Enhanced CPU

HIGHLIGHTS

This section of the manual contains the following topics:

© 2014-2019 Microchip Technology Inc.

1.0 INTRODUCTION

> **Note:** This family reference manual section is meant to serve as a complement to device data sheets. Depending on the device variant, this manual section may not apply to all dsPIC33/PIC24 devices.
>
> Please consult the notes at the beginning of the **"CPU"** chapter in the current device data sheet to check whether this document supports the device you are using.
>
> Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Website at: http://www.microchip.com.

The dsPIC33/PIC24 Enhanced CPU has a 16-bit (data) modified, Harvard architecture with an enhanced instruction set, including significant support for Digital Signal Processing (DSP). The CPU has a 24-bit instruction word with a variable length opcode field. The Program Counter (PC) is 24 bits wide and addresses up to 4M x 24 bits of user program memory space. The data space can be addressed as 32K words or 64 Kbytes and is split into two blocks, as X and Y data memory.

The CPU supports up to six addressing modes. An instruction prefetch mechanism helps maintain throughput and provides predictable execution. Most instructions execute in a single-cycle effective execution rate, with the exception of instructions that change the program flow, such as the double-word Move (`MOV.D`) instruction, Program Space Visibility (PSV) accesses and the table instructions. Overhead-free program loop constructs are supported using the `DO` and `REPEAT` instructions, both of which are interruptible at any point.

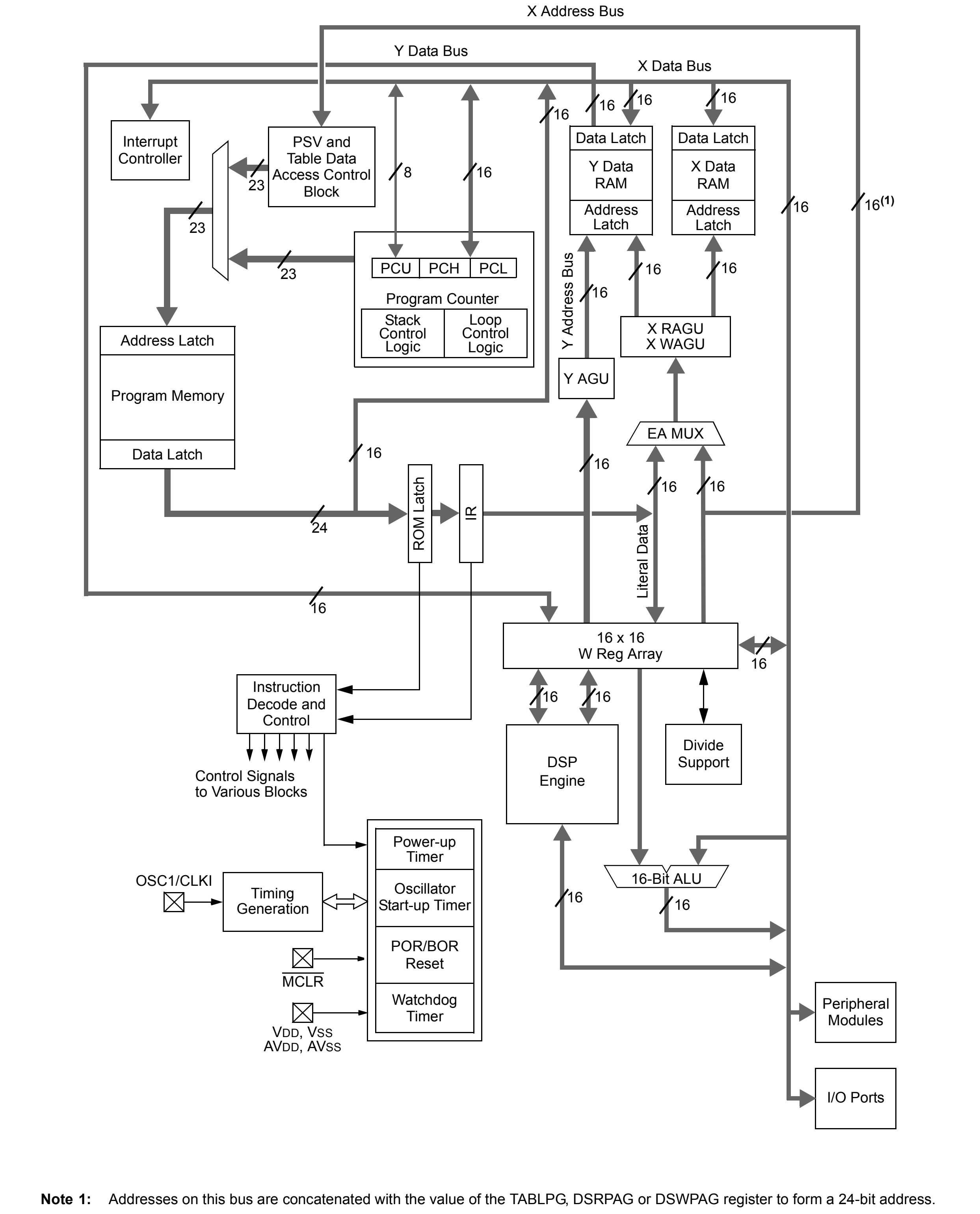
The dsPIC33/PIC24 Enhanced CPU described in this document also includes the following features:

- Alternate W register arrays allowing for hardware enhanced application context switching
- Different levels of CPU interrupt priority and exception handling
- Extensions to the dsPIC33/PIC24 instruction set to support hardware context switching and dual boot program memory arrays

Figure 1-1 illustrates the dsPIC33/PIC24 Enhanced CPU block diagram.

© 2014-2019 Microchip Technology Inc.

Figure 1-1: dsPIC33/PIC24 Enhanced CPU Block Diagram



Note 1: Addresses on this bus are concatenated with the value of the TABLPG, DSRPAG or DSWPAG register to form a 24-bit address.

1.1 Registers

The dsPIC33/PIC24 devices have one main Working (W) register set and up to four Alternate W register sets. All registers are 16 bits wide. The main W register set has sixteen registers (W0-W15) in the programmer's model. Each of the Working registers can act as a data, address or address offset register. W15 operates as a Software Stack Pointer (SSP) for interrupts and calls.

The Alternate W register sets each have fifteen registers (W0-W14) in the programmer's model. These registers are mapped to data memory when they are made active by the CTXTSWP instruction or by CPU interrupt exception processing. The use of the Alternate W registers is discussed in more detail in **Section 4.2 "Alternate Working Register Arrays"**. A summary of the registers associated with the dsPIC33/PIC24 Enhanced CPU is provided in Table 2-1.

© 2014-2019 Microchip Technology Inc.

2.0 REGISTER MAP

Table 2-1: dsPIC33/PIC24 Enhanced CPU Register Map[(1)]

SFR Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
W0	W0 (WREG) Register																0000
W1	W1 Register																0000
W2	W2 Register																0000
W3	W3 Register																0000
W4	W4 Register																0000
W5	W5 Register																0000
W6	W6 Register																0000
W7	W7 Register																0000
W8	W8 Register																0000
W9	W9 Register																0000
W10	W10 Register																0000
W11	W11 Register																0000
W12	W12 Register																0000
W13	W13 Register																0000
W14	W14 Register																0000
W15	W15 (Stack Pointer) Register																0000
SPLIM	Stack Pointer Limit Register																0000
ACCAL	Accumulator A, Low Word (ACCA[15:0]) Register																0000
ACCAH	Accumulator A, High Word (ACCA[31:16]) Register																0000
ACCAU	Sign Extension of ACCA[39] Register								Accumulator A, Upper Byte (ACCA[39:32] Register)								0000
ACCBL	Accumulator B, Low Word (ACCB[15:0]) Register																0000
ACCBH	Accumulator B, High Word (ACCB[31:16]) Register																0000
ACCBU	Sign Extension of ACCB[39] Register								Accumulator B, Upper Byte (ACCB[39:32]) Register								0000
PCL	Program Counter, Low Word (PC[15:0]) Register															0	0000
PCH	—	—	—	—	—	—	—	—	—	Program Counter, High Word (PC[23:16]) Register							0000
DSRPAG	—	—	—	—	—	—	Extended Data Space Read Page Register										0001
DSWPAG	—	—	—	—	—	—	—	Extended Data Space Write Page Register									0001
RCOUNT	REPEAT Loop Counter Register																xxxx
DCOUNT	DO Loop Counter Register																xxxx
DOSTARTL	DO Loop Start Address, Low Word Register															0	xxxx
DOSTARTH	—	—	—	—	—	—	—	—	—	DO Loop Start Address, High Word Register							00xx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: Refer to the device data sheet for specific core register map details.

2: This register is not available on all devices. Refer to the device-specific data sheet for availability.

Table 2-1: dsPIC33/PIC24 Enhanced CPU Register Map[1] (Continued)

SFR Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
DOENDL	DO Loop End Address, Low Word Register															0	xxxx
DOENDH	—	—	—	—	—	—	—	—	—	DO Loop End Address, High Word Register							00xx
SR	OA	OB	SA	SB	OAB	SAB	DA	DC	IPL[2:0]			RA	N	OV	Z	C	0000
CORCON	—	—	US[1:0]		EDT	DL[2:0]			SATA	SATB	SATDW	ACCSAT	IPL3	SFA	RND	IF	0020
MODCON	XMODEN	YMODEN	—	—	BWM[3:0]				YWM[3:0]				XWM[3:0]				0000
XMODSRT	X Modulo Buffer Start Address Register															0	xxxx
XMODEND	X Modulo Buffer End Address Register															1	xxxx
YMODSRT	Y Modulo Buffer Start Address Register															0	xxxx
YMODEND	Y Modulo Buffer End Address Register															1	xxxx
XBREV	BREN	X Modulo Bit Reverse Register															xxxx
DISICNT	—	—	DISI (Disable Interrupts) Counter Register														0000
TBLPAG	—	—	—	—	—	—	—	—	Table Page Address Register								0000
MSTRPR[2]	—	—	—	—	—	—	—	—	—	—	DMAPR	—	USBPR	—	—	NVMPR	0000
CTXTSTAT	—	—	—	—	—	CCTXI[2:0]			—	—	—	—	—	MCTXI[2:0]			0000

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: Refer to the device data sheet for specific core register map details.

2: This register is not available on all devices. Refer to the device-specific data sheet for availability.

2.1 Instruction Set

The dsPIC33/PIC24 instruction set has two classes of instructions: MCU instructions and DSP instructions. These two classes are seamlessly integrated into the architecture and execute from a single execution unit. The instruction set includes many addressing modes and was designed for optimum C compiler efficiency.

2.2 Data Space Addressing

The Base Data Space can be addressed as 32K words or 64 Kbytes and is split into two blocks 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. Certain DSP instructions operate through the X and Y AGUs to support dual operand reads, which splits the data address space into two parts. The X and Y data space boundary is device-specific.

The upper 32 Kbytes of the data space memory map can optionally be mapped into Program Space (PS) at any 16K program word boundary. The program-to-data-space mapping feature, known as Program Space Visibility (PSV), allows any instruction to read Program Space as if it were data space. Moreover, 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. The EDS can be addressed as 8M words or 16 Mbytes. Refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595) for more details on EDS, PSV and table accesses.

In dsPIC33/PIC24 devices, overhead-free circular buffers (Modulo Addressing mode) are supported in both X and Y address spaces. The Modulo Addressing removes the software boundary checking overhead for DSP algorithms. The X AGU Circular Addressing can be used with any of the MCU class of instructions. The X AGU also supports the Bit-Reversed Addressing mode to greatly simplify input or output data reordering for radix-2 FFT algorithms.

2.3 Addressing Modes

The CPU supports up to six addressing modes:

- Inherent (no operand)
- Relative
- Literal
- Memory Direct
- Register Direct
- Register Indirect

Each instruction is associated with a predefined addressing mode group, depending upon its functional requirements. For most instructions, the dsPIC33/PIC24 Enhanced CPU can execute all of the following functions in a single instruction cycle:

- Data memory read
- Working register (data) read
- Data memory write
- Program (instruction) memory read

As a result, three-operand instructions can be supported, allowing A + B = C operations to be executed in a single cycle.

2.4 DSP Engine and Instructions

The DSP engine features:

- A high-speed, 17-bit by 17-bit multiplier
- A 40-bit ALU
- Two 40-bit saturating accumulators
- A 40-bit bidirectional barrel shifter, 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 are designed for optimal real-time performance. The `MAC` instruction, and other associated instructions, can concurrently fetch two data operands from memory while multiplying two W 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 by assigning certain Working registers to each address space.

2.5 Exception Processing

The dsPIC33/PIC24 devices have a vectored exception scheme, with up to eight possible sources of non-maskable traps and up to 246 possible interrupt sources. Each interrupt source can be assigned to one of seven priority levels. The user can select between fixed and variable interrupt latency depending on the application requirements.

In addition, each of the Alternate W register contexts can be associated with its own Interrupt Priority Level (IPL) for exception handling. See **Section 4.2 "Alternate Working Register Arrays"** for more information.

For more information on interrupt latency, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

2.6 Dual Boot Support

The dsPIC33/PIC24 Enhanced CPU supports dual boot implementations of Flash program memory. This includes LiveUpdate, which permits an inactive program partition to be updated while the active partition executes code, and the `BOOTSWP` instruction to permit run-time swapping of active and inactive partitions. These features are explained in the *"dsPIC33/PIC24 Family Reference Manual"*, **"Dual Partition Flash Program Memory"** (DS70005156).

© 2014-2019 Microchip Technology Inc.

3.0 ENHANCED CPU REGISTER DESCRIPTIONS

3.1 SR: CPU STATUS Register

The dsPIC33/PIC24 Enhanced CPU has a 16-bit STATUS Register (SR). A detailed description of the CPU SR is shown in Register 3-1. The LSB of this register is referred to as the SRL (STATUS Register, Low Byte). The MSB is referred to as SRH (STATUS Register, High Byte).

SRL contains:

- All MCU ALU Operation Status flags
- The CPU Interrupt Priority Level Status bits, IPL[2:0]
- The `REPEAT` Loop Active Status bit, RA (SR[4])

During exception processing, SRL is concatenated with the MSB of the PC to form a complete word value, which is then stacked.

SRH contains:

- The DSP Adder/Subtracter Status bits
- The `DO` Loop Active bit, DA (SR[9])
- The Digit Carry bit, DC (SR[8])

The SR bits are readable/writable with the following exceptions:

- The DA bit (SR[9]) is read-only
- The RA bit (SR[4]) is read-only
- The OA, OB (SR[15:14]), OAB (SR[11]), SA, SB (SR[13:12]) and SAB (SR[10]) bits are readable and writable; however, once set, they remain set until cleared by the user application, regardless of the results from any subsequent DSP operations.

Note: Clearing the SAB bit also clears both the SA and SB bits. Similarly, clearing the OAB bit also clears both the OA and OB bits. A description of the STATUS Register bits affected by each instruction is provided in the *“16-Bit MCU and DSC Programmer’s Reference Manual”* (DS70000157).

3.2 CORCON: Core Control Register

The Core Control register (CORCON) has bits that control the operation of the DSP multiplier and `DO` loop hardware. The CORCON register also contains the IPL3 status bit, which is concatenated with IPL[2:0] (SR[7:5]), to form the CPU Interrupt Priority Level.

3.3 CTXTSTAT: Context Status Register

The CPU Context Status register is a read-only register. It contains the Current Context Identifier (CCTXI[2:0]) and Manual Context Identifier (MCTXI[2:0]) status bits.

The CCTXI[2:0] bits field indicates which CPU W register context is currently in use. This field is updated whenever the W register context is changed, either through automatic interrupt-based hardware switching, or as the result of a context change brought about by the execution of a `CTXTSWP` instruction.

The MCTXI[2:0] bits field indicates which CPU W register context was most recently selected as the result of a context change brought about by the execution of the `CTXTSWP` instruction. The MCTXI[2:0] field is used by the `RETFIE` instruction to track manual (software) context switches in the absence of any automatic interrupt-based context switching.

Register 3-1: SR: CPU STATUS Register

R/W-0	R/W-0	R/W-0	R/W-0	R/C-0	R/C-0	R-0	R/W-0
OA[2]	OB[2]	SA[1,2]	SB[1,2]	OAB[2]	SAB[2]	DA[2]	DC
bit 15							bit 8

R/W-0[3,4]	R/W-0[3,4]	R/W-0[3,4]	R-0	R/W-0	R/W-0	R/W-0	R/W-0
IPL2	IPL1	IPL0	RA	N	OV	Z	C
bit 7							bit 0

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[2]
1 = Accumulator A has overflowed
0 = Accumulator A has not overflowed

bit 14 **OB:** Accumulator B Overflow Status bit[2]
1 = Accumulator B has overflowed
0 = Accumulator B has not overflowed

bit 13 **SA:** Accumulator A Saturation 'Sticky' Status bit[1,2]
1 = Accumulator A is saturated or has been saturated at some time
0 = Accumulator A is not saturated

bit 12 **SB:** Accumulator B Saturation 'Sticky' Status bit[1,2]
1 = Accumulator B is saturated or has been saturated at some time
0 = Accumulator B is not saturated

bit 11 **OAB:** OA || OB Combined Accumulator Overflow Status bit[2]
1 = Accumulator A or B has overflowed
0 = Neither Accumulator A or B has overflowed

bit 10 **SAB:** SA || SB Combined Accumulator 'Sticky' Status bit[2]
1 = Accumulator A or B is saturated or has been saturated at some time
0 = Neither Accumulator A or B is saturated

bit 9 **DA:** DO Loop Active bit[2]
1 = DO loop is in progress
0 = DO loop is not in progress

bit 8 **DC:** MCU ALU Half Carry/$\overline{\text{Borrow}}$ 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: A data write to SR can modify the SA and SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA or SB bit write race condition, the SA and SB bits should not be modified using bit operations.

2: These bits are only present on dsPIC33/PIC24 devices. Please refer to the specific device data sheet for availability.

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

4: The IPL[2:0] Status bits are read-only when the NSTDIS bit (INTCON1[15]) = 1.

© 2014-2019 Microchip Technology Inc.

Register 3-1: SR: CPU STATUS Register (Continued)

bit 7-5 **IPL[2:0]:** CPU Interrupt Priority Level Status bits[3,4]

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/$\overline{\text{Borrow}}$ bit

1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note 1: A data write to SR can modify the SA and SB bits by either a data write to SA and SB or by clearing the SAB bit. To avoid a possible SA or SB bit write race condition, the SA and SB bits should not be modified using bit operations.

2: These bits are only present on dsPIC33/PIC24 devices. Please refer to the specific device data sheet for availability.

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

4: The IPL[2:0] Status bits are read-only when the NSTDIS bit (INTCON1[15]) = 1.

Register 3-2: CORCON: Core Control Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
—	—	US1[2]	US0[2]	EDT[1,2]	DL2[2]	DL1[2]	DL0[2]
bit 15							bit 8

R/W-0	R/W-0	R/W-1	R/W-0	R/C-0	R-0	R/W-0	R/W-0
SATA[2]	SATB[2]	SATDW[2]	ACCSAT[2]	IPL3[3]	SFA	RND[2]	IF[2]
bit 7							bit 0

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 15-14 **Unimplemented:** Read as ‘0’

bit 13-12 **US[1:0]:** DSP Multiply Unsigned/Signed Control bits[2]
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[1,2]
1 = Terminate executing DO loop at end of current loop iteration
0 = No effect

bit 10-8 **DL[2:0]:** DO Loop Nesting Level Status bits[2]
111 = 7 DO loops are active
•
•
•
001 = 1 DO loop is active
000 = 0 DO loops are active

bit 7 **SATA:** ACCA Saturation Enable bit[2]
1 = Accumulator A saturation is enabled
0 = Accumulator A saturation is disabled

bit 6 **SATB:** ACCB Saturation Enable bit[2]
1 = Accumulator B saturation is enabled
0 = Accumulator B saturation is disabled

bit 5 **SATDW:** Data Space Write from DSP Engine Saturation Enable bit[2]
1 = Data space write saturation is enabled
0 = Data space write saturation is disabled

bit 4 **ACCSAT:** Accumulator Saturation Mode Select bit[2]
1 = 9.31 saturation (super saturation)
0 = 1.31 saturation (normal saturation)

bit 3 **IPL3:** CPU Interrupt Priority Level Status bit 3[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’.
2: These bits are only present on dsPIC33/PIC24 devices. Please refer to the specific device data sheet for availability.
3: The IPL3 bit is concatenated with the IPL[2:0] bits (SR[7:5]) to form the CPU Interrupt Priority Level.

© 2014-2019 Microchip Technology Inc.

Register 3-2: CORCON: Core Control Register (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[2]
1 = Biased (conventional) rounding is enabled
0 = Unbiased (convergent) rounding is enabled

bit 0 **IF:** Integer or Fractional Multiplier Mode Select bit[2]
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: These bits are only present on dsPIC33/PIC24 devices. Please refer to the specific device data sheet for availability.
3: The IPL3 bit is concatenated with the IPL[2:0] bits (SR[7:5]) to form the CPU Interrupt Priority Level.

Register 3-3: CTXTSTAT: CPU Context Status Register

U-0	U-0	U-0	U-0	U-0	R-0	R-0	R-0
—	—	—	—	—	CCTXI[2:0]		
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	R-0	R-0	R-0
—	—	—	—	—	MCTXI[2:0]		
bit 7							bit 0

Legend:			
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 15-11 **Unimplemented:** Read as '0'

bit 10-8 **CCTXI[2:0]:** Current (W register) Context Identifier bits
111 = Reserved
110 = Reserved
101 = Reserved
100 = Alternate W Register Set 4 is currently in use
011 = Alternate W Register Set 3 is currently in use
010 = Alternate W Register Set 2 is currently in use
001 = Alternate W Register Set 1 is currently in use
000 = Default W register set is currently in use

bit 7-3 **Unimplemented:** Read as '0'

bit 2-0 **MCTXI[2:0]:** Manual (W register) Context Identifier bits
111 = Reserved
110 = Reserved
101 = Reserved
100 = Alternate W Register Set 4 was most recently selected
011 = Alternate W Register Set 3 was most recently selected
010 = Alternate W Register Set 2 was most recently selected
001 = Alternate W Register Set 1 was most recently selected
000 = Default W register set was most recently selected

© 2014-2019 Microchip Technology Inc.

3.4 Other dsPIC33/PIC24 Enhanced CPU Control Registers

The following registers are associated with the dsPIC33/PIC24 Enhanced CPU, but are described in further detail in other Microchip family reference manuals.

- **TBLPAG: Table Page Register**

  The TBLPAG register holds 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 further details, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“dsPIC33/PIC24 Program Memory”** (DS70000613).

- **DSRPAG: Extended Data Space Read Page Register**

  The 10-bit DSRPAG register extends X DS read access address space to a total of 32 Mbytes. DSRPAG page values, between 0x001 and 0x1FF, provide read access for a 16 Mbyte address space, referred to as Extended Data Space (EDS). DSRPAG page values, between 0x200 and 0x2FF, provide read access from the 8 Mbyte PSV address space (for least significant word (lsw) reads only). DSRPAG page values, between 0x300 and 0x3FF, duplicate the 8 Mbyte PSV address space, but allow the user to read the upper Most Significant Byte (MSB) of each PSV address. For further details on the DSRPAG register, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **DSWPAG: Extended Data Space Write Page Register**

  The 9-bit DSWPAG register extends DS write access space to 16 Mbytes (writes to PSV space are not permitted). DSWPAG page values, between 0x01 and 0x1FF, provide write access to EDS. For further details on the DSWPAG register, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **MODCON: Modulo Control Register**

  The MODCON register enables and configures Modulo Addressing (circular buffers). For further details on Modulo Addressing, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **XMODSRT, XMODEND: X Modulo Start and End Address Registers**

  The XMODSRT and XMODEND registers hold the start and end addresses for modulo (circular) buffers implemented in the X data memory address space. For further details on Modulo Addressing, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **YMODSRT, YMODEND: Y Modulo Start and End Address Registers**

  The YMODSRT and YMODEND registers hold the start and end addresses for modulo (circular) buffers implemented in the Y data memory address space. For further details on Modulo Addressing, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **XBREV: X Modulo Bit-Reversed Addressing Register**

  The XBREV register sets the buffer size used for Bit-Reversed Addressing. For further details on Bit-Reversed Addressing, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Data Memory”** (DS70595).

- **DISICNT: Disable Interrupts Count Register**

  The DISICNT register is used by the `DISI` instruction to disable interrupts of Priorities 1-6 for a specified number of cycles. For further information, refer to the *“dsPIC33/PIC24 Family Reference Manual”*, **“Interrupts”** (DS70000600).

© 2014-2019 Microchip Technology Inc.

4.0 PROGRAMMER’S MODEL

The programmer’s model for the dsPIC33/PIC24 Enhanced CPU is shown in Figure 4-1. All registers in the programmer’s model are memory-mapped and can be manipulated directly by instructions. Table 4-1 provides a description of each register in the programmer’s model.

In addition to the registers contained in the programmer’s model, the dsPIC33/PIC24 devices contain control registers for Modulo Addressing, Bit-Reversed Addressing and interrupts. These registers are described in subsequent sections of this document.

All registers associated with the programmer’s model are memory-mapped, as shown in Table 2-1.

Table 4-1: Programmer’s Model Register Descriptions

Register(s) Name	Description
W0 through W15(1)	Working Register Array (Default Context)
W0 through W14(1,2)	Working Register Array (Alternate Context 1)
W0 through W14(1,2)	Working Register Array (Alternate Context 2)
W0 through W14(1,2)	Working Register Array (Alternate Context 3)
W0 through W14(1,2)	Working Register Array (Alternate Context 4)
ACCA, ACCB(4)	40-Bit DSP Accumulators
PC	23-Bit Program Counter
SR(4)	ALU and DSP Engine Status Register
SPLIM	Stack Pointer Limit Value Register
TBLPAG	Table Memory Page Address Register
DSRPAG	Extended Data Space (EDS) Read Page Register
DSWPAG	Extended Data Space (EDS) Write Page Register
RCOUNT	REPEAT Loop Count Register
DCOUNT	DO Loop Count Register
DOSTARTH, DOSTARTL(3)	DO Loop Start Address Register (High and Low)
DOENDH, DOENDL	DO Loop End Address Register (High and Low)
CORCON(4)	DSP Engine and DO Loop Control bits
MSTRPR	EDS Bus Master Priority Control

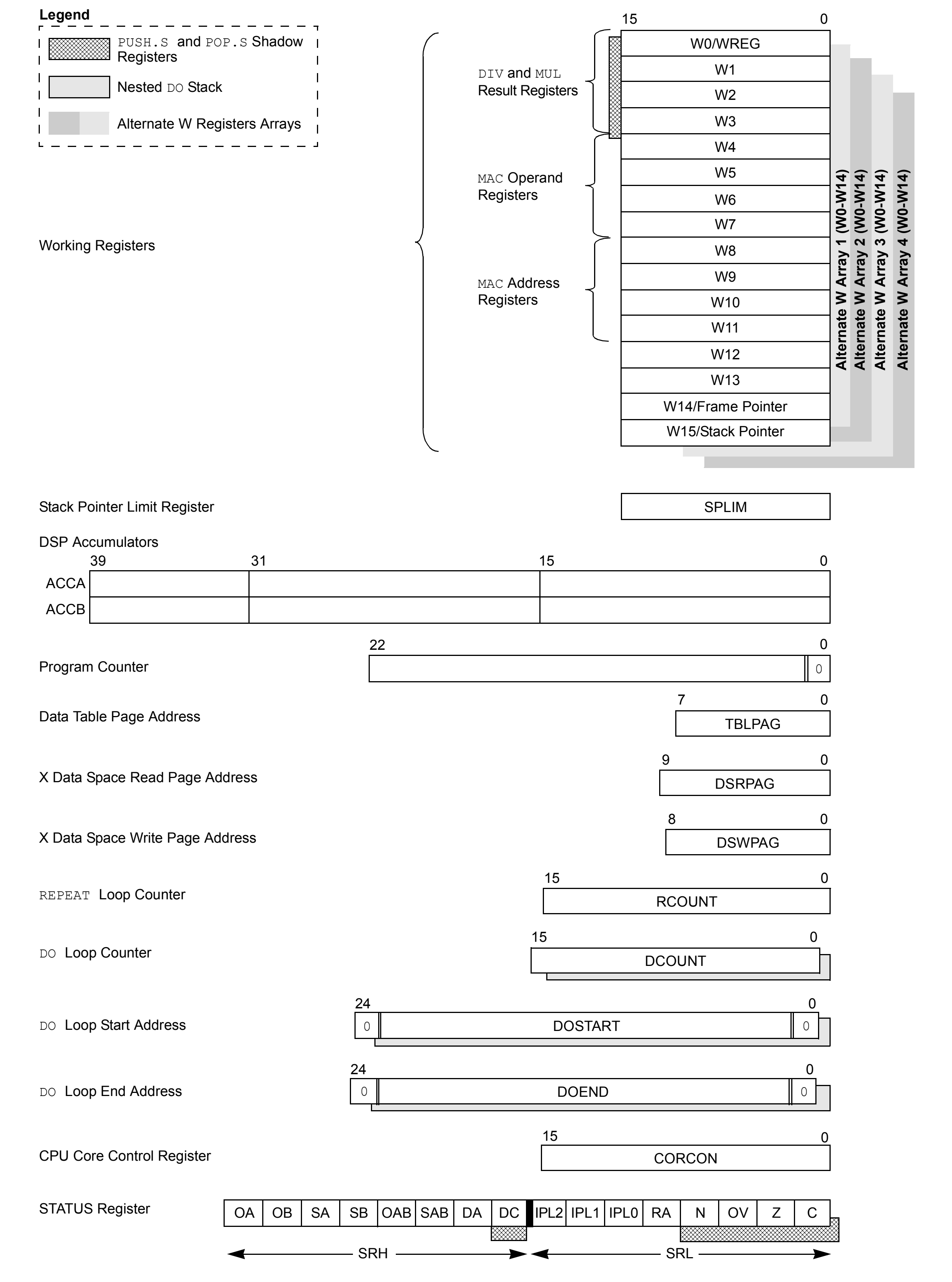
Note 1: W0 through W14 are mapped to memory based on the currently selected context. See text for details.

2: Not all Alternate W register sets are implemented on all devices. Refer to the device data sheet for specific information.

3: Read-only registers.

4: These registers support alternate context switching in select devices. See device-specific data sheet for availability.

Figure 4-1: dsPIC33 Enhanced CPU Programmer's Model



© 2014-2019 Microchip Technology Inc.

4.1 Working Register Array

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

The dsPIC33/PIC24 instruction set can be divided into two instruction types: Register instructions and File register instructions.

4.1.1 REGISTER INSTRUCTIONS

Register instructions can use each W register as a data value or an address offset value. Example 4-1 shows register instructions.

Example 4-1:

```
MOV     W0, W1                      ; move contents of W0 to W1
MOV     W0, [W1]                    ; move W0 to address contained in W1
ADD     W0, [W4], W5                ; add contents of W0 to contents pointed
                                    ; to by W4. Place result in W5.
```

4.1.2 FILE REGISTER INSTRUCTIONS

File register instructions operate on a specific memory address contained in the instruction opcode and register, W0. W0 is a special Working register used in File register instructions. Working registers, W1-W15, cannot be specified as target registers in File register instructions.

The File register instructions provide backward compatibility with existing PIC® MCU devices, which have only one W register. The label, ‘WREG’, is used in the assembler syntax to denote W0 in a File register instruction. Example 4-2 shows File register instructions.

Example 4-2:

```
MOV  WREG, 0x0100      ; move contents of W0 to address 0x0100
ADD  0x0100, WREG      ; add W0 to address 0x0100, store in W0
```

> **Note:** For a complete description of addressing modes and instruction syntax, refer to the “*16-Bit MCU and DSC Programmer’s Reference Manual*” (DS70000157).

4.1.3 W REGISTER MEMORY MAPPING

The W registers are memory-mapped, and thus, it is possible to access a W register in a File register instruction, as shown in the Example 4-3.

Example 4-3:

```
    MOV 0x0004, W10   ; equivalent to MOV W2, W10
```

Where:

```
    0x0004 = memory addresses of W2
```

It is also possible to execute an instruction that uses a W register as both an Address Pointer and operand destination, as shown in Example 4-4. In this example, the contents of W2 are 0x0004. Since W2 is used as an Address Pointer, it points to location, 0x0004, in memory. W2 is also mapped to this address in memory. Even though this is an unlikely event, it is impossible to detect until run time. The CPU ensures that the data write dominates, resulting in W2 = 0x1236.

Example 4-4:

```
   MOV  W1, [W2++]
```

Where:

```
   W1 = 0x1234
   W2 = 0x0004           ;[W2] addresses W2
```

4.1.4 W REGISTERS AND BYTE MODE INSTRUCTIONS

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

4.2 Alternate Working Register Arrays

Alternate Working register arrays are a subset of the Working registers (W0 through W15). Depending on the specific device, up to four Alternate Working register arrays may be implemented. Each set implements registers, W0 through W14, only.

The Alternate W registers are not separately mapped to data memory space in addition to the default W array. Instead, they are mapped to the SFR addresses of W0 through W14 when their particular Alternate W register set is active.

All W register arrays are persistent; that is to say, the contents of the default and Alternate W registers do not change whenever the CPU switches to another set. This saves time by reducing the amount of saving and restoring of register contents, making this very useful for time-critical applications.

Each alternate W array is typically associated with a particular Interrupt Priority Level (IPL) vector and Interrupt Service Routine (ISR) in application code. The IPL for each context is specified by the CTXTn[2:0] bits in the FALTREG Configuration register, where ‘n’ is the Alternate W register array’s number. Each can be assigned an Interrupt Priority Level between 1 and 7.

During an exception processing, the Interrupt Priority Control bits (IPC[3:0]) associated with an interrupt vector request are compared with the CTXTn bits’ field. If there is a match with any of the bit fields, the corresponding Alternate W register set is selected. Additionally, if there is a match, the CCTXI[2:0] bits (CTXTSTAT[10:8]) will be updated by hardware to reflect the most recent register set selected should the match be different from the one currently in use. If there is no match, the CPU continues using the current register set designated by CCTXI[2:0].

Depending on the device, different context Working register behavior can be observed with nested interrupts.

Consider the example, as shown in Figure 4-3, where there are nested interrupts. In this case, the system is configured as follows:

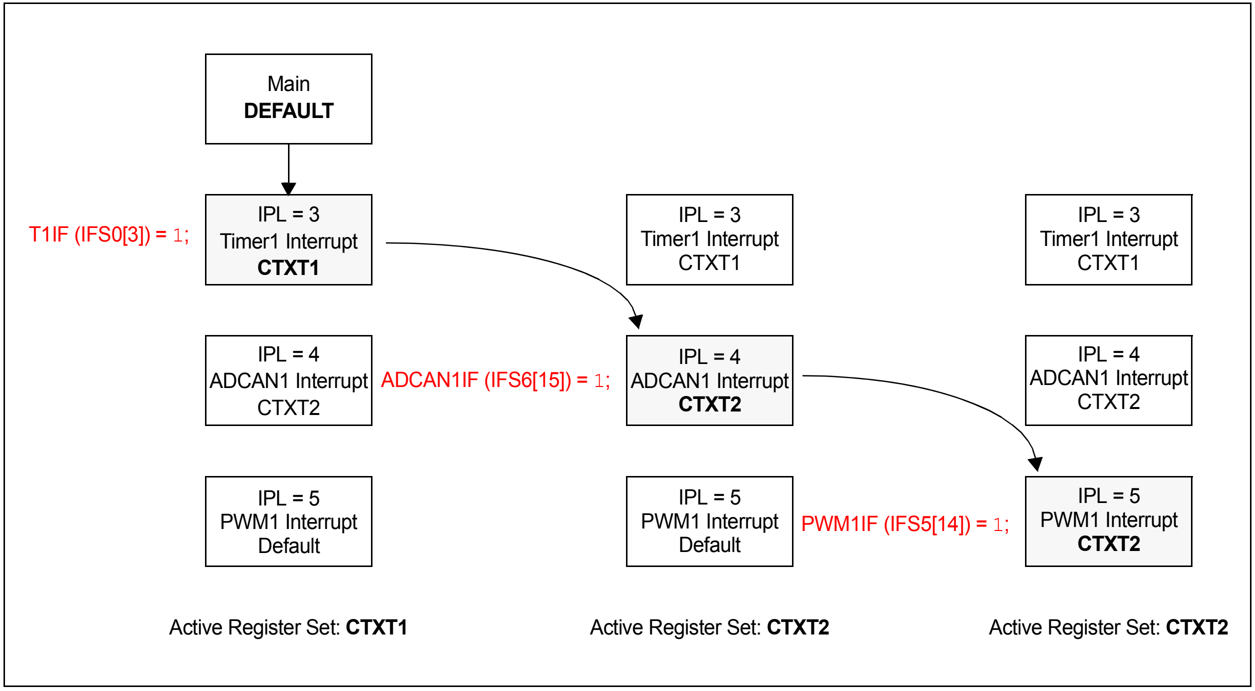
- Timer1 interrupt with an Interrupt Priority Level (IPL) of 3. The Alternate Working Register Set 1 (CTXT1) has an IPL of 3.
- ADCAN1 interrupt with an IPL of 4. The Alternate Working Register Set 2 (CTXT2) has an IPL of 4.
- PWM1 interrupt with an IPL of 5, using the default Working register set.

The application begins in the main function. At some point in time, the Timer1 interrupt flag is set and the program jumps to the Timer1 ISR. The register set switches from the default Working register set to the Alternate Working register set, CTXT1. At some point during the Timer1 ISR, the ADCAN1 conversion completes and its interrupt flag is set. Because it has a higher IPL, the program jumps to the ADCAN1 ISR. The register set switches from the CTXT1 Alternate Working register set to the Alternate Working register set, CTXT2. At some point during the ADCAN1 ISR, the PWM1 interrupt flag is set. Because the PWM1 IPL is higher than the ADCAN1 IPL, the program jumps to the PWM1 ISR and remains in the CTXT2 Working Register set.

Once the PWM ISR execution is completed, the program jumps back to the ADCAN1 ISR using CTXT2. Similarly, after the execution of the ADCAN1 ISR, the program jumps back to the Timer1 ISR using CTXT1.

© 2014-2019 Microchip Technology Inc.

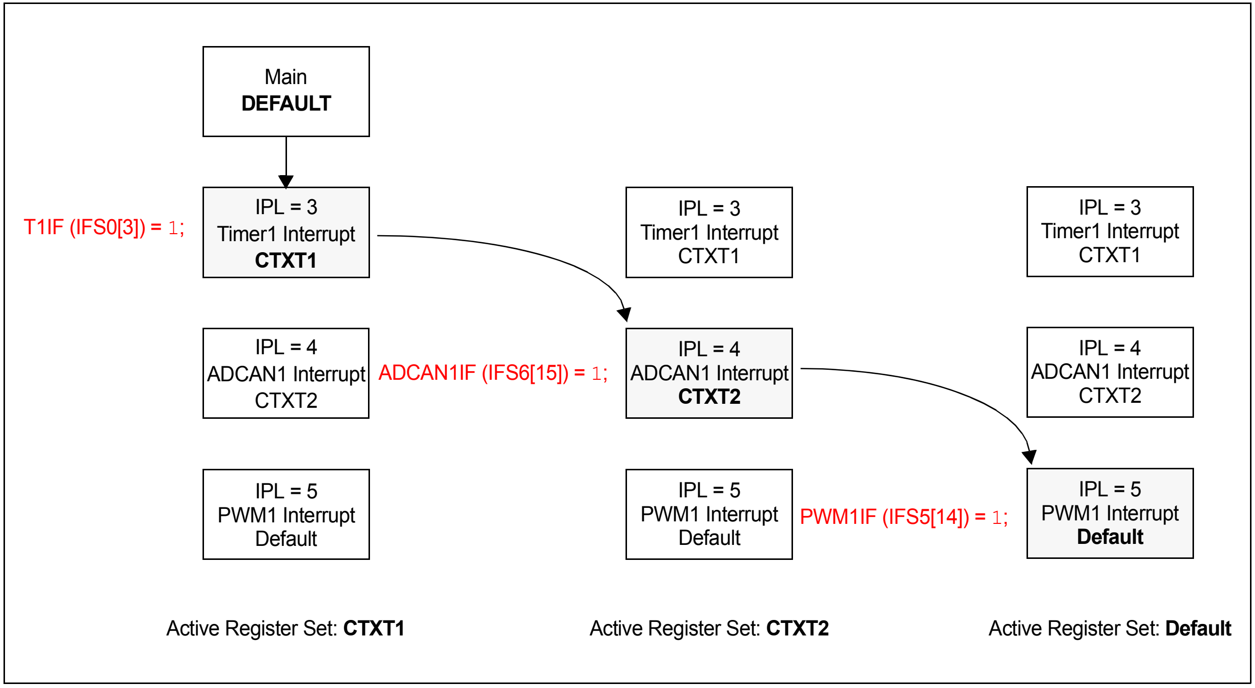
Figure 4-2: Nested Interrupt Context Flow for Devices Other than dsPIC33EPXXXGS70X/80X



4.2.1 dsPIC33EPXXXGS70X/80X DEVICE FAMILIES

Figure 4-2 is identical to Figure 4-3 except for the case when the PWM1 interrupt flag is set. In this case, the PWM1 ISR switches over to the default Working register set instead of using the current active Alternate Working register set, CTXT2. This switch occurs because the PWM1 ISR is not assigned to any of the Alternate Working register sets.

Figure 4-3: Nested Interrupt Context Flow for dsPIC33EPXXXGS70X/80X Devices



For more information on using IPL for context switching in exception handling, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

Alternatively, if an Alternate W register set is not assigned a unique IPL, the application may manually switch to it by executing the `CTXTSWP` instruction. `CTXTSWP` does not affect the CPU IPL; it is used to support software context switching for either context initialization, run-time usage of contexts within procedure calls or the like, thus operating independently from the interrupt system.

4.2.2 ADDITIONAL CPU REGISTER CONTEXTS FOR dsPIC33C DEVICES

dsPIC33C family devices support additional CPU register contexts to further support DSP related operations. In addition to the W registers, ACCA, ACCB, CORCON control bits (US[1:0], SATA, SATB, SATDW, ACCSAT, RND, IF) and SR Status bits (OA, OB, SA, SB, OAB and SAB) support context switching. Like the W registers, each context is persistent and is cleared during Reset.

A consequence of including the DSP CORCON control bits within the register context is that they need to be initialized for each context during application initialization. Writes to CORCON when IPL[3:0] = `0` (default register set) will automatically replicate across all DSP related CORCON bits, in all contexts until the first write to CORCON within the register context, other than default. A Reset will re-enable CORCON write replication.

4.3 Shadow Registers

Many of the registers in the programmer's model have an associated shadow register, as shown in Figure 4-1. None of the shadow registers are accessible directly.

The `PUSH.S` and `POP.S` instructions are useful for fast context save/restore during a function call or Interrupt Service Routine (ISR). The `PUSH.S` instruction transfers the following register values into their respective shadow registers:

- W0 through W3
- SR (N, OV, Z, C, DC bits only)

The `POP.S` instruction restores the values from the shadow registers into these register locations. Example 4-5 shows a code example using the `PUSH.S` and `POP.S` instructions.

Example 4-5:

```
MyFunction:
        PUSH.S                      ; Save W registers, MCU status
        MOV     #0x03, W0           ; load a literal value into W0
        ADD     RAM100              ; add W0 to contents of RAM100
        BTSC    SR, #Z              ; is the result 0?
        BSET    Flags, #IsZero      ; Yes, set a flag
        POP.S                       ; Restore W regs, MCU status
        RETURN
```

The `PUSH.S` instruction overwrites the contents previously saved in the shadow registers. The shadow registers are only one level in depth, so care must be taken if the shadow registers are to be used for multiple software tasks.

The user application must ensure that any task using the shadow registers is not interrupted by a higher priority task that also uses the shadow registers. If the higher priority task is allowed to interrupt the lower priority task, the contents of the shadow registers saved in the lower priority task are overwritten by the higher priority task.

4.4 Uninitialized W Register Reset

The W register array (with the exception of W15) is cleared during all Resets and is considered uninitialized until written to. An attempt to use an uninitialized register as an Address Pointer will reset the device.

A word write must be performed to initialize a W register. A byte write will not affect the initialization detection logic.

© 2014-2019 Microchip Technology Inc.

5.0 SOFTWARE STACK POINTER

The W15 register serves as a dedicated Software Stack Pointer (SSP) and is automatically modified by exception processing, subroutine calls 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 (for example, creating stack frames).

> **Note:** To protect against misaligned stack accesses, W15[0] is fixed to '0' by the hardware.

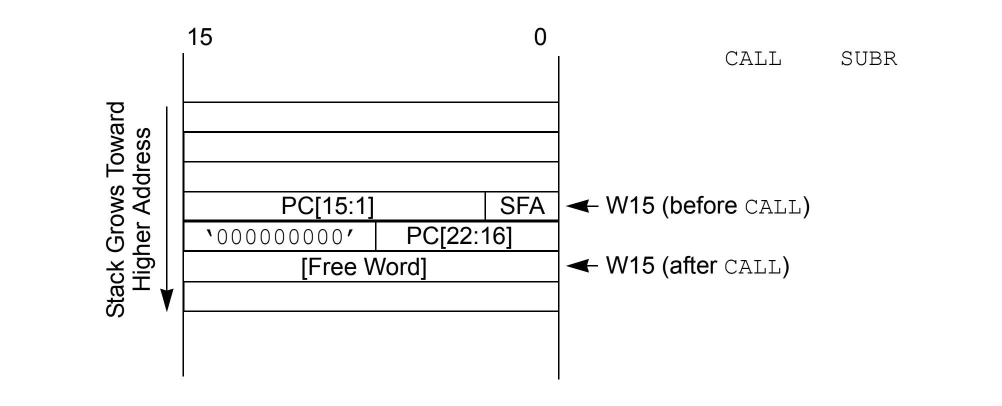
W15 is initialized to 0x1000 during all Resets. This address ensures that the Software Stack Pointer points to valid RAM in all dsPIC33/PIC24 devices and permits stack availability for non-maskable trap exceptions. These can occur before the SSP is initialized by the user software. You can reprogram the SSP during initialization to any location within data space.

The Software Stack Pointer always points to the first available free word in the data space (RAM) and fills the software stack, working from lower toward higher addresses. Figure 5-1 illustrates how it pre-decrements for a stack pop (read) and post-increments for a stack push (writes).

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 into the second available stack location. For a PC push during any CALL instruction, the MSB of the PC is zero-extended before the push, as shown in Figure 5-1. During exception processing, the MSB of the PC is concatenated with the lower eight bits of the CPU STATUS Register, SR. This allows the contents of SRL to be preserved automatically during interrupt processing.

> **Note:** The Stack Pointer, W15, is never subject to paging; therefore, stack addresses are restricted to the Base Data Space (0x0000-0xFFFF).

Figure 5-1: Stack Operation for a CALL Instruction



5.1 Software Stack Examples

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 as the Destination Pointer. For example, the contents of W0 can be pushed onto the stack by:

```
PUSH W0
```

This syntax is equivalent to:

```
MOV W0,[W15++]
```

The contents of the Top-of-Stack (TOS) can be returned to W0 by:

```
POP W0
```

This syntax is equivalent to:

```
MOV [--W15],W0
```

Figure 5-2 through Figure 5-5 illustrate examples of how the software stack is used. Figure 5-2 illustrates the software stack at device initialization. W15 has been initialized to 0x1000. This example assumes the values, 0x5A5A and 0x3636, have been written to W0 and W1, respectively. In Figure 5-3, the stack is pushed for the first time and the value contained in W0 is copied to the stack. W15 is automatically updated to point to the next available stack location (0x1002). In Figure 5-4, the contents of W1 are pushed onto the stack. Figure 5-5 illustrates how the stack is popped and the Top-of-Stack value (previously pushed from W1) is written to W3.

Figure 5-2: Stack Pointer at Device Reset

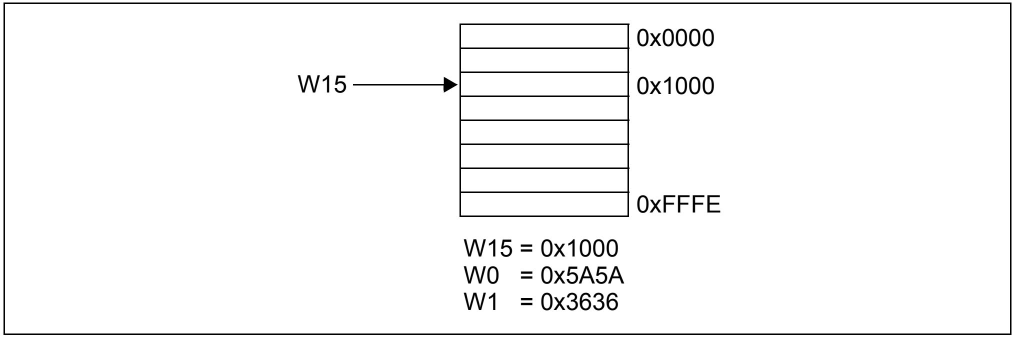


Figure 5-3: Stack Pointer After the First PUSH Instruction

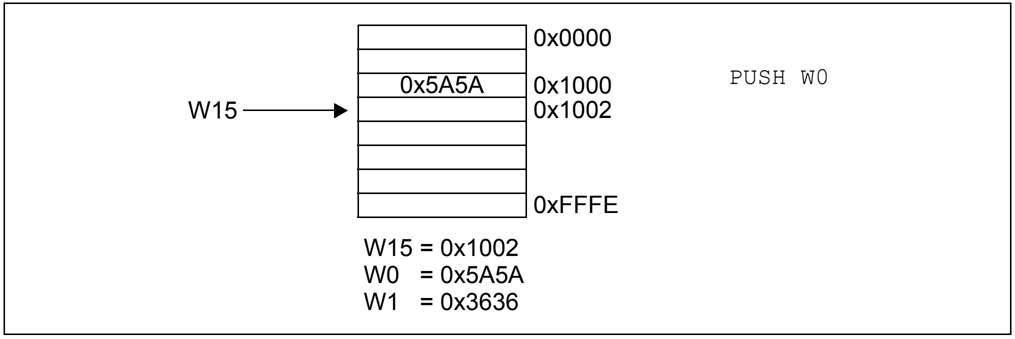


Figure 5-4: Stack Pointer After the Second PUSH Instruction

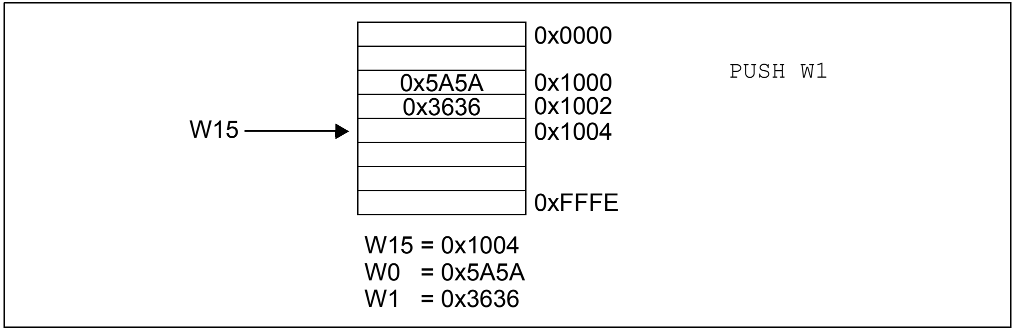
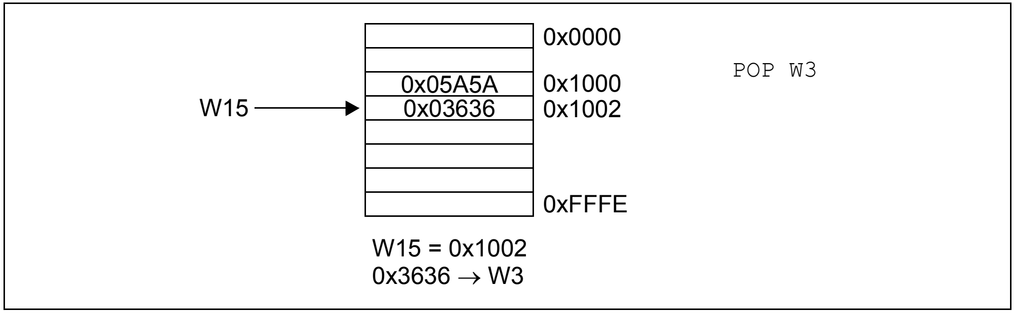


Figure 5-5: Stack Pointer After a POP Instruction



5.2 W14 Software Stack Frame Pointer

A frame is a user-defined section of memory in the stack that is used by a single function. The Working register, W14, can be used as a Stack Frame Pointer with the LNK (link) and ULNK (unlink) instructions. W14 can be used in a normal Working register by instructions when it is not used as a Frame Pointer.

For software examples that use W14 as a Stack Frame Pointer, refer to the "*16-Bit MCU and DSC Programmer's Reference Manual*" (DS70000157).

5.3 Stack Pointer Overflow

The Stack Pointer Limit (SPLIM) register specifies the size of the stack buffer. 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 is not enabled until a word write to SPLIM occurs. After this, it can only be disabled by a device Reset. All Effective Addresses (EAs), generated using W15 as a source or destination, are compared against the value in SPLIM. If the contents of the Stack Pointer (W15) exceed the contents of the SPLIM register by two, and a PUSH operation is performed, a stack error trap occurs on a subsequent PUSH operation. For example, if it is desirable to cause a stack error trap when the stack grows beyond address, 0x2000 in RAM, initialize the SPLIM with the value, 0x1FFE.

Note:	A stack error trap can be caused by any instruction that uses the contents of the W15 register to generate an Effective Address (EA). Therefore, if the contents of W15 are greater than the contents of the SPLIM register by a value of two, and a CALL instruction is executed or if an interrupt occurs, a stack error trap is generated. A stack error trap is also caused by a LNK instruction when the SFA bit is high or on a ULNK instruction when the SFA bit is '0'.

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

Note:	A write to the SPLIM should not be followed by an indirect read operation using W15.

For more information on the stack error trap, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

5.4 Stack Pointer Underflow

The stack is initialized to 0x1000 during a Reset. A stack error trap is initiated if the Stack Pointer address is less than 0x1000.

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

5.5 Stack Frame Active (SFA) Control

W15 is never subject to paging and is therefore, restricted to address range, 0x000000 to 0x00FFFF. However, the Stack Frame Pointer (W14) for any application 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 Stack Frame Active (SFA) status bit (CORCON[2]) achieves this function without additional S/W overhead.

When SFA 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`, `CALLW`, `CALLWL`, `RCALL` and `RCALLW` instructions or vectored interrupts also stack the SFA bit (placing it in the Least Significant bit (LSb) of the stacked PC), and clear the SFA bit after the stacking operation is complete. The called procedure, as well as interrupt vectoring, 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 executing the `LNK` instruction, and cleared by the `ULNK`, `CALL`, `CALLW`, `CALLWL`, `RCALL` and `RCALLW` instructions.

© 2014-2019 Microchip Technology Inc.

6.0 ARITHMETIC LOGIC UNIT (ALU)

The dsPIC33/PIC24 ALU is 16 bits wide and is capable of addition, subtraction, single bit shifts and logic operations. Unless otherwise mentioned, arithmetic operations are 2's complement in nature. Depending on the operation, the ALU can affect the values of the following bits in the STATUS Register:

- Carry (C)
- Zero (Z)
- Negative (N)
- Overflow (OV)
- Digit Carry (DC)

The C and DC Status bits operate as $\overline{\text{Borrow}}$ and $\overline{\text{Digit Borrow}}$ bits, respectively, for subtraction.

The ALU can perform 8-bit or 16-bit operations, depending on the mode of the instruction that is used. Data for the ALU operation can come from the W register array or data memory depending on the addressing mode of the instruction. Likewise, output data from the ALU can be written to the W register array or a data memory location.

For information on the SR bits affected by each instruction, addressing modes and 8-Bit/16-Bit Instruction modes, refer to the *"16-Bit MCU and DSC Programmer's Reference Manual"* (DS70000157).

Note 1: Byte operations use the 16-bit ALU and can produce results in excess of eight bits. However, to maintain backward compatibility with PIC® MCU devices, the ALU result from all byte operations is written back as a byte (i.e., the MSB is not modified) and the STATUS Register is updated based only upon the state of the LSB of the result.

2: All register instructions performed in Byte mode affect only the LSB of the W registers. The MSB of any W register can be modified by using File register instructions that access the memory-mapped contents of the W registers.

6.1 Byte to Word Conversion

The dsPIC33/PIC24 Enhanced CPU has two instructions that are helpful when mixing 8-bit and 16-bit ALU operations.

The Sign-Extend (`SE`) instruction takes a byte value in a W register or data memory and creates a sign-extended word value that is stored in a W register.

The Zero-Extend (`ZE`) instruction clears the 8 MSbs of a word value in a W register or data memory and places the result in a destination W register.

7.0 DSP ENGINE

The DSP engine is a block of hardware that is fed data from the W register array, but contains its own specialized result registers. The DSP engine is driven from the same instruction decoder that directs the MCU ALU. In addition, all operand Extended Addresses (EAs) are generated in the W register array. Concurrent operation with MCU instruction flow is not possible, though both the MCU ALU and DSP engine resources can be shared by all instructions in the instruction set.

The DSP engine consists of the following components:

- High-speed, 17-bit by 17-bit multiplier
- Barrel shifter
- 40-bit adder/subtracter
- Two target Accumulator registers
- Rounding logic with selectable modes
- Saturation logic with selectable modes

Data input to the DSP engine is derived from one of the following sources:

- Directly from the W array (registers: W4, W5, W6 or W7) for dual source operand DSP instructions. Data values for the W4, W5, W6 and W7 registers are pre-fetched via the X and Y memory data buses.
- From the X memory data bus for all other DSP instructions.

Data output from the DSP engine is written to one of the following destinations:

- The target accumulator, as defined by the DSP instruction being executed.
- The X memory data bus to any location in the data memory address space.

The DSP engine can perform inherent accumulator-to-accumulator operations that require no additional data.

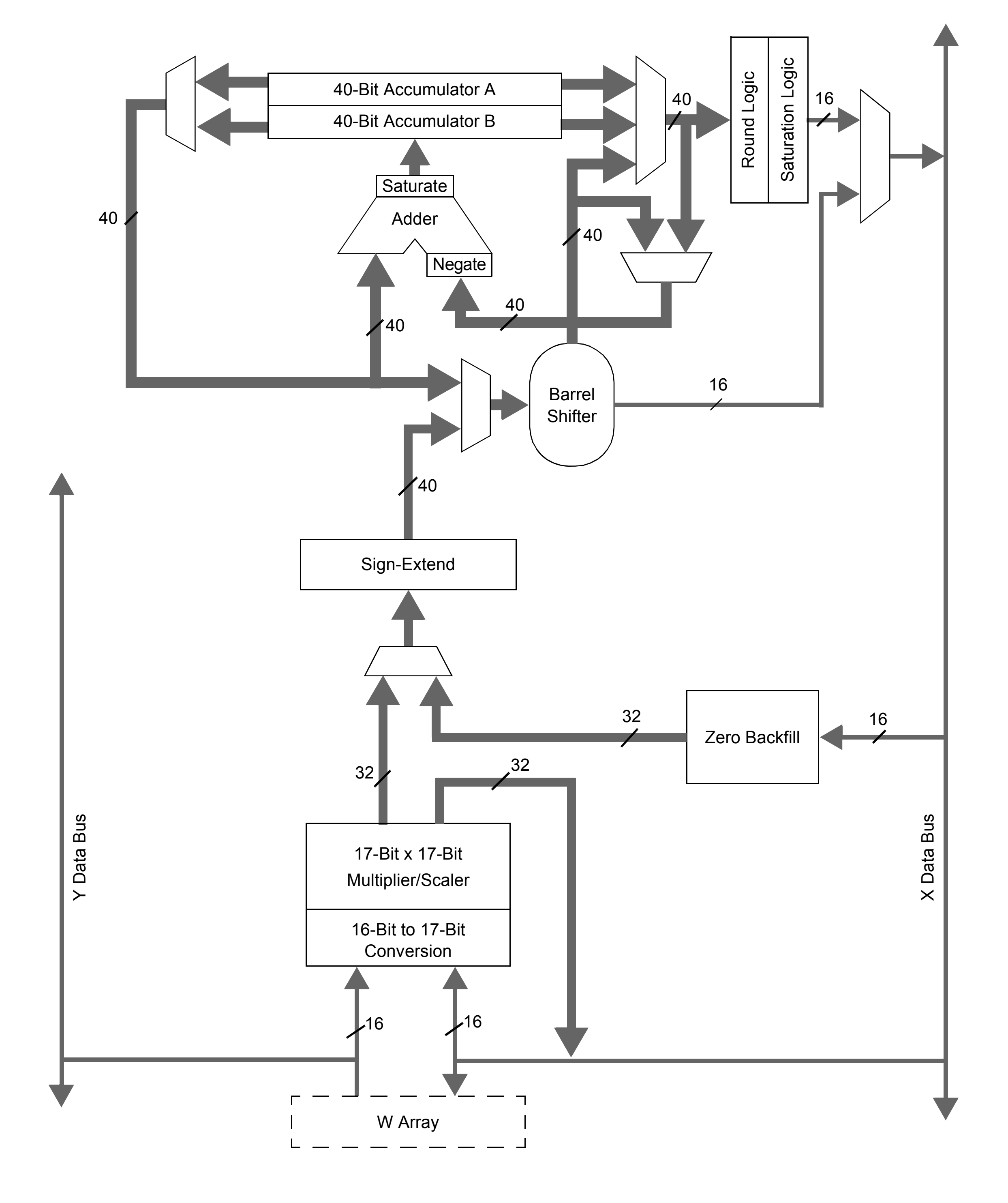
The MCU shift and multiply instructions use the DSP engine hardware to obtain their results. The X memory data bus is used for data reads and writes in these operations.

Figure 7-1 illustrates a block diagram of the DSP engine.

Note:	For detailed code examples and instruction syntax related to this section, refer to the *"16-Bit MCU and DSC Programmer's Reference Manual"* (DS70000157).

© 2014-2019 Microchip Technology Inc.

Figure 7-1: DSP Engine Block Diagram



7.1 Data Accumulators

Two 40-bit data accumulators, ACCA and ACCB, are the Result registers for the DSP instructions listed in Table 7-2. Each accumulator is memory-mapped to these three registers, where ‘x’ denotes the particular accumulator:

- ACCxL: ACCx[15:0]
- ACCxH: ACCx[31:16]
- ACCxU: ACCx[39:32]

For fractional operations that use the accumulators, the radix point is located to the right of bit 31. The range of fractional values that can be stored in each accumulator is -256.0 to (256.0 – 2^{-31}).

For integer operations that use the accumulators, the radix point is located to the right of bit 0. The range of integer values that can be stored in each accumulator is -549,755,813,888 to 549,755,813,887.

7.2 Multiplier

The dsPIC33/PIC24 devices feature a 17-bit-by-17-bit multiplier shared by both the MCU ALU and the DSP engine. The multiplier is capable of signed, unsigned or mixed-sign operation and supports either 1.31 fractional (Q.31), or 32-bit integer results.

The multiplier takes in 16-bit input data and converts the data to 17 bits. Signed operands to the multiplier are sign-extended. Unsigned input operands are zero-extended. The internal 17-bit representation of data in the multiplier allows correct execution of mixed-sign and unsigned 16-bit by 16-bit multiplication operations.

The representation of data in hardware for Integer and Fractional Multiplier modes is as follows:

- Integer data is inherently represented as a signed two’s complement value, where the Most Significant bit (MSb) 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).
- Fractional data is represented as a two’s complement fraction, where the MSb is defined as a Sign bit and the radix point is implied to lie just after the Sign bit (Q.X format). The range of an N-bit two’s complement fraction with this implied radix point is -1.0 to (1 – 2^{1-N}).

The range of data in both Integer and Fractional modes is listed in Table 7-1. Figure 7-2 and Figure 7-3 illustrate how the multiplier hardware interprets data in Integer and Fractional modes.

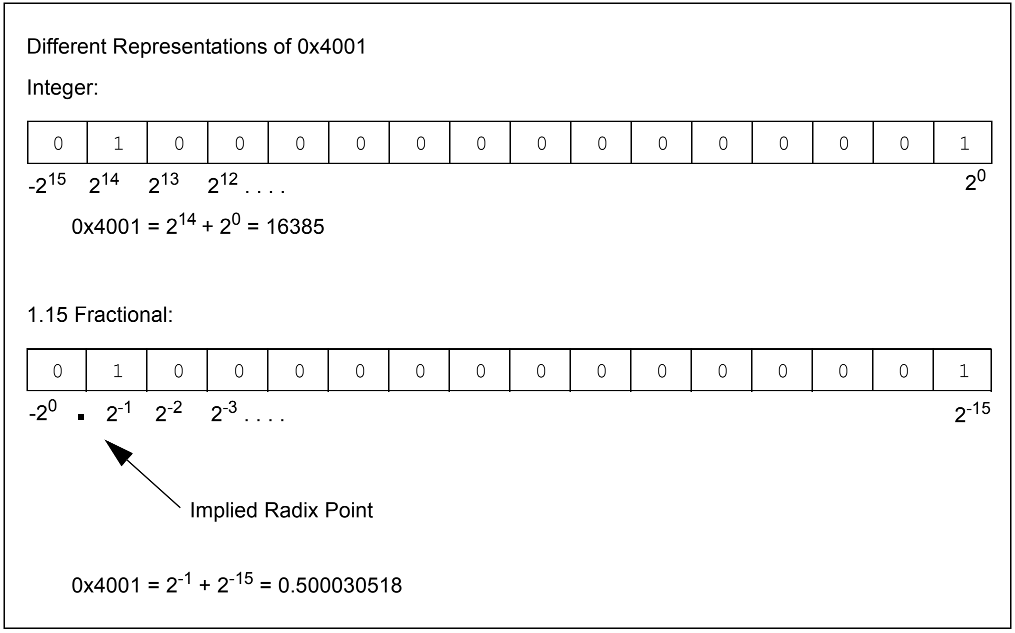
The Integer or Fractional Multiplier Mode Select (IF) bit (CORCON[0]) determines integer/fractional operation for the instructions listed in Table 7-2. The IF bit does not affect MCU multiply instructions listed in Table 7-3, which are always integer operations. The multiplier scales the result, one bit to the left, for fractional operation. The LSb of the result is always cleared. The multiplier defaults to Fractional mode for DSP operations at a device Reset.

Table 7-1: dsPIC33/PIC24 Data Ranges

Register Size	Integer Range	Fraction Range	Fraction Resolution
16-Bit	-32768 to 32767	-1.0 to (1.0 – 2^{-15}) (Q.15 Format)	3.052 x 10^{-5}
32-Bit	-2,147,483,648 to 2,147,483,647	-1.0 to (1.0 – 2^{-31}) (Q.31 Format)	4.657 x 10^{-10}
40-Bit	-549,755,813,888 to 549,755,813,887	-256.0 to (256.0 – 2^{-31}) (Q.31 Format with 8 Guard bits)	4.657 x 10^{-10}

© 2014-2019 Microchip Technology Inc.

Figure 7-2: Integer and Fractional Representation of 0x4001



Different Representations of 0x4001

Integer:

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
-2^{15}	2^{14}	2^{13}	2^{12}												2^{0}

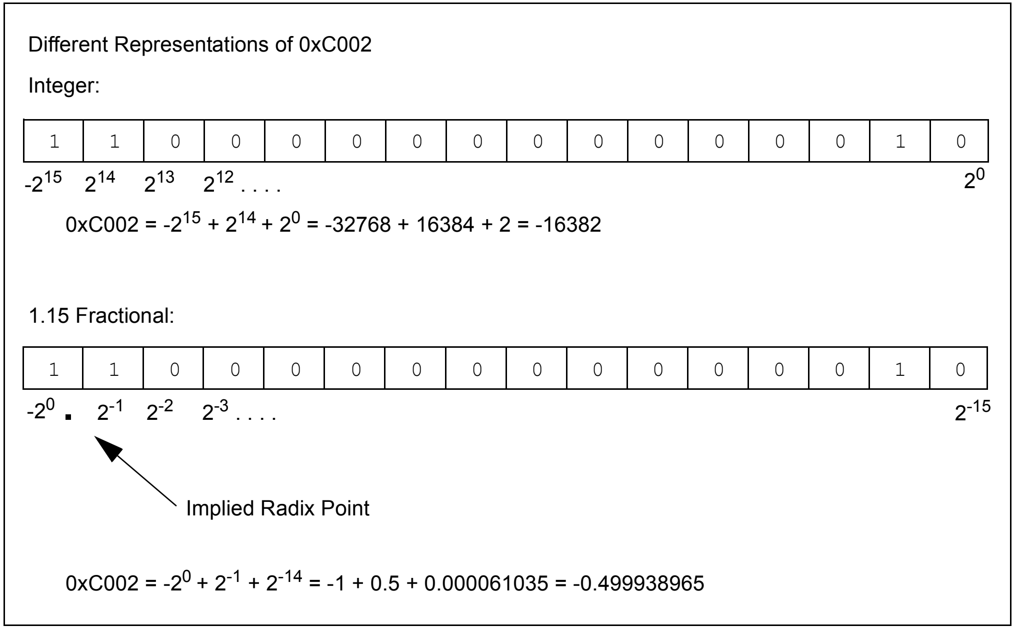
0x4001 = $2^{14} + 2^{0}$ = 16385

1.15 Fractional:

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
-2^{0} ▪	2^{-1}	2^{-2}	2^{-3}												2^{-15}

0x4001 = $2^{-1} + 2^{-15}$ = 0.500030518

Figure 7-3: Integer and Fractional Representation of 0xC002



Different Representations of 0xC002

Integer:

1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
-2^{15}	2^{14}	2^{13}	2^{12}												2^{0}

0xC002 = $-2^{15} + 2^{14} + 2^{0}$ = -32768 + 16384 + 2 = -16382

1.15 Fractional:

1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
-2^{0} ▪	2^{-1}	2^{-2}	2^{-3}												2^{-15}

0xC002 = $-2^{0} + 2^{-1} + 2^{-14}$ = -1 + 0.5 + 0.000061035 = -0.499938965

7.2.1 DSP MULTIPLY INSTRUCTIONS

The DSP instructions that use the multiplier are summarized in Table 7-2.

Table 7-2: DSP Instructions that Use the Multiplier

DSP Instruction[1]	Description	Algebraic Equivalent
`MAC`	Multiply and Add to Accumulator or Square and Add to Accumulator	$a = a + b * c$ $a = a + b^2$
`MSC`	Multiply and Subtract from Accumulator	$a = a - b * c$
`MPY`	Multiply	$a = b * c$
`MPY.N`	Multiply and Negate Result	$a = -b * c$
`ED`	Partial Euclidean Distance	$a = (b - c)^2$
`EDAC`	Add Partial Euclidean Distance to the Accumulator	$a = a + (b - c)^2$

Note 1: DSP instructions using the multiplier can operate in Fractional (1.15) or Integer modes.

The DSP Multiplier Unsigned/Signed Control (US[1:0]) bits (CORCON[13:12]) determine whether DSP multiply instructions are signed (default), unsigned or mixed-sign. The US[1:0] bits do not influence the MCU multiply instructions, which have specific instructions for signed or unsigned operation. If the USx bits are set to ‘`01`’, the input operands for instructions shown in Table 7-2 are considered as unsigned values, which are always zero-extended into the 17th bit of the multiplier value. If the USx bits are set to ‘`00`’, the operands are sign-extended.

If the USx bits (CORCON[13:12]) are set to ‘`10`’, the operands for the instructions listed above are considered as signed or unsigned values, depending upon the W register source. If the W register source is odd (W5 or W7), the operand is assumed to be signed. If the W register source is even, the operand is assumed to be unsigned. The result is sign-extended if one or both of the operands are signed; otherwise, it is zero-extended prior to any operation with the accumulator (which will always effectively be signed).

7.2.2 MCU MULTIPLY INSTRUCTIONS

The same multiplier supports the MCU multiply instructions, which include integer, 16-bit signed, unsigned and mixed-sign multiplies, as shown in Table 7-3. All multiplications performed by the `MUL` instruction produce integer results. The `MUL` instruction can be directed to use byte or word-sized operands. Byte input operands produce a 16-bit result and word input operands produce either a 16-bit result or a 32-bit result, either to the specified register(s) in the W array, or to an accumulator.

Table 7-3: MCU Instructions that Utilize the Multiplier

MCU Instruction[1]	Description
`MUL/MUL.UU`	Multiply two unsigned integers and generate 32-bit results.
`MUL.SS`	Multiply two signed integers and generate 32-bit results.
`MUL.SU/MUL.US`	Multiply a signed integer with an unsigned integer and generate 32-bit results.
`MULW.UU`	Multiply two unsigned integers and generate 16-bit results.
`MULW.SS`	Multiply two signed integers and generate 16-bit results.
`MULW.SU/MULW.US`	Multiply a signed integer with an unsigned integer and generate a 16-bit result.

Note 1: MCU instructions using the multiplier operate only in Integer mode.

© 2014-2019 Microchip Technology Inc.

7.3 Data Accumulator Adder/Subtracter

The data accumulators have a 40-bit adder/subtracter with automatic sign extension logic for the multiplier result (if signed). It can select one of two accumulators (A or B) as its pre-accumulation source and post-accumulation destination. For the `ADD` (accumulator) and `LAC` instructions, the data to be accumulated or loaded can optionally be scaled via the barrel shifter prior to accumulation.

The 40-bit adder/subtracter can optionally negate one of its operand inputs to change the sign of the result (without changing the operands). The negate is used during multiply and subtract (`MSC`) or multiply and negate (`MPY.N`) operations.

The 40-bit adder/subtracter has an additional saturation block that controls accumulator data saturation, if enabled.

7.3.1 ACCUMULATOR STATUS BITS

Six STATUS Register bits that support saturation and overflow are located in the CPU STATUS Register (SR) and are listed in Table 7-4.

Table 7-4: Accumulator Overflow and Saturation Status Bits

Status Bit (SR Location)	Description
OA ([15])	Accumulator A overflowed into guard bits (ACCA[39:32])
OB ([14])	Accumulator B overflowed into guard bits (ACCB[39:32])
SA ([13])	ACCA saturated (bit 31 overflow and saturation) or ACCA overflowed into guard bits and saturated (bit 39 overflow and saturation)
SB ([12])	ACCB saturated (bit 31 overflow and saturation) or ACCB overflowed into guard bits and saturated (bit 39 overflow and saturation)
OAB ([11])	OA logically ORed with OB, clearing OAB clears both OA and OB
SAB ([10])	SA logically ORed with SB, clearing SAB clears both SA and SB

The OA and OB bits are modified each time data passes through the accumulator add/subtract logic. When set, they indicate that the most recent operation has overflowed into the accumulator guard bits (ACCx[39:32]). This type of overflow is not catastrophic; the guard bits preserve the accumulator data. The OAB Status bit is the logically OR value of OA and OB.

The OA and OB bits, when set, can optionally generate an arithmetic error trap. The trap is enabled by setting the corresponding Overflow Trap Flag Enable bit (OVATE or OVBTE) in Interrupt Control Register 1 (INTCON1[10:9]) in the interrupt controller. The trap event allows the user to take immediate corrective action, if desired.

The SA and SB bits can be set each time data passes through the accumulator saturation logic. Once set, these bits remain set until cleared by the user application. The SAB Status bit indicates the logical OR value of SA and SB. When set, these bits indicate that the accumulator has overflowed its maximum range (bit 31 for 32-bit saturation or bit 39 for 40-bit saturation) and are saturated (if saturation is enabled).

When saturation is not enabled, the SA and SB bits indicate that a catastrophic overflow has occurred (the sign of the accumulator has been destroyed). If the Catastrophic Overflow Trap Enable (COVTE) bit (INTCON1[8]) is set, SA and SB bits will generate an arithmetic error trap when saturation is disabled. The SA and SB bits can be set in software, enabling efficient context state switching. For further information on arithmetic warning traps, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

> **Note:** The SA, SB and SAB Status bits can have different meanings depending on whether accumulator saturation is enabled. The Accumulator Saturation mode is controlled via the CORCON register.

7.3.2 SATURATION AND OVERFLOW MODES

The dsPIC33/PIC24 Enhanced CPU supports three Saturation and Overflow modes.

- **Accumulator 39-Bit Saturation**

  In this mode, the saturation logic loads the maximally positive 9.31 value (0x7FFFFFFFFF) or maximally negative 9.31 value (0x8000000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user application. This Saturation mode is useful for extending the dynamic range of the accumulator.

  To configure for this mode of saturation, set the Accumulator Saturation Mode Select (ACCSAT) bit (CORCON[4]). Additionally, set the ACCA Saturation Enable (SATA) bit (CORCON[7] and/or the ACCB Saturation Enable (SATB) bit (CORCON[6]) to enable accumulator saturation.

- **Accumulator 31-Bit Saturation**

  In this mode, the saturation logic loads the maximally positive 1.31 value (0x007FFFFFFF) or maximally negative 1.31 value (0xFF80000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. When this Saturation mode is in effect, the guard bits, 32 through 39, are not used except for sign extension of the accumulator value. Consequently, the OA, OB or OAB bits in SR are never set.

  To configure for this mode of overflow and saturation, the ACCSAT (CORCON[4]) bit must be cleared. Additionally, the SATA (CORCON[7]) and/or SATB (CORCON[6]) bits must be set to enable accumulator saturation.

- **Accumulator Catastrophic Overflow**

  If the SATA (CORCON[7]) and/or SATB (CORCON[6]) bits are not set, then no saturation operation is performed on the accumulator, and the accumulator is allowed to overflow all the way up to bit 39 (destroying its sign). If the Catastrophic Overflow Trap Enable (COVTE) bit (INTCON1[8] in the interrupt controller) is set, a catastrophic overflow initiates an arithmetic error trap.

Accumulator saturation and overflow detection can only result from the execution of a DSP instruction that modifies one of the two accumulators via the 40-bit DSP ALU. Saturation and overflow detection do not take place when the accumulators are accessed as memory-mapped registers via the MCU class of instructions. Furthermore, the Accumulator Status bits shown in Table 7-4 are not modified. However, the MCU Status bits (Z, N, C, OV, DC) will be modified, depending on the MCU instruction that accesses the accumulator. For further information on arithmetic error traps, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

7.3.3 DATA SPACE WRITE SATURATION

In addition to adder/subtracter saturation, writes to data space can be saturated without affecting the contents of the source accumulator. This feature allows data to be limited, while not sacrificing the dynamic range of the accumulator during intermediate calculation stages. Data space write saturation is enabled by setting the data space write from the DSP Engine Saturation Enable (SATDW) control bit (CORCON[5]). Data space write saturation is enabled by default at a device Reset.

The data space write saturation feature works with the `SAC` and `SAC.R` instructions. The value held in the accumulator is never modified when these instructions are executed. The hardware takes the following steps to obtain the saturated write result:

1. The read data is scaled based upon the arithmetic shift value specified in the instruction.
2. The scaled data is rounded (`SAC.R` only).
3. The scaled/rounded value is saturated to a 16-bit result based on the value of the guard bits. For data values greater than 0x007FFF, the data written to memory is saturated to the maximum positive 1.15 value, 0x7FFF. For input data less than 0xFF8000, data written to memory is saturated to the maximum negative 1.15 value, 0x8000.

© 2014-2019 Microchip Technology Inc.

7.3.4 ACCUMULATOR ‘WRITE BACK’

The MAC and MSC instructions can optionally write a rounded version of the accumulator that is not the target of the current operation into data space memory. The write is performed across the X-bus into the combined X and Y address space. This accumulator write-back feature is beneficial in certain algorithms, such as FFT and LMS filters.

Two addressing modes are supported by the accumulator write-back hardware:

- W13, Register Direct: The rounded contents of the non-target accumulator are written into W13 as a 1.15 fractional result.
- [W13]+ = 2, Register Indirect with Post-Increment: The rounded contents of the non-target accumulator are written into the address pointed to by W13 as a 1.15 fraction. W13 is then incremented by 2.

7.4 Round Logic

The round logic can perform a conventional (biased) or convergent (unbiased) round function during an accumulator write (store). The Round mode is determined by the state of the Rounding Mode Select (RND) bit (CORCON[1]). It generates a 16-bit, 1.15 data value, which is passed to the data space write saturation logic. If rounding is not indicated by the instruction, a truncated 1.15 data value is stored.

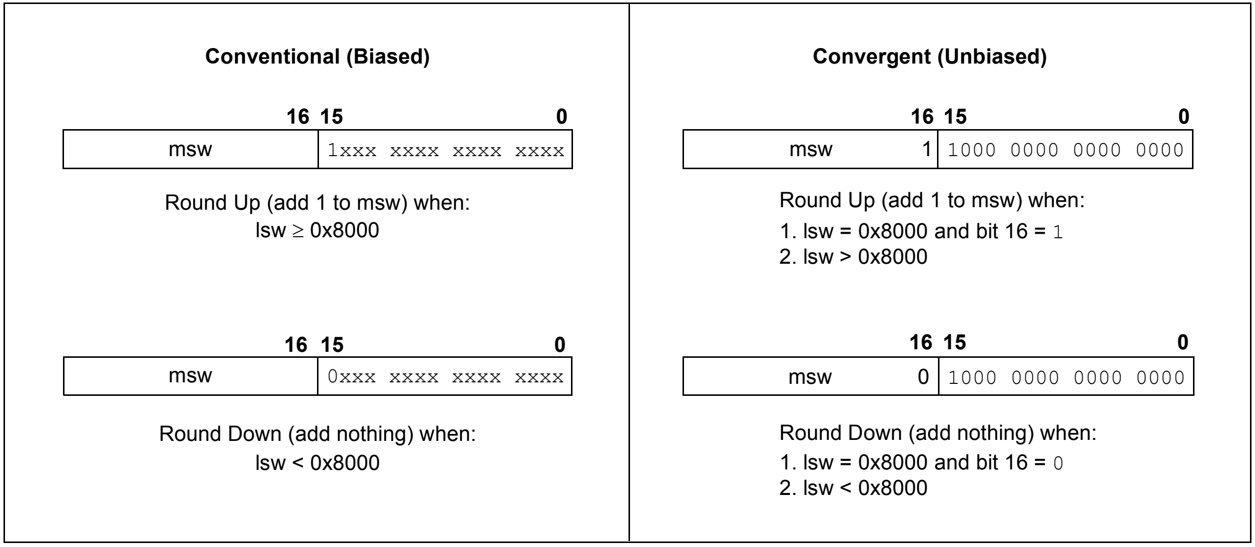
The two Rounding modes are shown in Figure 7-4. Conventional rounding takes bit 15 of the accumulator, zero-extends it and adds it to the most significant word (msw), excluding the guard or overflow bits (bits 16 through 31). If the least significant word (lsw) of the accumulator is between 0x8000 and 0xFFFF (0x8000 included), the msw is incremented. If the lsw of the accumulator is between 0x0000 and 0x7FFF, the msw remains unchanged. A consequence of this algorithm is that over a succession of random rounding operations, the value tends to be biased slightly positive.

Convergent (or unbiased) rounding operates in the same manner as conventional rounding except when the lsw equals 0x8000. If this is the case, the LSb of the msw (bit 16 of the accumulator) is examined. If it is ‘1’, the msw is incremented. If it is ‘0’, the msw is not modified. Assuming that bit 16 is effectively random in nature, this scheme removes any rounding bias that may accumulate.

The SAC and SAC.R instructions store either a truncated (SAC) or rounded (SAC.R) version of the contents of the target accumulator to data memory via the X-bus (subject to data saturation). For more information, refer to **Section 7.3.3 “Data Space Write Saturation”**.

For the MAC class of instructions, the accumulator write-back data path is always subject to rounding.

Figure 7-4: Conventional and Convergent Rounding Modes



7.5 Barrel Shifter

The barrel shifter can perform up to a 16-bit arithmetic right shift, or up to a 16-bit left shift, in a single cycle. DSP or MCU instructions can use the barrel shifter for multibit shifts.

The shifter requires a signed binary value to determine both the magnitude (number of bits) and direction of the shift operation:

- A positive value shifts the operand right
- A negative value shifts the operand left
- A value of '0' does not modify the operand

The barrel shifter is 40 bits wide to accommodate the width of the accumulators. A 40-bit output result is provided for DSP shift operations and a 16-bit result is provided for MCU shift operations.

Table 7-5 provides a summary of instructions that use the barrel shifter.

Table 7-5: Instructions that Use the DSP Engine Barrel Shifter

Instruction	Description
ASR	Arithmetic multibit right shift of data memory location.
LSR	Logical multibit right shift of data memory location.
SL	Multibit shift left of data memory location.
SAC	Store DSP accumulator with optional shift.
SFTAC	Shift DSP accumulator.

7.6 DSP Engine Mode Selection

These operational characteristics of the DSP engine, discussed in previous sections, can be selected through the CPU Core Configuration register (CORCON):

- Fractional or integer multiply operation
- Conventional or convergent rounding
- Automatic saturation on/off for ACCA
- Automatic saturation on/off for ACCB
- Automatic saturation on/off for writes to data memory
- Accumulator Saturation mode selection

7.7 DSP Engine Trap Events

Arithmetic error traps that can be generated for handling exceptions in the DSP engine are selected through the Interrupt Control Register 1 (INTCON1). These are:

- Trap on ACCA overflow enable using OVATE (INTCON1[10])
- Trap on ACCB overflow enable using OVBTE (INTCON1[9])
- Trap on catastrophic ACCA and/or ACCB overflow enable using COVTE (INTCON1[8])

Occurrence of the traps is indicated by these error status bits:

- OVAERR (INTCON1[14])
- OVBERR (INTCON1[13])
- COVAERR (INTCON1[12])
- COVBERR (INTCON1[11])

An arithmetic error trap is also generated when the user application attempts to shift a value beyond the maximum allowable range (±16 bits) using the SFTAC instruction. This trap source cannot be disabled and is indicated by the Shift Accumulator Error Status (SFTACERR) bit (INTCON1[7] in the interrupt controller). The instruction will execute, but the results of the shift are not written to the target accumulator.

For further information on bits in the INTCON1 register and arithmetic error traps, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

© 2014-2019 Microchip Technology Inc.

8.0 DIVIDE SUPPORT

The dsPIC33/PIC24 Enhanced CPU supports the following types of division operations:

- `DIVF`: 16/16 signed fractional divide (dsPIC33E devices only)
- `DIV.SD`: 32/16 signed divide
- `DIV.UD`: 32/16 unsigned divide
- `DIV.SW`: 16/16 signed divide
- `DIV.UW`: 16/16 unsigned divide

The quotient for all divide instructions is placed in Working register, W0. The remainder is placed in W1. The 16-bit divisor can be located in any W register. A 16-bit dividend can be located in any W register and a 32-bit dividend must be located in an adjacent pair of W registers.

All divide instructions are iterative operations and must be executed 18 times within a `REPEAT` loop. The developer is responsible for programming the `REPEAT` instruction. A complete divide operation takes 19 instruction cycles to execute.

The divide flow is interruptible, just like any other `REPEAT` loop. All data is restored into the respective data registers after each iteration of the loop, so the user application is responsible for saving the appropriate W registers in the ISR. Although they are important to the divide hardware, the intermediate values in the W registers have no meaning to the user application. The divide instructions must be executed 18 times in a `REPEAT` loop to produce a meaningful result.

A divide-by-zero error generates a math error trap. This condition is indicated by the Arithmetic Error Status (DIV0ERR) bit (INTCON1[6] in the interrupt controller).

For more information and programming examples for the divide instructions, refer to the "*16-Bit MCU and DSC Programmer's Reference Manual"* (DS70000157).

© 2014-2019 Microchip Technology Inc.

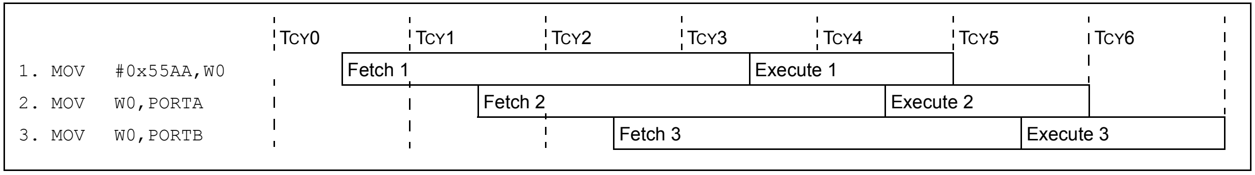
9.0 INSTRUCTION FLOW TYPES

Most instructions in the dsPIC33/PIC24 architecture occupy a single word of program memory and execute in a single cycle. An instruction prefetch mechanism facilitates single-cycle (1 TCY) execution. However, some instructions take two or more instruction cycles to execute. Consequently, there are seven different types of instruction flow in the dsPIC® DSC architecture. These are described in this section.

9.1 One Instruction Word, One Instruction Cycle

These instructions take one instruction cycle to execute, as shown in Figure 9-1. Most instructions are 1-word, 1-cycle instructions.

Figure 9-1: Instruction Flow – 1-Word, 1-Cycle

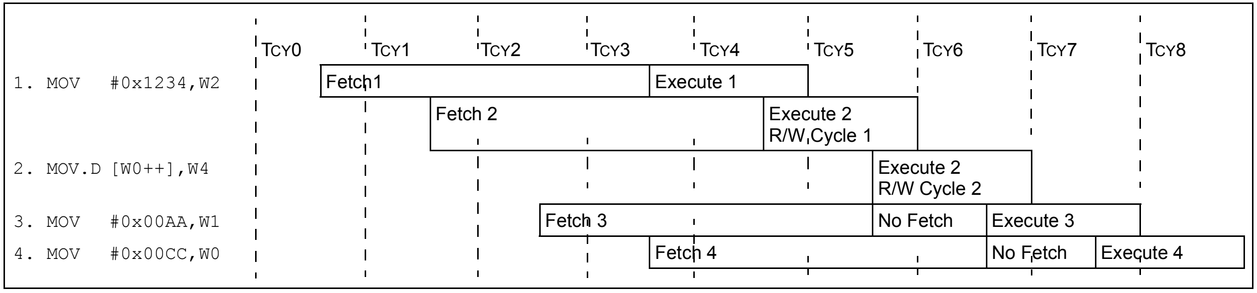


```
1. MOV   #0x55AA,W0
2. MOV   W0,PORTA
3. MOV   W0,PORTB
```

9.2 One Instruction Word, Two Instruction Cycles

In these instructions, there is no prefetch flush. The only instructions of this type are the `MOV.D` instructions (load and store double word), SFR reads and SFR bit operations. Two cycles are required to complete these instructions, as shown in Figure 9-2.

Figure 9-2: Instruction Flow – 1-Word, 2-Cycle (`MOV.D` Operation)

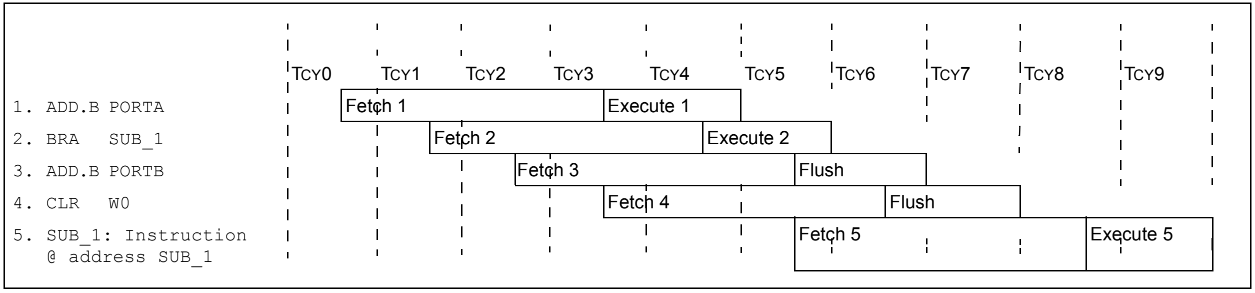


```
1. MOV   #0x1234,W2
2. MOV.D [W0++],W4
3. MOV   #0x00AA,W1
4. MOV   #0x00CC,W0
```

9.3 One Instruction Word, Two or Four Instruction Cycles (Program Flow Changes)

These instructions include relative call and branch instructions. When an instruction changes the PC (other than to increment it), the program memory prefetch data must be discarded. This makes the instruction take four effective cycles to execute, as shown in Figure 9-3.

Figure 9-3: Instruction Flow (Program Flow Changes)



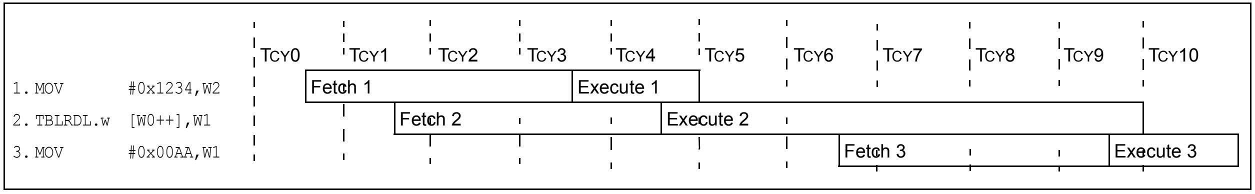
```
1. ADD.B PORTA
2. BRA   SUB_1
3. ADD.B PORTB
4. CLR   W0
5. SUB_1: Instruction
   @ address SUB_1
```

© 2014-2019 Microchip Technology Inc.

9.4 Table Read/Write Instructions

These instructions suspend fetching to insert a read or write cycle to the program memory. Figure 9-4 illustrates the instruction fetched while executing. The table operation is saved for one cycle and executed in the cycle immediately after the table operation.

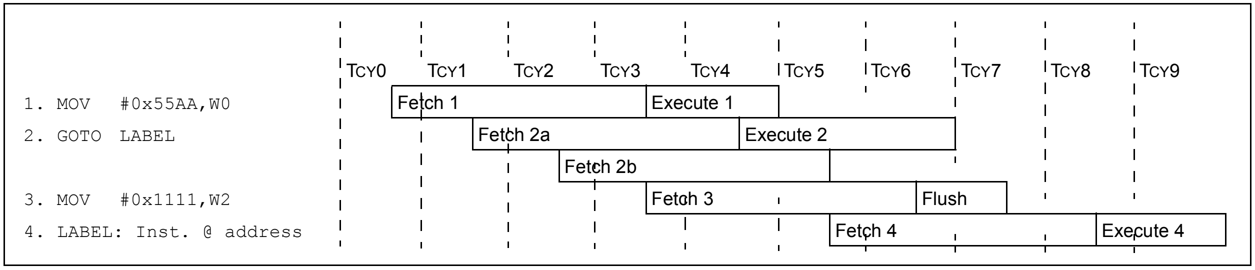
Figure 9-4: Instruction Flow (Table Operations)



9.5 Two Instruction Words, Four Instruction Cycles – `GOTO` or `CALL`

In these instructions, the fetch after the instruction contains data. This results in a 4-cycle instruction, as shown in Figure 9-5. The second word of a two-word instruction is encoded so that it executes as a `NOP` if it is fetched by the CPU when the CPU did not first fetch the first word of the instruction. This is important when a two-word instruction is skipped by a skip instruction (see Figure 9-5).

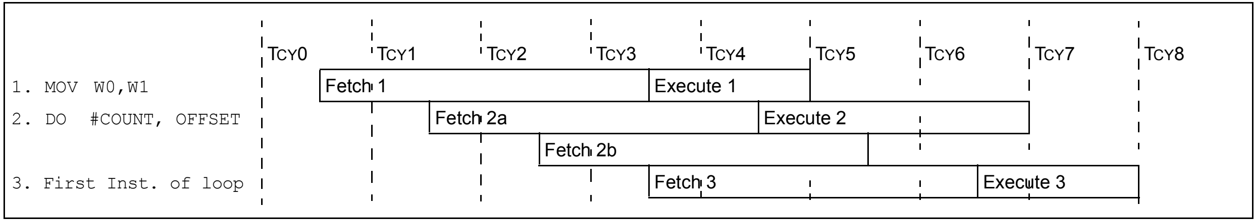
Figure 9-5: Instruction Flow (`GOTO` or `CALL`)



9.6 Two Instruction Words, Two Instruction Cycles – `DO`

In this instruction, the fetch after the instruction contains an address offset. This address offset is added to the first instruction address to generate the last loop instruction address.

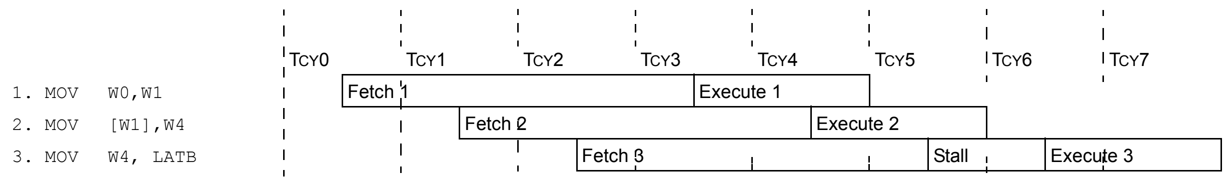
Figure 9-6: Instruction Flow (`DO`)



9.7 Address Register Dependencies

These are instructions subjected to a Stall due to a data address dependency between the X data space read and write operations. An additional cycle is inserted to resolve the resource conflict, as discussed in **Section 11.0 "Address Register Dependencies"**.

Figure 9-7: Instruction Pipeline Flow – 1-Word, 1-Cycle (with Instruction Stall)



9.8 Interrupt Processing

The instruction pipeline flow for interrupt processing is described in detail in the *"dsPIC33/PIC24 Family Reference Manual"*, **"Interrupts"** (DS70000600).

© 2014-2019 Microchip Technology Inc.

10.0 LOOP CONSTRUCTS

The dsPIC33/PIC24 Enhanced CPU supports both REPEAT and DO instruction constructs to provide unconditional automatic program loop control. The REPEAT instruction implements a single instruction program loop. The DO instruction implements a multiple instruction program loop. Both instructions use control bits within the CPU STATUS Register (SR) to temporarily modify CPU operation.

10.1 REPEAT Loop Construct

The REPEAT instruction causes the instruction that follows it to be repeated a specified number of times. A literal value contained in the instruction, or a value in one of the W registers, can be used to specify the REPEAT count value. The W register option enables the loop count to be a software variable.

An instruction in a REPEAT loop is executed at least once. The number of iterations for a REPEAT loop is the 15-bit literal value + 1 or Wn + 1. The syntax for the two forms is shown in Example 10-1.

Example 10-1:

```
;   Using a literal value as a counter
    REPEAT  #lit15              ; RCOUNT <-- lit15
    (Valid target Instruction)
;
;   Using a W register as a counter
    REPEAT  Wn                  ; RCOUNT <-- Wn
    (Valid target Instruction)
```

10.1.1 REPEAT OPERATION

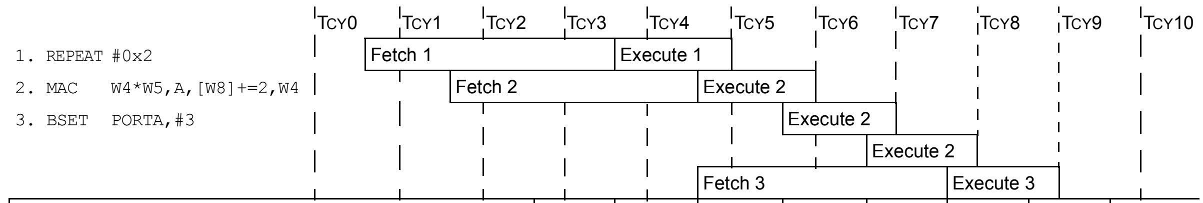
The loop count for REPEAT operations is held in the 16-bit Repeat Loop Counter register (RCOUNT), which is memory-mapped. RCOUNT is initialized by the REPEAT instruction. The REPEAT instruction sets the REPEAT Loop Active (RA) Status bit (SR[4]) to '1' if the RCOUNT is a non-zero value.

RA is a read-only bit and cannot be modified through software. For REPEAT loop count values greater than '0', the Program Counter is not incremented. Furthermore, Program Counter increments are inhibited until RCOUNT = 0. For an instruction flow example of a REPEAT loop, refer to Figure 10-1.

For a loop count value equal to '0', REPEAT has the effect of a NOP and the RA (SR[4]) bit is not set. The REPEAT loop is essentially disabled before it begins, allowing the target instruction to execute only once while pre-fetching the subsequent instruction (i.e., normal execution flow).

> **Note:** The instruction immediately following the REPEAT instruction (i.e., the target instruction) is always executed at least one time and it is always executed one time more than the value specified in the 15-bit literal or the W register operand.

Figure 10-1: REPEAT Instruction Pipeline Flow



PC (at end of instruction)	PC	PC+2	PC+2	PC+2	PC+4	PC+4	PC+4
RCOUNT (at end of instruction)	x	2	1	0	0	0	0
RA (at end of instruction)	0	1	1	0	0	0	0

> **Note:** A consequence of repeating the same instruction is that even when the repeated instruction is performing a PSV read, the first and last iteration incur 5 TCY and 6 TCY, respectively, due to Flash latency. All other iterations execute with an effective throughput of one instruction per cycle. However, this data pipelining is limited to certain addressing modes: post-increment or post-decrement by 1 or 2.

10.1.2 INTERRUPTING A REPEAT LOOP

A REPEAT instruction loop can be interrupted at any time.

The state of the RA bit is preserved on the stack during exception processing to enable the user application to execute further REPEAT loops from within any number of nested interrupts. After SRL is stacked, the RA Status bit is cleared to restore normal execution flow within the (ISR).

> **Note:** If a REPEAT loop has been interrupted, and an ISR is being processed, the user application must stack the Repeat Count register (RCOUNT) before it executes another REPEAT instruction within an ISR.
>
> If a REPEAT instruction is used within an ISR, the user application must unstack the RCOUNT register before it executes the RETFIE instruction.

Returning into a REPEAT loop from an ISR using the RETFIE instruction requires no special handling. Interrupts prefetch the repeated instruction during the fifth cycle of the RETFIE instruction. The stacked RA bit is restored when the SRL register is popped, and if set, the interrupted REPEAT loop is resumed.

10.1.2.1 Early Termination of a REPEAT Loop

An interrupted REPEAT loop can be terminated earlier than normal in the ISR by clearing the RCOUNT register in software.

10.1.3 RESTRICTIONS ON THE REPEAT INSTRUCTION

Any instruction can immediately follow a REPEAT except for the following:

- Program Flow Control instructions (any branch, compare and skip, subroutine calls, returns, etc.)
- Another REPEAT or DO instruction
- DISI, ULNK, LNK, PWRSAV or RESET instruction
- MOV.D instruction

> **Note:** Some instructions and/or Instruction Addressing modes can be executed within a REPEAT loop, but it might not make sense to repeat all instructions.

10.2 DO Loop Construct

The DO instruction can execute a group of instructions that follow it, a specified number of times, without software overhead. The set of instructions, up to and including the end address, is repeated. The repeat count value for the DO instruction can be specified by a 15-bit literal value + 1 or by the contents of a W register + 1 declared within the instruction. Example 10-2 provides examples of both forms.

Example 10-2:

```
; Syntax for the 15-bit literal form of the DO instruction:
       DO      #lit15,LOOP_END       ; DCOUNT <-- lit15
       Instruction1
       Instruction2
           :
   LOOP_END:  Instruction n
;
; Syntax for the W register declared form of the DO instruction:
       DO      Wn,LOOP_END           ; DCOUNT <-- Wn<15:0>
       Instruction1
       Instruction2
           :
   LOOP_END:  Instruction n
```

© 2014-2019 Microchip Technology Inc.

The following features are provided in the DO loop construct:

- The first instruction of a DO loop cannot be a PSV read or a Table Read
- A W register can be used to specify the loop count, which allows the loop count to be defined at run time
- The instruction execution order need not be sequential (i.e., there can be branches, subroutine calls, etc.)
- The loop end address need not be greater than the start address

10.2.1 DO LOOP REGISTERS AND OPERATION

The number of iterations executed by a DO loop will be the 15-bit literal value + 1 or the Wn value + 1. If a W register is used to specify the number of iterations, the two MSbs are not used to specify the loop count. The operation of a DO loop is similar to the DO-WHILE construct in the C programming language because the instructions in the loop will always be executed at least once.

The dsPIC33/PIC24 Enhanced CPU has three registers associated with DO loops:

- The DO Loop Start Address (DOSTART) register holds the starting address of the DO loop; it is a 22-bit register.
- The DO Loop End Address (DOEND) register holds the end address of the DO loop; it is a 22-bit register.
- The DO Loop Counter (DCOUNT) register holds the number of iterations to be executed by the loop; it is a 16-bit register.

These registers are memory-mapped and are automatically loaded by the hardware when the DO instruction is executed. The MSb and LSb of these registers are fixed to '0'. The LSb is not stored in these registers because PC[0] is always forced to '0'.

The DO Loop Active (DA) Status bit (SR[9]) indicates that a single DO loop (or nested DO loops) is active. When a DO instruction is executed, the DA bit is set, which enables the PC address to be compared with the DOEND register on each subsequent instruction cycle. When the PC matches the value in DOEND, DCOUNT is decremented.

If the DCOUNT register is not zero, the PC is loaded with the address contained in the DOSTART register to start another iteration of the DO loop. When DCOUNT reaches zero, the DO loop terminates. If no other nested DO loops are in progress, the DA bit is also cleared. DO loops can be interrupted at any time.

Note: The group of instructions in a DO loop construct is always executed at least one time. The DO loop is always executed one time more than the value specified in the literal or W register operand.

10.2.2 DO LOOP NESTING

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, in addition to the one that is executing.

The DO Loop Nesting Level (DL[2:0]) status bits (CORCON[10:8]) indicate the nesting level of the DO loop currently being executed. When the first DO instruction is executed, the DL[2:0] bits are set to '001' to indicate that one level of the DO loop is in progress. The DO Loop Active (DA) bit (SR[9]) is also set.

When another DO instruction is executed within the first DO loop, the DOSTART, DOEND and DCOUNT registers are transferred into the DO stack before they are updated with the new loop values. The DL[2:0] bits are set to '010' to indicate that a second, nested DO loop is in progress. The DA (SR[9]) bit also remains set. This continues for subsequent nested DO loops.

The DOSTART, DOEND and DCOUNT registers are automatically restored from their DO stack when a DO loop terminates.

Note: The DL[2:0] bits (CORCON[10:8]) are combined (logically ORed) to form the DA (SR[9]) bit. If nested DO loops are being executed, the DA bit is cleared only when the loop count associated with the outermost loop expires.

10.2.3 DO STACK

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
- First loop instruction
- Second loop instruction or second word of first loop instruction if it is a 2-word instruction

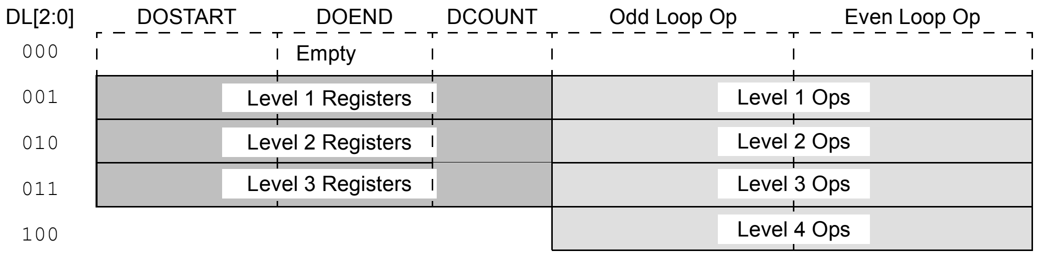
Note that the DO level status field (DL[2:0]) also acts as a pointer to address the DO stack. After the DO is executed, the DL[2:0] bits point to the next free entry.

The initial DO instruction executes without using the stack (actually stacks to a null (nonexistent) entry). Subsequent DO instructions start to fill up the DO stack until three entries are in place. At this point, DL[2:0] = 4 after the final DO instruction has executed, indicating that the initial DO loop, plus three nested DO loops, are executing. No further DO loops may be nested.

If the user attempts to nest an additional DO loop when DL[2:0] = 4 (at the start of the instruction, prior to the DO level increment), the DO Stack Overflow Soft Trap Status bit, DOOVR (INTCON3[4]), will be set and a generic soft trap is generated.

A conceptual representation of the DO stack is shown in Figure 10-2.

Figure 10-2: DO Stack Conceptual Diagram



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

2: For DO instruction buffer entries, the DL[2:0] bits represent the value after the DO stack is executed.

3: If DL[2:0] = 000, no DO loops are active (DA = 0).

10.2.4 EARLY TERMINATION OF THE DO LOOP

There are two ways to terminate a DO loop earlier than normal:

- The Early DO Loop Termination Control (EDT) bit (CORCON[11]) provides a means for the user application to terminate a DO loop before it completes all loops. Writing a '1' to the EDT bit forces the loop to complete the iteration underway and then terminate. If EDT is set during the next to last (penultimate) or last instruction of the loop, one more iteration of the loop occurs. EDT always reads as a '0'; clearing it has no effect. After the EDT bit is set, the user can optionally branch out of the DO loop.
- Alternately, the code can branch out of the loop at any point except from the last two instructions, which cannot be flow control instructions. Although the DA (SR[9]) bit enables the DO loop hardware, it has no effect unless the address of the penultimate instruction is encountered during an instruction prefetch. This is not a recommended method for terminating a DO loop.

> **Note:** Exiting a DO loop without using EDT is not recommended because the hardware will continue to check for DOEND addresses.

© 2014-2019 Microchip Technology Inc.

10.2.5 DO LOOP RESTRICTIONS

The use of DO loops imposes restrictions, such as:

- When the DOEND register can be read
- Certain instructions must not be used as the last two instructions in the loop
- Certain small loop lengths are prohibited (loop length refers to the size of the block of instructions that is being repeated in the loop)

10.2.6 DOEND REGISTER RESTRICTIONS

All DO loops must contain at least two instructions because the loop termination tests are performed in the penultimate instruction. REPEAT should be used for single instruction loops.

The Special Function Register, DOEND, cannot be read by user software in the instruction that immediately follows either a DO instruction or a File register write operation to the DOEND SFR.

The instruction before the penultimate instruction in a DO loop should not modify:

- CPU priority level governed by the CPU Interrupt Priority Level (IPL) status bits (SR[7:5])
- Peripheral Interrupt Enable bits governed by the Interrupt Enable Control registers (IECx)
- Peripheral Interrupt Priority bits governed by the Interrupt Priority Control registers (IPCx)

If these restrictions are not observed, the DO loop may execute incorrectly.

10.2.6.1 Restrictions on First Instruction

A PSV or Table Read cannot be the first instruction in the loop. This restriction is only applicable to devices that have Prefetch Units (PFU) with 3-cycle Flash accesses.

10.2.6.2 Restrictions on Last Two Instructions

The last two instructions in a DO loop should not be any of the following:

- Flow control instruction (e.g., any branch, compare and skip, GOTO, CALL, RCALL, TRAP)
- Another REPEAT or DO instruction
- Target instruction within a REPEAT loop; this restriction implies that the penultimate instruction also cannot be a REPEAT
- Any instruction that occupies two words in Program Space
- DISI instruction

RETURN, RETFIE and RETLW work correctly as one of the last two instructions of a DO loop, but the user application is responsible for returning to the loop to complete it.

10.2.6.3 Loop Length Restrictions

Loop length is defined as the signed offset of the last instruction from the first instruction in the DO loop. The loop length, when added to the address of the first instruction in the loop, forms the address of the last instruction of the loop. For example, a loop length of 1 implies a one-instruction loop. The loop length must not be -1, 0 or 1.

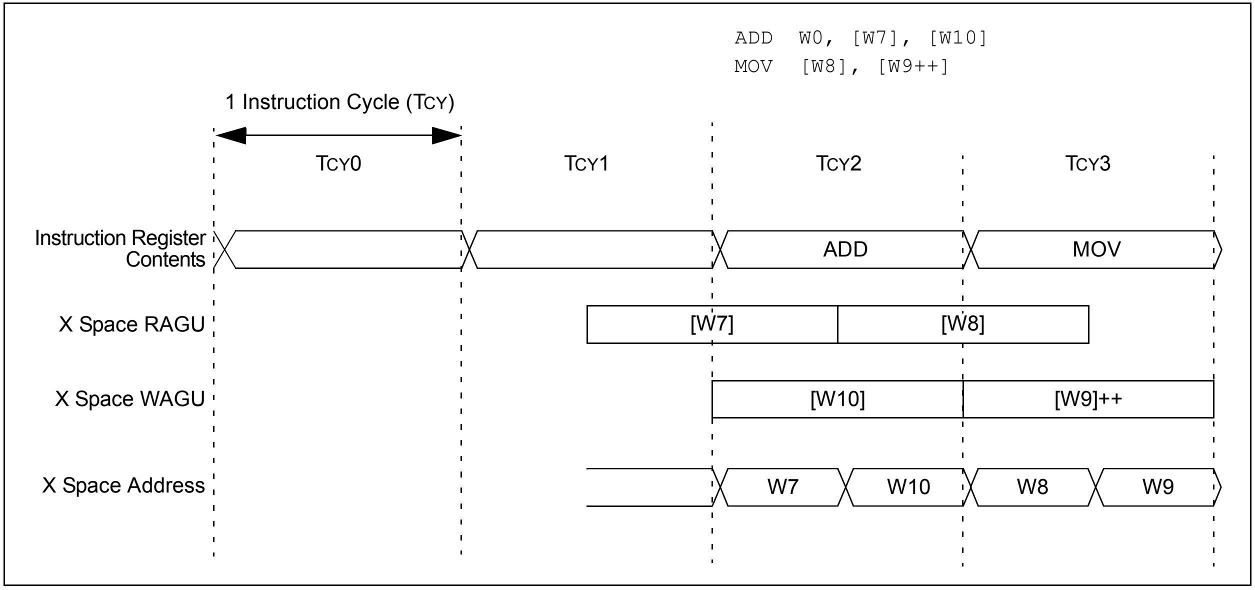
10.2.6.4 DO Loops and Emulation

DO loops cannot be executed outside of user space because DOEND[23] is always assumed to be ‘0’. Therefore, FEX, SSTEP and URUN instructions cannot be included within a DO loop.

11.0 ADDRESS REGISTER DEPENDENCIES

The dsPIC33/PIC24 architecture supports a data space read (source) and a data space write (destination) for most MCU class instructions. The EA calculation by the AGU, and subsequent data space read or write, each take one instruction cycle to complete. This timing causes the data space read and write operations for each instruction to partially overlap, as shown in Figure 11-1. A 'Read-After-Write' (RAW) data dependency can occur across instruction boundaries because of this overlap. RAW data dependencies are detected and handled at run time by the dsPIC33/PIC24 Enhanced CPU.

Figure 11-1: Data Space Access Timing



11.1 Read-After-Write Dependency Rules

If the W register is used as a write operation destination in the current instruction, and the W register being read in the pre-fetched instruction are the same, the following rules apply:

- If the destination write (current instruction) does not modify the contents of Wn, no Stalls will occur.
- If the source read (pre-fetched instruction) does not calculate an EA using Wn, no Stalls will occur.

During each instruction cycle, the dsPIC33/PIC24 hardware automatically checks to see if a RAW data dependency is about to occur. If the conditions specified above are not satisfied, the CPU automatically adds a one-instruction cycle delay before executing the pre-fetched instruction. The instruction Stall provides enough time for the destination W register write to occur before the next (pre-fetched) instruction uses the written data. Table 11-1 provides a summary of Read-After-Write dependency.

© 2014-2019 Microchip Technology Inc.

Table 11-1: Read-After-Write Dependency Summary

Destination Addressing Mode Using Wn	Source Addressing Mode Using Wn	Status	Examples (Wn = W2)
Direct	Direct	Allowed	ADD.w W0, W1, W2 MOV.w W2, W3
Direct	Indirect	Stall	ADD.w W0, W1, W2 MOV.w [W2], W3
Direct	Indirect with Modification	Stall	ADD.w W0, W1, W2 MOV.w [W2++], W3
Indirect	Direct	Allowed	ADD.w W0, W1, [W2] MOV.w W2, W3
Indirect	Indirect	Allowed	ADD.w W0, W1, [W2] MOV.w [W2], W3
Indirect	Indirect with Modification	Allowed	ADD.w W0, W1, [W2] MOV.w [W2++], W3
Indirect with Modification	Direct	Allowed	ADD.w W0, W1, [W2++] MOV.w W2, W3
Indirect	Indirect	Stall	ADD.w W0, W1, [W2] MOV.w [W2], W3 ; W2=0x0004 (mapped W2)
Indirect	Indirect with Modification	Stall	ADD.w W0, W1, [W2] MOV.w [W2++], W3 ; W2=0x0004 (mapped W2)
Indirect with Modification	Indirect	Stall	ADD.w W0, W1, [W2++] MOV.w [W2], W3
Indirect with Modification	Indirect with Modification	Stall	ADD.w W0, W1, [W2++] MOV.w [W2++], W3

11.2 Instruction Stall Cycles

An instruction Stall is essentially a Wait period instruction cycle, added in front of the read phase of an instruction, to allow the prior write to complete before the next read operation. For interrupt latency, the Stall cycle is associated with the instruction following the instruction where it was detected (i.e., Stall cycles always precede instruction execution cycles).

If a RAW data dependency is detected, the dsPIC33/PIC24 Enhanced CPU begins an instruction Stall. During an instruction Stall, the following events occur:

- The write operation in progress (for the previous instruction) is allowed to complete as normal
- Data space is not addressed until after the instruction Stall
- PC increment is inhibited until after the instruction Stall
- Further instruction fetches are inhibited until after the instruction Stall

11.2.1 INSTRUCTION STALL CYCLES AND INTERRUPTS

When an interrupt event coincides with two adjacent instructions that causes an instruction Stall, one of two possible outcomes can occur.

If the interrupt coincides with the first instruction, the first instruction is allowed to complete while the second instruction is executed after the ISR completes. In this case, the Stall cycle is eliminated from the second instruction because the exception process provides time for the first instruction to complete the write phase.

If the interrupt coincides with the second instruction, the second instruction and the appended Stall cycle are allowed to execute before the ISR. In this case, the Stall cycle associated with the second instruction executes normally. However, the Stall cycle is effectively absorbed into the exception process timing. The exception process proceeds as if an ordinary two-cycle instruction was interrupted.

11.2.2 INSTRUCTION STALL CYCLES AND FLOW CHANGE INSTRUCTIONS

The `CALL` and `RCALL` instructions write to the stack using Working register, W15, and can therefore, force an instruction Stall prior to the next instruction if the source read of the next instruction uses W15.

The `RETFIE` and `RETURN` instructions can never force an instruction Stall prior to the next instruction because they only perform read operations. However, the `RETLW` instruction can force a Stall because it writes to a W register during the last cycle.

The `GOTO` and branch instructions can never force an instruction Stall because they do not perform write operations.

11.2.3 INSTRUCTION STALLS AND `DO` AND `REPEAT` LOOPS

Other than the addition of instruction Stall cycles, RAW data dependencies do not affect the operation of either `DO` or `REPEAT` loops.

The pre-fetched instruction within a `REPEAT` loop does not change until the loop is complete or an exception occurs. Although register dependency checks occur across instruction boundaries, the dsPIC33/PIC24 devices effectively compare the source and destination of the same instruction during a `REPEAT` loop.

The last instruction of a `DO` loop either pre-fetches the instruction at the loop start address or the next instruction (outside the loop). The instruction Stall decision is based on the last instruction in the loop and the contents of the pre-fetched instruction.

11.2.4 INSTRUCTION STALLS AND PROGRAM SPACE VISIBILITY (PSV)

When Program Space (PS) is mapped to data space, and the X space EA falls within the visible Program Space window, the read or write cycle redirects to the address in Program Space. In general, any instruction accessing data from Program Space takes five instruction cycles, and therefore, incurs a Stall to ensure that the data is available.

Instructions operating in PSV address space are subject to RAW data dependencies and consequent instruction Stalls, just like any other instruction. In Example 11-1, the sequence of instructions would take seven instruction cycles to execute. The PSV access via W1 requires five instruction cycles, while an additional cycle is inserted to resolve the RAW data dependency caused by W2.

Example 11-1:

```
    ADD     W0,[W1],[W2++]      ; W1=0x8000, PSVPAG=0xAA, DSRPAG=0x0200
    MOV     [W2],[W3]
```

11.3 Data Space Arbiter Stalls

A CPU Stall can also be a result of competition for Extended Data Space resources. When the data space arbiter logic determines that the CPU cycle must be stalled to allow another bus master (e.g., DMA controller or USB module) access to data memory, instruction execution is suspended until the higher priority bus master completes the data access.

12.0 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC33/PIC24 device families, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the dsPIC33/PIC24 Enhanced CPU are:

Title	**Application Note #**
No related application notes at this time.	

Note: Please visit the Microchip website (www.microchip.com) for additional Application Notes and code examples for the dsPIC33/PIC24 families of devices.

13.0 REVISION HISTORY

Revision A (March 2014)

This is the initial release of this document.

Revision B (August 2016)

Added additional content to **Section 4.2 “Alternate Working Register Arrays”**.

Revision C (January 2019)

Updated document to include all dsPIC33 devices.

Moved **Table 2-1: “dsPIC33/PIC24 Enhanced CPU Register Map[(1)]”** to **Section 1.1 “Registers”**.

Updated Table 4-1.

Updated **Section 1.1 “Registers”** and **Section 2.2 “Data Space Addressing”**.

Added **Section 4.2.2 “Additional CPU Register Contexts for dsPIC33C Devices”**

Minor grammatical corrections throughout the document.

© 2014-2019 Microchip Technology Inc.

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

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

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

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

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 ═

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

© 2019, Microchip Technology Incorporated, All Rights Reserved.

ISBN: 978-1-5224-4102-1

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: 61-2-9868-6733

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: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

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 - Hsin Chu
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: 47-7288-4388

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 - Gothenberg
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

© 2014-2019 Microchip Technology Inc.

08/15/18