Indirect and Register Offset Addressing as described in Section 4.15.1 “MAC Prefetches”.

Operand AWB specifies the optional store of the "other" accumulator as described in Section 4.15.4 “MAC Write-Back”.

The 'm' bits select the operand registers, Wm and Wn, for the multiply.
The 'A' bit selects the accumulator for the result.
The 'x' bits select the prefetch Wxd destination.
The 'y' bits select the prefetch Wyd destination.
The 'i' bits select the Wx prefetch operation.
The 'j' bits select the Wy prefetch operation.
The 'a' bits select the accumulator Write-Back destination.

Note: The IF bit (CORCON<0>) determines if the multiply is fractional or an integer.

Words: 1
Cycles: 1
Example 1:

; 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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>9051</td>
</tr>
<tr>
<td>W7</td>
<td>7230</td>
</tr>
<tr>
<td>W8</td>
<td>0C00</td>
</tr>
<tr>
<td>W10</td>
<td>1C00</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 0567 8000</td>
</tr>
<tr>
<td>Data 0C00</td>
<td>D309</td>
</tr>
<tr>
<td>Data 1C00</td>
<td>100B</td>
</tr>
<tr>
<td>CORCON</td>
<td>0001</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>0500</td>
</tr>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>W11</td>
<td>1800</td>
</tr>
<tr>
<td>W12</td>
<td>0800</td>
</tr>
<tr>
<td>W13</td>
<td>6233</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 3738 5ED0</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 1000 0000</td>
</tr>
<tr>
<td>Data 2000</td>
<td>3579</td>
</tr>
<tr>
<td>CORCON</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
MUL Integer Unsigned Multiply f and WREG

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} MUL{.B} f

Operands:

f ∈ [0 ... 8191]

Operation:

For Byte Operation:

(WREG)<7:0> * (f)<7:0> → W2

For Word Operation:

(WREG) * (f) → W2:W3

Status Affected:

None

Encoding:

| 1011 | 1100 | 0B0f | ffff | ffff | ffff |

Description:

Multiply the default Working register WREG with the specified file register and place the result in the W2:W3 register pair. Both operands and the result are interpreted as unsigned integers. If this instruction is executed in Byte mode, the 16-bit result is stored in W2. In Word mode, the most significant word of the 32-bit result is stored in W3 and the least significant word of the 32-bit result is stored in W2.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte). The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

3: The IF bit (CORCON<0>) has no effect on this operation.

4: This is the only instruction which provides for an 8-bit multiply.

Words: 1

Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

Example 1:

MUL.B 0x800 ; Multiply (0x800)*WREG (Byte mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>9823</td>
<td>9823</td>
</tr>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>FFFF</td>
<td>13B0</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
</tr>
<tr>
<td>FFFF</td>
<td>FFFF</td>
</tr>
<tr>
<td>Data 0800</td>
<td>Data 0800</td>
</tr>
<tr>
<td>2690</td>
<td>2690</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
Example 2:  

MUL TMR1 ; Multiply (TMR1)*WREG (Word mode)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>F001</td>
</tr>
<tr>
<td>W2</td>
<td>0000</td>
</tr>
<tr>
<td>W3</td>
<td>0000</td>
</tr>
<tr>
<td>TMR1</td>
<td>3287</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>WREG (W0)</td>
<td>F001</td>
</tr>
<tr>
<td>W2</td>
<td>C287</td>
</tr>
<tr>
<td>W3</td>
<td>2F5E</td>
</tr>
<tr>
<td>TMR1</td>
<td>3287</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**MUL.SS**

### Integer 16x16-Bit Signed Multiply

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MUL.SS Wb, Ws, Wnd

- [Ws],
- [Ws++],
- [Ws--],
- [++Ws],
- [--Ws],

**Operands:**

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wnd ∈ [W0, W2, W4 ... W12]

**Operation:**

signed (Wb) * signed (Ws) → Wnd:Wnd + 1

**Status Affected:** None

**Encoding:**

| 1011 | 1001 | lwww | wddd | dppp | ssss |

**Description:**

Multiply the contents of Wb with the contents of Ws and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. Both source operands and the result Wnd are interpreted as two's complement signed integers. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Register Indirect Addressing may be used for Ws.

The 'w' bits select the address of the base register.
The 'd' bits select the address of the lower destination register.
The 'p' bits select the source addressing mode.
The 's' bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: Since the product of the multiplication is 32 bits, Wnd must be an even Working register. See Figure 4-2 for information on how double words are aligned in memory.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:**

1

**Cycles:**

1

**Example 1:**

MUL.SS W0, W1, W12  ; Multiply W0*W1
                     ; Store the result to W12:W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 9823</td>
<td>W0 9823</td>
</tr>
<tr>
<td>W1 67DC</td>
<td>W1 67DC</td>
</tr>
<tr>
<td>W12 FFFF</td>
<td>W12 D314</td>
</tr>
<tr>
<td>W13 FFFF</td>
<td>W13 D5DC</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

© 2005-2018 Microchip Technology Inc. DS70000157G-page 325
Example 2: \texttt{MUL.SS W2, [--W4], W0} ; Pre-decrement W4
; Multiply W2×[W4]
; Store the result to W0:W1

Before Instruction | After Instruction
\hline
W0        & \text{FFFF} & W0        & \text{28F8} \\
W1        & \text{FFFF} & W1        & \text{0000} \\
W2        & \text{0045} & W2        & \text{0045} \\
W4        & \text{27FE} & W4        & \text{27FC} \\
\text{Data 27FC} & \text{098} & \text{Data 27FC} & \text{098} \\
SR        & \text{0000} & SR        & \text{0000} \\

MUL.SS

Integer 16x16-Bit Signed Multiply with Accumulator Destination

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax: {label:} MUL.SS Wb, Ws, Acc

Operands:
- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Acc ∈ [A, B]

Operation:
signed (Wb) * signed (Ws) → Acc(A or B)

Status Affected: None

Encoding:

| 1011 | 1001 | lwww | w111 | Appp | ssss |

Description:
Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is sign-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). Both source operands are treated as signed values.

The ‘w’ bits select the address of the base register.

The ‘p’ bits select Source Addressing Mode 2.

The ‘A’ bit selects the destination accumulator for the product.

Note 1: This instruction operates in Word mode only.

2: The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

Words: 1
Cycles: 1 (1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

Example 1: MUL.SS W0, W1, A

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  9823</td>
<td>W0  9823</td>
</tr>
<tr>
<td>W1  67DC</td>
<td>W1  67DC</td>
</tr>
<tr>
<td>ACCA 00 0000 0000</td>
<td>ACCA FF D5DC D314</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
# MUL.SU

**Integer 16x16-Bit Signed-Unsigned Short Literal Multiply**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wb</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MUL.SU Wb, #lit5, Wnd

**Operands:**

Wb ∈ [W0 ... W15]

lit5 ∈ [0 ... 31]

Wnd ∈ [W0, W2, W4 ... W12]

**Operation:**

signed (Wb) * unsigned lit5 → Wnd:Wnd + 1

**Status Affected:**

None

**Encoding:**

| 1011 | 1001 | 0www | wddd | d11k | kkkk |

**Description:**

Multiply the contents of Wb with the 5-bit literal and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. The Wb operand and the result Wnd are interpreted as a two's complement signed integer. The literal is interpreted as an unsigned integer. Register Direct Addressing must be used for Wb and Wnd.

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.

**Note 1:** This instruction operates in Word mode only.

2: Since the product of the multiplication is 32 bits, Wnd must be an even numbered Working register. See Figure 4-3 for information on how double words are aligned in memory.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:**

1

**Cycles:**

1

**Example 1:**

MUL.SU W0, #0x1F, W2 ; Multiply W0 by literal 0x1F

; Store the result to W2:W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  C000</td>
<td>W0  C000</td>
</tr>
<tr>
<td>W2  1234</td>
<td>W2  4000</td>
</tr>
<tr>
<td>W3  C9BA</td>
<td>W3  FFF8</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>

**Example 2:**

MUL.SU W2, #0x10, W0 ; Multiply W2 by literal 0x10

; Store the result to W0:W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  ABCD</td>
<td>W0  2400</td>
</tr>
<tr>
<td>W1  89B3</td>
<td>W1  000F</td>
</tr>
<tr>
<td>W2  F240</td>
<td>W2  F240</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
**MUL.SU**  
**Integer 16x16-Bit Signed-Unsigned Multiply**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label;} MUL.SU Wb, Ws, Wnd

- [Ws],
- [Ws++],
- [Ws--],
- [+Ws],
- [-Ws],

**Operands:**

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wnd ∈ [W0, W2, W4 ... W12]

**Operation:**

signed (Wb) * unsigned (Ws) → Wnd:Wnd + 1

**Status Affected:**

None

**Encoding:**

| 1011 | 1001 | 0www | wddd | dppp | ssss |

**Description:**

Multiply the contents of Wb with the contents of Ws and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. The Wb operand and the result Wnd are interpreted as a two’s complement signed integer. The Ws operand is interpreted as an unsigned integer. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Register Indirect Addressing may be used for Ws.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the address of the lower destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: Since the product of the multiplication is 32 bits, Wnd must be an even Working register. See Figure 4-3 for information on how double words are aligned in memory.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:** 1

**Cycles:** 1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

---

© 2005-2018 Microchip Technology Inc.
Example 1: \texttt{MUL.SU W8, [W9], W0} ; Multiply W8*[W9]
\hspace{1cm} ; Store the result to W0:W1
\begin{tabular}{|c|c|}
\hline
Before & After \\
\hline
Instruction & Instruction \\
\hline
W0 & 68DC \\
W1 & AA40 \\
W8 & F000 \\
W9 & 178C \\
Data 178C & F000 \\
SR & 0000 \\
\hline
W0 & 0000 \\
W1 & F100 \\
W8 & F000 \\
W9 & 178C \\
Data 178C & F000 \\
SR & 0000 \\
\hline
\end{tabular}

Example 2: \texttt{MUL.SU W2, [++W3], W4} ; Pre-Increment W3
\hspace{1cm} ; Multiply W2*[W3]
\hspace{1cm} ; Store the result to W4:W5
\begin{tabular}{|c|c|}
\hline
Before & After \\
\hline
Instruction & Instruction \\
\hline
W2 & 0040 \\
W3 & 0280 \\
W4 & 1819 \\
W5 & 2021 \\
Data 0282 & 0068 \\
SR & 0000 \\
\hline
W2 & 0040 \\
W3 & 0282 \\
W4 & 1A00 \\
W5 & 0000 \\
Data 0282 & 0068 \\
SR & 0000 \\
\hline
\end{tabular}
### MUL.SU

**Integer 16x16-Bit Signed-Unsigned Multiply with Accumulator Destination**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} MUL.SU Wb, Ws, Acc

-[Ws],
-[Ws++],
-[Ws--],
[++Ws],
[--Ws],

**Operands:**

Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Acc ∈ [A, B]

**Operation:**

signed (Wb) * unsigned (Ws) → Acc(A or B)

**Status Affected:**

None

**Encoding:**

```
1011 1001 0www w111 Appp ssss
```

**Description:**

Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is sign-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). The first source operand is interpreted as a two's complement signed value and the second source operand is interpreted as an unsigned value.

The ‘w’ bits select the address of the base register.
The ‘p’ bits select Source Addressing Mode 2.
The ‘A’ bit selects the destination accumulator for the product.

**Note 1:**

- This instruction operates in Word mode only.
- The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

**Words:**

1

**Cycles:**

1\(^{(1)}\)

**Note 1:**
In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3 in Section 3.2.1 “Multi-Cycle Instructions”**.

**Example 1:**

MUL.SU W8, W9, A

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W8</td>
<td>W8</td>
</tr>
<tr>
<td>W9</td>
<td>W9</td>
</tr>
<tr>
<td>ACCA</td>
<td>ACCA</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>

Before Instruction:

- W8: F000
- W9: F000
- ACCA: 00 0000 0000
- SR: 0000

After Instruction:

- W8: F000
- W9: F000
- ACCA: FF F100 0000
- SR: 0000
MUL.SU

Integer 16x16-Bit Signed-Unsigned Short Literal Multiply with Accumulator Destination

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax: 

\{label:\} MUL.SU Wb, \#lit5, Acc

Operands:

Wb \in \{W0 \ldots W15\}

lit5 \in \{0 \ldots 31\}

Acc \in \{A, B\}

Operation:

signed (Wb) \times unsigned (lit5) \rightarrow Acc(A or B)

Status Affected:

None

Encoding:

\begin{array}{cccccccc}
1011 & 1001 & 0\text{www} & w111 & A\text{llk} & k\text{kkk} \\
\end{array}

Description:

Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is sign-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). The first source operand is interpreted as a two’s complement signed value and the second source operand is interpreted as an unsigned value.

The ‘w’ bits select the address of the base register.

The ‘k’ bits select the 5-bit literal value.

The ‘A’ bit selects the destination accumulator for the product.

Note 1: This instruction operates in Word mode only.

Note 2: The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

Words: 1

Cycles: 1

Example 1: MUL.SU W8, \#0x02, A

Before Instruction

| W8   | 0042 |
| ACCA | 00000 0000 |
| SR   | 0000 |

After Instruction

| W8   | 0042 |
| ACCA | 00000 0084 |
| SR   | 0000 |
Section 5. Instruction Descriptions

MUL.US  Integer 16x16-Bit Unsigned-Signed Multiply

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
{label:} MUL.US Wb, Ws, Wnd  

[Ws],  
[Ws++],  
[Ws--],  
[++Ws],  
[--Ws],

Operands:  
Wb ∈ [W0 ... W15]  
Ws ∈ [W0 ... W15]  
Wnd ∈ [W0, W2, W4 ... W12]

Operation:  
unsigned (Wb) * signed (Ws) → Wnd:Wnd + 1

Status Affected:  
None

Encoding:  
```
1011 1000 1www wddd dppp ssss
```

Description:  
Multiply the contents of Wb with the contents of Ws and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. The Wb operand is interpreted as an unsigned integer. The Ws operand and the result Wnd are interpreted as a two’s complement signed integer. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Register Indirect Addressing may be used for Ws.  
The ‘w’ bits select the address of the base register.  
The ‘d’ bits select the address of the lower destination register.  
The ‘p’ bits select the source addressing mode.  
The ‘s’ bits select the source register.

Note 1:  
This instruction operates in Word mode only.  
2:  
Since the product of the multiplication is 32 bits, Wnd must be an even numbered Working register. See Figure 4-3 for information on how double words are aligned in memory.  
3:  
Wnd may not be W14, since W15<0> is fixed to zero.  
4:  
The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words:  
1

Cycles:  
1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.
Example 1:  MUL.US W0, [W1], W2  ; Multiply W0*[W1] (unsigned-signed)
             ; Store the result to W2:W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>W0</td>
</tr>
<tr>
<td>C000</td>
<td>C000</td>
</tr>
<tr>
<td>W1</td>
<td>W1</td>
</tr>
<tr>
<td>2300</td>
<td>2300</td>
</tr>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>00DA</td>
<td>0000</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
</tr>
<tr>
<td>CC25</td>
<td>F400</td>
</tr>
<tr>
<td>Data 2300</td>
<td>Data 2300</td>
</tr>
<tr>
<td>F000</td>
<td>F000</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>W5</td>
</tr>
<tr>
<td>0C00</td>
<td>0C00</td>
</tr>
<tr>
<td>W6</td>
<td>W6</td>
</tr>
<tr>
<td>FFFF</td>
<td>FFFF</td>
</tr>
<tr>
<td>W10</td>
<td>W10</td>
</tr>
<tr>
<td>0908</td>
<td>8001</td>
</tr>
<tr>
<td>W11</td>
<td>W11</td>
</tr>
<tr>
<td>6EEB</td>
<td>7FFE</td>
</tr>
<tr>
<td>Data 0C00</td>
<td>Data 0C00</td>
</tr>
<tr>
<td>7FFF</td>
<td>7FFF</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
## MUL.US

**Integer 16x16-Bit Unsigned-Signed Multiply with Accumulator Destination**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

{label:} MUL.US \( Wb, Ws, Acc \)

- \[Ws\], \[Ws++\], \[Ws--\], \[++Ws\], \[-Ws\],

### Operands:

- \( Wb \in [W0 \ldots W15] \)
- \( Ws \in [W0 \ldots W15] \)
- \( Acc \in [A, B] \)

### Operation:

Unsigned (Wb) * signed (Ws) \( \rightarrow Acc(A \ or \ B) \)

### Status Affected:

None

### Encoding:

| 1011 | 1000 | 0www | w111 | Appp | ssss |

### Description:

Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is sign-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). The first source operand is interpreted as an unsigned value and the second source operand is interpreted as a two’s complement signed value.

The ‘w’ bits select the address of the base register.
The ‘p’ bits select Source Addressing Mode 2.
The ‘A’ bit selects the destination accumulator for the product.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

### Words: 1

### Cycles: 1\(^{(1)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in [Section 3.2.1 “Multi-Cycle Instructions”](#).

**Example 1:**

MUL.US \( W0, W1, B \)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W0 )</td>
<td>( W0 )</td>
</tr>
<tr>
<td>( W1 )</td>
<td>( W1 )</td>
</tr>
<tr>
<td>( ACCB )</td>
<td>( ACCB )</td>
</tr>
<tr>
<td>( SR )</td>
<td>( SR )</td>
</tr>
</tbody>
</table>
MUL.UU  Integer 16x16-Bit Unsigned Short Literal Multiply

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
{label;}  MUL.UU  Wb,  #lit5,  Wnd

Operands:

- Wb ∈ [W0 ... W15]
- lit5 ∈ [0 ... 31]
- Wnd ∈ [W0, W2, W4 ... W12]

Operation:

unsigned (Wb) * unsigned lit5 → Wnd:Wnd + 1

Status Affected: None

Encoding:

1011 1000 0www wddd d11k kkkk

Description:

Multiply the contents of Wb with the 5-bit literal and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. Both operands and the result are interpreted as unsigned integers. Register Direct Addressing must be used for Wb and Wnd.

Note 1: This instruction operates in Word mode only.
2: Since the product of the multiplication is 32 bits, Wnd must be an even Working register. See Figure 4-3 for information on how double words are aligned in memory.
3: Wnd may not be W14, since W15<0> is fixed to zero.
4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words: 1
Cycles: 1

Example 1:

MUL.UU  W0,  #0xF,  W12 ; Multiply W0 by literal 0xF
; Store the result to W12:W13

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>2323</td>
</tr>
<tr>
<td>W12</td>
<td>4512</td>
</tr>
<tr>
<td>W13</td>
<td>7821</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

MUL.UU  W7,  #0x1F,  W0 ; Multiply W7 by literal 0x1F
; Store the result to W0:W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>780B</td>
</tr>
<tr>
<td>W1</td>
<td>3805</td>
</tr>
<tr>
<td>W7</td>
<td>F240</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**MUL.UU**

**Integer 16x16-Bit Unsigned Multiply**

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} MUL.UU Wb, Ws, Wnd

[Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands: Wb $\in$ [W0 ... W15]

Ws $\in$ [W0 ... W15]

Wnd $\in$ [W0, W2, W4 ... W12]

Operation: unsigned (Wb) * unsigned (Ws) $\rightarrow$ Wnd:Wnd + 1

Status Affected: None

Encoding: 1011 1000 0www wddd dppp ssss

Description: Multiply the contents of Wb with the contents of Ws and store the 32-bit result in two successive Working registers. The least significant word of the result is stored in Wnd (which must be an even numbered Working register) and the most significant word of the result is stored in Wnd + 1. Both source operands and the result are interpreted as unsigned integers. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Indirect Addressing may be used for Ws.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the address of the lower destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: Since the product of the multiplication is 32 bits, Wnd must be an even numbered Working register. See Figure 4-3 for information on how double words are aligned in memory.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words: 1

Cycles: 1(1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multi-Cycle Instructions”.
**Example 1:**

\`\`MUL.UU W4, W0, W2 \`

; Multiply W4*W0 (unsigned-unsigned)

; Store the result to W2:W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>W0</td>
</tr>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
</tr>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>

Before Instruction: | After Instruction:
W0: FFFF | W0: FFFF
W2: 2300 | W2: 0001
W3: 00DA | W3: FFFE
W4: FFFF | W4: FFFF
SR: 0000 | SR: 0000

**Example 2:**

\`\`MUL.UU W0, [W1++], W4 \`

; Mult. W0*[W1] (unsigned-unsigned)

; Store the result to W4:W5

; Post-Increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>W0</td>
</tr>
<tr>
<td>W1</td>
<td>W1</td>
</tr>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>W5</td>
<td>W5</td>
</tr>
<tr>
<td>Data 2300</td>
<td>Data 2300</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>

Before Instruction: | After Instruction:
W0: 1024 | W0: 1024
W1: 2300 | W1: 2302
W4: 9654 | W4: 6D34
W5: BDBC | W5: 0D80
Data 2300: D625 | Data 2300: D625
SR: 0000 | SR: 0000
**MUL.UU**

Integer 16x16-Bit Unsigned Multiply with Accumulator Destination

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
</table>

Syntax: `{label:} MUL.UU Wb, Ws, Acc

[Ws],

[Ws++],

[Ws--],

[++Ws],

[--Ws],

Operands: 

Wb ∈ [W0 ... W15]

Ws ∈ [W0 ... W15]

Acc ∈ [A, B]

Operation: 

unsigned (Wb) * unsigned (Ws) → Acc(A or B)

Status Affected: None

Encoding:

| 1011 | 1000 | 0www | w111 | Appp | ssss |

Description:

Performs a 16-bit x 16-bit unsigned multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is zero-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). Both source operands are treated as unsigned values.

The 'w' bits select the address of the base register.

The 'p' bits select Source Addressing Mode 2.

The 'A' bit selects the destination accumulator for the product.

Note 1: This instruction operates in Word mode only.

2: The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

Words: 1

Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

Example 1: MUL.UU W4, W0, B

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>W4</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>ACCB</td>
<td>00 0000 0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
MUL.UU

Integer 16x16-Bit Unsigned Short Literal Multiply with Accumulator Destination

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{label:} MUL.UU Wb, #lit5, Acc

Operands:

Wb ∈ [W0 ... W15]

lit5 ∈ [0 ... 31]

Acc ∈ [A, B]

Operation:

unsigned (Wb) * unsigned (lit5) → Acc(A or B)

Status Affected:

None

Encoding:

1011 1000 0www w111 A1kk kkkk

Description:

Performs a 16-bit x 16-bit signed multiply with a 32-bit result, which is stored in one of the DSP engine accumulators: ACCA or ACCB. The 32-bit result is zero-extended to bit 39 prior to being loaded into the target accumulator.

The source operands are treated as integer or fractional values depending upon the operating mode of the DSP engine (as defined by the IF bit in CORCON<0>). Both source operands are treated as unsigned values.

The ‘w’ bits select the address of the base register.

The ‘k’ bits select the 5-bit literal.

The ‘A’ bit selects the destination accumulator for the product.

Note 1: This instruction operates in Word mode only.

2: The state of the Multiplier mode bits (US<1:0> in CORCON) has no effect upon the operation of this instruction.

Words: 1

Cycles: 1

Example 1:

MUL.UU W8, #0x02, A
**MULW.SS**

**Integer 16x16-Bit Signed Multiply with 16-Bit Result**

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{\text{label:}} \text{MULW.SS} \ Wb, \ Ws, \ Wnd

\[\text{Ws},\ \text{Ws}++,\ \text{Ws}--\],

\[++\text{Ws},\ --\text{Ws}\],

Operands:

- \text{Wb} \in [W0 \ldots W15]
- \text{Ws} \in [W0 \ldots W15]
- \text{Wnd} \in [W0, W2, W4 \ldots W12]

Operation:

\text{signed (Wb)} \times \text{signed (Ws)} \rightarrow \text{Wnd}

Status Affected:

None

Encoding:

| 1011 | 1001 | lwww | wddd | dppp | ssss |

Description:

Multiply the contents of \text{Wb} with the contents of \text{Ws} and store the result in a Working register, which must be an even numbered Working register. Both source operands and the result \text{Wnd} are interpreted as two’s complement signed integers. Register Direct Addressing must be used for \text{Wb} and \text{Wnd}. Register Direct or Register Indirect Addressing may be used for \text{Ws}.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the address of the lower destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: \text{Wnd} must be an even numbered Working register.

3: \text{Wnd} may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words:

1

Cycles:

1(t)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multi-Cycle Instructions”.
**Example 1:**  
```
MULW.SS W0, W1, W12 ; Multiply W0*W1
; Store the result to W12
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 9823</td>
<td>W0 9823</td>
</tr>
<tr>
<td>W1 67DC</td>
<td>W1 67DC</td>
</tr>
<tr>
<td>W12 FFFF</td>
<td>W12 D314</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**  
```
MULW.SS W2, [--W4], W0 ; Pre-decrement W4
; Multiply W2*[W4]
; Store the result to W0
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 FFFF</td>
<td>W0 28F8</td>
</tr>
<tr>
<td>W2 0045</td>
<td>W2 0045</td>
</tr>
<tr>
<td>W4 27FE</td>
<td>W4 27FC</td>
</tr>
<tr>
<td>Data 27FC</td>
<td>Data 27FC</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### MULW.SU

**Integer 16x16-Bit Signed-Unsigned Multiply with 16-Bit Result**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} MULW.SU Wb, Ws, Wnd
```

**Operands:**

- **Wb** ∈ [W0 ... W15]
- **Ws** ∈ [W0 ... W15]
- **Wnd** ∈ [W0, W2, W4 ... W12]

**Operation:**

```
signed (Wb) * unsigned (Ws) → Wnd
```

**Status Affected:**

None

**Encoding:**

```
1011 1001 0www wddd dppp ssss
```

**Description:**

Multiply the contents of Wb with the contents of Ws and store the result in a Working register, which must be an even numbered Working register. The Wb operand and the result Wnd are interpreted as a two’s complement signed integer. The Ws operand is interpreted as an unsigned integer. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Register Indirect Addressing may be used for Ws.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the address of the lower destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: Wnd must be an even numbered Working register.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:**

1

**Cycles:**

1\(^{(1)}\)

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.
**Example 1:**  
MULW.SU W8, [W9], W0  
; Multiply W8*W9  
; Store the result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>W0</td>
</tr>
<tr>
<td>W8</td>
<td>W8</td>
</tr>
<tr>
<td>W9</td>
<td>W9</td>
</tr>
<tr>
<td>Data 178C</td>
<td>Data 178C</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>

**Example 2:**  
MULW.SU W2, [++W3], W4  
; Pre-Increment W3  
; Multiply W2*[W3]  
; Store the result to W4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
</tr>
<tr>
<td>W4</td>
<td>W4</td>
</tr>
<tr>
<td>Data 0282</td>
<td>Data 0282</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
</tbody>
</table>
### MULW.SU

**Integer 16x16-Bit Signed-Unsigned Short Literal Multiply with 16-Bit Result**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label;} MULW.SU Wb, #lit5, Wnd

**Operands:**

- Wb ∈ [W0 ... W15]
- lit5 ∈ [0 ... 31]
- Wnd ∈ [W0, W2, W4 ... W12]

**Operation:**

signed (Wb) * unsigned (lit5) → Wnd

**Status Affected:**

None

**Encoding:**

```
1011 1001 0www wddd d11k kkkk
```

**Description:**

Multiply the contents of Wb with a 5-bit literal value and store the result in a Working register, which must be an even numbered Working register. The Wb operand and the result Wnd are interpreted as a two's complement signed integer. Register Direct Addressing must be used for Wb and Wnd.

- The ‘w’ bits select the address of the base register.
- The ‘d’ bits select the address of the lower destination register.
- The ‘k’ bits select the 5-bit literal value.

**Note 1:** This instruction operates in Word mode only.

2: Wnd must be an even numbered Working register.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:**

1

**Cycles:**

1

**Example 1:**

MULW.SU W8, #0x04, W0 ; Multiply W8 * #0x04

; Store the result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 68DC</td>
<td>W0 4000</td>
</tr>
<tr>
<td>W8 1000</td>
<td>W8 1000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
### MULW.US

**Integer 16x16-Bit Unsigned-Signed Multiply with 16-Bit Result**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
\{\text{label}\} \quad \text{MULW.US} \quad W_b, \quad W_s, \quad W_{nd}
\]

\[
[W_s],
[W_s^{++}],
[W_s^{-}],
[++W_s],
[--W_s],
\]

**Operands:**

- \(W_b \in [W_0 \ldots W_{15}]\)
- \(W_s \in [W_0 \ldots W_{15}]\)
- \(W_{nd} \in [W_0, W_2, W_4 \ldots W_{12}]\)

**Operation:**

\[
\text{unsigned (}W_b\text{) } \times \text{signed (}W_s\text{)} \rightarrow W_{nd}
\]

**Status Affected:** None

**Encoding:**

| 1011 | 1000 | lwww | wddd | dppp | ssss |

**Description:**

Multiply the contents of \(W_b\) with the contents of \(W_s\) and store the result in a Working register, which must be an even numbered Working register. The \(W_b\) operand is interpreted as an unsigned integer. The \(W_s\) operand and the result \(W_{nd}\) are interpreted as a two’s complement signed integer. Register Direct Addressing must be used for \(W_b\) and \(W_{nd}\). Register Direct or Register Indirect Addressing may be used for \(W_s\).

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the address of the lower destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** \(W_{nd}\) must be an even numbered Working register.

**Note 3:** \(W_{nd}\) may not be \(W_{14}\), since \(W_{15}<0>\) is fixed to zero.

**Note 4:** The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:** 1

**Cycles:** \(1^{(1)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multi-Cycle Instructions”.
### Example 1:

**MULW.US W0, [W1], W2**  
; Multiply W0*[W1] (unsigned-signed)  
; Store the result to W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>C000</td>
</tr>
<tr>
<td>W1</td>
<td>2300</td>
</tr>
<tr>
<td>W2</td>
<td>00DA</td>
</tr>
<tr>
<td>Data 2300</td>
<td>F000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:

**MULW.US W6, [W5++], W10**  
; Mult. W6*[W5] (unsigned-signed)  
; Store the result to W10  
; Post-Increment W5

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>0C00</td>
</tr>
<tr>
<td>W6</td>
<td>FFFF</td>
</tr>
<tr>
<td>W10</td>
<td>0908</td>
</tr>
<tr>
<td>Data 0C00</td>
<td>7FFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
MULW.UU

Integer 16x16-Bit Unsigned Multiply with 16-Bit Result

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} MULW.UU Wb, Ws, Wnd

Ws, [Ws], [Ws++], [Ws--], ++Ws, --Ws

Operands:

Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Wnd ∈ [W0, W2, W4 ... W12]

Operation:

unsigned (Wb) * unsigned (Ws) → Wnd

Status Affected:

None

Encoding:

```
 1011 1000 0www wddd dppp ssss
```

Description:

Multiply the contents of Wb with the contents of Ws and store the result in a Working register, which must be an even numbered Working register. Both source operands and the result are interpreted as unsigned integers. Register Direct Addressing must be used for Wb and Wnd. Register Direct or Indirect Addressing may be used for Ws.

The 'w' bits select the address of the base register.
The 'd' bits select the address of the lower destination register.
The 'p' bits select the source addressing mode.
The 's' bits select the source register.

**Note 1:** This instruction operates in Word mode only.

2: Wnd must be an even numbered Working register.

3: Wnd may not be W14, since W15<0> is fixed to zero.

4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

Words:

1

Cycles:

1(1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.**

Example 1:

```
MULW.UU W4, W0, W2 ; Multiply W4*W0 (unsigned-unsigned)
                   ; Store the result to W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0000</td>
</tr>
<tr>
<td>W2</td>
<td>0000</td>
</tr>
<tr>
<td>W4</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

### MULW.UU

**Integer 16x16-Bit Unsigned Short Literal Multiply with 16-Bit Result**

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**
{label:} MULW.UU Wb, #lit5, Wnd

**Operands:**
- Wb ∈ [W0 ... W15]
- lit5 ∈ [0 ... 31]
- Wnd ∈ [W0, W2, W4 ... W12]

**Operation:**
unsigned (Wb) * unsigned → Wnd

**Status Affected:**
None

**Encoding:**
```
1011 1000 0www wddd d11k kkkk
```

**Description:**
Multiply the contents of Wb with a 5-bit literal value and store the result in a Working register, which must be an even numbered Working register. Both source operands and the result are interpreted as unsigned integers. Register Direct Addressing must be used for Wb and Wnd.

The 'w' bits select the address of the base register.
The 'd' bits select the address of the lower destination register.
The 'k' bits select the 5-bit literal value.

**Note 1:** This instruction operates in Word mode only.
2: Wnd must be an even numbered Working register.
3: Wnd may not be W14, since W15<0> is fixed to zero.
4: The IF bit and the US<1:0> bits in the CORCON register have no effect on this operation.

**Words:** 1
**Cycles:** 1

**Example 1:**
MULW.UU W4, #0x04, W2 ; Multiply W4*W0 (unsigned-unsigned)
; Store the result to W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 2300</td>
<td>W2 4000</td>
</tr>
<tr>
<td>W4 1000</td>
<td>W4 1000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
NEG

Negate f

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} NEG{.B} f {,WREG}

Operands:

f \in [0 \ldots 8191]

Operation:

\((f) + 1 \rightarrow \text{destination designated by D})

Status Affected:

DC, N, OV, Z, C

Encoding:

\[
\begin{array}{cccc}
1110 & 1110 & 0BDf & ffff \\
\end{array}
\]

Description:

Compute the two’s complement of 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 ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: (1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:

NEG.B 0x880, WREG ; Negate (0x880) (Byte mode)

; Store result to WREG

Before Instruction

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>Data 0880</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>9080</td>
<td>2355</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>Data 0880</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>90AB</td>
<td>2355</td>
<td>0008 (N = 1)</td>
</tr>
</tbody>
</table>

SR = 0000

Example 2:

NEG 0x1200 ; Negate (0x1200) (Word mode)

Before Instruction

<table>
<thead>
<tr>
<th>Data 1200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>8923</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>Data 1200</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>76DD</td>
<td>0000</td>
</tr>
</tbody>
</table>
### Section 5. Instruction Descriptions

#### NEG
Negate Ws

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
\text{(label:)} \quad \text{NEG}[,B] \quad \text{Ws}, \quad \text{Wd}
\]

- \([\text{Ws}], \quad \text{[Wd]}\)
- \([\text{Ws}++], \quad \text{[Wd++]}\)
- \([\text{Ws}--], \quad \text{[Wd--]}\)
- \([++\text{Ws}], \quad [++\text{Wd}]\)
- \([-\text{Ws}], \quad [-\text{Wd}]\)

**Operands:**

- \(\text{Ws} \in [W0 \ldots W15]\)
- \(\text{Wd} \in [W0 \ldots W15]\)

**Operation:**

\((\text{Ws}) + 1 \rightarrow \text{Wd}\)

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

<table>
<thead>
<tr>
<th>B</th>
<th>q</th>
<th>d</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>1010</td>
<td>0Bqq</td>
<td>qddd</td>
<td>dppp</td>
</tr>
</tbody>
</table>

**Description:**

Compute the two’s complement of the contents of the source register Ws and place the result in the destination register Wd. Either Register Direct or Indirect Addressing may be used for both Ws and Wd.

- The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).
- The ‘q’ bits select the destination addressing mode.
- The ‘d’ bits select the destination register.
- The ‘p’ bits select the source addressing mode.
- The ‘s’ bits select the source register.

**Note:** 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

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

---

© 2005-2018 Microchip Technology Inc. DS70000157G-page 351
**Example 1:** `NEG.B W3, [W4++]`  
; Negate W3 and store to [W4] (Byte mode)  
; Post-increment W4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3  7839</td>
<td>W3  7839</td>
</tr>
<tr>
<td>W4  1005</td>
<td>W4  1006</td>
</tr>
<tr>
<td>Data 1004  2355</td>
<td>Data 1004  C755</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0008 (N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:** `NEG [W2++], [--W4]`  
; Pre-decrement W4 (Word mode)  
; Negate [W2] and store to [W4]  
; Post-increment W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2  0900</td>
<td>W2  0902</td>
</tr>
<tr>
<td>W4  1002</td>
<td>W4  1000</td>
</tr>
<tr>
<td>Data 0900  870F</td>
<td>Data 0900  870F</td>
</tr>
<tr>
<td>Data 1000  5105</td>
<td>Data 1000  78F1</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
NEG

Negate Accumulator

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

X X X X

Syntax: {label;} NEG Acc

Operands: Acc ∈ [A,B]

Operation:
If (Acc = A):
- ACCA → ACCA
Else:
- ACCB → ACCB

Status Affected: OA, OB, OAB, SA, SB, SAB

Encoding:

| 1100 | 1011 | A001 | 0000 | 0000 | 0000 |

Description:
Compute the two’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 specifies the selected accumulator.

Words: 1
Cycles: 1

Example 1:
NEG A ; Negate ACCA
; Store result to ACCA
; CORCON = 0x0000 (no saturation)

Before Instruction
ACCA 00 3290 59C8
CORCON 0000
SR 0000

After Instruction
ACCA FF CD6F A638
CORCON 0000
SR 0000

Example 2:
NEG B ; Negate ACCB
; Store result to ACCB
; CORCON = 0x00C0 (normal saturation)

Before Instruction
ACCB FF F230 10DC
CORCON 00C0
SR 0000

After Instruction
ACCB 00 0DCF EF24
CORCON 00C0
SR 0000
## NOP

No Operation

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

{label:} NOP

### Operands:
None

### Operation:
No Operation

### Status Affected:
None

### Encoding:

| 0000 | 0000 | xxxx | xxxx | xxxx | xxxx |

### Description:

No Operation is performed.
The 'x' bits can take any value.

### Words:

1

### Cycles:

1

---

**Example 1:**

NOP ; execute no operation

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC: 00 1092</td>
<td>PC: 00 1094</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

NOP ; execute no operation

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC: 00 08AE</td>
<td>PC: 00 08B0</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>
##NOPR No Operation

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**  
[label:] NOPR

**Operands:** None

**Operation:** No Operation

**Status Affected:** None

**Encoding:**

```
1111  llll  xxxx  xxxx  xxxx  xxxx
```

**Description:** No Operation is performed.  
The ‘x’ bits can take any value.

**Words:** 1

**Cycles:** 1

**Example 1:**  
NOPR ; execute no operation

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>00 2430</td>
<td>00 2432</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**  
NOPR ; execute no operation

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>00 1466</td>
<td>00 1468</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
**NORM**

**Normalize Accumulator**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>{label:} NORM Acc, Wd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operands:</td>
<td>Wnd ∈ [W0 ... W15]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc ∈ [A,B]</td>
<td>Refer to text.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Affected:</td>
<td>OA, SA or OB, SB, N, Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding:</td>
<td>1100 1110 A110 0000 0qqq dddd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Description:    | Normalize the contents of the target accumulator. If the accumulator contains an overflowed value, the contents of the accumulator are shifted right by the minimum number of bits required to remove the overflow. If the accumulator does not contain an overflowed value, the contents of the accumulator are shifted left by the minimum number of bits required to produce the largest fractional data value without an overflow.

If it is not possible to normalize the target accumulator (i.e., it is already normalized, or it is all '0's or all '1's), Wd is cleared, the Z bit is set and the N bit is cleared. The target accumulator is unaffected.

If it is possible to normalize the target accumulator, the exponent (shift value) required to normalize the target accumulator is written into Wd. A positive result indicates that a right shift of the accumulator was required for normalization. A negative result indicates that a left shift of the accumulator was required for normalization. The N bit is set to reflect the sign of the result and the Z bit is cleared.

The ‘A’ bit specifies the destination accumulator.
The ‘d’ bits select the address of the destination register.
The ‘q’ bits select Destination Address Mode 2.

**Note 1:** OA and SA or OB and SB Status bits are set based on the content of the target accumulator. Consequently, as the NORM instruction removes any overflow, OA or OB will always be cleared.

**Note 2:** The SA/SB bits will remain set if they were already set prior to execution of the NORM instruction, but these bits can never be affected by this instruction.

Words: 1
Cycles: 1
## POP

**Pop TOS to f**

### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

```
{label:} POP f
```

### Operands:

\( f \in [0 \ldots 65534] \)

### Operation:

\[
(W15) - 2 \rightarrow W15 \\
(TOS) \rightarrow f
\]

### Status Affected:

None

### Encoding:

```
1111 1001  ffff  ffff  ffff  fff0
```

### Description:

The Stack Pointer (W15) is pre-decremented by 2 and the Top-of-Stack (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 address of the file register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** The file register address must be word-aligned.

### Words:

1

### Cycles:

1

**Example 1:**

```
POP 0x1230 ; Pop TOS to 0x1230
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15 1006</td>
<td>W15 1004</td>
</tr>
<tr>
<td>Data 1004 A401</td>
<td>Data 1004 A401</td>
</tr>
<tr>
<td>Data 1230 2355</td>
<td>Data 1230 A401</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
POP 0x880 ; Pop TOS to 0x880
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15 2000</td>
<td>W15 1FFE</td>
</tr>
<tr>
<td>Data 0880 E3E1</td>
<td>Data 0880 A090</td>
</tr>
<tr>
<td>Data 1FFE A090</td>
<td>Data 1FFE A090</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
**POP**

### Pop TOS to Wd

#### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```
(label:) POP Wd

[Wd]

[Wd++]

[Wd--]

[--Wd]

[++]Wd

[Wd+Wb]
```

#### Operands:

- **Wd**: `Wd ∈ [W0 ... W15]`
- **Wb**: `Wb ∈ [W0 ... W15]`

#### Operation:

- `(W15) – 2 → W15`
- `(TOS) → Wd`

#### Status Affected: None

#### Encoding:

```
0111 1www w0hh hddd d100 1111
```

#### Description:

The Stack Pointer (W15) is pre-decremented by 2 and the Top-of-Stack (TOS) word is written to Wd. Either Register Direct or Indirect Addressing may be used for Wd.

The ‘w’ bits define the offset register Wb.

The ‘h’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** This instruction is a specific version of the “MOV Ws, Wd” instruction, `(MOV [--W15], Wd)`; it reverse assembles as MOV.

#### Words: 1

#### Cycles: 1

### Example 1:

```assembly
POP W4 ; Pop TOS to W4
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 EDA8</td>
<td>W4 C45A</td>
</tr>
<tr>
<td>W15 1008</td>
<td>W15 1006</td>
</tr>
<tr>
<td>Data 1006 C45A</td>
<td>Data 1006 C45A</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

### Example 2:

```assembly
POP [++]W10 ; Pre-increment W10
; Pop TOS to [W10]
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W10 0E02</td>
<td>W10 0E04</td>
</tr>
<tr>
<td>W15 1766</td>
<td>W15 1764</td>
</tr>
<tr>
<td>Data 0E04 E3E1</td>
<td>Data 0E04 C7B5</td>
</tr>
<tr>
<td>Data 1764 C7B5</td>
<td>Data 1764 C7B5</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

---

**POP.D**

Double Pop TOS to Wnd:Wnd+1

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} POP.D Wnd

**Operands:**

Wnd ∈ [W0, W2, W4, ... W14]

**Operation:**

\[
(W15) - 2 \rightarrow W15 \\
(TOS) \rightarrow Wnd + 1 \\
(W15) - 2 \rightarrow W15 \\
(TOS) \rightarrow Wnd
\]

**Status Affected:** None

**Encoding:**

```
1011 1110 0000 0ddd 0100 1111
```

**Description:** A double word is POPped from the Top-of-Stack (TOS) and stored to Wnd:Wnd + 1. The most significant word is stored to Wnd + 1 and the least significant word is stored to Wnd. Since a double word is POPped, the Stack Pointer (W15) gets decremented by 4.

The ‘d’ bits select the address of the destination register pair.

**Note 1:** This instruction operates on double words. See Figure 4-3 for information on how double words are aligned in memory.

**Note 2:** Wnd must be an even numbered Working register.

**Note 3:** This instruction is a specific version of the "MOV.D Ws, Wnd" instruction, (MOV.D [--W15], Wnd); it reverse assembles as MOV.D.

**Words:** 1

**Cycles:** 2

**Example 1:** POP.D W6 ; Double pop TOS to W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6 07BB</td>
<td>W6 3210</td>
</tr>
<tr>
<td>W7 89AE</td>
<td>W7 7654</td>
</tr>
<tr>
<td>W15 0850</td>
<td>W15 084C</td>
</tr>
<tr>
<td>Data 084C 3210</td>
<td>Data 084C 3210</td>
</tr>
<tr>
<td>Data 084E 7654</td>
<td>Data 084E 7654</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:** POP.D W0 ; Double pop TOS to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 673E</td>
<td>W0 791C</td>
</tr>
<tr>
<td>W1 DD23</td>
<td>W1 D400</td>
</tr>
<tr>
<td>W15 0BBC</td>
<td>W15 0BB8</td>
</tr>
<tr>
<td>Data 0BB8 791C</td>
<td>Data 0BB8 791C</td>
</tr>
<tr>
<td>Data 0BBA D400</td>
<td>Data 0BBA D400</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
POP.S Pop Shadow Registers

Implemented in:  

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} POP.S

Operands:  
None

Operation:  
POP shadow registers.

Status Affected:  
DC, N, OV, Z, C

Encoding:  

```
1111 1110 1000 0000 0000 0000
```

Description:  
The values in the shadow registers are copied into their respective primary registers. The following registers are affected: W0-W3, and the C, Z, OV, N and DC STATUS Register flags.

**Note 1:** The shadow registers are not directly accessible. They may only be accessed with **PUSH.S** and **POP.S**.

**Note 2:** The shadow registers are only one-level deep.

Words:  
1

Cycles:  
1

**Example 1:**  

```
POP.S ; Pop the shadow registers
; (See PUSH.S Example 1 for contents of shadows)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 07BB</td>
<td>W0 0000</td>
</tr>
<tr>
<td>W1 03FD</td>
<td>W1 1000</td>
</tr>
<tr>
<td>W2 9610</td>
<td>W2 2000</td>
</tr>
<tr>
<td>W3 7249</td>
<td>W3 3000</td>
</tr>
<tr>
<td>SR 00E0 (IPL = 7)</td>
<td>SR 00E1 (IPL = 7, C = 1)</td>
</tr>
</tbody>
</table>

**Note:** After instruction execution, the contents of shadow registers are NOT modified.
Section 5. Instruction Descriptions

PUSH

Push \( f \) to TOS

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( X )</td>
<td>( X )</td>
<td>( X )</td>
<td>( X )</td>
<td>( X )</td>
<td>( X )</td>
<td>( X )</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} PUSH \( f \)

Operands:

\( f \in [0 \ldots 65534] \)

Operation:

\((f) \rightarrow (TOS)\)

\((W15) + 2 \rightarrow W15\)

Status Affected:

None

Encoding:

```
1111 1000  ffff  ffff  ffff  fff0
```

Description:

The contents of the specified file register are written to the Top-of-Stack (TOS) location and then the Stack Pointer (W15) is incremented by 2.

The file register may reside anywhere in the lower 32K words of data memory.

The ‘\( f \)’ bits select the address of the file register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** The file register address must be word-aligned.

Words:

1

Cycles:

1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

**Example 1:**

```
PUSH 0x2004 ; Push (0x2004) to TOS
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15  0B00</td>
<td>W15  0B02</td>
</tr>
<tr>
<td>Data 0B00  791C</td>
<td>Data 0B00  D400</td>
</tr>
<tr>
<td>Data 2004  D400</td>
<td>Data 2004  D400</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
PUSH 0xC0E ; Push (0xC0E) to TOS
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15  0920</td>
<td>W15  0922</td>
</tr>
<tr>
<td>Data 0920  0000</td>
<td>Data 0920  67AA</td>
</tr>
<tr>
<td>Data 0C0E  67AA</td>
<td>Data 2004  67AA</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
### PUSH

**Push Ws to TOS**

#### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

\[
\{\text{label:}\} \quad \text{PUSH} \quad \text{Ws} \\
\{\text{Ws}\} \\
\{\text{Ws+}\} \\
\{\text{Ws-}\} \\
\{\text{--W}\} \\
\{++\text{W}\} \\
\{\text{W}+\text{W}\}
\]

#### Operands:

- **Ws** \(\in [W0 \ldots W15]\)
- **Wb** \(\in [W0 \ldots W15]\)

#### Operation:

\[(\text{Ws}) \rightarrow (\text{TOS})\]

\[(W15) + 2 \rightarrow W15\]

#### Status Affected:

None

#### Encoding:

\[
0111, 1\text{www}, 0001, 1111, 1\text{ggg}, \text{sss}s
\]

#### Description:

The contents of Ws are written to the Top-of-Stack (TOS) location and then the Stack Pointer (W15) is incremented by 2.

The 'w' bits define the offset register Wb.

The 'g' bits select the source addressing mode.

The 's' bits select the source register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** This instruction is a specific version of the "MOV Ws, Wd" instruction, (MOV Ws, [W15++]); it reverse assembles as MOV.

#### Words:

1

#### Cycles:

1\(^{(1)}\)

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

---

1\(^{(1)}\):
### Example 1: PUSH W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 6889</td>
<td>W2 6889</td>
</tr>
<tr>
<td>W15 1566</td>
<td>W15 1568</td>
</tr>
<tr>
<td>Data 1566</td>
<td>Data 1566</td>
</tr>
<tr>
<td>0000</td>
<td>6889</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Before instruction:
- W2: 6889
- W15: 1566
- Data 1566: 0000
- SR: 0000

After instruction:
- W2: 6889
- W15: 1568
- Data 1566: 6889
- SR: 0000

### Example 2: PUSH [W5+W10]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5 1200</td>
<td>W5 1200</td>
</tr>
<tr>
<td>W10 0044</td>
<td>W10 0044</td>
</tr>
<tr>
<td>W15 0806</td>
<td>W15 0808</td>
</tr>
<tr>
<td>Data 0806</td>
<td>Data 0806</td>
</tr>
<tr>
<td>216F</td>
<td>B20A</td>
</tr>
<tr>
<td>Data 1244</td>
<td>Data 1244</td>
</tr>
<tr>
<td>B20A</td>
<td>B20A</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Before instruction:
- W5: 1200
- W10: 0044
- W15: 0806
- Data 0806: 216F
- Data 1244: B20A
- SR: 0000

After instruction:
- W5: 1200
- W10: 0044
- W15: 0808
- Data 0806: B20A
- Data 1244: B20A
- SR: 0000
### PUSH.D

**Double Push Wns:Wns+1 to TOS**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} PUSH.D Wns

**Operands:**

Wns ∈ [W0, W2, W4 ... W14]

**Operation:**

(Wns) → (TOS)  
(W15) + 2 → W15  
(Wns + 1) → (TOS)  
(W15) + 2 → W15

**Status Affected:** None

**Encoding:**

```
1011 1110 1001 1111 1000 sss0
```

**Description:**

A double word (Wns:Wns + 1) is PUSHed to the Top-of-Stack (TOS). The least significant word (Wns) is PUSHed to the TOS first and the most significant word (Wns + 1) is PUSHed to the TOS last. Since a double word is PUSHed, the Stack Pointer (W15) gets incremented by 4.

The ‘s’ bits select the address of the source register pair.

**Note 1:** This instruction operates on double words. See Figure 4-3 for information on how double words are aligned in memory.

**Note 2:** Wns must be an even numbered Working register.

**Note 3:** This instruction is a specific version of the "MOV.D Wns, Wd" instruction, (MOV.D Wns, [W15++]); it reverse assembles as MOV.D.

**Words:** 1

**Cycles:** 2

---

**Example 1:**

```
PUSH.D W6 ; Push W6:W7 to TOS
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6 C451</td>
<td>W6 C451</td>
</tr>
<tr>
<td>W7 3380</td>
<td>W7 3380</td>
</tr>
<tr>
<td>W15 1240</td>
<td>W15 1244</td>
</tr>
<tr>
<td>Data 1240 B004</td>
<td>Data 1240 C451</td>
</tr>
<tr>
<td>Data 1242 0891</td>
<td>Data 1242 3380</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
PUSH.D W10 ; Push W10:W11 to TOS
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W10 80D3</td>
<td>W10 80D3</td>
</tr>
<tr>
<td>W11 4550</td>
<td>W11 4550</td>
</tr>
<tr>
<td>W15 00C8</td>
<td>W15 00C8</td>
</tr>
<tr>
<td>Data 00C8 79B5</td>
<td>Data 00C8 80D3</td>
</tr>
<tr>
<td>Data 00CA 008E</td>
<td>Data 00CA 4550</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
PUSH.S

Push Shadow Registers

Implemented in: PIC24F  PIC24H  PIC24E  dsPIC30F  dsPIC33F  dsPIC33E  dsPIC33C

Syntax: {label:} PUSH.S

Operands: None

Operation: Push shadow registers.

Status Affected: None

Encoding:

| 1111 | 1110 | 1010 | 0000 | 0000 | 0000 |

Description: The contents of the primary registers are copied into their respective shadow registers. The following registers are shadowed: W0-W3, and the C, Z, OV, N and DC STATUS Register flags.

Note 1: The shadow registers are not directly accessible. They may only be accessed with PUSH.S and POP.S.

2: The shadow registers are only one-level deep.

Words: 1

Cycles: 1

Example 1: PUSH.S ; Push primary registers into shadow registers

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 0000</td>
<td>W0 0000</td>
</tr>
<tr>
<td>W1 1000</td>
<td>W1 1000</td>
</tr>
<tr>
<td>W2 2000</td>
<td>W2 2000</td>
</tr>
<tr>
<td>W3 3000</td>
<td>W3 3000</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0001 (C = 1)</td>
</tr>
</tbody>
</table>

Note: After an instruction execution, the contents of the shadow registers are updated.
PWRSAV

Enter Power-Saving Mode

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: \( \text{label:} \) PWRSAV #lit1

Operands: lit1 \( \in [0,1] \)

Operation:
- 0 \( \rightarrow \) WDT Count register
- 0 \( \rightarrow \) WDT Prescaler A count
- 0 \( \rightarrow \) WDT Prescaler B count
- 0 \( \rightarrow \) WDTO (RCON<4>)
- 0 \( \rightarrow \) SLEEP (RCON<3>)
- 0 \( \rightarrow \) IDLE (RCON<2>)

If \( \text{lit1} = 0 \):
  - Enter Sleep mode
Else:
  - Enter Idle mode

Status Affected: None

Encoding:

```
1111 1110 0100 0000 0000 000k
```

Description: Place the processor into the specified power-saving mode. If \( \text{lit1} = 0 \), Sleep mode is entered. In Sleep mode, the clock to the CPU and peripherals is shut down. If an on-chip oscillator is being used, it is also shut down. If \( \text{lit1} = 1 \), 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 register (RCON) are reset.

Note 1: The processor will exit from Idle or Sleep through an interrupt, processor Reset or Watchdog Timer time-out. See the specific device data sheet for details.

2: If awakened from Idle mode, the IDLE bit (RCON<2>) is set to ‘1’ and the clock source is applied to the CPU.

3: If awakened from Sleep mode, the SLEEP bit (RCON<3>) is set to ‘1’ and the clock source is started.

4: If awakened from a Watchdog Timer time-out, the WDTO bit (RCON<4>) is set to ‘1’.

Words: 1

Cycles: 1

Example 1: PWRSAV #0 ; Enter SLEEP mode

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 0040 (IPL = 2)</td>
<td>SR 0040 (IPL = 2)</td>
</tr>
</tbody>
</table>

Example 2: PWRSAV #1 ; Enter IDLE mode

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 0020 (IPL = 1)</td>
<td>SR 0020 (IPL = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

RCALL Relative Call

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC32</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  {label:} RCALL Expr

Operands: Expr may be an absolute address, label or expression. Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... 32767].

Operation:

- (PC) + 2 → PC
- (PC<15:0>) → (TOS)
- (W15) + 2 → W15
- (PC<22:16>) → (TOS)
- (W15) + 2 → W15
- (PC) + (2 * Slit16) → PC

- NOP → Instruction Register

Status Affected: None

Encoding:

| 0000 | 0111 | nnnn | nnnn | nnnn | nnnn | nnnn |

Description: Relative subroutine call with a range of 32K program words forward or backward from the current PC. Before the call is made, the return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, the sign-extended 17-bit value (2 * Slit16) is added to the contents of the PC and the result is stored in the PC.

The 'n' bits are a signed literal that specifies the size of the relative call (in program words) from (PC + 2).

Note: When possible, this instruction should be used instead of CALL, since it only consumes one word of program memory.

Words: 1
Cycles: 2
Example 1:

```
012004    RCALL _Task1    ; Call _Task1
012006    ADD W0, W1, W2
.          ...
012458    _Task1: SUB W0, W2, W3    ; _Task1 subroutine
01245A    ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  01 2004</td>
<td>PC 01 2458</td>
</tr>
<tr>
<td>W15  0810</td>
<td>W15 0814</td>
</tr>
<tr>
<td>Data 0810</td>
<td>Data 0810 2006</td>
</tr>
<tr>
<td>Data 0812</td>
<td>Data 0812 0001</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

```
00620E    RCALL _Init    ; Call _Init
006210    MOV W0, [W4++]
.          ...
007000    _Init: CLR W2    ; _Init subroutine
007002    ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 620E</td>
<td>PC 00 7000</td>
</tr>
<tr>
<td>W15  0C50</td>
<td>W15 0C54</td>
</tr>
<tr>
<td>Data 0C50</td>
<td>Data 0C50 6210</td>
</tr>
<tr>
<td>Data 0C52</td>
<td>Data 0C52 0000</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
RCALL Relative Call

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
--- | --- | --- | --- | --- | --- | --- | --- |
   |   | X |   |   | X | X |

Syntax: \{label:\} RCALL Expr

Operands: Expr may be an absolute address, label or expression.
Expr is resolved by the linker to a Slit16, where Slit16 ∈ [-32768 ... 32767].

Operation:
- \((PC) + 2 \rightarrow PC\)
- \((PC<15:1>) \rightarrow TOS<15:1>, SFA Status bit \rightarrow TOS<0>\)
- \((W15) + 2 \rightarrow W15\)
- \((PC<22:16>) \rightarrow (TOS)\)
- \((W15) + 2 \rightarrow W15\)
- \(0 \rightarrow SFA Status bit\)
- \((PC) + (2 \times \text{Slit16}) \rightarrow PC\)
- \(\text{NOP} \rightarrow \text{Instruction Register}\)

Status Affected: SFA

Encoding:

| 0000 | 0111 | nnnn | nnnn | nnnn | nnnn |

Description: Relative subroutine call with a range of 32K program words forward or backward from the current PC. Before the call is made, the return address \((PC + 2)\) is PUSHed onto the stack. After the return address is stacked, the sign-extended 17-bit value \((2 \times \text{Slit16})\) is added to the contents of the PC and the result is stored in the PC.

The ‘n’ bits are a signed literal that specifies the size of the relative call (in program words) from \((PC + 2)\).

Note: When possible, this instruction should be used instead of CALL, since it only consumes one word of program memory.

Words: 1
Cycles: 4
Example 1:

```
012004 RCALL _Task1 ; Call _Task1
012006 ADD W0, W1, W2
     ... ...
012458 _Task1: SUB W0, W2, W3 ; _Task1 subroutine
01245A ...
```

Before Instruction
<table>
<thead>
<tr>
<th>PC</th>
<th>01 2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>0810</td>
</tr>
<tr>
<td>Data 0810</td>
<td>FFFF</td>
</tr>
<tr>
<td>Data 0812</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction
<table>
<thead>
<tr>
<th>PC</th>
<th>01 2458</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>0814</td>
</tr>
<tr>
<td>Data 0810</td>
<td>2006</td>
</tr>
<tr>
<td>Data 0812</td>
<td>0001</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

```
00620E CALL _Init ; Call _Init
006210 MOV W0, [W4++]
     ... ...
007000 _Init: CLR W2 ; _Init subroutine
007002 ...
```

Before Instruction
<table>
<thead>
<tr>
<th>PC</th>
<th>00 620E</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>0C50</td>
</tr>
<tr>
<td>Data 0C50</td>
<td>FFFF</td>
</tr>
<tr>
<td>Data 0C52</td>
<td>FFFF</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction
<table>
<thead>
<tr>
<th>PC</th>
<th>00 7000</th>
</tr>
</thead>
<tbody>
<tr>
<td>W15</td>
<td>0C54</td>
</tr>
<tr>
<td>Data 0C50</td>
<td>6210</td>
</tr>
<tr>
<td>Data 0C52</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
RCALL

Computed Relative Call

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:

{label:} RCALL Wn

Operands:

Wn ∈ [W0 ... W15]

Operation:

(PC) + 2 → PC
(PC<15:0>) → (TOS)
(W15) + 2 → W15
(PC<22:16>) → (TOS)
(W15) + 2 → W15
(PC) + (2 * (Wn)) → PC

NOP → Instruction Register

Status Affected:

None

Encoding:

| 0000 | 0001 | 0010 | 0000 | 0000 | ssss |

Description:

Computed, relative subroutine call specified by the Working register Wn. The range of the call is 32K program words forward or backward from the current PC. Before the call is made, the return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, the sign-extended 17-bit value (2 * (Wn)) is added to the contents of the PC and the result is stored in the PC. Register Direct Addressing must be used for Wn.

The ‘s’ bits select the source register.

Words:

1

Cycles:

2

Example 1:

00FF8C EX1: INC W2, W3 ; Destination of RCALL
00FF8E ... . ...
0100008
01000A RCALL W6 ; RCALL with W6
01000C MOVE W4, [W10]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 01 000A</td>
<td>PC 00 FF8C</td>
</tr>
<tr>
<td>W6 FFC0</td>
<td>W6 FFC0</td>
</tr>
<tr>
<td>W15 1004</td>
<td>W15 1008</td>
</tr>
<tr>
<td>Data 1004 98FF</td>
<td>Data 1004 000C</td>
</tr>
<tr>
<td>Data 1006 2310</td>
<td>Data 1006 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Example 2:

```
000302  RCALL  W2 ; RCALL with W2
000304  FF1L  W0, W1
     ... ...
000450  EX2:  CLR  W2 ; Destination of RCALL
000452  ... ...
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>00 0302</td>
</tr>
<tr>
<td>W2</td>
<td>00A6</td>
</tr>
<tr>
<td>W15</td>
<td>1004</td>
</tr>
<tr>
<td>Data 1004</td>
<td>32BB</td>
</tr>
<tr>
<td>Data 1006</td>
<td>901A</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**RCALL**

**Computed Relative Call**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

\{label\} RCALL Wn

**Operands:**

Wn ∈ [W0 ... W15]

**Operation:**

(PC) + 2 → PC

(PC<15:1>) → TOS<15:1>, SFA Status bit → TOS<0>

W15 + 2 → W15

(PC<22:16>) → (TOS)

W15 + 2 → W15

0 → SFA Status bit

(2 * (Wn) + (PC)) → PC

NOP → Instruction Register

**Status Affected:**

SFA

**Encoding:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>ssss</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td>0000</td>
<td>ss</td>
<td>ss</td>
</tr>
</tbody>
</table>

**Description:**

Computed, relative subroutine call specified by the Working register Wn. The range of the call is 32K program words forward or backward from the current PC. Before the call is made, the return address (PC + 2) is PUSHed onto the stack. After the return address is stacked, the sign-extended 17-bit value (2 * (Wn)) is added to the contents of the PC and the result is stored in the PC. Register Direct Addressing must be used for Wn.

The 's' bits select the source register.

**Words:**

1

**Cycles:**

4

**Example 1:**

```
00FF8C EX1: INC W2, W3 ; Destination of RCALL
00FF8E ... . . . .
010008
01000A RCALL W6 ; RCALL with W6
01000C MOVE W4, [W10]
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 01 000A</td>
<td>PC 00 FF8C</td>
</tr>
<tr>
<td>W6 FFC0</td>
<td>W6 FFC0</td>
</tr>
<tr>
<td>W15 1004</td>
<td>W15 1008</td>
</tr>
<tr>
<td>Data 1004 98FF</td>
<td>Data 1004 000C</td>
</tr>
<tr>
<td>Data 1006 2310</td>
<td>Data 1006 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Example 2:

```
000302  RCALL    W2         ; RCALL with W2
000304  FF1L    W0, W1

.       ... 

000450  EX2:    CLR    W2   ; Destination of RCALL
000452       ... 
```

### Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W2</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00A6</td>
<td>1004</td>
<td>32BB</td>
<td>0000</td>
</tr>
</tbody>
</table>

### After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W2</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00A6</td>
<td>1008</td>
<td>0304</td>
<td>0000</td>
</tr>
</tbody>
</table>
### REPEAT

**Repeat Next Instruction ‘lit14+1’ Times**

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} REPEAT #lit14

**Operands:**

lit14 ∈ [0 ... 16383]

**Operation:**

(lit14) → RCOUNT

(PO + 2) → 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 in the Instruction Register (IR) 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.

**Special Features, Restrictions:**

1. When the repeat literal is ‘0’, REPEAT has the effect of a NOP and the RA bit is not set.

2. The target REPEAT instruction cannot be:
   - An instruction that changes program flow
   - A DO, DISI, LNK, MOV.D, FWRSAV, REPEAT or UNLK instruction
   - A 2-word instruction

   Unexpected results may occur if these target instructions are used.

**Note:** The REPEAT and target instruction are interruptible.

**Words:**

1

**Cycles:**

1

**Example 1:**

000452 REPEAT #9 ; Execute ADD 10 times
000454 ADD [W0++], W1, [W2++] ; Vector update

```
Before Instruction
PC  00 0452
RCOUNT  0000
SR  0000

After Instruction
PC  00 0454
RCOUNT  0009
SR  0010 (RA = 1)
```

**Example 2:**

00089E REPEAT #0x3FF ; Execute CLR 1024 times
0008A0 CLR [W6++] ; Clear the scratch space

```
Before Instruction
PC  00 089E
RCOUNT  0000
SR  0000

After Instruction
PC  00 08A0
RCOUNT  03FF
SR  0010 (RA = 1)
```
### REPEAT

**Repeat Next Instruction ‘lit15+1’ Times**

- **Implemented in:**
  - PIC24F
  - PIC24H
  - PIC24E
  - dsPIC30F
  - dsPIC33F
  - dsPIC33E
  - dsPIC33C

- **Syntax:**
  ```
  (label:) REPEAT #lit15
  ```

- **Operands:**
  - `lit15 ∈ [0 ... 32767]`

- **Operation:**
  ```
  (lit15) → RCOUNT
  (PC) + 2 → PC
  ```

- **Enable code looping.**

- **Status Affected:** RA

- **Encoding:**
  ```
  0000 1001 0kkk kkkk kkkk kkkk
  ```

- **Description:**
  Repeat the instruction immediately following the `REPEAT` instruction (`lit15 + 1`) times. The repeated instruction (or target instruction) is held in the Instruction Register for all iterations and is only fetched once.

  When this instruction executes, the RCOUNT register is loaded with the repeat count value specified in the instruction. RCOUNT is decremented with each execution of the target instruction. When RCOUNT equals zero, the target instruction is executed one more time and then normal instruction execution continues with the instruction following the target instruction.

  The ‘k’ bits are an unsigned literal that specifies the loop count.

**Special Features, Restrictions:**

1. When the repeat literal is ‘0’, `REPEAT` has the effect of a `NOP` and the RA bit is not set.
2. The target `REPEAT` instruction cannot be:
   - An instruction that changes program flow
   - A `DISI`, `LNK`, `MOV.D`, `PWRSAV`, `REPEAT` or `UNLK` instruction
   - A 2-word instruction

   Unexpected results may occur if these target instructions are used.

**Note:** The `REPEAT` and target instruction are interruptible.

**Words:** 1

**Cycles:** 1

**Example 1:**
```
000452   REPEAT #9 ; Execute ADD 10 times
000454   ADD    [W0++], W1, [W2++] ; Vector update
```

**Before Instruction**
```
PC  00 0452
RCOUNT 0000
SR  0000
```

**After Instruction**
```
PC  00 0454
RCOUNT 0009
SR  0010 (RA = 1)
```

**Example 2:**
```
00089E   REPEAT #0x3FF    ; Execute CLR 1024 times
0008A0   CLR     [W6++]    ; Clear the scratch space
```

**Before Instruction**
```
PC  00 089E
RCOUNT 0000
SR  0000
```

**After Instruction**
```
PC  00 08A0
RCOUNT 03FF
SR  0010 (RA = 1)
```
REPEAT
Repeat Next Instruction Wn+1 Times

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:  
{label:} REPEAT Wn

Operands:  
Wn ∈ [W0 ... W15]

Operation:  
(Wn<13:0>) → RCOUNT
(PC) + 2 → PC
Enable code looping.

Status Affected:  
RA

Encoding:  
\[ \begin{array}{cccccc}
0000 & 1001 & 1000 & 0000 & 0000 & ssss \\
\end{array} \]

Description:  
Repeat the instruction immediately following the REPEAT instruction (Wn<13:0>) times. The instruction to be repeated (or target instruction) is held in the Instruction Register for all iterations and is only fetched once.

When this instruction executes, the RCOUNT register is loaded with the lower 14 bits of Wn. 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 ‘s’ bits specify the Wn register that contains the repeat count.

Special Features, Restrictions:
1. When (Wn) = 0, REPEAT has the effect of a NOP and the RA bit is not set.
2. The target REPEAT instruction cannot be:
   - An instruction that changes program flow
   - A DO, DISI, LNK, MOV.D, PWRSAV, REPEAT or ULNK instruction
   - A 2-word instruction

Unexpected results may occur if these target instructions are used.

Note: The REPEAT and target instruction are interruptible.

Words: 1
Cycles: 1

Example 1:

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 0A26</td>
<td>PC 00 0A28</td>
</tr>
<tr>
<td>W4 0023</td>
<td>W4 0023</td>
</tr>
<tr>
<td>RCOUNT 0000</td>
<td>RCOUNT 0023</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0010 (RA = 1)</td>
</tr>
</tbody>
</table>

Example 2:

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 089E</td>
<td>PC 00 08A0</td>
</tr>
<tr>
<td>W10 00FF</td>
<td>W10 00FF</td>
</tr>
<tr>
<td>RCOUNT 0000</td>
<td>RCOUNT 00FF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0010 (RA = 1)</td>
</tr>
</tbody>
</table>

Example:

000A26 REPEAT W4 ; Execute COM (W4+1) times
000A28 COM [W0++], [W2++] ; Vector complement

00089E REPEAT W10 ; Execute TBLRD (W10+1) times
0008A0 TBLRDL [W2++], [W3++] ; Decrement (0x840)
REPEAT
Repeat Next Instruction Wn+1 Times

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:
{label:} REPEAT Wn

Operands:
Wn ∈ [W0 ... W15]

Operation:
(Wn) → RCOUNT
(PC) + 2 → PC
Enable code looping.

Status Affected:
RA

Encoding:
0000 1001 1000 0000 0000 ssss

Description:
Repeat the instruction immediately following the REPEAT instruction (Wn) times. The instruction to be repeated (or target instruction) is held in the Instruction Register for all iterations and is only fetched once.

When this instruction executes, the RCOUNT register is loaded with Wn. 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 ‘s’ bits specify the Wn register that contains the repeat count.

Special Features, Restrictions:
1. When (Wn) = 0, REPEAT has the effect of a NOP and the RA bit is not set.
2. The target REPEAT instruction cannot be:
   • An instruction that changes program flow
   • A DO, DISI, LNK, MOV.D, PWRSAV, REPEAT or ULNK instruction
   • A 2-word instruction

   Unexpected results may occur if these target instructions are used.

   Note: The REPEAT and target instruction are interruptible.

Words: 1
Cycles: 1

Example 1:
000A26 REPEAT W4              ; Execute COM (W4+1) times
000A28 COM     [W0++], [W2++] ; Vector complement

Before Instruction          After Instruction
PC  00 0A26                   PC  00 0A28
W4  0023                      W4  0023
RCOUNT 0000                  RCOUNT 0023
SR  0000                      SR  0010 (RA = 1)

Example 2:
00089E REPEAT W10            ; Execute TBLRD (W10+1) times
0008A0 TBLRDL [W2++], [W3++] ; Decrement (0x840)

Before Instruction          After Instruction
PC  00 089E                   PC  00 08A0
W10  00FF                    W10  00FF
RCOUNT 0000                  RCOUNT 00FF
SR  0000                      SR  0010 (RA = 1)
**RESET**

**Reset**

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**  
\{label\} \textbf{RESET}

**Operands:** None

**Operation:** Force all registers that are affected by a MCLR Reset to their Reset condition.

1 → SWR (RCON<6>)

0 → PC

**Status Affected:** OA, OB, OAB, SA, SB, SAB, DA, DC, IPL<2:0>, RA, N, OV, Z, C, SFA

**Encoding:**

| 1111 | 1110 | 0000 | 0000 | 0000 | 0000 |

**Description:** This instruction provides a way to execute a software Reset. All core and peripheral registers will take their power-on value. The PC will be set to '0', the location of the \textbf{RESET GOTO} instruction. The SWR bit (RCON<6>) will be set to '1' to indicate that the \textbf{RESET} instruction was executed.

**Note:** Refer to the specific device family reference manual for the power-on value of all registers.

**Words:** 1

**Cycles:** 1

**Example 1:**

```assembly
00202A   RESET     ; Execute software RESET on dsPIC33F
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 202A</td>
<td>PC 00 0000</td>
</tr>
<tr>
<td>W0 8901</td>
<td>W0 0000</td>
</tr>
<tr>
<td>W1 08BB</td>
<td>W1 0000</td>
</tr>
<tr>
<td>W2 B87A</td>
<td>W2 0000</td>
</tr>
<tr>
<td>W3 872F</td>
<td>W3 0000</td>
</tr>
<tr>
<td>W4 C98A</td>
<td>W4 0000</td>
</tr>
<tr>
<td>W5 AAD4</td>
<td>W5 0000</td>
</tr>
<tr>
<td>W6 981E</td>
<td>W6 0000</td>
</tr>
<tr>
<td>W7 1809</td>
<td>W7 0000</td>
</tr>
<tr>
<td>W8 C341</td>
<td>W8 0000</td>
</tr>
<tr>
<td>W9 90F4</td>
<td>W9 0000</td>
</tr>
<tr>
<td>W10 F409</td>
<td>W10 0000</td>
</tr>
<tr>
<td>W11 1700</td>
<td>W11 0000</td>
</tr>
<tr>
<td>W12 1008</td>
<td>W12 0000</td>
</tr>
<tr>
<td>W13 6556</td>
<td>W13 0000</td>
</tr>
<tr>
<td>W14 231D</td>
<td>W14 0000</td>
</tr>
<tr>
<td>W15 1704</td>
<td>W15 0800</td>
</tr>
<tr>
<td>SPLIM 1800</td>
<td>SPLIM 0000</td>
</tr>
<tr>
<td>TBLPAG 000F</td>
<td>TBLPAG 0000</td>
</tr>
<tr>
<td>PSVPAG 0001</td>
<td>PSVPAG 0000</td>
</tr>
<tr>
<td>CORCON 00F0</td>
<td>CORCON 0020 (SATDW = 1)</td>
</tr>
<tr>
<td>RCON 0000</td>
<td>RCON 0040 (SWR = 1)</td>
</tr>
<tr>
<td>SR 0021 (IPL, C = 1)</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
RETFIE

Return from Interrupt

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax: (label:) RETFIE

Operands: None

Operation:
- \((W15) - 2 \rightarrow W15\)
- \((TOS<15:8>) \rightarrow (SR<7:0>)\)
- \((TOS<7>) \rightarrow (IPL3, CORCON<3>)\)
- \((TOS<6:0>) \rightarrow (PC<22:16>)\)
- \((W15) - 2 \rightarrow W15\)
- \((TOS<15:0>) \rightarrow (PC<15:0>)\)

NOP → Instruction Register

Status Affected: IPL<3:0>, RA, N, OV, Z, C

Encoding: 0000 0110 0100 0000 0000 0000

Description: Return from Interrupt Service Routine. The stack is POPped, which loads the low byte of the STATUS Register, IPL<3> (CORCON<3>) and the Most Significant Byte of the PC. The stack is POPped again, which loads the lower 16 bits of the PC.

Note 1: Restoring IPL<3> and the low byte of the STATUS Register restores the Interrupt Priority Level to the level before the execution was processed.

2: Before RETFIE is executed, the appropriate interrupt flag must be cleared in software to avoid recursive interrupts.

Words: 1

Cycles: 3 (2 if exception pending)

Example 1: 000A26 RETFIE ; Return from ISR

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 0A26</td>
<td>PC 01 0230</td>
</tr>
<tr>
<td>W15 0834</td>
<td>W15 0830</td>
</tr>
<tr>
<td>Data 0830 0230</td>
<td>Data 0832 8101</td>
</tr>
<tr>
<td>CORCON 0001</td>
<td>CORCON 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0003 (IPL = 4, C = 1)</td>
</tr>
</tbody>
</table>

Example 2: 008050 RETFIE ; Return from ISR

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 8050</td>
<td>PC 00 7008</td>
</tr>
<tr>
<td>W15 0926</td>
<td>W15 0922</td>
</tr>
<tr>
<td>Data 0922 7008</td>
<td>Data 0924 0300</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0003 (Z, C = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

RETFIE
Return from Interrupt

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:
{label:} RETFIE

Operands:
None

Operation:
(W15) – 2 → W15
(TOS<15:8>) → (SR<7:0>)
(TOS<7>) → (IPL3, CORCON<3>)
(TOS<6:0>) → (PC<22:16>)
(W15) – 2 → W15
(TOS<15:1>) → (PC<15:1>)
TOS<0> → SFA Status bit

NOP → Instruction Register

Status Affected:
IPL<3:0>, RA, N, OV, Z, C, SFA

Encoding:

```
0000 0110 0100 0000 0000 0000 0000
```

Description:
Return from Interrupt Service Routine. The stack is POPpFed, which loads the low byte of the STATUS Register, IPL<3> (CORCON<3>) and the Most Significant Byte of the PC. The stack is POPpFed again, which loads the lower 16 bits of the PC.

Note 1: Restoring IPL<3> and the low byte of the STATUS Register restores the Interrupt Priority Level to the level before the execution was processed.

2: Before RETFIE is executed, the appropriate interrupt flag must be cleared in software to avoid recursive interrupts.

Words: 1

Cycles: 6 (5 if exception pending)

Example 1:

```
000A26   RETFIE   ; Return from ISR
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 0A26</td>
<td>PC 01 0230</td>
</tr>
<tr>
<td>W15 0834</td>
<td>W15 0830</td>
</tr>
<tr>
<td>Data 0830</td>
<td>Data 0830</td>
</tr>
<tr>
<td>Data 0832</td>
<td>Data 0832</td>
</tr>
<tr>
<td>CORCON 0001</td>
<td>CORCON 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0081</td>
</tr>
</tbody>
</table>

(IPL = 4, C = 1)

Example 2:

```
008050   RETFIE   ; Return from ISR
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 8050</td>
<td>PC 00 7008</td>
</tr>
<tr>
<td>W15 0926</td>
<td>W15 0922</td>
</tr>
<tr>
<td>Data 0922</td>
<td>Data 0922</td>
</tr>
<tr>
<td>Data 0924</td>
<td>Data 0924</td>
</tr>
<tr>
<td>CORCON 0000</td>
<td>CORCON 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0003</td>
</tr>
</tbody>
</table>

(Z, C = 1)
RETLW

Return with Literal in Wn

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} RETLW.B \#lit10, Wn

Operands:

- lit10 ∈ [0 ... 255] for byte operation
- lit10 ∈ [0 ... 1023] for word operation
- Wn ∈ [W0 ... W15]

Operation:

- \((W15) - 2 \rightarrow W15\)
- TOS<15:8> → SR<7:0>
- TOS<7:0> → IPL<3>: PC<22:16>
- \((W15) - 2 \rightarrow W15\)
- (TOS) → (PC<15:0>)
- lit10 → Wn
- NOP → Instruction Register

Status Affected:

None

Encoding:

| 0000 | 0101 | 0Bkk | kkkk | kkkk | dddd |

Description:

Return from subroutine with the specified, unsigned 10-bit literal stored in Wn. The software stack is POPped twice to restore the PC and the signed literal is stored in Wn. Since two POPs are made, the Stack Pointer (W15) is decremented by 4.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'k' bits specify the value of the literal.

The 'd' bits select the destination register.

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.

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.

Words: 1

Cycles: 3 (2 if exception pending)

Example 1:

000440 RETLW.B \#0xA, W0 ; Return with 0xA in W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 0440</td>
<td>PC 00 7006</td>
</tr>
<tr>
<td>W0 9846</td>
<td>W0 980A</td>
</tr>
<tr>
<td>W15 1988</td>
<td>W15 1984</td>
</tr>
<tr>
<td>Data 1984 7006</td>
<td>Data 1984 7006</td>
</tr>
<tr>
<td>Data 1986 0000</td>
<td>Data 1986 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Example 2: 00050A RETLW #0x230, W2 ; Return with 0x230 in W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 00 050A</td>
<td>PC 01 7008</td>
</tr>
<tr>
<td>W2 0993</td>
<td>W2 0230</td>
</tr>
<tr>
<td>W15 1200</td>
<td>W15 11FC</td>
</tr>
<tr>
<td>Data 11FC 7008</td>
<td>Data 11FC 7008</td>
</tr>
<tr>
<td>Data 11FE 0001</td>
<td>Data 11FE 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
RETLW

Return with Literal in Wn

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
---|---|---|---|---|---|---|---|
| X | | | | X | X | |

Syntax:

\[
\{\text{label:}\} \text{RETLW}(.B) \#\text{lit10, Wn}
\]

Operands:

- \text{lit10} \in [0 \ldots 255] for byte operation
- \text{lit10} \in [0 \ldots 1023] for word operation
- \text{Wn} \in [W0 \ldots W15]

Operation:

- \( (W15) - 2 \rightarrow W15 \)
- \( \text{TOS}<15:1> \rightarrow \text{PC}<15:1> \)
- \( \text{TOS}<15:8> \rightarrow \text{SR}<15:0> \)
- \( \text{TOS}<7:0> \rightarrow \text{IPL}<3>: \text{PC}<22:16> \)
- \( \text{Wn} \rightarrow \text{Instruction Register} \)
- \( \text{Wn} \rightarrow \text{Instruction Register} \)

Status Affected:

- SFA

Encoding:

\[
\begin{array}{ccccccc}
0000 & 0101 & 0Bkk & kkkk & kkkk & dddd \\
\end{array}
\]

Description:

Return from subroutine with the specified, unsigned 10-bit literal stored in Wn. The software stack is POPped twice to restore the PC and the signed literal is stored in Wn. Since two POPs are made, the Stack Pointer (W15) is decremented by 4.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘k’ bits specify the value of the literal.

The ‘d’ bits select the destination register.

\textbf{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.

\textbf{Note 2:}\ For byte operations, the literal must be specified as an unsigned value [0:255]. See \textit{Section 4.6 “Using 10-Bit Literal Operands”} for information on using 10-bit literal operands in Byte mode.

Words: 1

Cycles: 6 (5 if exception pending)

**Example 1:** \(000440\ \text{RETLW}(.B) \#0xA, W0\); Return with \text{0xA} in \text{W0}
Example 2: `00050A RETLW #0x230, W2 ; Return with 0x230 in W2`

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>00 050A</td>
<td>01 7008</td>
</tr>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>0993</td>
<td>0230</td>
</tr>
<tr>
<td>W15</td>
<td>W15</td>
</tr>
<tr>
<td>1200</td>
<td>11FC</td>
</tr>
<tr>
<td>Data 11FC</td>
<td>Data 11FC</td>
</tr>
<tr>
<td>7008</td>
<td>7008</td>
</tr>
<tr>
<td>Data 11FE</td>
<td>Data 11FE</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
## RETURN

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

{label:} RETURN

### Operands:

None

### Operation:

(W15) – 2 → W15  
(TOS) → (PC<22:16>)  
(W15) – 2 → W15  
(TOS) → (PC<15:0>)  
NOP → Instruction Register

### Status Affected:

None

### Encoding:

| 0000 | 0110 | 0000 | 0000 | 0000 | 0000 |

### Description:

Return from subroutine. The software stack is POPped twice to restore the PC. Since two POPs are made, the Stack Pointer (W15) is decremented by 4.

### Words:

1

### Cycles:

3 (2 if exception pending)

#### Example 1:

```
001A06   RETURN     ; Return from subroutine
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 1A06</td>
<td>PC  01 0004</td>
</tr>
<tr>
<td>W15  1248</td>
<td>W15  1244</td>
</tr>
<tr>
<td>Data 1244 0004</td>
<td>Data 1244 0004</td>
</tr>
<tr>
<td>Data 1246 0001</td>
<td>Data 1246 0001</td>
</tr>
<tr>
<td>SR   0000</td>
<td>SR   0000</td>
</tr>
</tbody>
</table>

#### Example 2:

```
005404   RETURN     ; Return from subroutine
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC  00 5404</td>
<td>PC  00 0966</td>
</tr>
<tr>
<td>W15  090A</td>
<td>W15  0906</td>
</tr>
<tr>
<td>Data 0906 0966</td>
<td>Data 0906 0966</td>
</tr>
<tr>
<td>Data 0908 0000</td>
<td>Data 0908 0000</td>
</tr>
<tr>
<td>SR   0000</td>
<td>SR   0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

RETURN

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} RETURN

Operands: None

Operation:

(W15) – 2 → W15
(TOS) → (PC<22:16>)
(W15) – 2 → W15
(TOS<15:1) → (PC<15:1>)
TOS<0> → SFA Status bit
NOP → Instruction Register

Status Affected: SFA

Encoding:

0000 0110 0000 0000 0000 0000

Description: Return from subroutine. The software stack is POPped twice to restore the PC. Since two POPs are made, the Stack Pointer (W15) is decremented by 4.

Words: 1

Cycles: 6 (5 if exception pending)

Example 1:

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001A06</td>
<td>1248</td>
<td>1244</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>010004</td>
<td>1244</td>
<td>0004</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

Before Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>005404</td>
<td>090A</td>
<td>0906</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>PC</th>
<th>W15</th>
<th>Data</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>000966</td>
<td>0906</td>
<td>0966</td>
<td>0000</td>
</tr>
</tbody>
</table>
**RLC**

**Rotate Left f through Carry**

*Implemented in:*

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\{label: \} RLC{.B} f {,WREG}

**Operands:**

f ∈ [0 ... 8191]

**Operation:**

For Byte Operation:

- (C) → Dest<0>
- (f<6:0>) → Dest<7:1>
- (f<7>) → C

For Word Operation:

- (C) → Dest<0>
- (f<14:0>) → Dest<15:1>
- (f<15>) → C

**Status Affected:**

N, Z, C

**Encoding:**

| 1101 | 0110 | 1Bdf | ffff | ffff | ffff |

**Description:**

Rotate the contents of the file register f, one bit to the left through the Carry flag, and place the result in the destination register. The Carry flag of the STATUS Register is shifted into the Least Significant bit of the destination and it is then overwritten with the Most Significant bit of Ws.

The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for f, '1' for WREG).

The 'f' bits select the address of the file register.

**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.

2: The WREG is set to Working register W0.

**Words:**

1

**Cycles:**

1

**Example 1:**

RLC.B 0x1233 ; Rotate Left w/ C (0x1233) {Byte mode}

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1232 E807</td>
<td>Data 1232 D007</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**

RLC 0x820, WREG ; Rotate Left w/ C (0x820) {Word mode} ; Store result in WREG

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0) 5601</td>
<td>WREG (W0) 42DD</td>
</tr>
<tr>
<td>Data 0820 216E</td>
<td>Data 0820 216E</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0000 (C = 0)</td>
</tr>
</tbody>
</table>
# RLC
Rotate Left Ws through Carry

## Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

## Syntax:

{label:} RLC(.B) Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- [++Ws], [++Wd]
- [--Ws], [--Wd]

## Operands:

- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

## Operation:

**For Byte Operation:**

- (C) → Wd<0>
- (Ws<6:0>) → Wd<7:1>
- (Ws<7>) → C

**For Word Operation:**

- (C) → Wd<0>
- (Ws<14:0>) → Wd<15:1>
- (Ws<15>) → C

## Status Affected:

N, Z, C

## Encoding:

```
1101 0010 1Bqq qddd dppp ssss
```

## Description:

Rotate the contents of the source register Ws, one bit to the left through the Carry flag, and place the result in the destination register Wd. The Carry flag of the STATUS Register is shifted into the Least Significant bit of Wd and it is then overwritten with the Most Significant bit of Ws. Either Register Direct or Indirect Addressing 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 addressing mode.

The 'd' bits select the destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

**Note:** 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
**Example 1:** RLC.B W0, W3 ; Rotate Left w/C (W0) (Byte mode)
; Store the result in W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>W0</td>
</tr>
<tr>
<td>9976</td>
<td>9976</td>
</tr>
<tr>
<td>W3</td>
<td>W3</td>
</tr>
<tr>
<td>5879</td>
<td>58ED</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0001  (C = 1)</td>
<td>0009  (N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:** RLC [W2++], [W8] ; Rotate Left w/C [W2] (Word mode)
; Post-increment W2
; Store result in [W8]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>W2</td>
</tr>
<tr>
<td>2008</td>
<td>200A</td>
</tr>
<tr>
<td>W8</td>
<td>W8</td>
</tr>
<tr>
<td>094E</td>
<td>094E</td>
</tr>
<tr>
<td>Data 094E</td>
<td>Data 094E</td>
</tr>
<tr>
<td>3689</td>
<td>8082</td>
</tr>
<tr>
<td>Data 2008</td>
<td>Data 2008</td>
</tr>
<tr>
<td>C041</td>
<td>C041</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>0001  (C = 1)</td>
<td>0009  (N, C = 1)</td>
</tr>
</tbody>
</table>
RLNC

---

**Rotate Left f without Carry**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} RLNC{.B} f ,{WREG}
```

**Operands:**

- `f ∈ [0 ... 8191]`

**Operation:**

- **For Byte Operation:**
  
  \[
  (f<6:0>) \rightarrow \text{Dest}<7:1> \\
  (f<7>) \rightarrow \text{Dest}<0>
  \]

- **For Word Operation:**
  
  \[
  (f<14:0>) \rightarrow \text{Dest}<15:1> \]
  
  \[
  (f<15>) \rightarrow \text{Dest}<0>
  \]

**Status Affected:**

- N, Z

**Encoding:**

| 1101 | 0110 | 0BDf | ffff | ffff | ffff |

**Description:**

Rotate the contents of the file register `f`, one bit to the left, and place the result in the destination register. The Most Significant bit of `f` is stored in the Least Significant bit of the destination and the Carry flag is not affected.

The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.

The 'B' bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The 'D' bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The 'f' bits select the address of the file register.

**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.

**2:** The WREG is set to Working register W0.

**Words:**

1

**Cycles:**

1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

**Example 1:**

```
RLNC.B 0x1233 ; Rotate Left (0x1233) (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1232</td>
<td>Data 1233</td>
</tr>
<tr>
<td>E807</td>
<td>D107</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008</td>
</tr>
</tbody>
</table>

**Example 2:**

```
RLNC 0x820, WREG ; Rotate Left (0x820) (Word mode) ; Store result in WREG
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>5601</td>
<td>42DC</td>
</tr>
<tr>
<td>Data 0820</td>
<td>Data 0820</td>
</tr>
<tr>
<td>216E</td>
<td>216E</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0000 (C = 0)</td>
</tr>
</tbody>
</table>
### RLNC Rotate Left Ws without Carry

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```
{label:} RLNC{.B} Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [++Wd]

[--Ws], [--Wd]
```

#### Operands:

- **Ws** ∈ [W0 ... W15]
- **Wd** ∈ [W0 ... W15]

#### Operation:

**For Byte Operation:**

- \((Ws<6:0>) \rightarrow Wd<7:1>\)
- \((Ws<7>) \rightarrow Wd<0>\)

**For Word Operation:**

- \((Ws<14:0>) \rightarrow Wd<15:1>\)
- \((Ws<15>) \rightarrow Wd<0>\)

#### Status Affected:

- N, Z

#### Encoding:

```
1101 0010 0Bqq qddd dppp ssss
```

#### Description:

Rotate the contents of the source register **Ws**, one bit to the left, and place the result in the destination register **Wd**. The Most Significant bit of **Ws** is stored in the Least 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 ‘B’ bit selects byte or word operation (‘0’ for byte, ‘1’ for word).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note:** 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:**

\(1^{(t)}\)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

---

### Example 1:

**RLNC.B W0, W3**  
; Rotate Left (W0) (Byte mode)  
; Store the result in W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>9976</td>
</tr>
<tr>
<td>W3</td>
<td>5879</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>

### Example 2:

**RLNC [W2++], [W8]**  
; Rotate Left [W2] (Word mode)  
; Post-increment W2  
; Store result in [W8]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>2008</td>
</tr>
<tr>
<td>W8</td>
<td>094E</td>
</tr>
<tr>
<td>Data 094E</td>
<td>3689</td>
</tr>
<tr>
<td>Data 2008</td>
<td>C041</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
**RRC**

**Rotate Right f through Carry**

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} RRC{.B} f {,WREG}

Operands:

*f* ∈ [0 ... 8191]

Operation:

For Byte Operation:

(C) → Dest<7>

(f<7:1>) → Dest<6:0>

(f<0>) → C

For Word Operation:

(C) → Dest<15>

(f<15:1>) → Dest<14:0>

(f<0>) → C

Status Affected: N, Z, C

Encoding:

| 1101 | 0111 | 1BDF | fffe | fffe | fffe |

Description:

Rotate the contents of the file register *f*, one bit to the right through the Carry flag, and place the result in the destination register. The Carry flag of the STATUS Register is shifted into the Most Significant bit of the destination and it is then overwritten with the Least Significant bit of Ws.

The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.

The ‘B’ bit selects byte or word operation (‘0’ for byte, ‘1’ for word).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

**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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: 1

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in **Section 3.2.1 “Multicycle Instructions”**.
Example 1: \texttt{RRC.B 0x1233} ; Rotate Right w/C (0x1233) (Byte mode)

\begin{tabular}{|c|c|}
\hline
\textbf{Before} & \textbf{After} \\
\hline
Data 1232 & 1232 \\
SR & 0000 \\
\hline
\end{tabular}

Before Instruction: Data 1232 = E807, SR = 0000
After Instruction: Data 1232 = 7407, SR = 0000

Example 2: \texttt{RRC 0x820, WREG} ; Rotate Right w/C (0x820) (Word mode) ; Store result in WREG

\begin{tabular}{|c|c|}
\hline
\textbf{Before} & \textbf{After} \\
\hline
WREG (W0) & WREG (W0) \\
Data 0820 & Data 0820 \\
SR & SR \\
\hline
\end{tabular}

Before Instruction: WREG (W0) = 5601, Data 0820 = 216E, SR = 0001 (C = 1)
After Instruction: WREG (W0) = 90B7, Data 0820 = 216E, SR = 0008 (N = 1)
### RRC

**Rotate Right Ws through Carry**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} RRC(.B) Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- ++Ws, ++Wd
- --Ws, --Wd

**Operands:**

Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

**Operation:**

For Byte Operation:

- (C) → Wd<7>
- (Ws<7:1>) → Wd<6:0>
- (Ws<0>) → C

For Word Operation:

- (C) → Wd<15>
- (Ws<15:1>) → Wd<14:0>
- (Ws<0>) → C

**Status Affected:**

N, Z, C

**Encoding:**

```
| 1101 | 0011 | 1Bqq | qddd | dpdp | ssss |
```

**Description:** 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 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'q' bits select the destination addressing mode.

The 'd' bits select the destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

**Note:** 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

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 "Multicycle Instructions".
Example 1: \texttt{RRC.B \ W0, W3} ; Rotate Right w/ C (W0) (Byte mode)
; Store the result in W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 9976</td>
<td>W0 9976</td>
</tr>
<tr>
<td>W3 5879</td>
<td>W3 58BB</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>

Example 2: \texttt{RRC [W2++], [W8]} ; Rotate Right w/ C [W2] (Word mode)
; Post-increment W2
; Store result in [W8]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2 2008</td>
<td>W2 200A</td>
</tr>
<tr>
<td>W8 094E</td>
<td>W8 094E</td>
</tr>
<tr>
<td>Data 094E 3689</td>
<td>Data 094E E020</td>
</tr>
<tr>
<td>Data 2008 C041</td>
<td>Data 2008 C041</td>
</tr>
<tr>
<td>SR 0001 (C = 1)</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>
RRNC

Rotate Right f without Carry

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
\{label:\} RRNC{.B} f \{.WREG\}

Operands:  
f ∈ [0 ... 8191]

Operation:

For Byte Operation:

\[(f<7:1>) \rightarrow \text{Dest}<6:0>\]  
\[(f<0>) \rightarrow \text{Dest}<7>\]

For Word Operation:

\[(f<15:1>) \rightarrow \text{Dest}<14:0>\]  
\[(f<0>) \rightarrow \text{Dest}<15>\]

Status Affected: N, Z

Encoding:

| 1101 | 0111 | 0BDf | ffff | ffff | ffff |

Description:

Rotate the contents of the file register f, one bit to the right, and place the result in the destination register. The Least Significant bit of f is stored in the Most Significant bit of the destination and the Carry flag is not affected.

The optional WREG operand determines the destination register. If WREG is specified, the result is stored in WREG. If WREG is not specified, the result is stored in the file register.

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).

The 'D' bit selects the destination register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1: RRNC.B 0x1233 ; Rotate Right (0x1233) (Byte mode)

Before Instruction

<table>
<thead>
<tr>
<th>Data 1232</th>
<th>E807</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>Data 1232</th>
<th>7407</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2: RRNC 0x820, WREG ; Rotate Right (0x820) (Word mode)

; Store result in WREG

Before Instruction

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>5601</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0820</td>
<td>216E</td>
</tr>
<tr>
<td>SR</td>
<td>0001</td>
</tr>
</tbody>
</table>

After Instruction

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>10B7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0820</td>
<td>216E</td>
</tr>
<tr>
<td>SR</td>
<td>0001</td>
</tr>
</tbody>
</table>

(C = 1)
Section 5. Instruction Descriptions

RRNC
Rotate Right Ws without Carry

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} RRNC{.B} Ws, Wd

[Ws], [Wd]
[Ws++], [Wd++]
[Ws--], [Wd--]
[++Ws], [+Wd]
[--Ws], [--Wd]

Operands: Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:
For Byte Operation:
(Ws<7:1>) → Wd<6:0>
(Ws<0>) → Wd<7>

For Word Operation:
(Ws<15:1>) → Wd<14:0>
(Ws<0>) → Wd<15>

Status Affected: N, Z

Encoding:

1101 0011 0Bqq qddd dppp ssss

Description: 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 ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).
The ‘q’ bits select the destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

Note: 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\(^{(1)}\)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

\(^{(1)}\) Cycle count may be higher when using Indexed Addressing.
Example 1:  RRNC.B W0, W3  ; Rotate Right (W0) (Byte mode)
             ; Store the result in W3

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  9976</td>
<td>W0  9976</td>
</tr>
<tr>
<td>W3  5879</td>
<td>W3  583B</td>
</tr>
<tr>
<td>SR  0001 (C = 1)</td>
<td>SR  0001 (C = 1)</td>
</tr>
</tbody>
</table>

Example 2:  RRNC  [W2++], [W8]  ; Rotate Right [W2] (Word mode)
             ; Post-increment W2
             ; Store result in [W8]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2  2008</td>
<td>W2  200A</td>
</tr>
<tr>
<td>W8  094E</td>
<td>W8  094E</td>
</tr>
<tr>
<td>Data 094E 3689</td>
<td>Data 094E E020</td>
</tr>
<tr>
<td>Data 2008 C041</td>
<td>Data 2008 C041</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0008 (N = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

SAC

Store Accumulator

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  {label:} SAC Acc, {#Slit4,} Wd

[Wd]
[Wd++]
[Wd--]
[-Wd]
[++Wd]
[Wd + Wb]

Operands:

Acc ∈ [A,B]
Slit4 ∈ [-8 ... +7]
Wb, Wd ∈ [W0 ... W15]

Operation:

Shift\_Slit4(Acc) (optional)
(Acc[31:16]) → Wd

Status Affected:

None

Encoding:

| 1100 | 1100 | Awww | wrrr | rhhh | dddd |

Description:

Perform an optional, signed 4-bit shift of the specified accumulator, then store the shifted contents of ACCxH (Acc[31:16]) to Wd. The shift range is -8:7, where a negative operand indicates an arithmetic left shift and a positive operand indicates an arithmetic right shift. Either Register Direct or Indirect Addressing may be used for Wd.

The ‘A’ bit specifies the source accumulator.
The ‘w’ bits specify the offset register Wb.
The ‘r’ bits encode the optional accumulator preshift.
The ‘h’ bits select the destination addressing mode.
The ‘d’ bits specify the destination register Wd.

Note 1: This instruction does not modify the contents of Acc.

2: This instruction stores the truncated contents of Acc. The instruction, SAC R, may be used to store the rounded accumulator contents.

3: If data write saturation is enabled (SATDW (CORCON<5>) = 1), the value stored to Wd is subject to saturation after the optional shift is performed.

Words: 1

Cycles: 1
### Example 1:
```
SAC A, #4, W5
; Right shift ACCA by 4
; Store result to W5
; CORCON = 0x0010 (SATDW = 1)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>B900</td>
</tr>
<tr>
<td>ACCA</td>
<td>00 120F FF00</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0120</td>
</tr>
<tr>
<td></td>
<td>00 120F FF00</td>
</tr>
<tr>
<td></td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>ACCB</td>
<td>FF C891 1F4C</td>
</tr>
<tr>
<td>Data 2000</td>
<td>5BBE</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>FF C891 1F4C</td>
</tr>
<tr>
<td></td>
<td>Data 2000</td>
</tr>
<tr>
<td></td>
<td>8000</td>
</tr>
<tr>
<td></td>
<td>CORCON</td>
</tr>
<tr>
<td></td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td>SR</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
</tbody>
</table>
SAC.D  
Store Accumulator Double

### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Syntax:

{label:} SAC.D Acc, [ #Slit4], Wnd

- [Wnd]
- [Wnd++]
- [Wnd--]
- [--Wnd]
- [++Wnd]

### Operands:

- **Register Direct:** Wnd ∈ [W0, W2, W4, W6, W8, W10, W12, W14];
- **Register Indirect:** Wnd ∈ [W0 ... W15];
- **Acc** ∈ [A, B]
- **Slit4** ∈ [-8 ... +7]

### Operation:

- **Shift**<sub>Slit4</sub>(Acc) (optional); (Acc[31:0]) → Wnd

### Status Affected:

None

### Encoding:

```
1101 1100 A0qq qrrr r000 dddd
```  

### Description:

Optionally shift accumulator, then store accumulator, Acc<31:0>, to the destination Effective Address.

- The ‘A’ bit specifies the source accumulator.
- The ‘d’ bits specify the destination register Wnd.
- The ‘q’ bits select the destination addressing mode.
- The ‘r’ bits encode the optional operand Slit4, which determines the amount of the accumulator preshift; if the operand Slit4 is absent, a ‘0’ is encoded.

**Note 1:** Unlike SAC and SAC.R instructions, the SAC.D instruction does not support Indirect with Register Offset Addressing mode.

**Note 2:** Positive values of operand Slit4 represent arithmetic shift right. Negative values of operand Slit4 represent shift left.

**Note 3:** The SAC.D instruction cannot be executed within a REPEAT loop.

### Words:

1

### Cycles:

2
SAC.R

Store Rounded Accumulator

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: `{label:} SAC.R Acc, {#Slit4,} Wd

- [Wd]
- [Wd++]
- [Wd−]
- [−Wd]
- [++Wd]
- [Wd + Wb]

Operands:

- Acc ∈ [A,B]
- Slit4 ∈ [-8 ... +7]
- Wb ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

Operation:

- ShiftSLIT(Acc) (optional)
- Round(Acc)
- (Acc[31:16]) → Wd

Status Affected: None

Encoding:

```
1100 1101 Awww wrrr rhhh dddd
```

Description:

Perform an optional, signed 4-bit shift of the specified accumulator, then store the rounded contents of ACCxH (Acc[31:16]) to Wd. The shift range is -8:7, where a negative operand indicates an arithmetic left shift and a positive operand indicates an arithmetic right shift. The Rounding mode (Conventional or Convergent) is set by the RND bit (CORCON<1>). Either Register Direct or Indirect Addressing may be used for Wd.

The 'A' bit specifies the source accumulator.

The 'w' bits specify the offset register Wb.

The 'r' bits encode the optional accumulator preshift.

The 'h' bits select the destination addressing mode.

The 'd' bits specify the destination register Wd.

Note 1: This instruction does not modify the contents of the Acc.

Note 2: This instruction stores the rounded contents of Acc. The instruction, SAC, may be used to store the truncated accumulator contents.

Note 3: If data write saturation is enabled (SATDW (CORCON<5>) = 1), the value stored to Wd is subject to saturation after the optional shift is performed.

Words: 1

Cycles: 1
Example 1: SAC.R A, #4, W5
; Right shift ACCA by 4
; Store rounded result to W5
; CORCON = 0x0010 (SATDW = 1)

Before Instruction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>B900</td>
</tr>
<tr>
<td>ACCA</td>
<td>00120F FF00</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>0121</td>
</tr>
<tr>
<td>ACCA</td>
<td>00120F FF00</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

Before Instruction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>ACCB</td>
<td>FF F891 8F4C</td>
</tr>
<tr>
<td>Data 2000</td>
<td>5BBE</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Instruction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W5</td>
<td>2002</td>
</tr>
<tr>
<td>ACCB</td>
<td>FF F891 8F4C</td>
</tr>
<tr>
<td>Data 2000</td>
<td>8919</td>
</tr>
<tr>
<td>CORCON</td>
<td>0010</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
SE

**Sign-Extend Ws**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} SE Ws, Wnd

- [Ws],
- [Ws++],
- [Ws--],
- [++Ws],
- [--Ws],

**Operands:**

Ws ∈ [W0 ... W15]
Wnd ∈ [W0 ... W15]

**Operation:**

Ws<7:0> → Wnd<7:0>
If (Ws<7> = 1):

0xFF → Wnd<15:8>
Else:

0 → Wnd<15:8>

**Status Affected:**

N, Z, C

**Encoding:**

| 1111 | 1011 | 0000 | 0ddd | dppp | ssss |

**Description:**

Sign-extend the byte in Ws and store the 16-bit result in Wnd. Either Register Direct or Indirect Addressing may be used for Ws and Register Direct Addressing must be used for Wnd. The C flag is set to the complement of the N flag.

The 'd' bits select the destination register.
The 'p' bits select the source addressing mode.
The 's' bits select the source register.

**Note 1:** This operation converts a byte to a word and it uses no .B or .W extension.

**Note 2:** The source Ws is addressed as a byte operand, so any address modification is by ‘1’.

**Words:**

1

**Cycles:**

1(t)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1: \( SE \ W3, \ W4 \)

; Sign-extend \( W3 \) and store to \( W4 \)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W3 )</td>
<td>( 7839 )</td>
</tr>
<tr>
<td>( W4 )</td>
<td>( 1005 )</td>
</tr>
<tr>
<td>( SR )</td>
<td>( 0000 )</td>
</tr>
</tbody>
</table>

### Example 2: \( SE \ [W2++] , \ W12 \)

; Sign-extend \([W2]\) and store to \( W12 \)
; Post-increment \( W2 \)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W2 )</td>
<td>( 0900 )</td>
</tr>
<tr>
<td>( W12 )</td>
<td>( 1002 )</td>
</tr>
<tr>
<td>( Data \ 0900 )</td>
<td>( 008F )</td>
</tr>
<tr>
<td>( SR )</td>
<td>( 0000 )</td>
</tr>
<tr>
<td>( W2 )</td>
<td>( 0901 )</td>
</tr>
<tr>
<td>( W12 )</td>
<td>( FF8F )</td>
</tr>
<tr>
<td>( Data \ 0900 )</td>
<td>( 008F )</td>
</tr>
<tr>
<td>( SR )</td>
<td>( 0008 )</td>
</tr>
</tbody>
</table>

\( C = 1 \)

\( N = 1 \)
SETM

Set f or WREG

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} SETM{.B} f

Operands:

\( f \in [0 \ldots 8191] \)

Operation:

For Byte Operation:

\( 0xFF \rightarrow \text{destination designated by D} \)

For Word Operation:

\( 0xFFFF \rightarrow \text{destination designated by D} \)

Status Affected:

None

Encoding:

\[
\begin{array}{cccccc}
1110 & 1111 & 1Bdf & ffff & ffff & fffe \\
\end{array}
\]

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 ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘D’ bit selects the destination register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

Words: 1

Cycles: 1

Example 1:

\text{SETM.B 0x891} ; \text{Set 0x891 (Byte mode)}

\begin{tabular}{lcc}
Before & After \\
Instruction & Instruction \\
\hline
Data 0890 & 2739 & Data 0890 & FF39 \\
SR 0000 & 0000 & SR 0000 & 0000 \\
\end{tabular}

Example 2:

\text{SETM WREG} ; \text{Set WREG (Word mode)}

\begin{tabular}{lcc}
Before & After \\
Instruction & Instruction \\
\hline
WREG (W0) & 0900 & WREG (W0) & FFFF \\
SR 0000 & 0000 & SR 0000 & 0000 \\
\end{tabular}
**SETM**

**Set Ws**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

```
{label:} SETM{.B} Wd
[ Wd ]
[ Wd++]
[ Wd--]
[++Wd]
[--Wd]
```

**Operands:**

`Wd ∈ [W0 ... W15]`

**Operation:**

- **For Byte Operation:**
  
  \(0xFF \rightarrow Wd\) for byte operation

- **For Word Operation:**
  
  \(0xFFFF \rightarrow Wd\) for word operation

**Status Affected:**

None

**Encoding:**

```
1110 1011 1Bqq qddd d000 0000
```

**Description:**

All the bits of the specified register are set to ‘1’. Either Register Direct or Indirect Addressing may be used for `Wd`.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

**Note:**

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

**Example 1:**

```
SETM.B W13 ; Set W13 (Byte mode)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W13 2739</td>
<td>W13 27FF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
SETM [--W6] ; Pre-decrement W6 (Word mode)
```

; Set [W6]

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6 1250</td>
<td>W6 124E</td>
</tr>
<tr>
<td>Data 124E 3CD9</td>
<td>Data 124E FFFF</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
SFTAC  

Arithmetic Shift Accumulator by Slit6

Implemented in:  

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} SFTAC Acc, #Slit6

Operands:  

Acc ∈ [A,B]  
Slit6 ∈ [-16 ... 16]

Operation:  

Shiftk(Acc) → Acc

Status Affected:  

OA, OB, OAB, SA, SB, SAB

Encoding:  

| 1100 | 1000 | A000 | 0000 | 01kk | kkkk |

Description:  

Arithmetic shift the 40-bit contents of the specified accumulator by the signed, 6-bit literal and store the result back into the accumulator. The shift range is -16:16, where a negative operand indicates a left shift and a positive operand indicates a right shift. Any bits which are shifted out of the accumulator are lost.

The ‘A’ bit selects the accumulator for the result.

The ‘k’ bits determine the number of bits to be shifted.

Note 1:  
If saturation is enabled for the target accumulator (SATA, CORCON<7> or SATB, CORCON<6>), the value stored to the accumulator is subject to saturation.

2:  
If the shift amount is greater than 16 or less than -16, no modification will be made to the accumulator and an arithmetic trap will occur.

Words: 1

Cycles: 1

Example 1:  

```
SFTAC A, #12
; Arithmetic right shift ACCA by 12
; Store result to ACCA
; CORCON = 0x0080 (SATA = 1)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCA 00 120F FF00</td>
<td>ACCA 00 0001 20FF</td>
</tr>
<tr>
<td>CORCON 0080</td>
<td>CORCON 0080</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:  

```
SFTAC B, #-10
; Arithmetic left shift ACCB by 10
; Store result to ACCB
; CORCON = 0x0040 (SATB = 1)
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCB FF FFF1 8F4C</td>
<td>ACCB FF C63D 3000</td>
</tr>
<tr>
<td>CORCON 0040</td>
<td>CORCON 0040</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

SFTAC

Arithmetic Shift Accumulator by Wb

Implemented in: PIC24F PIC24H PIC24E dsPIC30F dsPIC33F dsPIC33E dsPIC33C

Syntax: {label:} SFTAC Acc, Wb

Operands: Acc ∈ [A,B]
Wb ∈ [W0 ... W15]

Operation: Shift\(_{(Wb)}\)(Acc) → Acc

Status Affected: OA, OB, OAB, SA, SB, SAB

Encoding:

\[
\begin{array}{ccccccccc}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & sss
\end{array}
\]

Description:

Arithmetic shift the 40-bit contents of the specified accumulator and store the result back into the accumulator. The Least Significant 6 bits of Wb are used to specify the shift amount. The shift range is -16:16, where a negative value indicates a left shift and a positive value indicates a right shift. Any bits which are shifted out of the accumulator are lost.

The 'A' bit selects the accumulator for the source/destination.

The 's' bits select the address of the Shift Count register.

Note 1: If saturation is enabled for the target accumulator (SATA, CORCON<7> or SATB, CORCON<6>), the value stored to the accumulator is subject to saturation.

2: If the shift amount is greater than 16 or less than -16, no modification will be made to the accumulator and an arithmetic trap will occur.

Words: 1
Cycles: 1

Example 1:

SFTAC A, W0

; Arithmetic shift ACCA by (W0)
; Store result to ACCA
; CORCON = 0x0000 (saturation disabled)

Before
Instruction
W0  FFFC
ACCA 00 320F AB09
CORCON 0000
SR 0000

After
Instruction
W0  FFFC
ACCA 03 20FA B090
CORCON 0000
SR 8800

Example 2:

SFTAC B, W12

; Arithmetic shift ACCB by (W12)
; Store result to ACCB
; CORCON = 0x0040 (SATB = 1)

Before
Instruction
W12 000F
ACCB FF FFF1 8F4C
CORCON 0040
SR 0000

After
Instruction
W12 000F
ACCB FF FFFF FFE3
CORCON 0040
SR 0000
**SL**

**Shift Left f**

### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

```{label:} SL{.B} f {,WREG}```

### Operands:

- `f` ∈ [0 ... 8191]

### Operation:

**For Byte Operation:**
- `(f<7>) → (C)`
- `(f<6:0>) → Dest<7:1>`
- `0 → Dest<0>`

**For Word Operation:**
- `(f<15>) → (C)`
- `(f<14:0>) → Dest<15:1>`
- `0 → Dest<0>`

### Status Affected:

N, Z, C

### Encoding:

```
1101 0100 0BDf ffff ffff ffff
```

### Description:

Shift the contents of the file register, one bit to the left, and place the result in the destination register. The Most Significant bit of the file register is shifted into the Carry bit of the STATUS Register and ‘0’ is shifted into the Least Significant bit of 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 register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

### 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.

### Words:

1

### Cycles:

1

### Note 1:

In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1:

**Instruction:** 
`SL.B 0x909`  
*Shift left (0x909) (Byte mode)*

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0908</td>
<td>9439</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>Data 0908</td>
<td>0839</td>
</tr>
<tr>
<td>SR</td>
<td>0001 <em>(C = 1)</em></td>
</tr>
</tbody>
</table>

### Example 2:

**Instruction:** 
`SL 0x1650, WREG`  
*Shift left (0x1650) (Word mode)*  
*Store result in WREG*

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>0900</td>
</tr>
<tr>
<td>Data 1650</td>
<td>4065</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>WREG (W0)</td>
<td>80CA</td>
</tr>
<tr>
<td>Data 1650</td>
<td>4065</td>
</tr>
<tr>
<td>SR</td>
<td>0008 <em>(N = 1)</em></td>
</tr>
</tbody>
</table>
SL  Shift Left Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

{label:} SL{.B} Ws, Wd  
[Ws], [Wd]  
[Ws++], [Wd++]  
[Ws--], [Wd--]  
[++Ws], [+Wd]  
[--Ws], [--Wd]

Operands:

Ws ∈ [W0 ... W15]  
Wd ∈ [W0 ... W15]

Operation:

For Byte Operation:

(Ws<7>) → C  
(Ws<6:0>) → Wd<7:1>  
0 → Wd<0>

For Word Operation:

(Ws<15>) → C  
(Ws<14:0>) → Wd<15:1>  
0 → Wd<0>

Status Affected:

N, Z, C

Encoding:

| 1101 | 0000 | 0Bqq | qddd | dppp | ssss |

Description:

Shift the contents of the source register Ws, one bit to the left, and place the result in the destination register Wd. The Most Significant bit of Ws is shifted into the Carry bit of the STATUS Register and ‘0’ is shifted into the Least Significant bit of Wd. Either Register Direct or Indirect Addressing 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 addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: 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(f)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1: SL.B W3, W4

; Shift left W3 (Byte mode)
; Store result to W4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>78A9</td>
</tr>
<tr>
<td>W4</td>
<td>1005</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2: SL [W2++], [W12]

; Shift left [W2] (Word mode)
; Store result to [W12]
; Post-increment W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>0900</td>
</tr>
<tr>
<td>W12</td>
<td>1002</td>
</tr>
<tr>
<td>Data</td>
<td>0900</td>
</tr>
<tr>
<td>Data</td>
<td>1002</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
**SL**

**Shift Left by Short Literal**

- **Implemented in:**
  - PIC24F: X
  - PIC24H: X
  - PIC24E: X
  - dsPIC30F: X
  - dsPIC33F: X
  - dsPIC33E: X
  - dsPIC33C: X

- **Syntax:** `{label:} SL Wb, #lit4, Wnd

- **Operands:**
  - Wb ∈ [W0 ... W15]
  - lit4 ∈ [0 ... 15]
  - Wnd ∈ [W0 ... W15]

- **Operation:**
  - lit4<3:0> → Shift_Val
  - Wnd<15:Shift_Val> = Wb<15-Shift_Val:0>
  - Wd<Shift_Val – 1:0> = 0

- **Status Affected:** N, Z

- **Encoding:**
  - 1101 1101 0www wddd d100 kkkk

- **Description:** Shift left the contents of the source register Wb by the 4-bit unsigned literal and store the result in the destination register Wnd. Any bits shifted out of the source register are lost. Direct Addressing must be used for Wb and Wnd.
  
The 'w' bits select the address of the base register.
The 'd' bits select the destination register.
The 'k' bits provide the literal operand, a five-bit integer number.

  **Note:** This instruction operates in Word mode only.

- **Words:** 1
- **Cycles:** 1

- **Example 1:**
  - SL W2, #4, W2 ; Shift left W2 by 4
  - ; Store result to W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>78A9</td>
</tr>
</tbody>
</table>
  | SR | 0000 | SR | 0008 | (N = 1)

- **Example 2:**
  - SL W3, #12, W8 ; Shift left W3 by 12
  - ; Store result to W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>0912</td>
</tr>
<tr>
<td>W8</td>
<td>1002</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### SL: Shift Left by Wns

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} SL Wb, Wns, Wnd

**Operands:**

- Wb ∈ [W0 ... W15]
- Wns ∈ [W0 ... W15]
- Wnd ∈ [W0 ... W15]

**Operation:**

- Wns<4:0> → Shift_Val
- Wnd<15:Shift_Val> = Wb<15 – Shift_Val:0>
- Wd<Shift_Val – 1:0> = 0

**Status Affected:** N, Z

**Encoding:**

```
1101 1101 0www wddd d000 ssss
```

**Description:**

Shift left the contents of the source register Wb by the 5 Least Significant bits of Wns (only up to 15 positions) and store the result in the destination register Wnd. Any bits shifted out of the source register are lost. Register Direct Addressing must be used for Wb, Wns and Wnd.

The ‘w’ bits select the address of the base register.

The ‘d’ bits select the destination register.

The ‘s’ bits select the source register.

**Note 1:** This instruction operates in Word mode only.

**Note 2:** If Wns is greater than 15, Wnd will be loaded with 0x0.

**Words:**

1

**Cycles:**

1

---

**Example 1:**

```
SL W0, W1, W2 ; Shift left W0 by W1<0:4>
; Store result to W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0: 09A4</td>
<td>W0: 09A4</td>
</tr>
<tr>
<td>W1: 8903</td>
<td>W1: 8903</td>
</tr>
<tr>
<td>W2: 78A9</td>
<td>W2: 4D20</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>

**Example 2:**

```
SL W4, W5, W6 ; Shift left W4 by W5<0:4>
; Store result to W6
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4: A409</td>
<td>W4: A409</td>
</tr>
<tr>
<td>W5: FF01</td>
<td>W5: FF01</td>
</tr>
<tr>
<td>W6: 0883</td>
<td>W6: 4812</td>
</tr>
<tr>
<td>SR: 0000</td>
<td>SR: 0000</td>
</tr>
</tbody>
</table>
SUB

Subtract WREG from f

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{(label:)\} SUB{.B} f {,WREG}\}

Operands:

f \in [0...8191]

Operation:

(f) – (WREG) \rightarrow \text{destination designated by D}

Status Affected:

DC, N, OV, Z, C

Encoding:

| 1011 | 0101 | 0Bdf | ffff | ffff | ffff |

Description:

Subtract the contents of the default Working register WREG 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 register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

**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.

**Note 2:** The WREG is set to Working register W0.

Words: 1

Cycles: 1

---

**Example 1:** SUB.B 0x1FFF ; Sub. WREG from (0x1FFF) (Byte mode)

; Store result to 0x1FFF

Before Instruction: | After Instruction:

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>Data 1FFE</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7804</td>
<td>9439</td>
<td>0000</td>
</tr>
<tr>
<td>9439</td>
<td>7804</td>
<td>0001</td>
</tr>
</tbody>
</table>

(C = 1)

**Example 2:** SUB 0xA04, WREG ; Sub. WREG from (0xA04) (Word mode)

; Store result to WREG

Before Instruction: | After Instruction:

<table>
<thead>
<tr>
<th>WREG (W0)</th>
<th>Data 0A04</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>6234</td>
<td>4523</td>
<td>0000</td>
</tr>
<tr>
<td>4523</td>
<td>E2EF</td>
<td>0008</td>
</tr>
</tbody>
</table>

(N = 1)

---

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 "Multicycle Instructions".
Section 5. Instruction Descriptions

SUB

Subtract Literal from Wn

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} SUB{.B} #lit10, Wn

Operands:
- lit10 ∈ [0 ... 255] for byte operation
- lit10 ∈ [0 ... 1023] for word operation
- Wn ∈ [W0 ... W15]

Operation: (Wn) – lit10 → Wn

Status Affected: DC, N, OV, Z, C

Encoding:

| 1011 0001 0Bkk kkkk kkkk dddd |

Description:
Subtract the 10-bit unsigned literal operand from the contents of the Working register Wn and store the result back in the Working register Wn. Register Direct Addressing must be used for Wn.

The ‘B’ bit selects byte or word operation.

The ‘k’ bits specify the literal operand.

The ‘d’ bits select the address of the Working register.

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.

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.

Words: 1

Cycles: 1

Example 1:
SUB.B #0x23, W0    ; Sub. 0x23 from W0 (Byte mode)
; Store result to W0

Before Instruction

| W0  | 7804 |
| SR  | 0000 |

After Instruction

| W0  | 78E1 |
| SR  | 0008 | (N = 1)

Example 2:
SUB    #0x108, W4   ; Sub. 0x108 from W4 (Word mode)
; Store result to W4

Before Instruction

| W4  | 6234 |
| SR  | 0000 |

After Instruction

| W4  | 612C |
| SR  | 0001 | (C = 1)
**SUB**

**Subtract Short Literal from Wb**

**Implemented in:**
- PIC24F
- PIC24H
- PIC24E
- dsPIC30F
- dsPIC33F
- dsPIC33E
- dsPIC33C

**Syntax:**
```
(label:) SUB{.B} Wb, #lit5, Wd
```

**Operands:**
- **Wb** \(\in [W0 \ldots W15]\)
- **lit5** \(\in [0 \ldots 31]\)
- **Wd** \(\in [W0 \ldots W15]\)

**Operation:**
\((Wb) - \text{lit5} \rightarrow Wd\)

**Status Affected:**
- DC, N, OV, Z, C

**Encoding:**
```
0101 0www wBqq qddd d11k kkkk
```

**Description:**
Subtract the 5-bit unsigned literal operand from the contents of the base register \(Wb\) and place the result in the destination register \(Wd\). Register Direct Addressing must be used for \(Wb\). Register Direct or Indirect Addressing must be used for \(Wd\).

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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘k’ bits provide the literal operand, a five-bit integer number.

**Note:** 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

**Example 1:**
```
SUB.B W4, #0x10, W5 ; Sub. 0x10 from W4 (Byte mode)
; Store result to W5
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 1782</td>
<td>W4 1782</td>
</tr>
<tr>
<td>W5 7804</td>
<td>W5 7872</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0005 (OV, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:**
```
SUB W0, #0x8, [W2++] ; Sub. 0x8 from W0 (Word mode)
; Store result to [W2]
; Post-increment W2
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 F230</td>
<td>W0 F230</td>
</tr>
<tr>
<td>W2 2004</td>
<td>W2 2006</td>
</tr>
<tr>
<td>Data 2004 A557</td>
<td>Data 2004 F228</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0009 (N, C = 1)</td>
</tr>
</tbody>
</table>

---

**SUB**

Subtract Ws from Wb

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | X |

Syntax: `{label:} SUB{.B} Wb, Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- [++Ws], [++Wd]
- [--Ws], [--Wd]

Operands:
- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

Operation: 
(Wb) – (Ws) → 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 or 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

**Note:** 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

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
**Example 1:** `SUB.B W0, W1, W0` ; Sub. W1 from W0 (Byte mode) ; Store result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 1732</td>
<td>W0 17EE</td>
</tr>
<tr>
<td>W1 7844</td>
<td>W1 7844</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0108 (DC, N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:** `SUB W7, [W8++], [W9++]` ; Sub. [W8] from W7 (Word mode) ; Store result to [W9] ; Post-increment W8 ; Post-increment W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 2450</td>
<td>W7 2450</td>
</tr>
<tr>
<td>W8 1808</td>
<td>W8 180A</td>
</tr>
<tr>
<td>W9 2020</td>
<td>W9 2022</td>
</tr>
<tr>
<td>Data 1808 92E4</td>
<td>Data 1808 92E4</td>
</tr>
<tr>
<td>Data 2020 A557</td>
<td>Data 2020 916C</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 010C (DC, N, OV = 1)</td>
</tr>
</tbody>
</table>
Section 5. Instruction Descriptions

SUB

Subtract Accumulators

Implemented in: PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C
X | X | X | X | X

Syntax: {label:} SUB Acc

Operands: Acc ∈ [A,B]

Operation:
If (Acc = A):
ACCA – ACCB → ACCA

Else:
ACCB – ACCA → ACCB

Status Affected: OA, OB, OAB, SA, SB, SAB

Encoding:

| 1100 1011 | A011 | 0000 | 0000 | 0000 |

Description:
Subtract the contents of the unspecified accumulator from the contents of Acc and store the result back into Acc. This instruction performs a 40-bit subtraction. The ‘A’ bit specifies the destination accumulator.

Words: 1
Cycles: 1

Example 1:
SUB A
; Subtract ACCB from ACCA
; Store the result to ACCA
; CORCON = 0x0000 (no saturation)

Before Instruction
ACCA 76 120F 098A
ACCB 23 F312 BC17
CORCON 0000
SR 0000

After Instruction
ACCA 52 1EFC 4D73
ACCB 23 F312 BC17
CORCON 0000
SR 1100 (OA, OB = 1)

Example 2:
SUB B
; Subtract ACCA from ACCB
; Store the result to ACCB
; CORCON = 0x0040 (SATB = 1)

Before Instruction
ACCA FF 9022 2EE1
ACCB 00 2456 8F4C
CORCON 0040
SR 0000

After Instruction
ACCA FF 9022 2EE1
ACCB 00 7FFF FFFF
CORCON 0040
SR 1400 (SB, SAB = 1)
SUBB
Subtract WREG and Carry Bit from f

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} SUBB{.B} f {,WREG} 

Operands: 
f ∈ [0 ... 8191]

Operation: (f) – (WREG) – (̅C) → destination designated by D

Status Affected: 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, ̅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 register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

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.

2: The WREG is set to Working register W0.

3: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1

Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1: SUBB.B 0x1FFF ; Sub. WREG and ̅C from (0x1FFF) (Byte mode) ; Store result to 0x1FFF

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>7804</td>
</tr>
<tr>
<td>Data 1FFE</td>
<td>9439</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>WREG (W0)</td>
<td>7804</td>
</tr>
<tr>
<td>Data 1FFE</td>
<td>8F39</td>
</tr>
<tr>
<td>SR</td>
<td>0011 (DC, C = 1)</td>
</tr>
</tbody>
</table>

Example 2: SUBB 0xA04, WREG ; Sub. WREG and ̅C from (0xA04) (Word mode) ; Store result to WREG

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>6234</td>
</tr>
<tr>
<td>Data 0A04</td>
<td>6235</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>WREG (W0)</td>
<td>0000</td>
</tr>
<tr>
<td>Data 0A04</td>
<td>6235</td>
</tr>
<tr>
<td>SR</td>
<td>0001 (C = 1)</td>
</tr>
</tbody>
</table>
SUBB
Subtract Wn from Literal with Borrow

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} SUBB{.B} #lit10, Wn

Operands:
lit10 ∈ [0 ... 255] for byte operation
lit10 ∈ [0 ... 1023] for word operation
Wn ∈ [W0 ... W15]

Operation: (Wn) – lit10 – (C) → Wn

Status Affected: DC, N, OV, Z, C

Encoding: 1011 0001 1Bkk kkkk kkkk dddd

Description:
Subtract the unsigned 10-bit literal operand and the Borrow flag (Carry flag inverse, Ĉ) from the contents of the Working register Wn, and store the result back in the Working register Wn. Register Direct Addressing must be used for Wn.

The ‘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.

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.

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.

3: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1
Cycles: 1

Example 1:
SUBB.B #0x23, W0 ; Sub. 0x23 and Ĉ from W0 (Byte mode)
; Store result to W0

Before Instruction
W0  7804
SR  0000

After Instruction
W0  78E0
SR  0108 (DC, N = 1)

Example 2:
SUBB #0x108, W4 ; Sub. 0x108 and Ĉ from W4 (Word mode)
; Store result to W4

Before Instruction
W4  6234
SR  0001 (C = 1)

After Instruction
W4  612C
SR  0001 (C = 1)
### SUBB

**Subtract Short Literal from Wb with Borrow**

#### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```
{label:} SUBB{.B} Wb, #lit5, Wd
```

- **Wb**: \[W0 ... W15\]
- **lit5**: \[0 ... 31\]
- **Wd**: \[W0 ... W15\]

#### Operands:

- **Wb**: \[W0 ... W15\]
- **lit5**: \[0 ... 31\]
- **Wd**: \[W0 ... W15\]

#### Operation:

\((Wb) - \text{lit5} - (\overline{C}) \rightarrow Wd\)

#### Status Affected:

- DC, N, OV, Z, C

#### Encoding:

```
0101 1www wBqq qddd d11k kkkk
```

#### Description:

Subtract the 5-bit unsigned literal operand and the Borrow flag (Carry flag inverse, \(\overline{C}\)) 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 or Indirect Addressing may be used for 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 destination addressing mode.

The 'd' bits select the destination register.

The 'k' bits provide the literal operand, a five-bit integer number.

**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.

**Note 2:** The Z flag is "sticky" for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

#### Words:

1

#### Cycles:

1
**Example 1:** SUBB.B W4, #0x10, W5 ; Sub. 0x10 and C from W4 (Byte mode) ; Store result to W5

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 1782</td>
<td>W4 1782</td>
</tr>
<tr>
<td>W5 7804</td>
<td>W5 7871</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0005 (OV, C = 1)</td>
</tr>
</tbody>
</table>

**Example 2:** SUBB W0, #0x8, [W2++] ; Sub. 0x8 and C from W0 (Word mode) ; Store result to [W2] ; Post-increment W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 0009</td>
<td>W0 0009</td>
</tr>
<tr>
<td>W2 2004</td>
<td>W2 2006</td>
</tr>
<tr>
<td>Data 2004 A557</td>
<td>Data 2004 0000</td>
</tr>
<tr>
<td>SR 0002 (Z = 1)</td>
<td>SR 0103 (DC, Z, C = 1)</td>
</tr>
</tbody>
</table>
SUBB

Subtract Ws from Wb with Borrow

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

(label;) SUBB{.B} Wb, Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[+Ws], [++Wd]

[--Ws], [--Wd]

Operands:

Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:

(Wb) – (Ws) – (C^) → Wd

Status Affected:

DC, N, OV, Z, C

Encoding:

| 0101 | lwww | wBqq | qddd | dppp | ssss |

Description:

Subtract the contents of the source register Ws and the Borrow flag (Carry flag inverse, C^) 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. Register Direct or 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

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.

2: The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words: 1
Cycles: 1

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.
### Example 1: SUBB.B W0, W1, W0
; Sub. W1 and C from W0 (Byte mode)
; Store result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 1732</td>
<td>W0 17ED</td>
</tr>
<tr>
<td>W1 7844</td>
<td>W1 7844</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0108 (DC, N = 1)</td>
</tr>
</tbody>
</table>

### Example 2: SUBB W7,[W8++],[W9++]
; Sub. [W8] and C from W7 (Word mode)
; Store result to [W9]
; Post-increment W8
; Post-increment W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7 2450</td>
<td>W7 2450</td>
</tr>
<tr>
<td>W8 1808</td>
<td>W8 180A</td>
</tr>
<tr>
<td>W9 2022</td>
<td>W9 2024</td>
</tr>
<tr>
<td>Data 1808 92E4</td>
<td>Data 1808 92E4</td>
</tr>
<tr>
<td>Data 2022 A557</td>
<td>Data 2022 916B</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 010C (DC, N, OV = 1)</td>
</tr>
</tbody>
</table>
SUBBR

Subtract f from WREG with Borrow

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
\{label:} SUBBR\{.B\} f {,WREG}  

Operands:  
f \in [0 ... 8191]

Operation:  
(WREG) – (f) – (\overline{C}) \rightarrow \text{destination designated by D}

Status Affected:  
DC, N, OV, Z, C

Encoding:  
\begin{array}{ccccccc} 
1011 & 1101 & 1Bdf & ffff & ffff & ffff 
\end{array}

Description:  
Subtract the contents of the specified file register f and the Borrow flag (Carry flag inverse, \(\overline{C}\)) from the contents of WREG, 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 register (‘0’ for WREG, ‘1’ for file register).
The ‘f’ bits select the address of the file register.

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.

2:  
The WREG is set to Working register W0.

3:  
The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

Words:  
1

Cycles:  
1\(^{(1)}\)

Note 1:  
In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

Example 1:  
SUBBR.B 0x803  
\; \text{Sub. (0x803) and \overline{C} from WREG (Byte mode)}  
\; \text{Store result to 0x803}

Before Instruction | After Instruction
\hline
WREG (W0) | 7804 | WREG (W0) | 7804 
Data 0802 | 9439 | Data 0802 | 6F39 
SR | 0002 (Z = 1) | SR | 0000 

Example 2:  
SUBBR 0xA04, WREG  
\; \text{Sub. (0xA04) and \overline{C} from WREG (Word mode)}  
\; \text{Store result to WREG}

Before Instruction | After Instruction
\hline
WREG (W0) | 6234 | WREG (W0) | FFFE 
Data 0A04 | 6235 | Data 0A04 | 6235 
SR | 0000 | SR | 0008 (N = 1)
### SUBBR

**Subtract Wb from Short Literal with Borrow**

#### Implemented in:
<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:
```
{label:} SUBBR{.B} Wb, #lit5, Wd
```

- `[Wd]`
- `[Wd++]`
- `[Wd--]`
- `[++Wd]`
- `[--Wd]`

#### Operands:
- `Wb` ∈ [W0 ... W15]
- `lit5` ∈ [0 ... 31]
- `Wd` ∈ [W0 ... W15]

#### Operation:
`lit5 – (Wb) – (C) → Wd`

#### Status Affected:
DC, N, OV, Z, C

#### Encoding:
```
0001 lwww wBqq qddd d11k kkkk
```

#### Description:
Subtract the contents of the base register `Wb` and the Borrow flag (Carry flag inverse, C) from the 5-bit unsigned literal, and place the result in the destination register `Wd`. Register Direct Addressing must be used for `Wb`. Register Direct or Indirect Addressing must be used for `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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘k’ bits provide the literal operand, a five-bit integer number.

**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.

**Note 2:** The Z flag is “sticky” for **ADDC, CPB, SUBB** and **SUBBR**. These instructions can only clear Z.

#### Words:
1

#### Cycles:
1
Example 1: \texttt{SUBBR.B W0, #0x10, W1}; Sub. \texttt{W0} and \texttt{C} from 0x10 (Byte mode)
; Store result to \texttt{W1}

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0: F310</td>
<td>W0: F310</td>
</tr>
<tr>
<td>W1: 786A</td>
<td>W1: 7800</td>
</tr>
<tr>
<td>SR: 0003 (Z, C = 1)</td>
<td>SR: 0103 (DC, Z, C = 1)</td>
</tr>
</tbody>
</table>

Example 2: \texttt{SUBBR W0, #0x8, [W2++]} ; Sub. \texttt{W0} and \texttt{C} from 0x8 (Word mode)
; Store result to \texttt{[W2]} ; Post-increment \texttt{W2}

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0: 0009</td>
<td>W0: 0009</td>
</tr>
<tr>
<td>Data 2004</td>
<td>Data 2004</td>
</tr>
<tr>
<td>A557</td>
<td>FFFE</td>
</tr>
<tr>
<td>SR: 0020 (Z = 1)</td>
<td>SR: 0108 (DC, N = 1)</td>
</tr>
</tbody>
</table>
### SUBBR

Subtract Wb from Ws with Borrow

#### Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

{label:} SUBBR{.B} Wb, Ws, Wd

- [Ws], [Wd]
- [Ws++], [Wd++]
- [Ws--], [Wd--]
- [++Ws], [++Wd]
- [--Ws], [--Wd]

#### Operands:

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

#### Operation:

(Ws) – (Wb) – (C\overline{)} \rightarrow Wd

#### Status Affected:

DC, N, OV, Z, C

#### Encoding:

| 0001 | lwww | wBqq | qddd | dppp | ssss |

#### Description:

Subtract the contents of the base register Wb and the Borrow flag (Carry flag inverse, \overline{C}) from the contents of the source register Ws, and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Register Direct or 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**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.

**Note 2:** The Z flag is “sticky” for ADDC, CPB, SUBB and SUBBR. These instructions can only clear Z.

#### Words:

1

#### Cycles:

1

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”.

---

© 2005-2018 Microchip Technology Inc.
**Example 1:**  
```
SUBBR.B W0, W1, W0  ; Sub. W0 and C from W1 (Byte mode)  
                   ; Store result to W0
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>1732</td>
</tr>
<tr>
<td>W1</td>
<td>7844</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**  
```
SUBBR W7, [W8++], [W9++]  ; Sub. W7 and C from [W8] (Word mode)  
                          ; Store result to [W9]  
                          ; Post-increment W8  
                          ; Post-increment W9
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7</td>
<td>2450</td>
</tr>
<tr>
<td>W8</td>
<td>1808</td>
</tr>
<tr>
<td>W9</td>
<td>2022</td>
</tr>
<tr>
<td>Data 1808</td>
<td>92E4</td>
</tr>
<tr>
<td>Data 2022</td>
<td>A557</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
## SUBR

Subtract f from WREG

### Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

{label:} SUBR{.B} f {,WREG}

### Operands:

f ∈ [0 ... 8191]

### Operation:

(WREG) – (f) → destination designated by D

### Status Affected:

DC, N, OV, Z, C

### Encoding:

```
1011 1101 0BDf ffff ffff ffff
```

### Description:

Subtract the contents of the specified file register from the contents of the default Working register WREG 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 register (‘0’ for WREG, ‘1’ for file register).

The ‘f’ bits select the address of the file register.

**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.

**Note 2:** The WREG is set to Working register W0.

### Words:

1

### Cycles:

1(1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.

### Example 1:

```
SUBR.B 0x1FFF ; Sub. (0x1FFF) from WREG (Byte mode)
; Store result to 0x1FFF
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0) 7804</td>
<td>WREG (W0) 7804</td>
</tr>
<tr>
<td>Data 1FFE 9439</td>
<td>Data 1FFE 7039</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

### Example 2:

```
SUBR 0xA04, WREG   ; Sub. (0xA04) from WREG (Word mode)
; Store result to WREG
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0) 6234</td>
<td>WREG (W0) FFFF</td>
</tr>
<tr>
<td>Data 0A04 6235</td>
<td>Data 0A04 6235</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008 (N = 1)</td>
</tr>
</tbody>
</table>
**SUBR**

**Subtract Wb from Short Literal**

Implemented in: | PIC24F | PIC24H | PIC24E | dsPIC30F | dsPIC33F | dsPIC33E | dsPIC33C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
{label;} SUBR{.B} Wb, #lit5, Wd  

- [Wd]  
- [Wd++]  
- [Wd--]  
- [++Wd]  
- [--Wd]

Operands:  
- Wb ∈ [W0 ... W15]  
- lit5 ∈ [0 ... 31]  
- Wd ∈ [W0 ... W15]

Operation:  
lit5 – (Wb) → Wd

Status Affected:  
DC, N, OV, Z, C

Encoding:  
0001 0www wBqq qddd d11k kkkk

Description: 
Subtract the contents of the base register Wb from the unsigned 5-bit literal operand and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or Indirect Addressing may be used for 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘k’ bits provide the literal operand, a five-bit integer number.

**Note:** 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

**Example 1:**  
SUBR.B W0, #0x10, W1  ; Sub. W0 from 0x10 (Byte mode)  
; Store result to W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>F310</td>
</tr>
<tr>
<td>W1</td>
<td>786A</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W0</td>
<td>F310</td>
</tr>
<tr>
<td>W1</td>
<td>7800</td>
</tr>
</tbody>
</table>
| SR | 0103 | (DC, Z, C = 1)

**Example 2:**  
SUBR W0, #0x8, [W2++]  ; Sub. W0 from 0x8 (Word mode)  
; Store result to [W2]  
; Post-increment W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0009</td>
</tr>
<tr>
<td>W2</td>
<td>2004</td>
</tr>
<tr>
<td>Data 2004</td>
<td>A557</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
<tr>
<td>W0</td>
<td>0009</td>
</tr>
<tr>
<td>W2</td>
<td>2006</td>
</tr>
<tr>
<td>Data 2004</td>
<td>FFFF</td>
</tr>
</tbody>
</table>
| SR | 0108 | (DC, N = 1)
**Section 5. Instruction Descriptions**

### SUBR

**Subtract Wb from Ws**

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} SUBR{.B} Wb, Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[+Ws], [+Wd]

[-Ws], [-Wd]

**Operands:**

- Wb ∈ [W0 ... W15]
- Ws ∈ [W0 ... W15]
- Wd ∈ [W0 ... W15]

**Operation:**

(Ws) – (Wb) → Wd

**Status Affected:**

DC, N, OV, Z, C

**Encoding:**

```
0001 0www wBqq qddd dpdp ssst
```

**Description:** Subtract the contents of the base register Wb from the contents of the source register Ws and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

**Note:** 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

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multicycle Instructions”.
Example 1: SUBR.B W0, W1, W0 ; Sub. W0 from W1 (Byte mode)
; Store result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  1732</td>
<td>W0  1712</td>
</tr>
<tr>
<td>W1  7844</td>
<td>W1  7844</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0001 (C = 1)</td>
</tr>
</tbody>
</table>

Example 2: SUBR W7, [W8++], [W9++] ; Sub. W7 from [W8] (Word mode)
; Store result to [W9]
; Post-increment W8
; Post-increment W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W7  2450</td>
<td>W7  2450</td>
</tr>
<tr>
<td>W8  1808</td>
<td>W8  180A</td>
</tr>
<tr>
<td>W9  2022</td>
<td>W9  2024</td>
</tr>
<tr>
<td>Data 1808</td>
<td>Data 1808</td>
</tr>
<tr>
<td>Data 2022</td>
<td>Data 2022</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0005 (OV, C = 1)</td>
</tr>
</tbody>
</table>
## SWAP

**Instruction: SWAP Byte or Nibble Swap Wn**

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Syntax:

{label:} SWAP{.B} Wn

### Operands:

Wn ∈ [W0 ... W15]

### Operation:

**For Byte Operation:**

(Wn)<7:4> ↔ (Wn)<3:0>

**For Word Operation:**

(Wn)<15:8> ↔ (Wn)<7:0>

### Status Affected:

None

### Encoding:

| 1111 | 1101 | 1B00 | 0000 | 0000 | ssss |

### Description:

Swap the contents of the Working register Wn. In Word mode, the two bytes of Wn are swapped. In Byte mode, the two nibbles of the Least Significant Byte of Wn are swapped and the Most Significant Byte of Wn is unchanged. Register Direct Addressing must be used for Wn.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘s’ bits select the address of the Working register.

**Note:** 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

#### Example 1:

SWAP.B W0 ; Nibble swap (W0)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  AB87</td>
<td>W0  AB78</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>

#### Example 2:

SWAP W0 ; Byte swap (W0)

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0  8095</td>
<td>W0  9580</td>
</tr>
<tr>
<td>SR  0000</td>
<td>SR  0000</td>
</tr>
</tbody>
</table>
## TBLRDH

**Table Read High**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\{
label: \}

TBLRDH{.B} \[Ws\], \[Ws++\], \[Ws--\], \[Wd\], \[Wd++\], \[Wd--\], \[+++Wd\], \[--Wd\]

**Operands:**

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

**Operation:**

For Byte Operation:

- If (LSB(Ws) = 1):
  - 0 → Wd
- Else:
  - Program Mem [(TBLPAG),(Ws)] <23:16> → Wd

For Word Operation:

- Program Mem [(TBLPAG),(Ws)] <23:16> → Wd <7:0>
- 0 → Wd <15:8>

**Status Affected:**

None

**Encoding:**

| 1011 | 1010 | 1Bqq | qddd | dppp | ssss |

**Description:**

Read the contents of the most significant word of program memory and store it to the destination register Wd. The target word address of program 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.

In Byte mode, the source address depends on the contents of Ws. If Ws is not word-aligned, zero is stored to the destination register (due to non-existent program memory). If Ws is word-aligned, the third program memory byte (PM<23:16>), at the specified program memory address, is stored to the destination register.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

**Note:** The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

**Words:**

1

**Cycles:**

2 (PIC24F, PIC24H, dsPIC30F, dsPIC33F)

5 (PIC24E, dsPIC33E, dsPIC33C)
Example 1: TBLRDH.B [W0], [W1++] ; Read PM (TBLPAG:[W0]) (Byte mode)
                ; Store to [W1]
                ; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0812</td>
</tr>
<tr>
<td>W1</td>
<td>0F71</td>
</tr>
<tr>
<td>Data 0F70</td>
<td>0944</td>
</tr>
<tr>
<td>Program 01 0812</td>
<td>EF 2042</td>
</tr>
<tr>
<td>TBLPAG</td>
<td>0001</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2: TBLRDH [W6++], W8 ; Read PM (TBLPAG:[W6]) (Word mode)
                ; Store to W8
                ; Post-increment W6

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>3406</td>
</tr>
<tr>
<td>W8</td>
<td>65B1</td>
</tr>
<tr>
<td>Program 00 3406</td>
<td>29 2E40</td>
</tr>
<tr>
<td>TBLPAG</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
### TBLRDL

**Implemented in:**

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBLRDL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} TBLRDL{.B} [Ws], Wd

[Ws++], [Wd]

[Ws--], [Wd++]

[++Ws], [Wd--]

[--Ws], [++Wd]

[--Wd]

**Operands:**

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

**Operation:**

For Byte Operation:

If (LSB(Ws) = 1):

Program Mem [(TBLPAG),(Ws)] <15:8> → Wd

Else:

Program Mem [(TBLPAG),(Ws)] <7:0> → Wd

For Word Operation:

Program Mem [(TBLPAG),(Ws)] <15:0> → Wd

**Status Affected:** None

**Encoding:**

| 1011 | 1010 | 0Bqq | qddd | dppp | ssss |

**Description:**

Read the contents of the least significant word of program memory and store it to the destination register Wd. The target word address of program 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, the lower 2 bytes of program memory are stored to the destination register. In Byte mode, the source address depends on the contents of Ws. If Ws is not word-aligned, the second byte of the program memory word (PM<15:7>) is stored to the destination register. If Ws is word-aligned, the first byte of the program memory word (PM<7:0>) is stored to the destination register.

The 'B' bit selects byte or word operation ('0' for word mode, '1' for byte).

The 'q' bits select the destination addressing mode.

The 'd' bits select the destination register.

The 'p' bits select the source addressing mode.

The 's' bits select the source register.

**Note:** The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

**Words:** 1

**Cycles:**

2 (PIC24F, PIC24H, dsPIC30F, dsPIC33F)

5 (PIC24E, dsPIC33E, dsPIC33C)
### Example 1:

**TBLRDL.B** \([W0++]\), W1 ; Read PM (TBLPAG:[W0]) (Byte mode)

; Store to W1

; Post-increment W0

#### Before Instruction

<table>
<thead>
<tr>
<th>W0</th>
<th>W1</th>
<th>Data 0F70</th>
<th>Program 01 0812</th>
<th>TBLPAG</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0813</td>
<td>0F71</td>
<td>0944</td>
<td>EF 2042</td>
<td>0001</td>
<td>0000</td>
</tr>
</tbody>
</table>

#### After Instruction

<table>
<thead>
<tr>
<th>W0</th>
<th>W1</th>
<th>Data 0F70</th>
<th>Program 01 0812</th>
<th>TBLPAG</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0814</td>
<td>0F20</td>
<td>EF44</td>
<td>EF 2042</td>
<td>0001</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:

**TBLRDL** \([W6] , [W8++]\) ; Read PM (TBLPAG:[W6]) (Word mode)

; Store to W8

; Post-increment W8

#### Before Instruction

<table>
<thead>
<tr>
<th>W6</th>
<th>W8</th>
<th>Data 1202</th>
<th>Program 00 3406</th>
<th>TBLPAG</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3406</td>
<td>1202</td>
<td>658B</td>
<td>29 2E40</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

#### After Instruction

<table>
<thead>
<tr>
<th>W6</th>
<th>W8</th>
<th>Data 1202</th>
<th>Program 00 3406</th>
<th>TBLPAG</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3406</td>
<td>1204</td>
<td>2E40</td>
<td>29 2E40</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
TBLWTH

Table Write High

Implemented in:

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  

{label:} TBLWTH{.B} Ws, [Wd]  

[Ws], [Wd++]  

[Ws++], [Wd--]  

[Ws--], [++Wd]  

[++Ws], [--Wd]  

[--Ws],  

Operands:  

Ws ∈ [W0 ... W15]  

Wd ∈ [W0 ... W15]  

Operation:  

For Byte Operation:  

If (LSB(Wd) = 1):  

NOP  

Else:  

(Ws) → Program Mem [(TBLPAG),(Wd)]<23:16>  

For Word Operation:  

(Ws)<7:0> → Program Mem [(TBLPAG),(Wd)] <23:16>  

Status Affected:  

None  

Encoding:  

| 1011 | 1011 | 1Bqq | qddd | dppp | ssss |

Description:  

Store the contents of the working source register Ws to the most significant word of program memory. The destination word address of program memory is formed by concatenating the 8-bit Table Pointer register, TBLPAG<7:0>, with the Effective Address specified by Wd. Either Direct or Indirect Addressing may be used for Ws and Indirect Addressing must be used for Wd.

Since program memory is 24 bits wide, this instruction can only write to the upper byte of program memory (PM<23:16>). This may be performed using a Wd that is word-aligned in Byte mode or Word mode. If Byte mode is used with a Wd that is not word-aligned, no operation is performed.

The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).

The ‘q’ bits select the destination addressing mode.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

Note: The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

Words:  

1  

Cycles:  

2(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multicycle Instructions”. 
### Example 1:

**TBLWTH.B [W0++], [W1]**  
; Write [W0]... (Byte mode)  
; to PM Latch High (TBLPAG:[W1])  
; Post-increment W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 0812</td>
<td>W0 0814</td>
</tr>
<tr>
<td>W1 0F70</td>
<td>W1 0F70</td>
</tr>
<tr>
<td>Data 0812</td>
<td>Data 0944</td>
</tr>
<tr>
<td>Program 01 0F70</td>
<td>Program 01 0F70</td>
</tr>
<tr>
<td>TBLPAG 0001</td>
<td>TBLPAG 0001</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Note:** Only the program latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

### Example 2:

**TBLWTH W6, [W8++]**  
; Write W6... (Word mode)  
; to PM Latch High (TBLPAG:[W8])  
; Post-increment W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6 0026</td>
<td>W6 0026</td>
</tr>
<tr>
<td>W8 0870</td>
<td>W8 0872</td>
</tr>
<tr>
<td>Program 00 0870</td>
<td>Program 00 0870</td>
</tr>
<tr>
<td>TBLPAG 0000</td>
<td>TBLPAG 0000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Note:** Only the program latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.
### TBLWTL

**Table Write Low**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

#### Syntax:

```
{label:} TBLWTL{.B} Ws, [Wd]
[Ws], [Wd++]
[Ws++], [Wd--]
[Ws--], [++Wd]
[++Ws], [--Wd]
[--Ws], [--Wd]
```

#### Operands:

- $Ws \in [W0 \ldots W15]$
- $Wd \in [W0 \ldots W15]$

#### Operation:

**For Byte Operation:**

- If $(\text{LSB}(Wd) = 1)$:
  - $(Ws) \rightarrow \text{Program Mem} [\text{TBLPAG},(Wd)] <15:8>$
- else:
  - $(Ws) \rightarrow \text{Program Mem} [\text{TBLPAG},(Wd)] <7:0>$

**For Word Operation:**

- $(Ws) \rightarrow \text{Program Mem} [\text{TBLPAG},(Wd)] <15:0>$

#### Status Affected:

None

#### Encoding:

```
1011 1011 0Bqq qddd dppp ssss
```

#### Description:

Store the contents of the working source register $Ws$ to the least significant word of program memory. The destination word address of program memory is formed by concatenating the 8-bit Table Pointer register, TBLPAG<7:0>, with the Effective Address specified by $Wd$. Either Direct or Indirect Addressing may be used for $Ws$ and Indirect Addressing must be used for $Wd$.

In Word mode, $Ws$ is stored to the lower 2 bytes of program memory. In Byte mode, the Least Significant bit of $Wd$ determines the destination byte. If $Wd$ is not word-aligned, $Ws$ is stored to the second byte of program memory (PM<15:8>). If $Wd$ is word-aligned, $Ws$ is stored to the first byte of program memory (PM<7:0>).

The 'B' bit selects byte or word operation ('0' for word, '1' for byte).
The 'q' bits select the destination addressing mode.
The 'd' bits select the destination register.
The 'p' bits select the source addressing mode.
The 's' bits select the source register.

**Note:** The extension .B in the instruction denotes a byte move rather than a word move. You may use a .W extension to denote a word move, but it is not required.

#### Words:

1

#### Cycles:

2(1)

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 "Multicycle Instructions".
Section 5. Instruction Descriptions

Example 1:  
TBLWTL.B W0, [W1++] ; Write W0... (Byte mode)
; to PM Latch Low (TBLPAG:[W1])
; Post-increment W1

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>6628</td>
</tr>
<tr>
<td>W1</td>
<td>1225</td>
</tr>
<tr>
<td>Program 00 1224</td>
<td>Program 01 1224</td>
</tr>
<tr>
<td>TBLPAG</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Note: Only the program latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.

Example 2:  
TBLWTL [W6], [W8] ; Write [W6]... (Word mode)
; to PM Latch Low (TBLPAG:[W8])
; Post-increment W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W6</td>
<td>1600</td>
</tr>
<tr>
<td>W8</td>
<td>7208</td>
</tr>
<tr>
<td>Data 1600</td>
<td>Data 1600</td>
</tr>
<tr>
<td>Program 01 7208</td>
<td>Program 01 7208</td>
</tr>
<tr>
<td>TBLPAG</td>
<td>0001</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Note: Only the program latch is written to. The contents of program memory are not updated until the Flash memory is programmed using the procedure described in the specific device family reference manual.
ULNK

Deallocate Stack Frame

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} ULNK

Operands:

None

Operation:

W14 → W15
(W15) – 2 → W15
(TOS) → W14

Status Affected:

None

Encoding:

1111 1010 1000 0000 0000 0000

Description:

This instruction deallocates a stack frame for a subroutine calling sequence. The stack frame is deallocated by setting the Stack Pointer (W15) equal to the Frame Pointer (W14) and then POPping the stack to reset the Frame Pointer (W14).

Words: 1

Cycles: 1

Example 1:

ULNK ; Unlink the stack frame

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W14 2002</td>
<td>W14 2000</td>
</tr>
<tr>
<td>W15 20A2</td>
<td>W15 2000</td>
</tr>
<tr>
<td>Data 2000</td>
<td>Data 2000</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

Example 2:

ULNK ; Unlink the stack frame

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W14 0802</td>
<td>W14 0800</td>
</tr>
<tr>
<td>W15 0812</td>
<td>W15 0800</td>
</tr>
<tr>
<td>Data 0800</td>
<td>Data 0800</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
ULNK

Deallocate Stack Frame

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax: {label:} ULNK

Operands: None

Operation:

$W14 \rightarrow W15$

$(W15) - 2 \rightarrow W15$

$(TOS) \rightarrow W14$

$0 \rightarrow SFA$ bit

Status Affected: SFA

Encoding:

```
1111 1010 1000 0000 0000 0000
```

Description: This instruction deallocates a stack frame for a subroutine calling sequence. The stack frame is deallocated by setting the Stack Pointer ($W15$) equal to the Frame Pointer ($W14$) and then POPping the stack to reset the Frame Pointer ($W14$).

Words: 1

Cycles: 1

Example 1:

```
ULNK ; Unlink the stack frame
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W14$</td>
<td>2002</td>
</tr>
<tr>
<td>$W15$</td>
<td>20A2</td>
</tr>
<tr>
<td>Data</td>
<td>2000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

Example 2:

```
ULNK ; Unlink the stack frame
```

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W14$</td>
<td>0802</td>
</tr>
<tr>
<td>$W15$</td>
<td>0812</td>
</tr>
<tr>
<td>Data</td>
<td>0800</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
VFSLV  Verify Slave Processor Program RAM

Implemented in:  
<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Syntax:  
{label:} VFSLV [Wns], [Wnd++], #lit2
        [Wns++]

Operands:  
Wns ∈ [W0 ... W15];
Wnd ∈ [W0 ... W15];
lit2 ∈ [0 ... 3];

Operation:  
If Master (EAs) ≠ Slave EAd
          Master VERFERR (MSIxSTAT<11>) = 1

Status Affected:  None

Encoding:  
0000 0011 10kk 0ddd d0p1 ssss

Description:  
This instruction reads a single instruction word from the target Slave PRAM image (held in the Master program space Flash) and compares it to the value in the Slave PRAM at the destination address. The source address must be located within PSV address space (i.e., DSRPAG ≥ 0x200). The destination address uses DSWPAG and the destination EA to create a 24-bit Slave program space PRAM write address.

Starting with an aligned double instruction word (destination address, see Note 1), the contents of the source Effective Address (in Master program space) are compared with the destination Effective Address (in the Slave PRAM address space) in order to verify the PRAM contents.

If the (single instruction word) destination address is even, the data is captured in the Slave PRAM wrapper. If the (single instruction word) destination address is odd, the ECC parity bits are calculated from the current and captured source data (48 bits), and compared. If the data and ECC parity are not the same, the VERFERR (MSIxSTAT<11>) status bit is set.

The target Slave processor is selected by the value defined by lit2.

The instruction may be regarded as a PSV operation, and therefore, may be executed within a REPEAT loop to accelerate data processing.

The 's' bits select the address of the source register.
The 'd' bits select the address of the destination register.
The 'k' bits select the target Slave processor.
The 'p' bit selects the destination addressing mode (see Note 1).

Note 1:  This instruction supports a subset of addressing modes. The source addressing mode bit field is constrained to 2 options and the destination addressing mode bit field is not required.

2:  An aligned double instruction word destination address is an even address that addresses the least significant word of a double instruction word.

3:  This instruction only supports Word mode.

Words:  1
Cycles:  1
Section 5. Instruction Descriptions

XOR

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax: {label:} XOR{.B} f {,WREG}

Operands: f ∈ [0 ... 8191]

Operation: (f).XOR.(WREG) → destination designated by D

Status Affected: N, Z

Encoding:

| 1011 | 0110 | 0BDf | ffff | ffff | ffff |

Description: Compute the logical exclusive OR operation of the contents of the default Working register WREG and 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 register ('0' for WREG, '1' for file register).

The 'f' bits select the address of the file register.

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.

2: The WREG is set to working register W0.

Words: 1
Cycles: 1(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.

Example 1: XOR.B 0x1FFF ; XOR (0x1FFF) and WREG (Byte mode)
; Store result to 0x1FFF

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>Data 1FFE</td>
<td>Data 1FFE</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008</td>
</tr>
</tbody>
</table>

Example 2: XOR 0xA04, WREG ; XOR (0xA04) and WREG (Word mode)
; Store result to WREG

Before Instruction | After Instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WREG (W0)</td>
<td>WREG (W0)</td>
</tr>
<tr>
<td>Data 0A04</td>
<td>Data 0A04</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0008</td>
</tr>
</tbody>
</table>
**XOR**

**Exclusive OR Literal and Wn**

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:  
\{label:\} XOR{.B} #lit10, Wn

Operands:
- lit10 \(\in [0 \ldots 255]\) for byte operation
- lit10 \(\in [0 \ldots 1023]\) for word operation
- Wn \(\in [W0 \ldots W15]\)

Operation:  
lit10.XOR.(Wn) \(\rightarrow\) Wn

Status Affected:  
N, Z

Encoding:

| 1011 | 0010 | 1Bkk | kkkk | kkkk | dddd |

Description:
Compute the logical exclusive OR operation of the unsigned 10-bit literal operand and the contents of the Working register Wn, and store the result back in the Working register Wn. Register Direct Addressing must be used for Wn.

The ‘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.

**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.

**Note 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.

Words: 1

Cycles: 1

**Example 1:**
XOR.B #0x23, W0  ; XOR 0x23 and W0 (Byte mode)
; Store result to W0

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 7804</td>
<td>W0 7827</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>

**Example 2:**
XOR #0x108, W4      ; XOR 0x108 and W4 (Word mode)
; Store result to W4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4 6134</td>
<td>W4 603C</td>
</tr>
<tr>
<td>SR 0000</td>
<td>SR 0000</td>
</tr>
</tbody>
</table>
## XOR

**Exclusive OR Wb and Short Literal**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

\[
{\text{label:}} \quad \text{XOR}\{.B\} \quad \text{Wb}, \quad \#\text{lit5}, \quad \text{Wd}
\]

- \[\text{[Wd]}\]
- \[\text{Wd}++\]
- \[\text{Wd}--\]
- \[\text{++Wd}\]
- \[-\text{Wd}\]

**Operands:**

- \(\text{Wb} \in \{W0 \ldots W15\}\)
- \(\text{lit5} \in \{0 \ldots 31\}\)
- \(\text{Wd} \in \{W0 \ldots W15\}\)

**Operation:**

\((\text{Wb}).\text{XOR.lit5} \rightarrow \text{Wd}\)

**Status Affected:**

\(N, Z\)

**Encoding:**

```
0110 1www wBqq qddd dllk kkkk
```

**Description:**

Compute the logical exclusive OR operation of the contents of the base register Wb and the unsigned 5-bit literal operand, and place the result in the destination register Wd. Register Direct Addressing must be used for Wb. Either Register Direct or Indirect Addressing may be used for 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 destination addressing mode.

The ‘d’ bits select the destination register.

The ‘k’ bits provide the literal operand, a 5-bit integer number.

**Note:** 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

**Example 1:**

\(\text{XOR.B W4, } \#0x14, \text{ W5} \); XOR W4 and 0x14 (Byte mode)

\(\); Store result to W5

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W4</td>
<td>C822</td>
</tr>
<tr>
<td>W5</td>
<td>1200</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Example 2:**

\(\text{XOR W2, } \#0x1F, [W8++]\); XOR W2 by 0x1F (Word mode)

\(\); Store result to [W8]

\(\); Post-increment W8

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>8505</td>
</tr>
<tr>
<td>W8</td>
<td>1004</td>
</tr>
<tr>
<td>Data 1004</td>
<td>6628</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

| Data 1004          | 851A              |
| SR                 | 0008\(\text{ (N = 1)}\) |
XOR

Exclusive OR Wb and Ws

Implemented in:

<table>
<thead>
<tr>
<th></th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Syntax:

\{label:\} XOR\{.B\} Wb, Ws, Wd

\[Ws\], \[Wd\]

\[Ws++\], \[Wd++\]

\[Ws--\], \[Wd--\]

\[++Ws\], \[++Wd\]

\[--Ws\], \[--Wd\]

Operands:

Wb ∈ [W0 ... W15]
Ws ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation:

(Wb).XOR.(Ws) → Wd

Status Affected:

N, Z

Encoding:

| 0110 | lwww | wBqg | qddd | dppp | ssss |

Description:

Compute the logical exclusive OR operation of the contents of the source register Ws and 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 or 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 destination addressing mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source addressing mode.
The ‘s’ bits select the source register.

Note: 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(1)

Note 1: In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see Note 3 in Section 3.2.1 “Multi-Cycle Instructions”.
## Section 5. Instruction Descriptions

**Example 1:** XOR B W1, [W5++], [W9++] ; XOR W1 and [W5] (Byte mode) ; Store result to [W9] ; Post-increment W5 and W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>AAAA</td>
</tr>
<tr>
<td>W5</td>
<td>2000</td>
</tr>
<tr>
<td>W9</td>
<td>2600</td>
</tr>
<tr>
<td>Data 2000</td>
<td>115A</td>
</tr>
<tr>
<td>Data 2600</td>
<td>0000</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>AAAA</td>
</tr>
<tr>
<td>W5</td>
<td>2001</td>
</tr>
<tr>
<td>W9</td>
<td>2601</td>
</tr>
<tr>
<td>Data 2000</td>
<td>115A</td>
</tr>
<tr>
<td>Data 2600</td>
<td>00F0</td>
</tr>
<tr>
<td>SR</td>
<td>0008 (N = 1)</td>
</tr>
</tbody>
</table>

**Example 2:** XOR W1, W5, W9 ; XOR W1 and W5 (Word mode) ; Store the result to W9

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>FEDC</td>
</tr>
<tr>
<td>W5</td>
<td>1234</td>
</tr>
<tr>
<td>W9</td>
<td>A34D</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>FEDC</td>
</tr>
<tr>
<td>W5</td>
<td>1234</td>
</tr>
<tr>
<td>W9</td>
<td>ECE8</td>
</tr>
<tr>
<td>SR</td>
<td>0008 (N = 1)</td>
</tr>
</tbody>
</table>
**ZE**

Zero-Extend Ws

---

**Implemented in:**

<table>
<thead>
<tr>
<th>PIC24F</th>
<th>PIC24H</th>
<th>PIC24E</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
<th>dsPIC33E</th>
<th>dsPIC33C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label;} ZE Ws, Wnd

Ws, Ws++, Ws--,

[Ws],

[--Ws],

[++]Ws,

[––Ws],

**Operands:**

Ws ∈ [W0 ... W15]

Wnd ∈ [W0 ... W15]

**Operation:**

Ws<7:0> → Wnd<7:0>

0 → Wnd<15:8>

**Status Affected:**

N, Z, C

**Encoding:**

```
1111 1011 1000 0ddd dppp ssss
```

**Description:**

Zero-extend the Least Significant Byte in source Working register Ws to a 16-bit value and store the result in the destination Working register Wnd. Either Register Direct or Indirect Addressing may be used for Ws and Register Direct Addressing must be used for Wnd. The N flag is cleared and the C flag is set because the zero-extended word is always positive.

The ‘d’ bits select the destination register.

The ‘p’ bits select the source addressing mode.

The ‘s’ bits select the source register.

- **Note 1:** This operation converts a byte to a word and it uses no .B or .W extension.
- **Note 2:** The source Ws is addressed as a byte operand, so any address modification is by one.

**Words:**

1

**Cycles:**

1\(^{(1)}\)

---

**Note 1:** In dsPIC33E, dsPIC33C and PIC24E devices, the listed cycle count does not apply to read and Read-Modify-Write operations on non-CPU Special Function Registers. For more details, see **Note 3** in Section 3.2.1 “Multi-Cycle Instructions”.

---
### Example 1:

ZE W3, W4  
; zero-extend W3  
; Store result to W4

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>7839</td>
</tr>
<tr>
<td>W4</td>
<td>1005</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>

### Example 2:

ZE [W2++], W12  
; Zero-extend [W2]  
; Store to W12  
; Post-increment W2

<table>
<thead>
<tr>
<th>Before Instruction</th>
<th>After Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>0900</td>
</tr>
<tr>
<td>W12</td>
<td>1002</td>
</tr>
<tr>
<td>Data 0900</td>
<td>268F</td>
</tr>
<tr>
<td>SR</td>
<td>0000</td>
</tr>
</tbody>
</table>
Section 6. Built-in Functions

HIGHLIGHTS

This section of the manual contains the following major topics:

6.1 Introduction ................................................................................................................... 460
6.2 Built-in Function List...................................................................................................... 461
6.1 INTRODUCTION

This section describes the built-in functions that are specific to the MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs (formerly MPLAB C30).

Built-in functions give the C programmer access to assembler operators or machine instructions that are currently only accessible using in-line assembly, but are sufficiently useful that they are applicable to a broad range of applications. Built-in functions are coded in C source files syntactically like function calls, but they are compiled to assembly code that directly implements the function and does not involve function calls or library routines.

There are a number of reasons why providing built-in functions is preferable to requiring programmers to use in-line assembly. They include the following:
1. Providing built-in functions for specific purposes simplifies coding.
2. Certain optimizations are disabled when in-line assembly is used. This is not the case for built-in functions.
3. For machine instructions that use dedicated registers, coding in-line assembly while avoiding register allocation errors can require considerable care. The built-in functions make this process simpler as you do not need to be concerned with the particular register requirements for each individual machine instruction.

The built-in functions are listed below followed by their individual detailed descriptions.

- __builtin_addab
- __builtin_add
- __builtin_btg
- __builtin_clr
- __builtin_clr_prefetch
- __builtin_divf
- __builtin_divmodsd
- __builtin_divmodud
- __builtin_divsd
- __builtin_divud
- __builtin_dmaoffset
- __builtin_ed
- __builtin_edac
- __builtin_edsoffset
- __builtin_edspage
- __builtin_fbcl
- __builtin_lac
- __builtin_mac
- __builtin_modsd
- __builtin_modud
- __builtin_movsac
- __builtin_mpy
- __builtin_mpy
- __builtin_mpy
- __builtin_msc
- __builtin_mulss
- __builtin_mulsu
- __builtin_mul
- __builtin_nop
- __builtin_psvpage
- __builtin_psvoffset
- __builtin_readsfr
- __builtin_return_address
- __builtin_sac
- __builtin_sacr
- __builtin_sftac
- __builtin_subab
- __builtin_tbladdress
- __builtin_tbp
- __builtin_tbplpage
- __builtin_tbloffset
- __builtin_sblac
- __builtin_szac
- __builtin_tbdh
- __builtin_tbdl
- __builtin_tblwth
- __builtin_tbw

This section describes only the built-in functions related to the CPU operations. The compiler provides additional built-in functions for operations, such as writing to Flash program memory and changing the oscillator settings. Refer to the "MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide" (DS51284) for a complete list of compiler built-in functions.
6.2 BUILT-IN FUNCTION LIST

This section describes the programmer interface to the compiler built-in functions. Since the functions are “built-in”, there are no header files associated with them. Similarly, there are no command-line switches associated with the built-in functions – they are always available. The built-in function names are chosen such that they belong to the compiler’s namespace (they all have the prefix: __builtin_), so they will not conflict with function or variable names in the programmer’s namespace.

__builtin_addab

Description:
Adds Accumulators A and B with the result written back to the specified accumulator. For example:

```c
register int result asm("A");
register int B asm("A");
result = __builtin_addab(result, B);
```

will generate:

add A

Prototype:

```c
int __builtin_addab(int Accum_a, int Accum_b);
```

Argument:

- `Accum_a` First accumulator to add.
- `Accum_b` Second accumulator to add.

Return Value:

Returns the addition result to an accumulator.

Assembler Operator/Machine Instruction:

add

Error Messages:

An error message appears if the result is not an Accumulator register.
__builtin_add

Description:
Adds value to the accumulator specified by result with a shift specified by literal shift. For example:

```c
register int result asm("A");
int value;
result = __builtin_add(result,value,0);
```

If value is held in w0, the following will be generated:
```
add w0, #0, A
```

Prototype:
```
int __builtin_add(int Accum, int value, const int shift);
```

Argument:
- Accum:
  Accumulator to add.
- value:
  Integer number to add to accumulator value.
- shift:
  Amount to shift resultant accumulator value.

Return Value:
Returns the shifted addition result to an accumulator.

Assembler Operator/Machine Instruction:
```
add
```

Error Messages:
An error message appears if:
- The result is not an Accumulator register
- Argument 0 is not an accumulator
- The shift value is not a literal within range
**__builtin_btg**

**Description:**
This function will generate a `btg` machine instruction. Some examples include:

```c
int i;   /* near by default */
int l __attribute__((far));

struct foo {
    int bit1:1;
} barbits;

int bar;

void some_bittoggles() {
    register int j asm("w9");
    int k;
    k = i;
    __builtin_btg(&i,1);
    __builtin_btg(&j,3);
    __builtin_btg(&k,4);
    __builtin_btg(&l,11);
    return j+k;
}
```

Note that taking the address of a variable in a register will produce a warning by the compiler and cause the register to be saved onto the stack (so that its address may be taken); this form is not recommended. This caution only applies to variables explicitly placed in registers by the programmer.

**Prototype:**

```c
void __builtin_btg(unsigned int *, unsigned int 0xn);
```

**Argument:**
- `*` A pointer to the data item for which a bit should be toggled.
- `0xn` A literal value in the range of 0 to 15.

**Return Value:**
Returns a `btg` machine instruction.

**Assembler Operator/Machine Instruction:**

`btg`

**Error Messages:**
An error message appears if the parameter values are not within range.
__builtin_clr

Description:
Clears the specified accumulator. For example:

register int result asm("A");
result = __builtin_clr();

will generate:
clr A

Prototype:
int __builtin_clr(void);

Argument:
None.

Return Value:
Returns the cleared value result to an accumulator.

Assembler Operator/Machine Instruction:
clr

Error Messages:
An error message appears if the result is not an Accumulator register.
Section 6. Built-in Functions

__builtin_clr_prefetch

Description:
Clears an accumulator and prefetch data ready for a future MAC operation.
xptr may be null to signify no X prefetch to be performed; in which case, the values of xincr and xval are ignored, but required.
yptr may be null to signify no Y prefetch to be performed; in which case, the values of yincr and yval are ignored, but required.
xval and yval nominate the address of a C variable where the prefetched value will be stored.
xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.
If AWB is non-null, the other accumulator will be written back into the referenced variable.

For example:

```
register int result asm("A");
register int B asm("B");
int x_memory_buffer[256]
__attribute__((space(xmemory)));
int y_memory_buffer[256]
__attribute__((space(ymemory)));
int *xmemory;
int *ymemory;
int awb;
int xVal, yVal;
xmemory = x_memory_buffer;
ymemory = y_memory_buffer;
result = __builtin_clr(&xmemory, &xVal, 2,
&ymemory, &yVal, 2, &awb, B);
```

May generate:

```
clr A, [w8]+=2, w4, [w10]+=2, w5, w13
```

The compiler may need to spill w13 to ensure that it is available for the Write-Back. It may be recommended to users that the register be claimed for this purpose.

After this instruction:
• Result will be cleared
• xVal will contain x_memory_buffer[0]
• yVal will contain y_memory_buffer[0]
• xmemory and ymemory will be incremented by 2, ready for the next MAC operation

Prototype:

```
int __builtin_clr_prefetch(
    int **xptr, int *xval, int xincr,
    int **yptr, int *yval, int yincr, int *AWB,
    int AWB_accum);
```
__builtin_clr_prefetch (Continued)

Argument:

- `xptr`  Integer Pointer to X prefetch.
- `xval`  Integer value of X prefetch.
- `xincr` Integer increment value of X prefetch.
- `yptr`  Integer Pointer to Y prefetch.
- `yval`  Integer value of Y prefetch.
- `yincr` Integer increment value of Y prefetch.
- `AWB`   Accumulator Write-Back location.
- `AWB_accum` Accumulator to Write-Back.

Note: The arguments, `xptr` and `yptr`, must point to the arrays located in the X data memory and Y data memory, respectively.

Return Value:

Returns the cleared value result to an accumulator.

Assembler Operator/Machine Instruction:

`clr`

Error Messages:

An error message appears if:

- The result is not an Accumulator register
- `xval` is a null value but `xptr` is not null
- `yval` is a null value but `yptr` is not null
- `AWB_accum` is not an accumulator and `AWB` is not null
### __builtin_divf

**Description:**
Computes the quotient: num / den. A math error exception occurs if den is zero. Function arguments are unsigned, as is the function result.

**Prototype:**
unsigned int __builtin_divf(unsigned int num, unsigned int den);

**Argument:**
- num Numerator.
- den Denominator.

**Return Value:**
Returns the unsigned integer value of the quotient: num / den.

**Assembler Operator/Machine Instruction:**
div.f

### __builtin_divmodsd

**Description:**
Issues the 16-bit architecture's native signed divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture both the quotient and remainder.

**Prototype:**
signed int __builtin_divmodsd(
signed long dividend, signed int divisor,
signed int *remainder);

**Argument:**
- dividend Number to be divided.
- divisor Number to divide by.
- remainder Pointer to remainder.

**Return Value:**
Quotient and remainder.

**Assembler Operator/Machine Instruction:**
divmodsd

**Error Messages:**
None.
__builtin_divmodud

Description:
Issues the 16-bit architecture’s native unsigned divide support. Notably, if the quotient does not
fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in
function will capture both the quotient and remainder.

Prototype:
unsigned int __builtin_divmodud(
unsigned long dividend, unsigned int divisor,
unsigned int *remainder);

Argument:
dividend Number to be divided.
divisor Number to divide by.
remainder Pointer to remainder.

Return Value:
Quotient and remainder.

Assembler Operator/Machine Instruction:
divmodud

Error Messages:
None.

__builtin_divsd

Description:
Computes the quotient: num / den. A math error exception occurs if den is zero. Function
arguments are signed, as is the function result. The command-line option, -Wconversions,
can be used to detect unexpected sign conversions.

Prototype:
int __builtin_divsd(const long num, const int den);

Argument:
um Numerator.
den Denominator.

Return Value:
Returns the signed integer value of the quotient: num / den.

Assembler Operator/Machine Instruction:
div.sd
__builtin_divud

Description:
Computes the quotient: $num / den$. A math error exception occurs if $den$ is zero. Function arguments are unsigned, as is the function result. The command-line option, `-Wconversions`, can be used to detect unexpected sign conversions.

Prototype:
unsigned int __builtin_divud(const unsigned long num, const unsigned int den);

Argument:
num Numerator.
den Denominator.

Return Value:
Returns the unsigned integer value of the quotient: $num / den$.

Assembler Operator/Machine Instruction:
div.ud

__builtin_dmaoffset

Description:
Obtains the offset of a symbol within DMA memory.
For example:

```c
unsigned int result;
char buffer[256] __attribute__((space(dma)));
result = __builtin_dmaoffset(&buffer);
```

May generate:
```
mov #dmaoffset(buffer), w0
```

Prototype:
unsigned int __builtin_dmaoffset(const void *p);

Argument:
*p Pointer to DMA address value.

Return Value:
Returns the offset to a variable located in DMA memory.

Assembler Operator/Machine Instruction:
dmaoffset

Error Messages:
An error message appears if the parameter is not the address of a global symbol.
__builtin_ed

Description:
Squares \( sqr \), returning it as the result. Also prefetches data for future square operation by computing \( xptr - yptr \) and storing the result in \( distance \).
xincr and yincr may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```c
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_ed(distance,
    &xmemory, 2,
    &ymemory, 2,
    &distance);
```

May generate:
```
ed w4*w4, A, [w8]+=2, [W10]+=2, w4
```

Prototype:
```
int __builtin_ed(int sqr, int **xptr, int xincr,
    int **yptr, int yincr, int *distance);
```

Argument:
- \( sqr \) Integer squared value.
- \( xptr \) Integer Pointer to pointer to X prefetch.
- \( xincr \) Integer increment value of X prefetch.
- \( yptr \) Integer Pointer to pointer to Y prefetch.
- \( yincr \) Integer increment value of Y prefetch.
- \( distance \) Integer Pointer to distance.

**Note:** The arguments, \( xptr \) and \( yptr \), must point to the arrays located in the X data memory and Y data memory, respectively.

Return Value:
Returns the squared result to an accumulator.

Assembler Operator/Machine Instruction:
```
ed
```

Error Messages:
An error message appears if:
- The result is not an Accumulator register
- \( xptr \) is null
- \( yptr \) is null
- \( distance \) is null
Section 6. Built-in Functions

__builtin_edac

Description:
Squares \( s^{qr} \) and sums with the nominated Accumulator register, returning it as the result. Also prefetches data for future square operation by computing \( \star x^{ptr} - \star y^{ptr} \) and storing the result in \( \star distance \).

\( x^{incr} \) and \( y^{incr} \) may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

For example:

```c
register int result asm("A");
int *xmemory, *ymemory;
int distance;

result = __builtin_ed(result, distance,
                      &xmemory, 2,
                      &ymemory, 2,
                      &distance);
```

May generate:

```
edac w4\times w4, A, [w8]+=2, [W10]+=2, w4
```

Prototype:

```c
int __builtin_edac(int Accum, int sqr,
                   int **xptr, int xincr, int **yptr, int yincr,
                   int *distance);
```

Argument:

- **Accum**
  - Accumulator to sum.
- **sqr**
  - Integer squared value.
- **xptr**
  - Integer Pointer to pointer to X prefetch.
- **xincr**
  - Integer increment value of X prefetch.
- **yptr**
  - Integer Pointer to pointer to Y prefetch.
- **yincr**
  - Integer increment value of Y prefetch.
- **distance**
  - Integer Pointer to distance.

**Note:** The arguments, \( x^{ptr} \) and \( y^{ptr} \), must point to the arrays located in the X data memory and Y data memory, respectively.

Return Value:

Returns the squared result to specified accumulator.

**Assembler Operator/Machine Instruction:**

```
edac
```

**Error Messages:**

An error message appears if:

- The result is not an Accumulator register
- **Accum** is not an Accumulator register
- **xptr** is null
- **yptr** is null
- **distance** is null
__builtin_edsoffset

Description:
Returns the EDS page offset of the object whose address is given as a parameter. The argument \textit{p} must be the address of an object in Extended Data Space; otherwise, an error message is produced and the compilation fails. See the \texttt{space} attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

Prototype:

\begin{verbatim}
unsigned int __builtin_edsoffset(int *p);
\end{verbatim}

Argument:
\textit{p} Object address.

Return Value:
Returns the EDS page number of the object whose address is given as a parameter.

Assembler Operator/Machine Instruction:
edsoffset

__builtin_edspage

Description:
Returns the EDS page number of the object whose address is given as a parameter. The argument \textit{p} must be the address of an object in Extended Data Space; otherwise, an error message is produced and the compilation fails. See the \texttt{space} attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

Prototype:

\begin{verbatim}
unsigned int __builtin_edspage(int *p);
\end{verbatim}

Argument:
\textit{p} Object address.

Return Value:
Returns the EDS page number of the object whose address is given as a parameter.

Assembler Operator/Machine Instruction:
edspage
__builtin_fbcl

**Description:**
Finds the first bit change from left in value. This is useful for dynamic scaling of fixed-point data. For example:

```c
int result, value;
result = __builtin_fbcl(value);
```

May generate:

```
fbcl w4, w5
```

**Prototype:**

```c
int __builtin_fbcl(int value);
```

**Argument:**

- `value` Integer number of first bit change.

**Return Value:**

Returns the shifted addition result to an accumulator.

**Assembler Operator/Machine Instruction:**

```
fbcl
```

**Error Messages:**

An error message appears if the result is not an Accumulator register.

__builtin_lac

**Description:**
Shifts value by `shift` (a literal between -8 and 7) and returns the value to be stored into the Accumulator register. For example:

```c
register int result asm("A");
int value;
result = __builtin_lac(value, 3);
```

May generate:

```
lac w4, #3, A
```

**Prototype:**

```c
int __builtin_lac(int value, int shift);
```

**Argument:**

- `value` Integer number to be shifted.
- `shift` Literal amount to shift.

**Return Value:**

Returns the shifted addition result to an accumulator.

**Assembler Operator/Machine Instruction:**

```
lac
```

**Error Messages:**

An error message appears if:

- The result is not an Accumulator register
- The shift value is not a literal within range
__builtin_mac

**Description:**
Computes $a \times b$ and sums with accumulator; also, prefetches data ready for a future MAC operation.

$xptr$ may be null to signify no X prefetch to be performed; in which case, the values of $xincr$ and $xval$ are ignored, but required.

$yptr$ may be null to signify no Y prefetch to be performed; in which case, the values of $yincr$ and $yval$ are ignored, but required.

$xval$ and $yval$ nominate the address of a C variable where the prefetched value will be stored.

$xincr$ and $yincr$ may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If $AWB$ is non-null, the other accumulator will be written back into the referenced variable.

For example:
```c
register int result asm("A");
register int B asm("B");
int   *xmemory;
int   *ymemory;
int   xVal, yVal;

result = __builtin_mac(result, xVal, yVal,
                       &xmemory, &xVal, 2,
                       &ymemory, &yVal, 2, 0, B);
```

May generate:
```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

**Prototype:**
```c
int __builtin_mac(int Accum, int a, int b,
                   int **xptr, int *xval, int xincr,
                   int **yptr, int *yval, int yincr, int *AWB,
                   int AWB_accum);
```

**Argument:**
- $Accum$ Accumulator to sum.
- $a$ Integer multiplicand.
- $b$ Integer multiplier.
- $xptr$ Integer Pointer to pointer to X prefetch.
- $xval$ Integer Pointer to value of X prefetch.
- $xincr$ Integer increment value of X prefetch.
- $yptr$ Integer Pointer to pointer to Y prefetch.
- $yval$ Integer Pointer to value of Y prefetch.
- $yincr$ Integer increment value of Y prefetch.
- $AWB$ Accumulator Write-Back location.
- $AWB_accum$ Accumulator to Write-Back.

**Note:** The arguments, $xptr$ and $yptr$, must point to the arrays located in the X data memory and Y data memory, respectively.

**Return Value:**
Returns the cleared value result to an accumulator.

**Assembler Operator/Machine Instruction:**
```
mac
```
Error Messages:

An error message appears if:

- The result is not an Accumulator register
- `Accum` is not an Accumulator register
- `xval` is a null value but `xptr` is not null
- `yval` is a null value but `y.ptr` is not null
- `AWB_accum` is not an Accumulator register and `AWN` is not null
__builtin_modsd

**Description:**
Issues the 16-bit architecture’s native signed divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture only the remainder.

**Prototype:**
```c
signed int __builtin_modsd(signed long dividend, signed int divisor);
```

**Argument:**
- `dividend` Number to be divided.
- `divisor` Number to divide by.

**Return Value:**
Remainder.

**Assembler Operator/Machine Instruction:**
`modsd`

**Error Messages:**
None.

__builtin_modud

**Description:**
Issues the 16-bit architecture’s native unsigned divide support. Notably, if the quotient does not fit into a 16-bit result, the results (including remainder) are unexpected. This form of the built-in function will capture only the remainder.

**Prototype:**
```c
unsigned int __builtin_modud(unsigned long dividend, unsigned int divisor);
```

**Argument:**
- `dividend` Number to be divided.
- `divisor` Number to divide by.

**Return Value:**
Remainder.

**Assembler Operator/Machine Instruction:**
`modud`

**Error Messages:**
None.
__builtin_movsac

Description:
Computes nothing, but prefetches data ready for a future MAC operation.

\( xptr \) may be null to signify no X prefetch to be performed; in which case, the values of \( xincr \) and \( xval \) are ignored, but required.

\( yptr \) may be null to signify no Y prefetch to be performed; in which case, the values of \( yincr \) and \( yval \) are ignored, but required.

\( xval \) and \( yval \) nominate the address of a C variable where the prefetched value will be stored.

\( xincr \) and \( yincr \) may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If \( AWB \) is not null, the other accumulator will be written back into the referenced variable.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_movsac(&xmemory, &xVal, 2,
                          &ymemory, &yVal, 2, 0, 0);
```

May generate:

```
movsac A, [w8]+=2, w4, [w10]+=2, w5
```

Prototype:

```c
int __builtin_movsac(
  int **xptr, int *xval, int xincr,
  int **yptr, int *yval, int yincr, int *AWB
  int AWB_accum);
```

Argument:

- \( xptr \) Integer Pointer to pointer to X prefetch.
- \( xval \) Integer Pointer to value of X prefetch.
- \( xincr \) Integer increment value of X prefetch.
- \( yptr \) Integer Pointer to pointer to Y prefetch.
- \( yval \) Integer Pointer to value of Y prefetch.
- \( yincr \) Integer increment value of Y prefetch.
- \( AWB \) Accumulator Write-Back location.
- \( AWB_accum \) Accumulator to Write-Back.

**Note:** The arguments, \( xptr \) and \( yptr \), must point to the arrays located in the X data memory and Y data memory, respectively.

Return Value:

Returns prefetch data.

Assembler Operator/Machine Instruction:

```
movsac
```

Error Messages:

An error message appears if:

- The result is not an Accumulator register
- \( xval \) is a null value but \( xptr \) is not null
- \( yval \) is a null value but \( yptr \) is not null
- \( AWB\_accum \) is not an Accumulator register and \( AWB \) is not null

© 2005-2018 Microchip Technology Inc.
__builtin_mpy

**Description:**
Computes $a \times b$; also, prefetches data ready for a future MAC operation.

$xptr$ may be null to signify no X prefetch to be performed; in which case, the values of $xincr$ and $xval$ are ignored, but required.

$yptr$ may be null to signify no Y prefetch to be performed; in which case, the values of $yincr$ and $yval$ are ignored, but required.

$xval$ and $yval$ nominate the address of a C variable where the prefetched value will be stored.

$xincr$ and $yincr$ may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

**For example:**
```c
register int result asm("A");
int  *xmemory;
int  *ymemory;
int xVal, yVal;

result = __builtin_mpy(xVal, yVal,
                        &xmemory, &xVal, 2,
                        &ymemory, &yVal, 2);
```

May generate:
```assembly
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

**Prototype:**
```c
int __builtin_mpy(int a, int b,
                  int **xptr, int *xval, int xincr,
                  int **yptr, int *yval, int yincr);
```

**Argument:**

- $a$ Integer multiplicand.
- $b$ Integer multiplier.
- $xptr$ Integer Pointer to pointer to X prefetch.
- $xval$ Integer Pointer to value of X prefetch.
- $xincr$ Integer increment value of X prefetch.
- $yptr$ Integer Pointer to pointer to Y prefetch.
- $yval$ Integer Pointer to value of Y prefetch.
- $yincr$ Integer increment value of Y prefetch.
- $AWB$ Integer Pointer to accumulator selection.

**Note:** The arguments, $xptr$ and $yptr$, must point to the arrays located in the X data memory and Y data memory, respectively.

**Return Value:**
Returns the cleared value result to an accumulator.

**Assembler Operator/Machine Instruction:**
```
mpy
```

**Error Messages:**
An error message appears if:
- The result is not an Accumulator register
- $xval$ is a null value but $xptr$ is not null
- $yval$ is a null value but $yptr$ is not null
__builtin_mpyn

Description:
Computes \(-a \times b\); also, prefetches data ready for a future MAC operation.

\(xptr\) may be null to signify no X prefetch to be performed; in which case, the values of \(xincr\)
and \(xval\) are ignored, but required.

\(yptr\) may be null to signify no Y prefetch to be performed; in which case, the values of \(yincr\)
and \(yval\) are ignored, but required.

\(xval\) and \(yval\) nominate the address of a C variable where the prefetched value will be stored.
\(xincr\) and \(yincr\) may be the literal values: \(-6, -4, -2, 0, 2, 4, 6\) or an integer value.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;

result = __builtin_mpy(xVal, yVal,
                        &xmemory, &xVal, 2,
                        &ymemory, &yVal, 2);
```

May generate:
```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

Prototype:
```
int __builtin_mpyn(int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr);
```

Argument:
- \(a\) Integer multiplicand.
- \(b\) Integer multiplier.
- \(xptr\) Integer Pointer to pointer to X prefetch.
- \(xval\) Integer Pointer to value of X prefetch.
- \(xincr\) Integer increment value of X prefetch.
- \(yptr\) Integer Pointer to pointer to Y prefetch.
- \(yval\) Integer Pointer to value of Y prefetch.
- \(yincr\) Integer increment value of Y prefetch.
- \(AWB\) Integer Pointer to accumulator selection.

Note: The arguments, \(xptr\) and \(yptr\), must point to the arrays located in the X data
memory and Y data memory, respectively.

Return Value:
Returns the cleared value result to an accumulator.

Assembler Operator/Machine Instruction:
```
mpyn
```

Error Messages:
An error message appears if:
- The result is not an Accumulator register
- \(xval\) is a null value but \(xptr\) is not null
- \(yval\) is a null value but \(yptr\) is not null
__builtin_msc

Description:
Computes $a \times b$ and subtracts from accumulator; also, prefetches data ready for a future MAC operation.

$xptr$ may be null to signify no X prefetch to be performed; in which case, the values of $xincr$ and $xval$ are ignored, but required.

$yptr$ may be null to signify no Y prefetch to be performed; in which case, the values of $yincr$ and $yval$ are ignored, but required.

$xval$ and $yval$ nominate the address of a C variable where the prefetched value will be stored.

$xincr$ and $yincr$ may be the literal values: -6, -4, -2, 0, 2, 4, 6 or an integer value.

If $AWB$ is non-null, the other accumulator will be written back into the referenced variable.

For example:

```c
register int result asm("A");
int *xmemory;
int *ymemory;
int xVal, yVal;
result = __builtin_msc(result, xVal, yVal, &xmemory, &xVal, 2, &ymemory, &yVal, 2, 0, 0);
```

May generate:

```
msc w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

Prototype:

```c
int __builtin_msc(int Accum, int a, int b, int **xptr, int *xval, int xincr, int **yptr, int *yval, int yincr, int *AWB, int AWB_accum);
```

Argument:

- **Accum**: Accumulator to sum.
- **a**: Integer multiplicand.
- **b**: Integer multiplier.
- **xptr**: Integer Pointer to pointer to X prefetch.
- **xval**: Integer Pointer to value of X prefetch.
- **xincr**: Integer increment value of X prefetch.
- **yptr**: Integer Pointer to pointer to Y prefetch.
- **yval**: Integer Pointer to value of Y prefetch.
- **yincr**: Integer increment value of Y prefetch.
- **AWB**: Accumulator Write-Back location.
- **AWB_accum**: Accumulator to Write-Back.

**Note:** The arguments, $xptr$ and $yptr$, must point to the arrays located in the X data memory and Y data memory, respectively.

Return Value:

Returns the cleared value result to an accumulator.

Assembler Operator/Machine Instruction:

```
msc`
__builtin_msc (Continued)

Error Messages:

An error message appears if:

- The result is not an Accumulator register
- `Accum` is not an Accumulator register
- `xval` is a null value but `xptr` is not null
- `yval` is a null value but `yptr` is not null
- `AWB_accum` is not an Accumulator register and `AWB` is not null
__builtin_mulss

Description:
Computes the product \( p_0 \times p_1 \). Function arguments are signed integers and the function result is a signed long integer. The command-line option, \(-Wconversions\), can be used to detect unexpected sign conversions.

Prototype:
signed long __builtin_mulss(const signed int p0, const signed int p1);

Argument:
p0   Multiplicand.
p1   Multiplier.

Return Value:
Returns the signed long integer value of the product \( p_0 \times p_1 \).

Assembler Operator/Machine Instruction:
mul.ss

__builtin_mulsu

Description:
Computes the product \( p_0 \times p_1 \). Function arguments are integers with mixed signs and the function result is a signed long integer. The command-line option, \(-Wconversions\), can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including Immediate mode for operand \( p_1 \).

Prototype:
signed long __builtin_mulsu(const signed int p0, const unsigned int p1);

Argument:
p0   Multiplicand.
p1   Multiplier.

Return Value:
Returns the signed long integer value of the product \( p_0 \times p_1 \).

Assembler Operator/Machine Instruction:
mul.su
__builtin_mulus

Description:
Computes the product $p_0 \times p_1$. Function arguments are integers with mixed signs and the function result is a signed long integer. The command-line option, -Wconversions, can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction.

Prototype:
```c
signed long __builtin_mulus(const unsigned int p0, const signed int p1);
```

Argument:
- $p_0$  Multiplicand.
- $p_1$  Multiplier.

Return Value:
Returns the signed long integer value of the product $p_0 \times p_1$.

Assembler Operator/Machine Instruction:
mul.us

__builtin_muluu

Description:
Computes the product $p_0 \times p_1$. Function arguments are unsigned integers and the function result is an unsigned long integer. The command-line option, -Wconversions, can be used to detect unexpected sign conversions. This function supports the full range of addressing modes of the instruction, including Immediate mode for operand $p_1$.

Prototype:
```c
unsigned long __builtin_muluu(const unsigned int p0, const unsigned int p1);
```

Argument:
- $p_0$  Multiplicand.
- $p_1$  Multiplier.

Return Value:
Returns the signed long integer value of the product $p_0 \times p_1$.

Assembler Operator/Machine Instruction:
mul.uu
__builtin_nop

Description:
Generates a NOP instruction.

Prototype:
void __builtin_nop(void);

Argument:
None.

Return Value:
Returns a no operation (NOP).

Assembler Operator/Machine Instruction:
NOP

__builtin_psvoffset

Description:
Returns the PSV page offset of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise, an error message is produced and the compilation fails. See the space attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

Prototype:
unsigned int __builtin_psvoffset(const void *p);

Argument:
\( p \) Object address.

Return Value:
Returns the PSV page number offset of the object whose address is given as a parameter.

Assembler Operator/Machine Instruction:
psvoffset

Error Messages:
The following error message is produced when this function is used incorrectly:
“Argument to __builtin_psvoffset() is not the address of an object in code, PSV or EE data section”.

The argument must be an explicit object address.

For example, if \( obj \) is an object in an executable or read-only section, the following syntax is valid:

unsigned page = __builtin_psvoffset(&obj);
__builtin_psvpage

Description:
Returns the PSV page number of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise, an error message is produced and the compilation fails. See the space attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

Prototype:

\[
\text{unsigned int} \quad \text{__builtin_psvpage} (\text{const void *} p);
\]

Argument:

\( p \quad \text{Object address.} \)

Return Value:

Returns the PSV page number of the object whose address is given as a parameter.

Assembler Operator/Machine Instruction:

\text{psvpage}

Error Messages:

The following error message is produced when this function is used incorrectly:

“Argument to \text{__builtin_psvpage}() is not the address of an object in code, PSV or EE data section”.

The argument must be an explicit object address.

For example, if \text{obj} is an object in an executable or read-only section, the following syntax is valid:

\[
\text{unsigned page} = \text{__builtin_psvpage}(\&\text{obj});
\]

__builtin_readsfr

Description:

Reads the SFR.

Prototype:

\[
\text{unsigned int} \quad \text{__builtin_readsfr} (\text{const void *} p);
\]

Argument:

\( p \quad \text{Object address.} \)

Return Value:

Returns the SFR.

Assembler Operator/Machine Instruction:

\text{readsfr}
__builtin_return_address

Description:
Returns the return address of the current function or of one of its callers. For the \texttt{level} argument, a value of 0 yields the return address of the current function, a value of 1 yields the return address of the caller of the current function, and so forth. When \texttt{level} exceeds the current stack depth, 0 will be returned. This function should only be used with a non-zero argument for debugging purposes.

Prototype:

\begin{verbatim}
int __builtin_return_address (const int level);
\end{verbatim}

Argument:
\begin{itemize}
  \item \texttt{level} \quad Number of frames to scan up the call stack.
\end{itemize}

Return Value:
Returns the return address of the current function or of one of its callers.

Assembler Operator/Machine Instruction:
\begin{verbatim}
return_address
\end{verbatim}

__builtin_sac

Description:
Shifts value by \texttt{shift} (a literal between -8 and 7) and returns the value.

For example:
\begin{verbatim}
register int value asm("A");
int result;
result = __builtin_sac(value, 3);
\end{verbatim}

May generate:
\begin{verbatim}
sac A, #3, w0
\end{verbatim}

Prototype:

\begin{verbatim}
int __builtin_sac(int value, int shift);
\end{verbatim}

Argument:
\begin{itemize}
  \item \texttt{value} \quad Integer number to be shifted.
  \item \texttt{shift} \quad Literal amount to shift.
\end{itemize}

Return Value:
Returns the shifted result to an accumulator.

Assembler Operator/Machine Instruction:
\begin{verbatim}
sac
\end{verbatim}

Error Messages:
An error message appears if:
\begin{itemize}
  \item The result is not an Accumulator register
  \item The shift value is not a literal within range
__builtin_sacr

Description:
Shifts value by \textit{shift} (a literal between -8 and 7) and returns the value, which is rounded using the Rounding mode determined by the RND (CORCON<1>) control bit.

For example:
```c
register int value asm("A");
int result;

result = __builtin_sacr(value,3);
```
May generate:
```asm
sac.r A, #3, w0
```

Prototype:
```
int __builtin_sacr(int value, int shift);
```

Argument:
- \texttt{value}  Integer number to be shifted.
- \texttt{shift}  Literal amount to shift.

Return Value:
Returns the shifted result to the CORCON register.

Assembler Operator/Machine Instruction:
```
sacr
```

Error Messages:
An error message appears if:
- The result is not an Accumulator register
- The shift value is not a literal within range
__builtin_sftac

Description:
Shifts accumulator by shift. The valid shift range is -16 to 16.

For example:
```
register int result asm("A");
int i;

result = __builtin_sftac(result, i);
```

May generate:
sftac A, w0

Prototype:
```
int __builtin_sftac(int Accum, int shift);
```

Argument:
```
Accum    Accumulator to shift.
shift    Amount to shift.
```

Return Value:
Returns the shifted result to an accumulator.

Assembler Operator/Machine Instruction:
sftac

Error Messages:
An error message appears if:
- The result is not an Accumulator register
- Accum is not an Accumulator register
- The shift value is not a literal within range
**__builtin_subab**

Description:
Subtracts Accumulators A and B with the result written back to the specified accumulator. For example:

```c
register int result asm("A");
register int B asm("B");
result = __builtin_subab(result,B);
```

Will generate:

```asm
sub A
```

Prototype:

```c
int __builtin_subab(int Accum_a, int Accum_b);
```

Argument:
- `Accum_a`: Accumulator from which to subtract.
- `Accum_b`: Accumulator to subtract.

Return Value:
Returns the subtraction result to an accumulator.

Assembler Operator/Machine Instruction:

```
sub
```

Error Messages:
An error message appears if the result is not an Accumulator register.

---

**__builtin_tbladdress**

Description:
Returns a value that represents the address of an object in program memory. The argument `p` must be the address of an object in an EE data, PSV or executable memory space; otherwise, an error message is produced and the compilation fails. See the `space` attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

Prototype:

```c
unsigned long __builtin_tblpage(const void *p);
```

Argument:
- `p`: Object address.

Return Value:
Returns an `unsigned long` value that represents the address of an object in program memory.

Assembler Operator/Machine Instruction:

```
tbladdress
```

Error Messages:
The following error message is produced when this function is used incorrectly:

“Argument to __builtin_tbladdress() is not the address of an object in code, PSV or EE data section”.

The argument must be an explicit object address.
For example, if `obj` is an object in an executable or read-only section, the following syntax is valid:

```c
unsigned long page = __builtin_tbladdress(&obj);
```
_builtin_tbloffset

Description:
Returns the table page offset of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise, an error message is produced and the compilation fails. See the \textit{space} attribute in \textit{Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide” (DS51284)}.

Prototype:

\[
\text{unsigned int \_\_builtin_tbloffset(const void \*p);}
\]

Argument:
\( p \) Object address.

Return Value:
Returns the table page number offset of the object whose address is given as a parameter.

Assembler Operator/Machine Instruction:
tbloffset

Error Messages:
The following error message is produced when this function is used incorrectly:

“Argument to \_\_builtin_tbloffset() is not the address of an object in code, PSV or EE data section”.

The argument must be an explicit object address.

For example, if \( \text{obj} \) is an object in an executable or read-only section, the following syntax is valid:

\[
\text{unsigned page = \_\_builtin_tbloffset(&obj);} 
\]
### __builtin_tblpage

**Description:**
Returns the table page number of the object whose address is given as a parameter. The argument \( p \) must be the address of an object in an EE data, PSV or executable memory space; otherwise, an error message is produced and the compilation fails. See the `space` attribute in Section 2.3.1 “Specifying Attributes of Variables” of the “MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User’s Guide” (DS51284).

**Prototype:**
```
unsigned int __builtin_tblpage(const void *p);
```

**Argument:**
- \( p \) Object address.

**Return Value:**
Returns the table page number of the object whose address is given as a parameter.

**Assembler Operator/Machine Instruction:**
`tblpage`

**Error Messages:**
The following error message is produced when this function is used incorrectly:

“Argument to __builtin_tblpage() is not the address of an object in code, PSV or EE data section”.

The argument must be an explicit object address.
For example, if \( obj \) is an object in an executable or read-only section, the following syntax is valid:
```
unsigned page = __builtin_tblpage(&obj);
```

### __builtin_tblrdh

**Description:**
Issues the \texttt{TBLRDH.W} instruction to read a word from Flash or EE data memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of: `__builtin_tbloffset()` and `__builtin_tblpage()`.

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

**Prototype:**
```
unsigned int __builtin_tblrdh(unsigned int offset);
```

**Argument:**
- `offset` Desired memory offset.

**Return Value:**
None.

**Assembler Operator/Machine Instruction:**
`tblrdh`

**Error Messages:**
None.
__builtin_tblrdl

Description:
Issues the TBLRD.L.W instruction to read a word from Flash or EE data memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of:
__builtin_tbloffset() and __builtin_tblpage().
Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

Prototype:
unsigned int __builtin_tblrdl(unsigned int offset);

Argument:
offset  Desired memory offset.

Return Value:
None.

Assembler Operator/Machine Instruction:
tblrld

Error Messages:
None.

__builtin_tblwth

Description:
Issues the TBLWTH.W instruction to write a word to Flash or EE data memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of:
__builtin_tbloffset() and __builtin_tblpage().
Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

Prototype:
void __builtin_tblwth(unsigned int offset
unsigned int data);

Argument:
offset  Desired memory offset.
data     Data to be written.

Return Value:
None.

Assembler Operator/Machine Instruction:
tblwth

Error Messages:
None.
__builtin_tblwtl

Description:
Issues the TBLRD.W instruction to write a word to Flash or EE data memory. You must set up the TBLPAG to point to the appropriate page. To do this, you may make use of: __builtin_tbloffset() and __builtin_tblpage().

Please refer to the specific device data sheet or the appropriate family reference manual for complete details regarding reading and writing program Flash.

Prototype:

void __builtin_tblwtl(unsigned int offset
unsigned int data);

Argument:
offset Desired memory offset.
data Data to be written.

Return Value:
None.

Assembler Operator/Machine Instruction:
tblwtl

Error Messages:
None.
16-Bit MCU and DSC Programmer’s Reference Manual
Example 6-1:

Additional In-Line Functions

#include "p33fxxxx.h"
volatile
volatile
volatile
volatile
volatile

long
long
long
long
long

Result_mpy1616;
Result_addab;
Result_subab;
Result_mpy3216;
Result_div3216;

register
register

int
int

Accu_A asm("A");
Accu_B asm("B");

inline

static

long mpy_32_16 (long, int);

inline
static long mpy_32_16 (long x, int y)
{
long
result;
int
temp1, temp2;
temp1 = (x>>1)&0x7FFF;
temp2 = x>>16;
Accu_A = __builtin_mpy (temp1, y, 0,0,0,0,0,0);
Accu_A = __builtin_sftac (15);
Accu_A = __builtin_mac (temp2, y, 0,0,0,0,0,0,0);
asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(result) : "w"(Accu_A));
return result;
}
int main (void)
{
// Variable declarations
int
Input1;
int
Input2;
int
Input3;
int
Input4;
long
Input5;
int
Input6;
long
Input7;
int
Input8;
// Enable 32-bit saturation, signed and fractional modes for both ACCA
and ACCB
CORCON = 0x00C0;
// Example of 16*16-bit fractional multiplication using ACCA
Input1 = 32767;
Input2 = 32767;
Accu_A = __builtin_mpy (Input1, Input2, 0,0,0,0,0,0);
asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(Result_mpy1616) : "w"(Accu_A));
// Example of 16*16-bit fractional multiplication using ACCB
Input3 = 16384;
Input4 = 16384;
Accu_B = __builtin_mpy (Input3, Input4, 0,0,0,0,0,0);
asm("mov _ACCBL,%0\n\t"
"mov _ACCBH,%d0" : "=r"(Result_mpy1616) : "w"(Accu_B));
// Example of 32-bit addition using ACCA (ACCA = ACCA + ACCB)
Accu_A = __builtin_addab();
asm("mov _ACCAL,%0\n\t"
"mov _ACCAH,%d0" : "=r"(Result_addab) : "w"(Accu_A));
// Example of 32-bit subtraction using ACCB (ACCB = ACCB - ACCA)
Accu_B = __builtin_subab();
asm("mov _ACCBL,%0\n\t"
"mov _ACCBH,%d0" : "=r"(Result_subab) : "w"(Accu_B));
// Example of 32*16-bit fractional multiplication using ACCA
Input5 = 0x7FFFFFFF;
Input6 = 32767;
Result_mpy3216 = mpy_32_16 (Input5, Input6);
while(1);
}

DS70000157G-page 494

© 2005-2018 Microchip Technology Inc.


Example 6-2: Divide_32_by_16

```c
#include <p33Fxxxx.h>
#include "divide.h"

_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_NONE);
_FWDT(FWDTEN_OFF);

unsigned int divide_(long a, int b) {
    union convert {
        unsigned long l;
        unsigned int i[2];
    } c;
    int sign;
    unsigned int result;

    c.l = a;
    sign = c.i[1] ^ b;

    if (a < 0) a = (-a);
    if (b < 0) b = -b;
    result = __builtin_divud(a,b);
    result >>= 1;
    if (sign < 0) result = -result;
    return result;
}

int main(void) {
    unsigned long dividend;
    unsigned int divisor;
    unsigned int quotient;

    dividend = 0x3FFFFFFF;
    divisor = 0x7FFF;

    quotient = divide_((long)dividend, (int)divisor);
    while(1);
}
```
Section 7. Reference

HIGHLIGHTS

This section of the manual contains the following major topics:

7.1 Instruction Bit Map ........................................................................................................ 498
7.2 Instruction Set Summary Table ..................................................................................... 501
7.3 Revision History ............................................................................................................ 511
7.1 INSTRUCTION BIT MAP

Instruction encoding for the 16-bit MCU and DSC family devices is summarized in Table 7-1. This table contains the encoding for the MSB of each instruction. The first column in the table represents bits<23:20> of the opcode and the first row of the table represents bits 19:16 of the opcode. The first byte of the opcode is formed by taking the first column bit value and appending the first row bit value. For instance, the MSB of the PUSH instruction (last row, ninth column) is encoded with '1111000b' (0xF8).

Note: The complete opcode for each instruction may be determined by the instruction descriptions in Section 5. “Instruction Descriptions”, using Table 5-1 through Table 5-15.
### Table 7-1: Instruction Encoding

<table>
<thead>
<tr>
<th>Opcode&lt;19:16&gt;</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NOP</td>
<td>BRA</td>
<td>CALL</td>
<td>GOTO</td>
<td>LDSLV</td>
<td>VFSLV</td>
<td>GOTO</td>
<td>RETLW</td>
<td>RETFIE</td>
<td>RETURN</td>
<td>RCALL</td>
<td>DO†</td>
<td>REPEAT</td>
<td>BFEXT</td>
<td>BFINS</td>
<td>BRA† (OA)</td>
</tr>
<tr>
<td>0001</td>
<td>SUBR</td>
<td>SUBBR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>MOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>MOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>ADD</td>
<td>ADDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>SUB</td>
<td>SUBB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>AND</td>
<td>XOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>IOR</td>
<td>MOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>MOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>MOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>BSET</td>
<td>BCLR</td>
<td>BTG</td>
<td>BTST</td>
<td>BTSTS</td>
<td>BTST</td>
<td>BTSS</td>
<td>BTSC</td>
<td>BSET</td>
<td>BCLR</td>
<td>BTG</td>
<td>BTST</td>
<td>BTSTS</td>
<td>BSW</td>
<td>BTSS</td>
<td>BTSC</td>
</tr>
<tr>
<td>1011</td>
<td>ADDC</td>
<td>SUBB</td>
<td>AND</td>
<td>XOR</td>
<td>IOR</td>
<td>MOV</td>
<td>ADDC</td>
<td>SUBB</td>
<td>AND</td>
<td>XOR</td>
<td>IOR</td>
<td>MOV</td>
<td>MUL.US</td>
<td>MUL.UU</td>
<td>MUL.SS</td>
<td>MUL.SU</td>
</tr>
<tr>
<td>1100</td>
<td>MAC†</td>
<td>MPY†</td>
<td>MPY.N†</td>
<td>MSC†</td>
<td>CLRAC†</td>
<td>MAC†</td>
<td>MPY†</td>
<td>MPY.N†</td>
<td>MSC†</td>
<td>MOVSAC†</td>
<td>SFAC†</td>
<td>ADD†</td>
<td>LAC†</td>
<td>LAC.D†</td>
<td>ADD†</td>
<td>NEQ†</td>
</tr>
<tr>
<td>1101</td>
<td>SL</td>
<td>ASR</td>
<td>LSR</td>
<td>RLC</td>
<td>RLNC</td>
<td>RRC</td>
<td>SL</td>
<td>ASR</td>
<td>LSR</td>
<td>RLC</td>
<td>RLNC</td>
<td>RRC</td>
<td>DIV.S</td>
<td>DIV.U</td>
<td>DIV2.S†</td>
<td>DIV2.U†</td>
</tr>
</tbody>
</table>

**Note 1:** This instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C family devices.

**Note 2:** This instruction is only available in PIC24E and dsPIC33E family devices.

**Note 3:** This instruction is only available in dsPIC33C and some dsPIC33E family devices.

**Note 4:** This instruction is only available in dsPIC33C family devices.

**Note 5:** This instruction is only available in some dsPIC33C, dsPIC33E and PIC24F family devices.
### Table 7-1: Instruction Encoding (Continued)

<table>
<thead>
<tr>
<th>Opcode&lt;19:16&gt;</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>CP0</td>
<td>CP</td>
<td>CPB</td>
<td>CP</td>
<td>CPB</td>
<td>FLIM(4)</td>
<td>FLIM.V(4)</td>
<td>CPBGT(2)</td>
<td>CPBGT(2)</td>
<td>CPBGT(2)</td>
<td>CPBGT(2)</td>
<td>CPBGT(2)</td>
<td>CPBGT(2)</td>
<td>INC</td>
<td>INC2</td>
<td>DEC</td>
</tr>
<tr>
<td>1111</td>
<td>ED(1)</td>
<td>EDAC(1)</td>
<td>MAC(1)</td>
<td>MPY(1)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>PUSH</td>
<td>POP</td>
<td>LNK</td>
<td>ULNK</td>
<td>SE</td>
<td>ZE</td>
<td>DISI</td>
<td>DAW</td>
<td>EXCH</td>
</tr>
</tbody>
</table>

**Note 1:** This instruction is only available in dsPIC30F, dsPIC33F, dsPIC33E, and dsPIC33C family devices.

**Note 2:** This instruction is only available in PIC24E and dsPIC33E family devices.

**Note 3:** This instruction is only available in dsPIC33C and some dsPIC33E family devices.

**Note 4:** This instruction is only available in dsPIC33C family devices.

**Note 5:** This instruction is only available in some dsPIC33C, dsPIC33E, and PIC24F family devices.
## 7.2 INSTRUCTION SET SUMMARY TABLE

The complete 16-bit MCU and DSC device instruction set is summarized in Table 7-2. This table contains an alphabetized listing of the instruction set. It includes instruction assembly syntax, description, size (in 24-bit words), execution time (in instruction cycles), affected Status bits and the page number in which the detailed description can be found. Table 1-2 identifies the symbols that are used in the Instruction Set Summary Table.

**Note:** The instruction cycle counts listed here are for PIC24F, PIC24H, dsPIC30F and dsPIC33F devices. Some instructions require additional cycles in PIC24E and dsPIC33E devices. Refer to Section 3.3 “Instruction Set Summary Tables” and Section 5.4 “Instruction Descriptions” for details.

### Table 7-2: Instruction Set Summary Table

<table>
<thead>
<tr>
<th>Assembly Syntax Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA</th>
<th>OB</th>
<th>SA</th>
<th>SB</th>
<th>OAB</th>
<th>SAB</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD f, (WREG)</td>
<td>Destination = f + WREG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD #lit10,Wn</td>
<td>Wn = lit10 + Wn</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD Wb, #lit5, Wd</td>
<td>Wd = Wb + lit5</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD Wb, Ws, Wd</td>
<td>Wd = Wb + Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD Acc(2)</td>
<td>Add Accumulators</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD Ws, #lit4, Acc</td>
<td>16-Bit Signed Add to Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ADCC f, (WREG)</td>
<td>Destination = f + WREG + (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADCC #lit10, Wn</td>
<td>Wn = lit10 + Wn + (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADCC Wb, #lit5, Wd</td>
<td>Wd = Wb + lit5 + (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADCC Wb, Ws, Wd</td>
<td>Wd = Wb + Ws + (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND f, (WREG)</td>
<td>Destination = f .AND. WREG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND #lit10, Wn</td>
<td>Wn = lit10 .AND. Wn</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND Wb, #lit5, Wd</td>
<td>Wd = Wb .AND. lit5</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND Wb, Ws, Wd</td>
<td>Wd = Wb .AND. Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASR f, (WREG)</td>
<td>Destination = Arithmetic Right Shift f, LSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASR Wb, Ws</td>
<td>Wd = Arithmetic Right Shift Ws, LSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASR Wb, #lit4, Wm</td>
<td>Wnd = Arithmetic Right Shift Wb by lit4, LSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASR Wb, Wm, Wns</td>
<td>Wnd = Arithmetic Right Shift Wb by Wns, LSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BCLR f, #lit4</td>
<td>Bit Clear f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Legend:**
- `0` set or cleared; `1` may be cleared, but never set; `0` may be set, but never cleared; `'1' always set; `'0' always cleared; — unchanged

**Note:**
1. SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
2. This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3. This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4. This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
5. This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
6. This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
7. These instructions are only available in dsPIC33C devices.
8. These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9. These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA</th>
<th>OB</th>
<th>SA</th>
<th>SB</th>
<th>OAB</th>
<th>SAB</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCLR Ws,#bit4</td>
<td>Bit Clear Ws</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>BFEXT #bit4,#wid5,Ws,Wb</td>
<td>Bit Field Extract from Ws to Wb</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>BFEXT #bit4,#wid5,f,Wb</td>
<td>Bit Field Extract from f to Wb</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>131</td>
</tr>
<tr>
<td>BFINS #bit4,#wid5,Wb,Ws</td>
<td>Bit Field Insert from Wb into Ws</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>132</td>
</tr>
<tr>
<td>BFINS #bit4,#wid5,Wb,f</td>
<td>Bit Field Insert from Wb into f</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>133</td>
</tr>
<tr>
<td>BFINS #bit4,#wid5,#lit8,Ws</td>
<td>Bit Field Insert from #lit8 into Ws</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>134</td>
</tr>
<tr>
<td>BOOTSWP</td>
<td>Swap Active and Inactive Program Flash Spaces</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>135</td>
</tr>
<tr>
<td>BRA Expr</td>
<td>Branch Unconditionally</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>136</td>
</tr>
<tr>
<td>BRA Wn</td>
<td>Computed Branch</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>137</td>
</tr>
<tr>
<td>BRA Wn</td>
<td>Computed Branch</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>138</td>
</tr>
<tr>
<td>BRA C Expr</td>
<td>Branch if Carry</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>139</td>
</tr>
<tr>
<td>BRA GE Expr</td>
<td>Branch if Signed Greater Than or Equal</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>141</td>
</tr>
<tr>
<td>BRA GEU Expr</td>
<td>Branch if Unsigned Greater Than or Equal</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>142</td>
</tr>
<tr>
<td>BRA GT Expr</td>
<td>Branch if Signed Greater Than</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>143</td>
</tr>
<tr>
<td>BRA GTU Expr</td>
<td>Branch if Unsigned Greater Than</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>144</td>
</tr>
<tr>
<td>BRA LE Expr</td>
<td>Branch if Signed Less Than or Equal</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>145</td>
</tr>
<tr>
<td>BRA LEU Expr</td>
<td>Branch if Unsigned Less Than or Equal</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>146</td>
</tr>
<tr>
<td>BRA LT Expr</td>
<td>Branch if Signed Less Than</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>BRA LTU Expr</td>
<td>Branch if Unsigned Less Than</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>148</td>
</tr>
<tr>
<td>BRA N Expr</td>
<td>Branch if Negative</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>BRA NC Expr</td>
<td>Branch if Not Carry</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>BRA NN Expr</td>
<td>Branch if Not Negative</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>BRA NOV Expr</td>
<td>Branch if Not Overflow</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>BRA NZ Expr</td>
<td>Branch if Not Zero</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>BRA QA Expr</td>
<td>Branch if Accumulator A Overflow</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>BRA QAB Expr</td>
<td>Branch if Accumulator B Overflow</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>BRA QV Expr</td>
<td>Branch if Overflow</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>BRA SA Expr</td>
<td>Branch if Accumulator A Saturation</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>BRA SAB Expr</td>
<td>Branch if Accumulator B Saturation</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>158</td>
</tr>
<tr>
<td>BRA Z Expr</td>
<td>Branch if Zero</td>
<td>1</td>
<td>1(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>159</td>
</tr>
<tr>
<td>BSET f,#bit4</td>
<td>Bit Set in f</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>160</td>
</tr>
</tbody>
</table>

Legend: 0 set or cleared; 1 may be cleared, but never set; 0 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged

Note: 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC33F and dsPIC33F devices.
6: This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
7: These instructions are only available in dsPIC33C devices.
8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(1,2)</th>
<th>OB(1,2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Set</td>
<td>Bit Set in Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>161</td>
</tr>
<tr>
<td>Bit Write</td>
<td>Bit Write in Ws&lt;Wb&gt;</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>163</td>
</tr>
<tr>
<td>Bit Toggle</td>
<td>Bit Toggle in f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>165</td>
</tr>
<tr>
<td>Bit Test</td>
<td>Bit Test in Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>166</td>
</tr>
<tr>
<td>Bit Test f, Skip if Clear</td>
<td>Bit Test f, Skip if Clear</td>
<td>1</td>
<td>1 (2 or 3)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>168</td>
</tr>
<tr>
<td>Bit Test Ws, Skip if Clear</td>
<td>Bit Test Ws, Skip if Clear</td>
<td>1</td>
<td>1 (2 or 3)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>Bit Test f, Skip if Set</td>
<td>Bit Test f, Skip if Set</td>
<td>1</td>
<td>1 (2 or 3)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>172</td>
</tr>
<tr>
<td>Bit Test Ws, Skip if Set</td>
<td>Bit Test Ws, Skip if Set</td>
<td>1</td>
<td>1 (2 or 3)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Bit Test f</td>
<td>Bit Test in f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>175</td>
</tr>
<tr>
<td>Bit Test Ws, Skip if Set in f</td>
<td>Bit Test Ws, Skip if Set in f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>Bit Test Ws, Skip if Set in Ws</td>
<td>Bit Test Ws, Skip if Set in Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Call</td>
<td>Call Subroutine</td>
<td>2</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Call</td>
<td>Call Subroutine</td>
<td>2</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Call</td>
<td>Call Indirect Subroutine</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>187</td>
</tr>
<tr>
<td>CALL</td>
<td>Call Indirect Subroutine</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>189</td>
</tr>
<tr>
<td>CALL.L</td>
<td>Call Indirect Subroutine Long (long address)</td>
<td>1</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>191</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear f or WREG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>192</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Wd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>193</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>194</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>196</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>197</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Wd</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>198</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>201</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>202</td>
</tr>
</tbody>
</table>

Legend: § set or cleared;  δ may be cleared, but never set;  0 may be set, but never cleared;  ‘i’ always set;  ‘0’ always cleared;  — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.

3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.

4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.

5: This instruction/operand is only available in PIC24F, dsPIC33F and dsPIC33E devices.

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

7: These instructions are only available in dsPIC33C devices.

8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).

9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(2)</th>
<th>OB(2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP Wb,Wn</td>
<td>Compare (Wb – Wn)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>CP0 f</td>
<td>Compare (f – 0x0)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>CP0 Wn</td>
<td>Compare (Wns – 0x0)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>CPB f</td>
<td>Compare with Borrow (f – WREG – C)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>CPB Wb,#lit5</td>
<td>Compare with Borrow (Wb – lit5 – C)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>CPB Wb,#lit8</td>
<td>Compare with Borrow (Wb – lit8 – C)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>CPB Wb,Ws</td>
<td>Compare with Borrow (Wb – Ws – C)</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>CPBEQ Wb,Wn,Expr</td>
<td>Compare Wb with Wn, Branch if =</td>
<td></td>
<td>1</td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBGT Wb,Wn,Expr</td>
<td>Signed Compare Wb with Wn, Branch if &gt;</td>
<td></td>
<td>1</td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBLT Wb,Wn,Expr</td>
<td>Signed Compare Wb with Wn, Branch if &lt;</td>
<td></td>
<td>1</td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBEQ Wb,Wn,Expr</td>
<td>Compare Wb with Wn, Branch if =</td>
<td></td>
<td>1</td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSEQ Wb,Wn,(3)</td>
<td>Compare (Wb with Wn), Skip if =</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSGT Wb,Wn,(6)</td>
<td>Signed Compare (Wb with Wn), Skip if &gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSGT Wb,Wn,(3)</td>
<td>Signed Compare (Wb with Wn), Skip if &gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSGT Wb,Wn,(6)</td>
<td>Signed Compare (Wb with Wn), Skip if &gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSLT Wb,Wn,(3)</td>
<td>Signed Compare (Wb with Wn), Skip if &lt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSLT Wb,Wn,(6)</td>
<td>Signed Compare (Wb with Wn), Skip if &lt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSNE Wb,Wn,(6)</td>
<td>Signed Compare (Wb with Wn), Skip if ≠</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSNE Wb,Wn,(3)</td>
<td>Signed Compare (Wb with Wn), Skip if ≠</td>
<td></td>
<td>1</td>
<td>1</td>
<td>(2 or 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTXTSWP #lit3</td>
<td>CPU Register Context Swap Literal</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTXTSWP Wn</td>
<td>CPU Register Context Swap Wn</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAW.B Wn</td>
<td>Wn = Decimal Adjust Wn</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC f, (WREG)</td>
<td>Destination = f – 1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC Ws,Wd</td>
<td>Wd = Ws – 1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: 0 set or cleared; 0 may be cleared, but never set; 1 may be set, but never cleared; ’1’ always set; ‘0’ always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC33F and dsPIC33F devices.
6: This instruction/operand is only available in dsPIC33F and dsPIC33F devices.
7: These instructions are only available in dsPIC33C devices.
8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
### Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(2)</th>
<th>OB(2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRC2 f {,WREG}</td>
<td>Destination = f – 2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>DRC2 Ws,Wd</td>
<td>Wd = Ws – 2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>230</td>
</tr>
<tr>
<td>DIS1 #lit14</td>
<td>Disable Interrupts for lit14 Instruction Cycles</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>232</td>
</tr>
<tr>
<td>DIV.S Wm,Wn</td>
<td>Signed 16/16-Bit Integer Divide</td>
<td>1</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>233</td>
</tr>
<tr>
<td>DIV.V Wm,Wn</td>
<td>Signed 16/16-Bit Fractional Divide</td>
<td>1</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>236</td>
</tr>
<tr>
<td>DIV2.S Wm,Wn</td>
<td>Signed 16/16-Bit Integer Divide (W1:W0 preserved)</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>238</td>
</tr>
<tr>
<td>DIV2.0 Wm,Wn</td>
<td>Unsigned 16/16-Bit Integer Divide (W1:W0 preserved)</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>240</td>
</tr>
<tr>
<td>DO #lit14,Expr</td>
<td>Do Code to PC + Expr, (lit14 + 1) Times</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>242</td>
</tr>
<tr>
<td>DO #lit15,Expr</td>
<td>Do Code to PC + Expr, (lit15 + 1) Times</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>244</td>
</tr>
<tr>
<td>DO Wn,Expr</td>
<td>Do Code to PC + Expr, (Wn + 1) Times</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>246</td>
</tr>
<tr>
<td>DO Wn,Expr</td>
<td>Do Code to PC + Expr, (Wn + 1) Times</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>248</td>
</tr>
<tr>
<td>ED Wn*Wn,Acc,[Wx],[Wy],Wxd</td>
<td>Euclidean Distance (no accumulate)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>EDAC Wn*Wn,Acc,[Wx],[Wy],Wxd</td>
<td>Euclidean Distance</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>EXCH Wns,Wnd</td>
<td>Swap Wns and Wnd</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>254</td>
<td></td>
</tr>
<tr>
<td>FBCL Ws,Wnd</td>
<td>Find First Bit Change from Left (MSb) Side</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>FF1L Ws,Wnd</td>
<td>Find First One from Left (MSb) Side</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>257</td>
<td></td>
</tr>
<tr>
<td>FF1R Ws,Wnd</td>
<td>Find First One from Right (LSb) Side</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>FLIM Wb,Ws</td>
<td>Force (signed) Data Range Limit</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>261</td>
<td></td>
</tr>
<tr>
<td>FLIM.V Wb,Ws,Wd</td>
<td>Force (signed) Data Range Limit with Limit Excess Result</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>262</td>
<td></td>
</tr>
<tr>
<td>GOTO Expr</td>
<td>Unconditional Jump</td>
<td>2</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>263</td>
<td></td>
</tr>
<tr>
<td>GOTO Wn</td>
<td>Unconditional Indirect Jump</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>264</td>
<td></td>
</tr>
<tr>
<td>GOTO Wn</td>
<td>Unconditional Indirect Jump</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>GOTO.L Wn</td>
<td>Unconditional Indirect Jump Long</td>
<td>1</td>
<td>4</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>INC f {,WREG}</td>
<td>Destination = f + 1</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td>INC Ws,Wd</td>
<td>Wd = Ws + 1</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>268</td>
<td></td>
</tr>
<tr>
<td>INC2 f {,WREG}</td>
<td>Destination = f + 2</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>269</td>
<td></td>
</tr>
<tr>
<td>INC2 Ws,Wd</td>
<td>Wd = Ws + 2</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>IOR f {,WREG}</td>
<td>Destination = f .IOR. WREG</td>
<td>1</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>271</td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- 0 = set or cleared; 1 = may be cleared, but never set; 2 = may be set, but never cleared; ‘‘1’’ always set; ‘0’ always cleared; --- unchanged

**Note 1:**
- SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
- 1: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
- 2: This instruction/operand is only available in dsPIC30F, dsPIC33E and dsPIC33C devices.
- 3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
- 4: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
- 5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
- 6: This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
- 7: These instructions are only available in dsPIC33C devices.
- 8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
- 9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(2)</th>
<th>OB(2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOR #lit10,Wn</td>
<td>Wn = lit10 .IOR. Wn</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>272</td>
</tr>
<tr>
<td>IOR Wb,#lit5,Wd</td>
<td>Wd = Wb .IOR. lit5</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>273</td>
</tr>
<tr>
<td>IOR Wb,Ws,Wd</td>
<td>Wd = Wb .IOR. Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>274</td>
</tr>
<tr>
<td>LAC Ws,#Slit4,Acc(4)</td>
<td>Load Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>276</td>
</tr>
<tr>
<td>LAC.D Ws,#Slit4,Acc(4)</td>
<td>Load Accumulator Double Word</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>278</td>
</tr>
<tr>
<td>LSDLV [Wns],[Wnd++],#lit7(4)</td>
<td>Move Single Instruction Word from Master to Slave PRAM</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>279</td>
</tr>
<tr>
<td>LNK #lit5(4)</td>
<td>Link Frame Pointer</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>280</td>
</tr>
<tr>
<td>LNK #lit6(4)</td>
<td>Link Frame Pointer</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>281</td>
</tr>
<tr>
<td>LSR f ,WREG</td>
<td>Destination = Logical Right Shift f, MSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>282</td>
</tr>
<tr>
<td>LSR Ws,Wd</td>
<td>Wd = Logical Right Shift Ws, MSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>283</td>
</tr>
<tr>
<td>LSR Wb,#Slit4,Wd</td>
<td>Wd = Logical Right Shift Wb by lit4, MSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>284</td>
</tr>
<tr>
<td>LSR Wb,Wns,Wd</td>
<td>Wd = Logical Right Shift Wb by Wns, MSb → C</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>285</td>
</tr>
<tr>
<td>MAC Wm*Wn,Acc,[Wx],[Wyd],[Wy],Wxd,AWB</td>
<td>Multiply and Accumulate</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>288</td>
</tr>
<tr>
<td>MAC Wm*Wm,Acc,[Wx],[Wyd],[Wy],Wyd</td>
<td>Square and Accumulate</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>290</td>
</tr>
<tr>
<td>MAX Acc(4)</td>
<td>Force Accumulator Maximum Data Range Limit</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>292</td>
</tr>
<tr>
<td>MAX.V Acc,Wd(4)</td>
<td>Force Accumulator Maximum Data Range Limit and Store Limit Excess Result</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>293</td>
</tr>
<tr>
<td>MIN Acc(4)</td>
<td>Force Accumulator Minimum Data Range Limit</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>294</td>
</tr>
<tr>
<td>MIN.V Acc,Wd(4)</td>
<td>Force Accumulator Minimum Data Range Limit and Store Limit Excess Result</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>295</td>
</tr>
<tr>
<td>MINZ Acc(4)</td>
<td>Conditionally Force Accumulator Minimum Data Range Limit if Z Flag is Set</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>296</td>
</tr>
<tr>
<td>MINZ.V Acc,Wd(4)</td>
<td>Conditionally Force Accumulator Minimum Data Range Limit and Store Limit Excess Result if Z Flag is Set</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>297</td>
</tr>
<tr>
<td>MOV f ,WREG</td>
<td>Move f to Destination</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>299</td>
</tr>
<tr>
<td>MOV WREG,f</td>
<td>Move WREG to f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>300</td>
</tr>
<tr>
<td>MOV f ,Wnd</td>
<td>Move f to Wnd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>301</td>
</tr>
<tr>
<td>MOV Wns,f</td>
<td>Move Wns to f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>302</td>
</tr>
<tr>
<td>MOV.B #lit10,Wnd</td>
<td>Move 8-Bit Unsigned Literal to Wnd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>303</td>
</tr>
<tr>
<td>MOV #lit16,Wnd</td>
<td>Move 16-Bit Literal to Wnd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>304</td>
</tr>
</tbody>
</table>

Legend: 0 set or cleared; 0 may be cleared, but never set; 0 may be set, but never cleared; 1 always set; 1 always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
Note 2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
Note 3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
Note 4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
Note 5: This instruction/operand is only available in PIC24F, dsPIC24H, dsPIC30F and dsPIC33F devices.
Note 6: This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
Note 7: These instructions are only available in dsPIC33C devices.
Note 8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
Note 9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(2)</th>
<th>OB(2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV [Ws+Slit10],Wnd</td>
<td>Move [Ws + Slit10] to Wnd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>305</td>
</tr>
<tr>
<td>MOV Ws,Wd</td>
<td>Move Ws to Wd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>307</td>
</tr>
<tr>
<td>MOV.D Wns,Wnd</td>
<td>Move Double Wns:Wns + 1 to Wnd</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>309</td>
</tr>
<tr>
<td>MOV.PAG #lit10,DSRPAG</td>
<td>Move 10-Bit Literal to DSRPAG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>311</td>
</tr>
<tr>
<td>MOV.Wn,DSRPAG</td>
<td>Move Wn to DSRPAG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>312</td>
</tr>
<tr>
<td>MOV.Ws,Wnd</td>
<td>Move Ws to Wnd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>313</td>
</tr>
<tr>
<td>MOV.SAC Acc,[Wx],Wxd,[Wy],Wyd</td>
<td>Move [Wx] to Wxd and [Wy] to Wyd</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>315</td>
</tr>
<tr>
<td>MPY Wm*Wn,Acc</td>
<td>Multiply Wm by Wn to Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>317</td>
</tr>
<tr>
<td>MPY.N Wm*Wn,Acc</td>
<td>-(Multiply Wn by Wm) to Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>319</td>
</tr>
<tr>
<td>MSC Wm*Wn,Acc</td>
<td>Multiply and Subtract from Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>321</td>
</tr>
<tr>
<td>MUL f</td>
<td>W3:W2 = f * WREG</td>
<td>—</td>
<td>—</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>MUL.SS Wb,Ws,Wnd</td>
<td>(Wnd + 1,Wnd) = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>325</td>
</tr>
<tr>
<td>MUL.SS Wb,Ws,Acc(4)</td>
<td>Accumulator = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>326</td>
</tr>
<tr>
<td>MUL.SU Wb,#lit5,Wnd</td>
<td>(Wnd + 1,Wnd) = Signed(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>328</td>
</tr>
<tr>
<td>MUL.SU Wb,Ws,Wnd</td>
<td>(Wnd + 1,Wnd) = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>329</td>
</tr>
<tr>
<td>MUL.SU Wb,#lit5,Acc(4)</td>
<td>Accumulator = Signed(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>331</td>
</tr>
<tr>
<td>MUL.SU Wb,Ws,Acc(4)</td>
<td>Accumulator = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>332</td>
</tr>
<tr>
<td>MUL.US Wb,Ws,Wnd</td>
<td>(Wnd + 1,Wnd) = Signed(Wb) * Unsigned(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>333</td>
</tr>
<tr>
<td>MUL.US Wb,Ws,Acc(4)</td>
<td>Accumulator = Unsigned(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>335</td>
</tr>
<tr>
<td>MUL.UU Wb,Ws,Wnd</td>
<td>(Wnd + 1,Wnd) = Unsigned(Wb) * Unsigned(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>336</td>
</tr>
<tr>
<td>MUL.UU Wb,Ws,Acc(4)</td>
<td>Accumulator = Unsigned(Wb) * Unsigned(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>337</td>
</tr>
<tr>
<td>MUL.UU Wb,#lit5,Acc(4)</td>
<td>Accumulator = Unsigned(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>338</td>
</tr>
<tr>
<td>MUL.UU Wb,Ws,Wnd</td>
<td>(Wnd + 1,Wnd) = Unsigned(Wb) * Unsigned(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>339</td>
</tr>
<tr>
<td>MUL.UU Wb,#lit5,Wnd</td>
<td>(Wnd + 1,Wnd) = Unsigned(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>340</td>
</tr>
<tr>
<td>MUL.WS Wb,Ws,Wnd</td>
<td>Wn = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>341</td>
</tr>
<tr>
<td>MUL.WS Wb,Ws,Acc(4)</td>
<td>Wn = Signed(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>342</td>
</tr>
<tr>
<td>MUL.WS Wb,#lit5,Wnd</td>
<td>Wn = Signed(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>343</td>
</tr>
<tr>
<td>MUL.WS Wb,#lit5,Acc(4)</td>
<td>Wn = Signed(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>344</td>
</tr>
<tr>
<td>MUL.WS Wb,Ws,Wnd</td>
<td>Wn = Unsigned(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>345</td>
</tr>
<tr>
<td>MUL.WS Wb,Ws,Acc(4)</td>
<td>Wn = Unsigned(Wb) * Signed(Ws)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>346</td>
</tr>
<tr>
<td>MUL.WS Wb,#lit5,Wnd</td>
<td>Wn = Unsigned(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>348</td>
</tr>
<tr>
<td>MUL.WS Wb,#lit5,Acc(4)</td>
<td>Wn = Unsigned(Wb) * Unsigned(lit5)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>349</td>
</tr>
</tbody>
</table>

Legend: θ set or cleared; 0 may be cleared, but never set; θ may be set, but never cleared; ‘1’ always set; ‘0’ always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.

2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
6: This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
7: These instructions are only available in dsPIC33C devices.
8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA(2)</th>
<th>OB(2)</th>
<th>SA(1,2)</th>
<th>SB(1,2)</th>
<th>OAB(2)</th>
<th>SAB(1,2)</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG f (,WREG)</td>
<td>Destination = f + 1</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEG Ws,Wd</td>
<td>Wd = Ws + 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>351</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEG Acc(2)</td>
<td>Negate Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>353</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>354</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOPR</td>
<td>No Operation</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>355</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORM Acc,Wd(4)</td>
<td>Normalize Accumulator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>356</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP f</td>
<td>POP TOS to f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>357</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP Wd</td>
<td>POP TOS to Wd</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>358</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP,D Wnd</td>
<td>POP Double from TOS to Wnd:Wnd + 1</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>359</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP,S</td>
<td>POP Shadow Registers</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH f</td>
<td>PUSH f to TOS</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>361</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH Ws</td>
<td>PUSH Ws to TOS</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>362</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH,D Wns</td>
<td>PUSH Double Wns;Wns + 1 to TOS</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>363</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH,S</td>
<td>PUSH Shadow Registers</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>364</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRSAV #lit1</td>
<td>Enter Power-Saving Mode</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>365</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCALL Exp[6]</td>
<td>Relative Call</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>367</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCALL Exp[8]</td>
<td>Relative Call</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>369</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCALL Wd[3]</td>
<td>Computed Relative Call</td>
<td>1</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT #lit14[6]</td>
<td>Repeat Next Instruction (lit14 + 1) Times</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>375</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT #lit15[6]</td>
<td>Repeat Next Instruction (lit15 + 1) Times</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>376</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT Wd[4]</td>
<td>Repeat Next Instruction (Wn + 1) Times</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>377</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT Wd[4]</td>
<td>Repeat Next Instruction (Wn + 1) Times</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>378</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>Software Device Reset</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>379</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETFIE[5]</td>
<td>Return from Interrupt Enable</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>380</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETFIE[3]</td>
<td>Return from Interrupt Enable</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>381</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETLM #lit10,Wn[8]</td>
<td>Return with lit10 in Wn</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>382</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETLM #lit10,Wn[8]</td>
<td>Return with lit10 in Wn</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>384</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN[5]</td>
<td>Return from Subroutine</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>386</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN[3]</td>
<td>Return from Subroutine</td>
<td>1</td>
<td>3</td>
<td>(2)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>387</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: 0 set or cleared; 0 may be cleared, but never set; 0 may be set, but never cleared; '1' always set; '0' always cleared; — unchanged

Note 1: SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
2: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
5: This instruction/operand is only available in dsPIC30F, PIC24F, dsPIC30F and dsPIC33F devices.
6: This instruction/operand is only available in dsPIC30F and dsPIC33F devices.
7: These instructions are only available in dsPIC33C devices.
8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
### Table 7-2: Instruction Set Summary Table (Continued)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Words</th>
<th>Cycles</th>
<th>OA</th>
<th>OB</th>
<th>SA</th>
<th>SB</th>
<th>OAB</th>
<th>SAB</th>
<th>DC</th>
<th>N</th>
<th>OV</th>
<th>Z</th>
<th>C</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLC f (,WREG)</td>
<td>Destination = Rotate Left through Carry f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>RLC Ws,Wd</td>
<td>Wd = Rotate Left through Carry Ws f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>389</td>
<td></td>
</tr>
<tr>
<td>RLNC f (,WREG)</td>
<td>Destination = Rotate Left (no Carry) f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>RLNC Ws,Wd</td>
<td>Wd = Rotate Left (no Carry) Ws f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>392</td>
<td></td>
</tr>
<tr>
<td>RRC f (,WREG)</td>
<td>Destination = Rotate Right through Carry f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RRC Ws,Wd</td>
<td>Wd = Rotate Right through Carry Ws f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>396</td>
<td></td>
</tr>
<tr>
<td>RRNC f (,WREG)</td>
<td>Destination = Rotate Right (no Carry) f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>398</td>
<td></td>
</tr>
<tr>
<td>RRNC Ws,Wd</td>
<td>Wd = Rotate Right (no Carry) Ws f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>399</td>
<td></td>
</tr>
<tr>
<td>SAC Acc,#lit4,Wd</td>
<td>Store Accumulator</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>401</td>
</tr>
<tr>
<td>SAC.D Acc,#lit4,Wd</td>
<td>Store Accumulator Double Word</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>403</td>
</tr>
<tr>
<td>SAC.R Acc,#lit4,Wd</td>
<td>Store Rounded Accumulator</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>404</td>
</tr>
<tr>
<td>SE Ws,Wd</td>
<td>Wd = Sign-Extended Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SETM f</td>
<td>f = 0xFFFF</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SETM Wd</td>
<td>Wd = 0xFFFF</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SFTCAc Acc,#lit6</td>
<td>Arithmetic Shift Accumulator by #lit6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SFTCAc Acc,Wb</td>
<td>Arithmetic Shift Accumulator by (Wb)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SL f (,WREG)</td>
<td>Destination = Arithmetic Left Shift f</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SL Ws,Wd</td>
<td>Wd = Arithmetic Left Shift Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SL Wb,#lit4,Wd</td>
<td>Wd = Left Shift Wb by #lit4</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SL Wb,Wns,Wd</td>
<td>Wd = Left Shift Wb by Wns</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUB f (,WREG)</td>
<td>Destination = f – WREG</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUB #lit10,Wn</td>
<td>Wn = Wn – #lit10</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUB Wb,#lit5,Wd</td>
<td>Wd = Wb – #lit5</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUB Wb,Ws,Wd</td>
<td>Wd = Wb – Ws</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUB Acc(8)</td>
<td>Subtract Accumulators</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUBB f (,WREG)</td>
<td>Destination = f – WREG – (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUBB #lit10,Wn</td>
<td>Wn = Wn – #lit10 – (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUBB Wb,#lit5,Wd</td>
<td>Wd = Wb – #lit5 – (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUBB Wb,Ws,Wd</td>
<td>Wd = Wb – Ws – (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SUBH f (,WREG)</td>
<td>Destination = WREG – f – (C)</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Legend:**
- 1: set or cleared; 0: may be cleared, but never set; 0: may be set, but never cleared; ‘1’ always set; ‘0’ always cleared; — unchanged

**Note:**
1. SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
2. This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
3. This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
4. This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
5. This instruction/operand is only available in PIC24F, PIC24H, dsPIC30F and dsPIC33F devices.
6. This instruction/operand is only available in dsPIC33F devices.
7. These instructions are only available in dsPIC33C devices.
8. These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
9. These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
### Table 7-2: Instruction Set Summary Table (Continued)

| Assembly Syntax          | Description                                                                 | Words | Cycles | OA | OB | SA | SB | OAB | SAB | DC | N  | OV | Z  | C  | Page Number |
|--------------------------|------------------------------------------------------------------------------|-------|--------|----|----|----|----|-----|-----|----|----|----|----|    |            |
| SUBBR Wb,#lit5,Wd       | $Wd = \text{lit5} - Wb - (\overline{C})$                                 | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 431 |
| SUBBR Wb,Ws,Wd          | $Wd = Ws - Wb - (\overline{C})$                                            | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 433 |
| SUBR f (WREG)           | Destination = WREG - f                                                       | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 435 |
| SUBBR Wb,#lit5,Wd       | $Wd = \text{lit5} - Wb$                                                      | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 436 |
| SUBBR Wb,Ws,Wd          | $Wd = Ws - Wb$                                                               | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 437 |
| SWAP Wn                  | $Wn = \text{Byte or Nibble Swap} Wn$                                         | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 439 |
| TBLRDH [Ws],Wd          | Read High Program Word to Wd                                                | 1     | 2      |    |    |    |    |     |     |    |    |    |    | 440 |
| TBLRDL [Ws],Wd          | Read Low Program Word to Wd                                                 | 1     | 2      |    |    |    |    |     |     |    |    |    |    | 442 |
| TBLWTH Ws,[Ws]          | Write Ws to High Program Word                                               | 1     | 2      |    |    |    |    |     |     |    |    |    |    | 444 |
| TBLWTL Ws,[Ws]          | Write Ws to Low Program Word                                                | 1     | 2      |    |    |    |    |     |     |    |    |    |    | 446 |
| ULNK(5)                 | Deallocate Stack Frame                                                       | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 448 |
| ULNR(6)                 | Deallocate Stack Frame                                                       | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 449 |
| VFSLV Wns,Wnd,#lit2(7)  | Verify Slave Processor Program RAM                                          | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 450 |
| XOR f (WREG)            | Destination = f .XOR. WREG                                                   | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 451 |
| XOR #lit10,Wn           | $Wn = \text{lit10} .XOR. Wn$                                                | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 452 |
| XOR Wb,#lit5,Wd         | $Wd = Wb .XOR. \text{lit5}$                                                 | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 453 |
| ZE Ws,Wnd               | $Wd = \text{Zero-Extended Ws}$                                              | 1     | 1      |    |    |    |    |     |     |    |    |    |    | 454 |

**Legend:**
- $\overline{0}$ set or cleared; $\overline{1}$ may be cleared, but never set; $\overline{0}$ may be set, but never cleared; ‘$\checkmark$‘ always set; ‘$\times$‘ always cleared; $\overline{---}$ unchanged

**Note 1:**
- SA, SB and SAB are only modified if the corresponding saturation is enabled, otherwise unchanged.
- 1: This instruction/operand is only available in dsPIC30F, dsPIC33F, dsPIC33E and dsPIC33C devices.
- 2: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
- 3: This instruction/operand is only available in PIC24E, dsPIC33E and dsPIC33C devices.
- 4: This instruction/operand is only available in dsPIC33E and dsPIC33C devices.
- 5: This instruction/operand is only available in PIC24F, PIC24H, dsPIC33F and dsPIC333F devices.
- 6: This instruction/operand is only available in dsPIC33F and dsPIC33F devices.
- 7: These instructions are only available in dsPIC33C devices.
- 8: These instructions are only available in all dsPIC33C devices and some dsPIC33E devices (see device data sheet for details).
- 9: These instructions are only available in all dsPIC33C devices, and some PIC24F and dsPIC33E devices (see device data sheet for details).
7.3 REVISION HISTORY

Revision A (May 2005)
This is the initial release of this document.

Revision B (September 2005)
This revision incorporates all known errata at the time of this document update.

Revision C (February 2008)
This revision includes the following corrections and updates:
- Instruction Updates:
  - Updated BRA Instruction (see “BRA”)
  - Updated DIVF Instruction (see “DIVF”)
  - Updated DO Instruction (see “DO”)
  - Updated SUB instruction (see “SUB”)

Revision D (November 2009)
This revision includes the following corrections and updates:
- Document has been completely redesigned to accommodate all current 16-bit families: dsPIC30F, dsPIC33F, PIC24F and PIC24H

Revision E (June 2010)
This revision includes the following corrections and updates:
- Information specific to dsPIC33E and PIC24E devices has been added throughout the document

Revision F (July 2011)
This revision includes the following corrections and updates:
- Added a new section “Built-in Functions”
- Added and updated the cross-references throughout the document
- Updated the bit characteristics from U to U-0 in Register 2-4 and Register 2-6
- Added a note throughout the document specifying the requirement of an additional cycle for read and read-modify-write operations on non-CPU special function registers in dsPIC33E and PIC24E devices
- Updates to formatting and minor text changes were incorporated throughout the document

Revision G (April 2018)
This revision includes the following corrections and updates:
- Information specific to dsPIC33C devices has been added throughout the document
- Updates to formatting and minor text changes were incorporated throughout the document
**INDEX**

<table>
<thead>
<tr>
<th>Symbols</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>__builtin_add</td>
<td>462</td>
</tr>
<tr>
<td>__builtin_addab</td>
<td>461</td>
</tr>
<tr>
<td>__builtin_btg</td>
<td>463</td>
</tr>
<tr>
<td>__builtin_clr</td>
<td>464</td>
</tr>
<tr>
<td>__builtin_clr_fetch</td>
<td>465</td>
</tr>
<tr>
<td>__builtin_divf</td>
<td>467</td>
</tr>
<tr>
<td>__builtin_divmodd</td>
<td>470</td>
</tr>
<tr>
<td>__builtin_divmodd</td>
<td>471</td>
</tr>
<tr>
<td>__builtin_divmod</td>
<td>472</td>
</tr>
<tr>
<td>__builtin_fcl</td>
<td>473</td>
</tr>
<tr>
<td>__builtin_lac</td>
<td>473</td>
</tr>
<tr>
<td>__builtin_mac</td>
<td>474</td>
</tr>
<tr>
<td>__builtin_modsd</td>
<td>476</td>
</tr>
<tr>
<td>__builtin_modud</td>
<td>476</td>
</tr>
<tr>
<td>__builtin_movsac</td>
<td>477</td>
</tr>
<tr>
<td>__builtin_mpy</td>
<td>478</td>
</tr>
<tr>
<td>__builtin_mpy</td>
<td>479</td>
</tr>
<tr>
<td>__builtin_msc</td>
<td>480</td>
</tr>
<tr>
<td>__builtin_mualss</td>
<td>482</td>
</tr>
<tr>
<td>__builtin_mulsu</td>
<td>482</td>
</tr>
<tr>
<td>__builtin_mul</td>
<td>483</td>
</tr>
<tr>
<td>__builtin_mulu</td>
<td>483</td>
</tr>
<tr>
<td>__builtin_nop</td>
<td>484</td>
</tr>
<tr>
<td>__builtin_psoffset</td>
<td>484</td>
</tr>
<tr>
<td>__builtin_psvpage</td>
<td>485</td>
</tr>
<tr>
<td>__builtin_readsf</td>
<td>485</td>
</tr>
<tr>
<td>__builtin_return_address</td>
<td>486</td>
</tr>
<tr>
<td>__builtin_sac</td>
<td>486</td>
</tr>
<tr>
<td>__builtin_sacr</td>
<td>487</td>
</tr>
<tr>
<td>__builtin_sfacc</td>
<td>488</td>
</tr>
<tr>
<td>__builtin_subab</td>
<td>489</td>
</tr>
<tr>
<td>__builtin_tbladdress</td>
<td>489</td>
</tr>
<tr>
<td>__builtin_tbloffset</td>
<td>490</td>
</tr>
<tr>
<td>__builtin_tblpage</td>
<td>491</td>
</tr>
<tr>
<td>__builtin_tblrdh</td>
<td>491</td>
</tr>
<tr>
<td>__builtin_tblrdi</td>
<td>492</td>
</tr>
<tr>
<td>__builtin_tblwth</td>
<td>492</td>
</tr>
<tr>
<td>__builtin_tblwfl</td>
<td>493</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator A, Accumulator B</td>
<td>20</td>
</tr>
<tr>
<td>Accumulator Access</td>
<td>86</td>
</tr>
<tr>
<td>Accumulator Selection</td>
<td>100</td>
</tr>
<tr>
<td>Accumulator Usage</td>
<td>85</td>
</tr>
<tr>
<td>Addressing Modes for Wd Destination Register</td>
<td>97</td>
</tr>
<tr>
<td>Addressing Modes for Ws Source Register</td>
<td>97</td>
</tr>
<tr>
<td>Architecture Overview</td>
<td>10</td>
</tr>
<tr>
<td>Assigned Working Register Usage</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Field Insert/Extract Instructions</td>
<td>71</td>
</tr>
<tr>
<td>Block Diagrams</td>
<td></td>
</tr>
<tr>
<td>DO Stack Conceptual</td>
<td>26</td>
</tr>
<tr>
<td>dsPIC30F/dsPIC33F Programmer’s Model</td>
<td>16</td>
</tr>
<tr>
<td>dsPIC33C Programmer’s Model</td>
<td>18</td>
</tr>
<tr>
<td>dsPIC33E Programmer’s Model</td>
<td>17</td>
</tr>
<tr>
<td>PIC24E Programmer’s Model</td>
<td>15</td>
</tr>
<tr>
<td>PIC24F/PIC24H Programmer’s Model</td>
<td>14</td>
</tr>
<tr>
<td>Built-In Functions</td>
<td></td>
</tr>
<tr>
<td>__builtin_add</td>
<td>462</td>
</tr>
<tr>
<td>__builtin_addab</td>
<td>461</td>
</tr>
<tr>
<td>__builtin_btg</td>
<td>463</td>
</tr>
<tr>
<td>__builtin_clr</td>
<td>464</td>
</tr>
<tr>
<td>__builtin_clr_fetch</td>
<td>465</td>
</tr>
<tr>
<td>__builtin_divf</td>
<td>467</td>
</tr>
<tr>
<td>__builtin_divmodd</td>
<td>470</td>
</tr>
<tr>
<td>__builtin_fcl</td>
<td>473</td>
</tr>
<tr>
<td>__builtin_lac</td>
<td>473</td>
</tr>
<tr>
<td>__builtin_mac</td>
<td>474</td>
</tr>
<tr>
<td>__builtin_modsd</td>
<td>476</td>
</tr>
<tr>
<td>__builtin_modud</td>
<td>476</td>
</tr>
<tr>
<td>__builtin_movsac</td>
<td>477</td>
</tr>
<tr>
<td>__builtin_mpy</td>
<td>478</td>
</tr>
<tr>
<td>__builtin_mpy</td>
<td>479</td>
</tr>
<tr>
<td>__builtin_msc</td>
<td>480</td>
</tr>
<tr>
<td>__builtin_mualss</td>
<td>482</td>
</tr>
<tr>
<td>__builtin_mulsu</td>
<td>482</td>
</tr>
<tr>
<td>__builtin_mul</td>
<td>483</td>
</tr>
<tr>
<td>__builtin_mulu</td>
<td>483</td>
</tr>
<tr>
<td>__builtin_nop</td>
<td>484</td>
</tr>
<tr>
<td>__builtin_psoffset</td>
<td>484</td>
</tr>
<tr>
<td>__builtin_psvpage</td>
<td>485</td>
</tr>
<tr>
<td>__builtin_readsf</td>
<td>485</td>
</tr>
<tr>
<td>__builtin_return_address</td>
<td>486</td>
</tr>
<tr>
<td>__builtin_sac</td>
<td>486</td>
</tr>
<tr>
<td>__builtin_sacr</td>
<td>487</td>
</tr>
<tr>
<td>__builtin_sfacc</td>
<td>488</td>
</tr>
<tr>
<td>__builtin_subab</td>
<td>489</td>
</tr>
<tr>
<td>__builtin_tbladdress</td>
<td>489</td>
</tr>
<tr>
<td>__builtin_tbloffset</td>
<td>490</td>
</tr>
<tr>
<td>__builtin_tblpage</td>
<td>491</td>
</tr>
<tr>
<td>__builtin_tblrdh</td>
<td>491</td>
</tr>
<tr>
<td>__builtin_tblrdi</td>
<td>492</td>
</tr>
<tr>
<td>__builtin_tblwth</td>
<td>492</td>
</tr>
<tr>
<td>__builtin_tblwfl</td>
<td>493</td>
</tr>
</tbody>
</table>

© 2005-2018 Microchip Technology Inc.
Index

Index

DO (Initialize Hardware Loop Wn) .......................... 242, 244

ED (Euclidean Distance) ................................. 246, 248

ED (Euclidean Distance, No Accumulate) ............ 250

EDAC (Euclidean Distance) .............................. 252

EXCH (Exchange Wns and Wnd) .......................... 254

FBCL (Find First Bit Change from Left) ............... 255

FF1L (Find First One from Left) ......................... 257

FF1R (Find First One from Right) ....................... 259

FLIM (Force (Signed) Data Range Limit) ............... 261

FLIM.V (Force (Signed) Data Range Limit with Limit Excess Result) ............... 262

GOTO (Unconditional Jump) .............................. 263

GOTO.L (Unconditional Jump Long) ..................... 265

GOTO (Unconditional Indirect Jump) ................. 264, 265

GOTO.L (Unconditional Indirect Jump Long) .......... 266

INC (Increment f) .......................................... 267

INC (Increment Ws) ........................................ 268

INC2 (Increment f by 2) .................................. 269

INC2 (Increment Ws by 2) ................................ 270

IOR (Inclusive OR f and WREG) ......................... 271

IOR (Inclusive OR Ws and Wn) ........................... 272

IOR (Inclusive OR Wb and Short Literal) .............. 273

IOR (Inclusive OR Wb and Ws) .......................... 274

LAC (Load Accumulator) .................................. 276

LAC.D (Load Accumulator Double) ....................... 278

LDHSV (Load Slave Processor Program RAM) ......... 279

LNK (Allocate Stack Frame) ............................ 280, 281

LSR (Logical Shift Right by Short Literal) .......... 282

LSR (Logical Shift Right by Ws) ......................... 283

LSR (Logical Shift Right) ............................... 284

MAC (Multiply and Accumulate) ......................... 288

MAC (Square) .............................................. 290

MAX (Accumulator Force Maximum Data Range Limit) ............... 292

MIN.V (Accumulator Force Minimum Data Range Limit with Limit Excess Result) ............... 293

MIN (Accumulator Force Minimum Data Range Limit) ............... 294

MIN.V (Accumulator Force Minimum Data Range Limit with Limit Excess Result) ............... 295

MINZ (Accumulator Force Minimum Data Range Limit) ............... 296

MINZ.V (Accumulator Force Minimum Data Range Limit with Limit Excess Result) ............... 297

MOV (Move 16-Bit Literal to Wnd) ....................... 304

MOV (Move f to Destination) ............................ 299

MOV (Move f to Wnd) ...................................... 301

MOV (Move Ws to f) ...................................... 302

MOV (Move WREG to f) ................................... 300

MOV (Move Ws to Wd) ..................................... 307

MOV.B (Move 8-Bit Literal to Wnd) ..................... 303

MOV.D (Double-Word Move from Source to Wnd) .... 309

MPY (Multiply Wm by Wn) ................................ 315

MPY (Multiply f by Wn) ................................... 314

MPY (Multiply w by Wn) ................................... 315

MPY (Square to Accumulator) ............................ 317

MUL (Integer Unsigned Multiply f and WREG) ........ 323

MUL (Integer Unsigned Multiply w and WREG) ....... 323

MUL.SS (Integer 16x16-Bit Signed Multiply with Accumulator Destination) ............... 327

MUL.S (Integer 16x16-Bit Signed Multiply with Accumulator Destination) ............... 327

MUL.SU (Integer 16x16-Bit Signed-Unsigned Multiply with Accumulator Destination) ............... 331

© 2005-2018 Microchip Technology Inc.
MUL SU (Integer 16x16-Bit Signed-Unsigned Multiply)..........................329
MUL US (Integer 16x16-Bit Signed-Unsigned Short Literal Multiply with Accumulator Destination)..........................332
MUL SU (Integer 16x16-Bit Signed-Unsigned Short Literal Multiply)..................................................................328
MUL SU (Integer 16x16-Bit Signed-Unsigned Multiply with Accumulator Destination)..........................335
MUL UU (Integer 16x16-Bit Unsigned Multiply with 16-Bit Result)..................................................................341
MUL W SU (Integer 16x16-Bit Signed-Unsigned Multiply with 16-Bit Result)..................................................343
MUL W US (Integer 16x16-Bit Signed-Unsigned Short Literal Multiply with 16-Bit Result)..........................345
MUL UU (Integer 16x16-Bit Unsigned Multiply with 16-Bit Result)..............................................................346
MUL UU (Integer 16x16-Bit Unsigned Short Literal Multiply with 16-Bit Result)............................................348
MUL UU (Integer 16x16-Bit Unsigned Short Literal Multiply with 16-Bit Result)............................................349
NEG (Negate Accumulator)..................................................................................................................353
NEG (Negate f)..................................................................................................................................350
NEG (Negate Ws)..................................................................................................................................351
NOP (No Operation)............................................................................................................................354
NOPR (No Operation)............................................................................................................................355
NORM (Normalize Accumulator)...........................................................................................................356
POP (Pop TOS to f)................................................................................................................................357
POP (Pop TOS to Wd)................................................................................................................................358
POP D (Double Pop TOS to Wnd/Wnd+1).................................................................................................359
POP S (Pop Shadow Registers)..............................................................................................................360
PUSH (Push f to TOS).............................................................................................................................361
PUSH (Push Ws to TOS)..........................................................................................................................362
PUSH D (Double Push Ws/Wns+1 to TOS).................................................................................................364
PUSH S (Push Shadow Registers)..........................................................................................................365
PWSAV (Power-Saving Mode)..................................................................................................................366
RCALL (Computed Relative Call)...........................................................................................................371
RCALL (Relative Call)............................................................................................................................367
REP E (Repeat Next Instruction 'lit14+1' Times)......................................................................................375
REP E (Repeat Next Instruction 'lit15+1' Times)......................................................................................376
REP E (Repeat Next Instruction Wn+1 Times).........................................................................................377
RESET (Reset).........................................................................................................................................379
RET I F E (Return from Interrupt)............................................................................................................380
RETLW (Return with Literal in Wn)......................................................................................................382
RETURN (Return)..................................................................................................................................386
RLC (Rotate Left f through Carry).........................................................................................................388
RLC (Rotate Left Ws through Carry)......................................................................................................389
RLNC (Rotate Left f without Carry).....................................................................................................391
RLWC (Rotate Left Ws without Carry).................................................................................................392
RRC (Rotate Right f through Carry)....................................................................................................394
RRC (Rotate Right Ws through Carry).................................................................................................396
RRNC (Rotate Right f without Carry)..................................................................................................398
RRNC (Rotate Right Ws without Carry)...............................................................................................399
SAC (Store Accumulator)..........................................................................................................................401
SAC D (Store Accumulator Double)......................................................................................................403
SAC R (Store Rounded Accumulator).....................................................................................................404
SE (Sign-Extend Ws)................................................................................................................................406
SE T M (Set f or WREG)..........................................................................................................................408
SET M (SetWs)......................................................................................................................................409
SFT AC (Arithmetic Shift Accumulator by S8fH).......................................................................................410
SFT AC (Arithmetic Shift Accumulator by Wb).......................................................................................411
SL (Shift Left by Short Literal)................................................................................................................416
SL (Shift Left by Wns)............................................................................................................................417
SL (Shift Left f).......................................................................................................................................412
SL (Shift Left Ws)....................................................................................................................................414
SUB (Subtract Accumulators)..................................................................................................................423
SUB (Subtract Literal from Wn)..............................................................................................................419
SUB (Subtract Short Literal from Wb)....................................................................................................420
SUB (Subtract WREG from f)..................................................................................................................418
SUB (Subtract Ws from Wb)....................................................................................................................421
SUBB (Subtract Short Literal from Wb with Borrow).............................................................................426
SUBB (Subtract Wn from Literal with Borrow)......................................................................................424
SUBB (Subtract WREG and Carry Bit from f)..........................................................................................424
SUBB (Subtract Ws from Wb with Borrow)............................................................................................426
SUBBR (Subtract f from WREG with Borrow).......................................................................................430
SUBBR (Subtract Wb from Short Literal with Borrow)............................................................................431
SUBBR (Subtract Wb from Ws with Borrow)...........................................................................................433
SUBBR (Subtract Short f from WREG).....................................................................................................435
SUBR (Subtract Wb from Short Literal)..................................................................................................436
SUBR (Subtract Wb from Ws)..................................................................................................................437
SWAP (Byte or Nibble Swap Wn)............................................................................................................439
TBL R DH (Table Read High)..................................................................................................................440
TBL RL DH (Table Read Low)................................................................................................................442
TBL WTH (Table Write High)................................................................................................................444
TBL W TL (Table Write Low)................................................................................................................446
UL NK (Deallocate Stack Frame)............................................................................................................448
VFSL V (Verify Slave Processor Program RAM)....................................................................................450
XOR (Exclusive OR f and WREG)..........................................................................................................451
XOR (Exclusive OR Literal and Wn)........................................................................................................452
XOR (Exclusive OR WREG and Wn)........................................................................................................453
XOR (Exclusive OR WREG and Short Literal).......................................................................................454
ZE (Zero-Extend Ws).............................................................................................................................456
Instruction Encoding..............................................................................................................................499
Instruction Encoding Field Descriptors Introduction............................................................................96
Instruction Set Overview..........................................................................................................................40
Bit Instructions........................................................................................................................................47
Compare/Skip and Compare/Branch Instructions..................................................................................48
Control Instructions..................................................................................................................................51
DSP Instructions......................................................................................................................................52
Instruction Groups....................................................................................................................................40
Logic Instructions.................................................................................................................................45
Math Instructions.....................................................................................................................................43
Move Instructions....................................................................................................................................42
Program Flow Instructions....................................................................................................................49
Rotate/Shift Instructions..........................................................................................................................46
Shadow/Stack/Context Instructions.......................................................................................................51
Instruction Set Summary Table..............................................................................................................501
Instruction Set Symbols.........................................................................................................................8
(text)......................................................................................................................................................8
[text]........................................................................................................................................................8
{ label }..................................................................................................................................................8
#text.....................................................................................................................................................8
<n;m>...................................................................................................................................................8
Acc...........................................................................................................................................................8
AWB.......................................................................................................................................................8
bit4............................................................................................................................................................8
Expr..........................................................................................................................................................8
f.................................................................................................................................................................8
lit1..............................................................................................................................................................8
lit10............................................................................................................................................................8
Index

P
PIC Microcontroller Compatibility ........................................ 81
PRODH
  PRODL Register Pair ........................................ 81
Program Addressing Modes ........................................... 63
  Methods of Modifying Flow ................................ 63
Program Counter (PC) .................................................. 21
Programmer's Model ..................................................... 14
  Register Descriptions ........................................... 19
  PSVPAG Register ................................................. 21

R
RCOUNT Register ....................................................... 21
Register Direct Addressing ........................................... 55
Register Indirect Addressing ......................................... 56
  and the Instruction Set .......................................... 59
Displays................................................................. 56
Registers
  CORCON (Core Control - dsPIC30F, dsPIC33F) ............ 34
  CORCON (Core Control - dsPIC33E, dsPIC33C) ........... 36
  CORCON (Core Control - PIC24E) ............................ 33
  CORCON (Core Control - PIC24F, PIC24H) ................. 32
  SR (CPU STATUS - dsPIC30F, dsPIC33F) .................. 28
  SR (CPU STATUS - dsPIC33E, dsPIC33C) .................. 30
  SR (CPU STATUS - PIC24H, PIC24F, PIC24E) ............ 27
Revision History ....................................................... 511

S
Scaling Data with the FBCL Instruction ......................... 90
  Scaling Examples .................................................. 91
Shadow Registers ....................................................... 25
  Automatic Usage .................................................. 25
Software Stack Frame Pointer ................................... 20, 74
  Example .................................................................. 75
  Overflow .............................................................. 76
  Underflow ............................................................ 77
Software Stack Pointer ................................................ 72
  Example .................................................................. 73
Software Stack Pointer (SSP) ....................................... 20
Stack Frame Active (SFA) Control ................................ 77
Stack Pointer Limit Register (SPLIM) ........................... 20
STATUS Register ....................................................... 22
  DO Loop Active (DA) Status Bit ................................. 23
  DSP ALU Status Bits ............................................ 23
  Interrupt Priority Level Bits ................................... 24
  MCU ALU Status Bits ........................................... 22
  REPEAT Loop Active (RA) Status Bit ........................ 23
  Style and Symbol Conventions ................................. 7
  Document Conventions ........................................... 7

T
TBLPAG Register ....................................................... 21
to Wnd) ................................................................. 305

U
Using 10-Bit Literal Operands ...................................... 71
  10-Bit Literal Coding ............................................ 71

© 2005-2018 Microchip Technology Inc.
### Instruction Descriptions

#### W
- MOV (Move Wns to) ................................................... 306
- Word Move Operations ............................................. 68
- Data Alignment in Memory ...................................... 68
- Working Register Array ........................................... 19

#### MOV (Move)
- Instruction Descriptions ........................................ 305

#### X
- X Data Space Prefetch Operation ............................... 98

#### Y
- Y Data Space Prefetch Destination .......................... 99
- Y Data Space Prefetch Operation ......................... 99

#### Z
- Z Status Bit ....................................................... 79
## WORLDWIDE SALES AND SERVICE

### AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277

**Technical Support:**
http://www.microchip.com/support

**Web Address:**
www.microchip.com

**Atlanta**
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

**Austin, TX**
Tel: 512-257-3370

**Boston**
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

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

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

**Detroit**
Novi, MI
Tel: 248-848-4000

**Houston, TX**
Tel: 281-894-5983

**Indianapolis**
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

**Raleigh, NC**
Tel: 919-844-7510

**New York, NY**
Tel: 631-435-6000

**San Jose, CA**
Tel: 408-735-9110
Tel: 408-436-4270

**Canada - Toronto**
Tel: 905-695-1980
Fax: 905-695-2078

### ASIA/PACIFIC

**Australia - Sydney**
Tel: 61-2-9888-6733

**China - Beijing**
Tel: 86-10-8569-7000

**China - Chengdu**
Tel: 86-28-8665-5511

**China - Chongqing**
Tel: 86-23-8980-9588

**China - Dongguan**
Tel: 86-769-8702-9880

**China - Guangzhou**
Tel: 86-20-8755-8029

**China - Hangzhou**
Tel: 86-571-8792-8115

**China - Hong Kong SAR**
Tel: 852-2943-5100

**China - Nanjing**
Tel: 86-25-8473-2460

**China - Qingdao**
Tel: 86-532-8502-7355

**China - Shanghai**
Tel: 86-21-3326-8000

**China - Shenyang**
Tel: 86-24-2334-2829

**China - Shenzhen**
Tel: 86-755-8864-2200

**China - Suzhou**
Tel: 86-186-6233-1526

**China - Wuhan**
Tel: 86-27-5802-5300

**China - Xian**
Tel: 86-29-8833-7252

**China - Xiamen**
Tel: 86-592-2386138

**China - Zhuhai**
Tel: 86-756-3210040

### ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-3090-4444

**India - New Delhi**
Tel: 91-11-4160-8631

**India - Pune**
Tel: 91-20-4121-0141

**Japan - Osaka**
Tel: 81-6-6152-7160

**Japan - Tokyo**
Tel: 81-3-6880-3770

**Korea - Daegu**
Tel: 82-53-744-4301

**Korea - Seoul**
Tel: 82-2-554-7200

**Malaysia - Kuala Lumpur**
Tel: 60-3-7651-7906

**Malaysia - Penang**
Tel: 60-4-227-8870

**Philippines - Manila**
Tel: 63-2-634-9065

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

**Taiwan - Hsinchu**
Tel: 886-3-577-8366

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830

**Taiwan - Taipei**
Tel: 886-2-2508-8600

**Thailand - Bangkok**
Tel: 66-2-694-1351

**Vietnam - Ho Chi Minh**
Tel: 84-28-5448-2100

### EUROPE

**Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

**Denmark - Copenhagen**
Tel: 45-4450-2828
Fax: 45-4485-2829

**Finland - Espoo**
Tel: 358-9-4520-820

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

**Germany - Garching**
Tel: 49-8931-9700

**Germany - Haan**
Tel: 49-2129-3766400

**Germany - Heilbronn**
Tel: 49-7131-67-3636

**Germany - Karlsruhe**
Tel: 49-721-625370

**Germany - Munich**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Germany - Rosenheim**
Tel: 49-8031-354-560

**Israel - Ra'anana**
Tel: 972-9-744-7705

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Italy - Padova**
Tel: 39-049-7625286

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**Norway - Trondheim**
Tel: 47-7289-7561

**Poland - Warsaw**
Tel: 48-22-3325737

**Romania - Bucharest**
Tel: 40-21-407-87-50

**Spain - Madrid**
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

**Sweden - Gothenburg**
Tel: 46-31-704-60-40

**Sweden - Stockholm**
Tel: 46-8-5090-4654

**UK - Wokingham**
Tel: 44-118-921-5800
Fax: 44-118-921-5820