Interrupts

HIGHLIGHTS

This section of the manual contains the following major topics:

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 note at the beginning of the **"Interrupt Controller"** 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 Web site at: http://www.microchip.com

The dsPIC33/PIC24 Interrupt Controller module reduces the numerous peripheral interrupt request signals to a single interrupt request signal to the dsPIC33/PIC24 CPU. This module includes the following major features:

- Up to eight processor exceptions and software traps
- Seven user-selectable priority levels
- Interrupt Vector Table (IVT) with up to 246 vectors
- Alternate Interrupt Vector Table (AIVT) with up to 246 vectors
- A unique vector for each interrupt or exception source
- Fixed priority within a specified user priority level
- Fixed interrupt entry and return latencies

1.1 Interrupt Vector Table (IVT)

The IVT, as shown in Figure 1-1, resides in program memory. The IVT contains 254 vectors, consisting of up to eight non-maskable trap vectors and up to 246 interrupt sources. In general, each interrupt source has its own vector. Each interrupt vector contains a 24-bit wide address. The value programmed into each interrupt vector location is the starting address of the associated Interrupt Service Routine (ISR).

On devices with Auxiliary Flash, a single Auxiliary Interrupt Vector exists in the Auxiliary Flash memory. This interrupt vector contains a 24-bit wide starting address of the ISR that will be executed when a trap, or any enabled interrupt, occurs while program is being executed from the Auxiliary Flash. Refer to the **"Memory Organization"** and **"Interrupt Controller"** chapters in the specific device data sheet for availability and location of the Auxiliary Interrupt Vector.

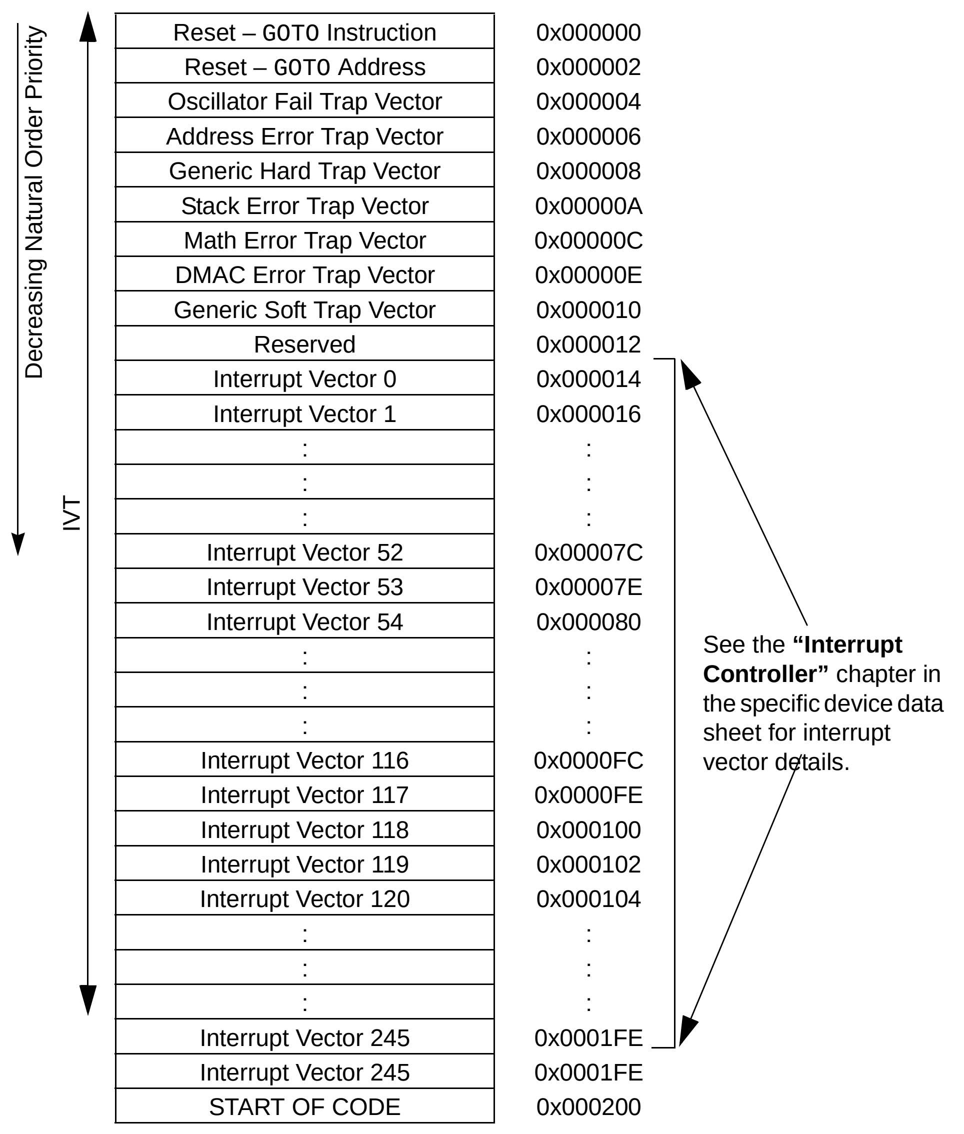
1.2 Alternate Interrupt Vector Table (AIVT)

Some devices feature an Alternate Interrupt Vector Table (AIVT) which can support emulation and debugging by providing a means to switch between an application, and a support environment without reprogramming the interrupt vector. This feature also enables switching between applications for evaluation of different software algorithms at run time.

The AIVT is available only if the Boot Segment has been defined and the AIVT has been enabled. To enable the AIVT, both the Configuration bit (AIVTDIS) in the FSEC<15> Configuration register, and the AIVTEN bit (INTCON2<8>) in the Special Function Register, have to be set. When the AIVT is enabled, all interrupts and exception processes use the alternate vectors instead of the default vectors. The AIVT begins at the start of the first half of the last page of the Boot Segment memory section, defined by the BSLIM<12:0> bits. The second half of the page is no longer usable space. The Boot Segment must be at least 2 pages to enable the AIVT. Note that the units of the BSLIM<12:0> bits are in pages.

For example, if the user decides to create three pages of Boot Segment memory, the application software would then set BSLIM<12:0> = 0x1FFC. Note that the BSLIM<12:0> bits in the unprogrammed state are all '1's. Because three pages are required, the first four Least Significant Bits (LSbs) become '0b1100' or 0xC. The nine Most Significant bits (MSbs) remain unchanged in their unprogrammed state, thus having BSLIM<12:0> = 0x1FFC. See Figure 1-3 for more information.

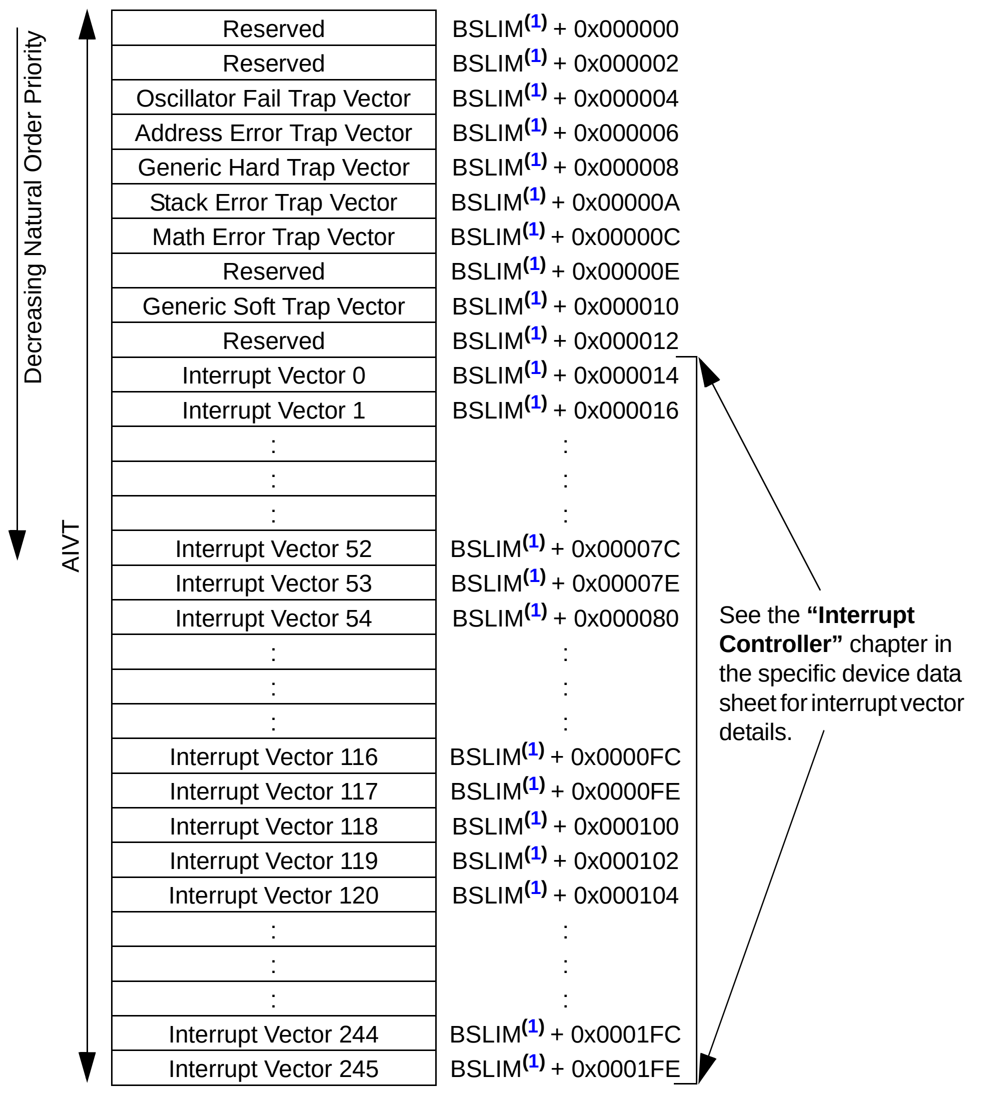
Figure 1-1: Interrupt Vector Table



Reset – `GOTO` Instruction	0x000000
Reset – `GOTO` Address	0x000002
Oscillator Fail Trap Vector	0x000004
Address Error Trap Vector	0x000006
Generic Hard Trap Vector	0x000008
Stack Error Trap Vector	0x00000A
Math Error Trap Vector	0x00000C
DMAC Error Trap Vector	0x00000E
Generic Soft Trap Vector	0x000010
Reserved	0x000012
Interrupt Vector 0	0x000014
Interrupt Vector 1	0x000016
:	:
:	:
:	:
Interrupt Vector 52	0x00007C
Interrupt Vector 53	0x00007E
Interrupt Vector 54	0x000080
:	:
:	:
:	:
Interrupt Vector 116	0x0000FC
Interrupt Vector 117	0x0000FE
Interrupt Vector 118	0x000100
Interrupt Vector 119	0x000102
Interrupt Vector 120	0x000104
:	:
:	:
:	:
Interrupt Vector 245	0x0001FE
Interrupt Vector 245	0x0001FE
START OF CODE	0x000200

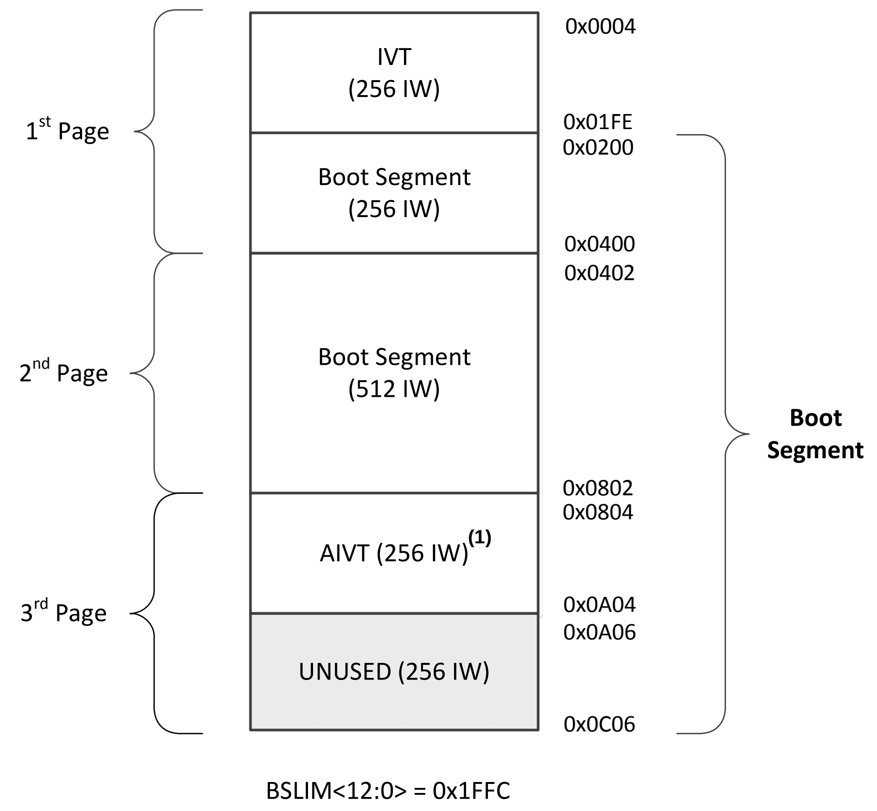
See the **"Interrupt Controller"** chapter in the specific device data sheet for interrupt vector details.

Figure 1-2: Alternate Interrupt Vector Table



Note 1: The address depends on the size of the Boot Segment defined by BSLIM<12:0>. [($\overline{\text{BSLIM<12:0>}}$ – 1) x 0x400] + Offset.

Figure 1-3: Boot Segment Memory and AIVT Implementation



Note 1: To enable the AIVT, set Configuration bit, AIVTDIS (FSEC<15>), and configure the BSLIM<12:0> bits (FBSLIM). Setting AIVTEN (INTCON2<8>) = 1 will switch between the IVT and the AIVT.

1.3 Reset Sequence

A device Reset is not a true exception because the Interrupt Controller is not involved in the Reset process. The dsPIC33/PIC24 device clears its registers during Reset, which forces the Program Counter (PC) to zero. The processor then begins program execution at location, 0x000000. The user application programs a `GOTO` instruction at the Reset address, which redirects program execution to the appropriate start-up routine.

Note: Any unimplemented or unused vector locations in the IVT must be programmed with the address of a default interrupt handler routine that contains a `RESET` instruction.

On devices with Auxiliary Flash, the Reset location can be optionally set to Auxiliary Flash. Refer to the **“Memory Organization”** and **“Special Features”** chapters in the specific device data sheet for availability and how to change the Reset location to Auxiliary Flash.

1.4 CPU Priority Status

The CPU can operate at one of the 16 priority levels that range from 0-15. An interrupt or a trap source must have a priority level greater than the current CPU priority to initiate an exception process. Peripherals and external interrupt sources can be programmed to priority levels between 1-7. CPU Priority Levels 8-15 are reserved for trap sources.

A trap is a non-maskable interrupt source, intended to detect hardware and software problems (see **Section 2.0 "Non-Maskable Traps"**). The priority level for each trap source is fixed. Only one trap is assigned to a priority level. An interrupt source programmed to Priority Level 0 is effectively disabled because it can never be greater than the CPU priority.

The current CPU priority level is indicated by the following status bits:

- CPU Interrupt Priority Level Status bits (IPL<2:0>) in the CPU STATUS Register (SR<7:5>)
- CPU Interrupt Priority Level Status bit 3 (IPL3) in the Core Control register (CORCON<3>)

The IPL<2:0> status bits are readable and writable, so the user application can modify these bits to disable all sources of interrupts below a given priority level. For example, if IPL<2:0> = `011`, the CPU would not be interrupted by any source with a programmed priority level of 1, 2 or 3.

Trap events have higher priority than any user interrupt source. When the IPL3 bit is set, a trap event is in progress. The IPL3 bit can be cleared, but not set, by the user application. In some applications, the IPL3 bit will need to be cleared when a trap has occurred and branch to an instruction other than the instruction immediately after the one that originally caused the trap to occur. All user interrupt sources can be disabled by setting IPL<2:0> = `111`.

1.5 Interrupt Priority

Each peripheral interrupt source can be assigned to one of the seven priority levels. The user-assignable interrupt priority control bits for each individual interrupt are located in the Least Significant three bits of each nibble within the IPCx registers. Bit 3 of each nibble is not used and is read as a '`0`'. These bits define the priority level assigned to a particular interrupt. The usable priority levels are 1 (lowest priority) through 7 (highest priority). If all the IPCx bits associated with an interrupt source are cleared, the interrupt source is effectively disabled.

More than one interrupt request source can be assigned to a specific priority level. To resolve priority conflicts within a given user-assigned level, each source of an interrupt has a natural priority order based on its location in the IVT/AIVT (refer to the **"Interrupt Controller"** chapter in the specific device data sheet for the exact Interrupt Vector Table information). The lower numbered interrupt vectors have higher natural priority, while the higher numbered vectors have lower natural priority. The overall priority level for any pending source of an interrupt is first determined by the user application-assigned priority of that source in the IPCx register, then by the natural order priority within the IVT/AIVT.

Natural order priority is used only to resolve conflicts between simultaneous pending interrupts with the same user application-assigned priority level. Once the priority conflict is resolved and the exception process begins, the CPU can be interrupted only by a source with a higher user application-assigned priority. Interrupts with the same user application-assigned priority, but a higher natural order priority that becomes pending during the exception process, remain pending until the current exception process completes.

Each interrupt source can be assigned to one of seven priority levels. This enables the user application to assign a low natural order priority and a very high overall priority level to an interrupt. For example, the UART1 RX Interrupt can be assigned to Priority Level 7, and the External Interrupt 0 (INT0) can be assigned to Priority Level 1, thereby giving it a very low effective priority.

Note: The peripherals and sources of interrupt available in the IVT/AIVT vary depending on the specific dsPIC33/PIC24 device. The sources of interrupt shown in this document represent a comprehensive listing of all interrupt sources found on dsPIC33/PIC24 devices. Refer to the specific device data sheet for further details.

2.0 NON-MASKABLE TRAPS

Traps are non-maskable, nestable interrupts that adhere to a fixed priority structure. Traps provide a means to correct erroneous operation during debugging and the operation of the application. If the user application does not intend to correct a trap error condition, these vectors must be loaded with the address of a software routine to reset the device. Otherwise, the user application programs the trap vector with the address of a service routine that corrects the trap condition.

The following sources of non-maskable traps are implemented in dsPIC33/PIC24 devices:

- Oscillator Failure Trap
- Stack Error Trap
- Address Error Trap
- Math Error Trap
- DMAC Error Trap
- Generic Hard Trap
- Generic Soft Trap

For many of the trap conditions, the instruction that caused the trap is allowed to complete before exception processing begins. Therefore, the user application may have to correct the action of the instruction that caused the trap.

Each trap source has a fixed priority as defined by its position in the IVT/AIVT. An oscillator failure trap has the highest priority, while a DMA Controller (DMAC) error trap has the lowest priority (see Figure 1-1). In addition, trap sources are classified into two distinct categories: soft traps and hard traps.

2.1 Soft Traps

The DMAC error trap (Priority Level 10), math error trap (Priority Level 11) and stack error trap (Priority Level 12) are categorized as soft trap sources. Soft traps can be treated like non-maskable sources of an interrupt that adhere to the priority assigned by their position in the IVT/AIVT. Soft traps are processed like interrupts and require two cycles to be sampled and Acknowledged prior to exception processing. Therefore, additional instructions may be executed before a soft trap is Acknowledged.

2.1.1 STACK ERROR TRAP (SOFT TRAP, LEVEL 12)

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

A Stack Limit register (SPLIM) associated with the Stack Pointer is uninitialized at Reset. The stack overflow check is not enabled until a word is written to the SPLIM register.

All Effective Addresses (EAs), generated using W15 as a source or destination pointer, are compared against the value in the SPLIM register. If the EA is greater than the contents of the SPLIM register, a stack error trap is generated. In addition, a stack error trap is generated if the EA calculation wraps over the end of data space (0xFFFF).

A stack error can be detected in software by polling the Stack Error Trap Status bit (STKERR) in the INTCON1 register. To avoid re-entry into the Trap Service Routine (TSR), the STKERR status flag must be cleared in software.

2.1.2 MATH ERROR TRAP (SOFT TRAP, LEVEL 11)

Any of the following events can generate a math error trap:

- Accumulator A overflow
- Accumulator B overflow
- Catastrophic accumulator overflow
- Divide-by-zero
- Shift Accumulator (SFTAC) operation that exceeds ±16 bits

The following three bits in the INTCON1 register enable three types of accumulator overflow traps:

- The Accumulator A Overflow Trap Enable bit (OVATE) in the INTCON1 register enables traps for an Accumulator A overflow event
- The Accumulator B Overflow Trap Enable bit (OVBTE) in the INTCON1 register enables traps for an Accumulator B overflow event
- The Catastrophic Overflow Trap Enable bit (COVTE) in the INTCON1 register enables traps for a catastrophic overflow of either accumulator. When this trap is detected, these corresponding Error bits are set in the INTCON1 register:
  - Accumulator A Overflow Trap Flag bit (OVAERR)
  - Accumulator B Overflow Trap Flag bit (OVBERR)
  - Accumulator A Catastrophic Overflow Trap Flag bit (COVAERR)
  - Accumulator B Catastrophic Overflow Trap Flag bit (COVBERR)

An Accumulator A or Accumulator B overflow event is defined as a carry-out from bit 31. The accumulator overflow cannot occur if the 31-Bit Saturation mode is enabled for the accumulator. A catastrophic accumulator overflow is defined as a carry-out from bit 39 of either accumulator. The catastrophic overflow cannot occur if accumulator saturation (31-bit or 39-bit) is enabled.

Divide-by-zero traps cannot be disabled. The divide-by-zero check is performed during the first iteration of the REPEAT loop that executes the divide instruction. The Divide-by-Zero Error Status bit (DIV0ERR) in the INTCON1 register is set when this trap is detected.

Accumulator shift traps cannot be disabled. The SFTAC instruction can be used to shift the accumulator by a literal value or a value in one of the W registers. If the shift value exceeds ±16 bits, an arithmetic trap is generated and the Shift Accumulator Error Status bit (SFTACERR) in the INTCON1 register is set. The SFTAC instruction executes, but the results of the shift are not written to the target accumulator.

A math error trap can be detected in software by polling the Math Error Status bit (MATHERR) in the INTCON1 register. To avoid re-entry into the Trap Service Routine, the MATHERR status flag must be cleared in software. Before the MATHERR status bit can be cleared, all conditions that caused the trap to occur must also be cleared. If the trap was due to an accumulator overflow, the Accumulator Overflow bits (OA and OB) in the SR register must be cleared.

2.1.3 DMAC ERROR TRAP (SOFT TRAP, LEVEL 10)

A Direct Memory Access (DMAC) error trap occurs with these conditions:

- RAM write collision
- DMA ready peripheral RAM write collision

Write collision errors are a serious enough threat to system integrity to warrant a non-maskable CPU trap event. If both the CPU and a DMA channel attempt to write to a target address, the CPU is given priority and the DMA write is ignored. In this case, a DMAC error trap is generated and the DMAC Error Trap Status bit (DMACERR) in the INTCON1 register is set.

2.1.4 GENERIC SOFT TRAP (LEVEL 13)

A generic soft trap occurs when any of the bits within the INTCON3 register are set. Each bit within the INTCON3 register is assigned to a specific trap error condition.

2.1.5 USB ADDRESS ERROR SOFT TRAP (UAE)

All USB endpoints are implemented as buffers in RAM. The CPU and the USB module both have access to the buffers.

The application specifies the location of the endpoint buffers and other data through an endpoint Buffer Descriptor Table (BDT). The size of the endpoint BDT is 512 bytes. The endpoint BDT contains entries called endpoint buffer descriptors for each endpoint. Space for the endpoint buffer descriptor is allocated regardless of the endpoint enable/disable status.

The Start address of the endpoint BDT is specified by the application. This 32-bit address should be aligned at the 512-byte boundary (i.e., the last nine bits of the address should be zero) and is specified in the UxBDTP1, UxBDTP2 and UxBDTP3 registers.

If the UxBDTP1, UxBDTP2 and UxBDTP3 registers are initialized pointing to unimplemented memory, or any of the 512 bytes of the buffer, this results in an unimplemented memory access by the USB module, and a USB address error soft trap will be issued; the UAE bit will be set. Refer to **“USB On-The-Go (OTG)”** (DS70175) for additional information.

2.1.6 DMA ADDRESS ERROR SOFT TRAP (DAE)

The Direct Memory Access (DMA) controller transfers data between peripheral data registers and data space, SRAM. If the DMA module attempts to access any unimplemented memory address, a DMA address error soft trap will be issued and the DAE bit will be set. Refer to **“Direct Memory Access (DMA)”** (DS70348) for additional information.

2.1.7 `DO` STACK OVERFLOW SOFT TRAP (DOOVR)

Up to 4 levels of `DO` loops can be executed and nested in hardware. The `DO` Level bits (DL<2:0>) in the CORCON register indicate the `DO` nesting level and are used to address the `DO` stack. They are automatically updated for all nested `DO` loops. A `DO` level of 0 (DL<2:0> = `000`) indicates that no `DO` loops are nested (i.e., no `DO` state needs to be preserved). A `DO` level of 4 (DL<2:0> = `100`) indicates that four prior `DO` loops are in progress and nested.

If the user attempts to nest a `DO` loop when the `DO` stack is already full (DL<2:0> = `100`, or four `DO` loops are already underway), a `DO` stack overflow soft trap will be issued (DOOVR = `1`). The `DO` instruction that caused the trap will not change any of the `DO` states prior to the instruction execution, nor will it modify the `DO` stack. The user may choose to try and recover the Fault condition, abort the task or just reset the device.

2.2 Hard Traps

Hard traps include exceptions of Interrupt Priority Level 13 through Level 15, inclusive. The address error (Level 14) and oscillator error (Level 15) traps fall into this category.

Like soft traps, hard traps are non-maskable sources of interrupt. The difference between hard traps and soft traps is that hard traps force the CPU to stop code execution after the instruction causing the trap has completed. Normal program execution flow does not resume until the trap has been Acknowledged and processed.

2.2.1 TRAP PRIORITY AND HARD TRAP CONFLICTS

If a higher priority trap occurs while any lower priority trap is in progress, the processing of the lower priority trap is suspended. The higher priority trap is Acknowledged and processed. The lower priority trap remains pending until the processing of the higher priority trap completes.

Each hard trap that occurs must be Acknowledged before code execution of any type can continue. If a lower priority hard trap occurs while a higher priority trap is pending, Acknowledged or is being processed, a hard trap conflict occurs because the lower priority trap cannot be Acknowledged until the processing for the higher priority trap completes.

The device is automatically reset in a hard trap conflict condition. The Trap Reset Flag Status bit (TRAPR) in the Reset Control register (RCON<15>) is set when the Reset occurs, so that the condition can be detected in software. For further details, refer to **“Reset”** (DS70602) in the *“dsPIC33/PIC24 Family Reference Manual”.*

2.2.2 OSCILLATOR FAILURE TRAP (HARD TRAP, LEVEL 15)

An oscillator failure trap event is generated for any of these reasons:

- The Fail-Safe Clock Monitor (FSCM) is enabled and has detected a loss of the system clock source
- A loss of PLL lock has been detected during normal operation using the PLL
- The FSCM is enabled and the PLL fails to achieve lock at a Power-on Reset (POR)

An oscillator failure trap event can be detected in software by polling the Oscillator Failure Trap bit (OSCFAIL) in the INTCON1 register or the Clock Fail bit (CF) in the OSCCON register. To avoid re-entry into the Trap Service Routine, the OSCFAIL status flag must be cleared in software. For more information about the Fail-Safe Clock Monitor, refer to **“Oscillator”** (DS70580) and **“Device Configuration”** (DS70618) in the *“dsPIC33/PIC24 Family Reference Manual”*.

2.2.3 ADDRESS ERROR TRAP (HARD TRAP, LEVEL 14)

Operating conditions that can generate an address error trap include the following:

- A misaligned data word fetch is attempted. This condition occurs when an instruction performs a word access with the Least Significant bit (LSb) of the Effective Address (EA) set to ‘`1`’. The dsPIC33/PIC24 CPU requires all word accesses to be aligned to an even address boundary.
- A bit manipulation instruction uses the Indirect Addressing mode with the LSb of the Effective Address set to ‘`1`’.
- A data fetch is attempted from unimplemented data address space.
- Execution of a `BRA #literal` instruction or a `GOTO #literal` instruction, where ‘`literal`’ is an unimplemented program memory address.
- A data read or write is attempted with Paged Addressing when the dsr/dsw page is 0.
- Execution of instructions after the PC has been modified to point to unimplemented program memory addresses. The PC can be modified by loading a value into the stack and executing a `RETURN` instruction.

When an address error trap occurs, data space writes are inhibited so that data is not destroyed.

An address error can be detected in software by polling the ADDRERR status bit (INTCON1<3>). To avoid re-entry into the Trap Service Routine, the ADDRERR status flag must be cleared in software.

Note: In the `MAC` class of instructions, the data space is split into X and Y spaces. In these instructions, unimplemented X space includes all of Y space and unimplemented Y space includes all of X space.

2.2.4 GENERIC HARD TRAP

A generic hard trap occurs when the following occurs:

- The SWTRAP bit in the INTCON2 register is set
- Any bit within the INTCON4 register is set

Note: If the SWTRAP bit (INTCON2<13>) is set to ‘`1`’ by the user, the SGHT bit (INTCON4<0>) is automatically set to ‘`1`’, which causes code execution to enter the generic hard trap handler. Both the SWTRAP and SGHT bits should be cleared (i.e., set to ‘`0`’) in the trap handler to avoid repetitive traps.

2.3 Disable Interrupts Instruction

The `DISI` (Disable Interrupts) instruction can disable interrupts for up to 16384 instruction cycles. This instruction is useful for executing time critical code segments.

The `DISI` instruction only disables interrupts with Priority Levels 1-6. Priority Level 7 interrupts, and all trap events, can still interrupt the CPU when the `DISI` instruction is active.

The `DISI` instruction works in conjunction with the Disable Interrupts Count register (DISICNT) in the CPU. When the DISICNT register is non-zero, Priority Levels 1-6 interrupts are disabled. The DISICNT register is decremented on each subsequent instruction cycle. When the DISICNT register counts down to zero, Priority Levels 1-6 interrupts are re-enabled. The value specified in the `DISI` instruction includes all cycles due to PSV accesses, instruction stalls and so on.

The DISICNT register is both readable and writable. The user application can terminate the effect of a previous `DISI` instruction early by clearing the DISICNT register. The duration for which the interrupts are disabled can also be increased by writing to, or adding to, the DISICNT register.

If the DISICNT register is zero, interrupts cannot be disabled by simply writing a non-zero value to the register. Interrupts must first be disabled by using the `DISI` instruction. Once the `DISI` instruction has executed and DISICNT holds a non-zero value, the application can extend the interrupt disable time by modifying the contents of the DISICNT register.

The `DISI` Instruction Status bit (DISI) in the INTCON2 register is set whenever interrupts are disabled as a result of the `DISI` instruction.

Note: The `DISI` instruction can be used to quickly disable all user interrupt sources if no source is assigned to CPU Priority Level 7.

2.3.1 GLOBAL INTERRUPT ENABLE (GIE)

A Global Interrupt Enable bit (GIE) is used to enable or disable all interrupts globally. When the GIE bit is cleared, it causes the Interrupt Controller to behave as if the CPU's IPLx bits are set to 7 (see Register 4-1) and disables all interrupts except the traps. When the GIE bit is set again, the Interrupt Controller acts based on the IPL value and the system will return to the previous operating state, depending on the prior interrupt priority bit settings.

Note 1: The GIE bit does not modify the CPU's IPLx bits.

2: There is one cycle delay between clearing the GIE bit and the interrupts being disabled.

2.4 Interrupt Operation

All interrupt event flags are sampled during each instruction cycle. A pending Interrupt Request (IRQ) is indicated by the flag bit = `1` in an IFSx register. The IRQ causes an interrupt if the corresponding bit in the Interrupt Enable Control (IECx) registers is set. During the rest of the instruction cycle in which the IRQ is sampled, the priorities of all pending interrupt requests are evaluated.

No instruction is aborted when the CPU responds to the IRQ. When the IRQ is sampled, the instruction in progress is completed before the Interrupt Service Routine (ISR) is executed.

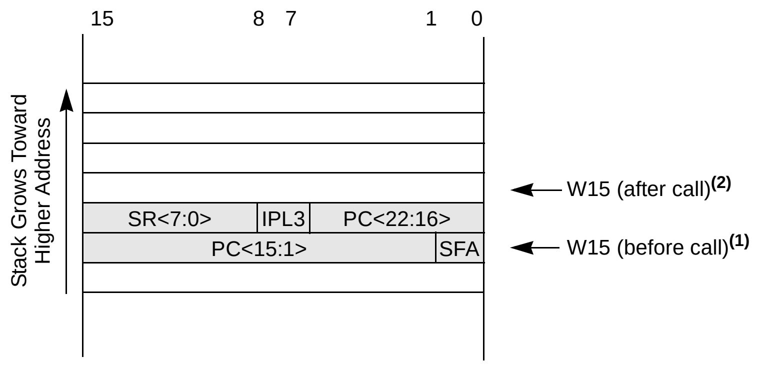
If there is a pending IRQ with a user application-assigned priority level greater than the current processor priority level, indicated by the IPL<2:0> status bits (SR<7:5>), an interrupt is presented to the processor. The processor then saves the following information on the software stack:

- Current PC value
- Low byte of the processor STATUS Register (SRL) containing the processor priority level (IPL<2:0>) just prior to the beginning of the interrupt cycle
- IPL3 status bit (CORCON<3>)
- SFA: Stack Frame Active (CORCON<2>)

These three values allow the return PC address value, MCU status bits and current processor priority level to be automatically saved.

After this information is saved on the stack, the CPU writes the priority level of the pending interrupt into the IPL<2:0> bit locations. This action disables all interrupts of lower or equal priority until the ISR is terminated using the `RETFIE` instruction.

Figure 2-1: Stack Operation for Interrupt Event



Note 1: SFA is at a logic High state.
2: SFA is at a logic Low state (cleared).

2.4.1 RETURN FROM INTERRUPT

The `RETFIE` (Return from Interrupt) instruction unstacks the PC return address, IPL3 status bit, SFA bit and SRL register to return the processor to the state and priority level that existed before the interrupt sequence.

2.4.2 INTERRUPT NESTING

Interrupts are nestable by default. Any ISR in progress can be interrupted by another source of interrupt with a higher user application-assigned priority level. Interrupt nesting can be disabled by setting the Interrupt Nesting Disable bit (NSTDIS) in the INTCON1 register. When the NSTDIS control bit is set, all interrupts in progress force the CPU priority to Level 7 by setting IPL<2:0> = `111`. This action effectively masks all other sources of interrupt until a `RETFIE` instruction is executed. When interrupt nesting is disabled, the user application-assigned Interrupt Priority Levels (IPLs) have no effect except to resolve conflicts between simultaneous pending interrupts.

The IPL<2:0> bits (SR<7:5>) become read-only when interrupt nesting is disabled. This prevents the user application from setting IPL<2:0> to a lower value, which would effectively re-enable interrupt nesting.

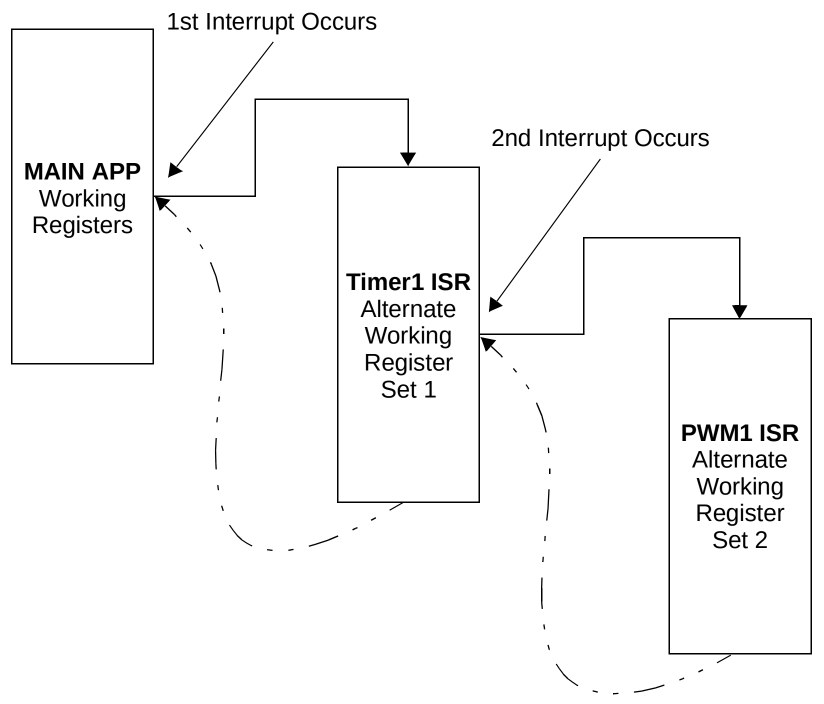
For example, Figure 2-2 demonstrates a typical nested interrupt sequence using two peripherals and the Alternate Working registers. Note that not all devices have the Alternate Working register feature. Refer to the specific device data sheet to determine if this feature is included. It is assumed that the Interrupt Priority Level for Timer1 (T1IP) is set to 4 and the Interrupt Priority Level for the PWM1 module (PWM1IP) is set to 6. Therefore, the PWM1 module has a higher priority than the Timer1 module. Additionally, the Alternate Working Register Set 1 priority level is set to 4 and the Alternate Working Register Set 2 priority level is set to 6. Refer to the *“dsPIC33/ PIC24 Family Reference Manual”*, **“CPU”** for further details regarding the Alternate Working register sets.

As shown in Figure 2-2, the application begins in the main application code using the default Working registers. As soon as there is an interrupt due to the Timer1 module, the Timer1 Interrupt Flag (T1IF) is set. The Timer1 Interrupt Priority Level is compared to the CTXT1 and CTXT2 bit fields within the FALTREG register. As noted above, we know that the Timer1 (T1IP) is equal to CTXT1. Therefore, the Alternate Working Register Set 1 is used for the processing of the Timer1 ISR. During the Timer1 ISR, there is an additional interrupt due to the PWM1 module. The PWM1 Interrupt Flag (PWM1IF) is set. Since the PWM1 Interrupt Priority Level is higher than the Interrupt Priority Level of the Timer1 module, the code jumps to service the PWM1 ISR. The PWM1 Interrupt Priority Level (PWM1IP) is compared to CTXT1 and CTXT2. Since the PWM1 Interrupt Priority Level is equal to CTXT2, the Alternate Working Register Set 2 is used for the PWM ISR processing. Once the PWM1 ISR processing finishes, the PWM1IF flag is cleared, and the `RETFIE` (Return from Interrupt) instruction unstacks the Program Counter and status registers to return the processor to its state prior to the interrupt sequence.

Control is returned to the Timer1 ISR, whereby the Alternate Working Register Set 1 is again utilized. Once the Timer1 ISR completes its processing, the T1IF flag is cleared and the `RETFIE` instruction is again called. This returns the application to its initial state, utilizing the default Working register set prior to any peripheral interrupts.

Note: For the special case of disabling nested interrupts, the Alternate Working registers should not be used for those devices that provide the Alternate Working registers as an option.

Figure 2-2: Nested Interrupt Sequence


1st Interrupt Occurs
MAIN APP Working Registers
2nd Interrupt Occurs
Timer1 ISR Alternate Working Register Set 1
PWM1 ISR Alternate Working Register Set 2

1. First Interrupt Process:

a) Timer1 Interrupt Flag (T1IF) is set.

b) T1IP is compared to both CTXT1 and CTXT2 in the FALTREG register.

c) Because T1IP is equal to the CTXT1 Input Priority Level, Set 1 of the Alternative Working registers is used for Timer1 ISR processing.

d) PWM1 Interrupt Flag (PWM1IF) is set during Timer1 ISR.

e) Return from PWM ISR.

f) Timer1 ISR process continues and then completes. T1IF flag is cleared.

g) `RETFIE` (Return from Interrupt) instruction unstacks the Program Counter and status registers to return the process to its state prior to the interrupt sequence.
   - Returns to main application and Working registers.

2. Second Interrupt Process:

a) PWM1IP is compared to both CTXT1 and CTXT2 in the FALTREG register.

b) Because PWM1IP is equal to the CTXT2 Input Priority Level, Set 2 of the Alternative Working registers is used for PWM1 ISR processing.

c) PWM1 ISR process completes and the PWM1IF flag is cleared.

d) `RETFIE` (Return from Interrupt) instruction unstacks the Program Counter and status registers to return the processor to its state prior to the interrupt sequence.

Note: Figure 2-2 assumes that the application software includes the configuration as shown in Example 2-1.

Example 2-1: Configuration Register Setup

```
_FALTREG (CTXT1_IPL4 & CTXT2_IPL6);       // Where CTXT1 represents IPL for working register set 1
                                          // and CTXT2 represents IPL for working register set 2

IPC0bits.T1IP    = 4;                     // Timer1 interrupt priority level = 4
IPC23bits.PWM1IP = 6;                     // PWM1 interrupt priority level = 6
```

2.5 Wake-up from Sleep and Idle

Any source of interrupt that is individually enabled, using its corresponding control bit in the IECx registers, can wake-up the processor from Sleep or Idle mode. When the interrupt status flag for a source is set and the interrupt source is enabled by the corresponding bit in the IECx Control registers, a wake-up signal is sent to the dsPIC33/PIC24 CPU. When the device wakes from Sleep or Idle mode, one of two actions occur:

- If the Interrupt Priority Level for that source is greater than the current CPU priority level, the processor will process the interrupt and branch to the ISR for the interrupt source.
- If the user application-assigned Interrupt Priority Level for the source is lower than, or equal to, the current CPU priority level, the processor will continue execution, starting with the instruction immediately following the `PWRSAV` instruction that previously put the CPU in Sleep or Idle mode.

Note: User interrupt sources that are assigned to Priority Level 0 cannot wake the CPU from Sleep or Idle mode because the interrupt source is effectively disabled. To use an interrupt as a wake-up source, the program must assign the user interrupt source to a Priority Level 1 or greater.

2.6 External Interrupt Support

The dsPIC33/PIC24 supports up to five external interrupt pin sources (INT0-INT4). Each external interrupt pin has edge detection circuitry to detect the interrupt event. The INTCON2 register has five control bits (INT0EP-INT4EP) that select the polarity of the edge detection circuitry. Each external interrupt pin can be programmed to interrupt the CPU on a rising edge or falling edge event. See Register 4-4 for further details.

2.6.1 ANALOG-TO-DIGITAL CONVERTER (ADC) EXTERNAL CONVERSION REQUEST

On some devices, the INT0 external interrupt request pin is shared with the ADC as an external conversion request signal. The INT0 interrupt source has programmable edge polarity, which is also available to the ADC external conversion request feature.

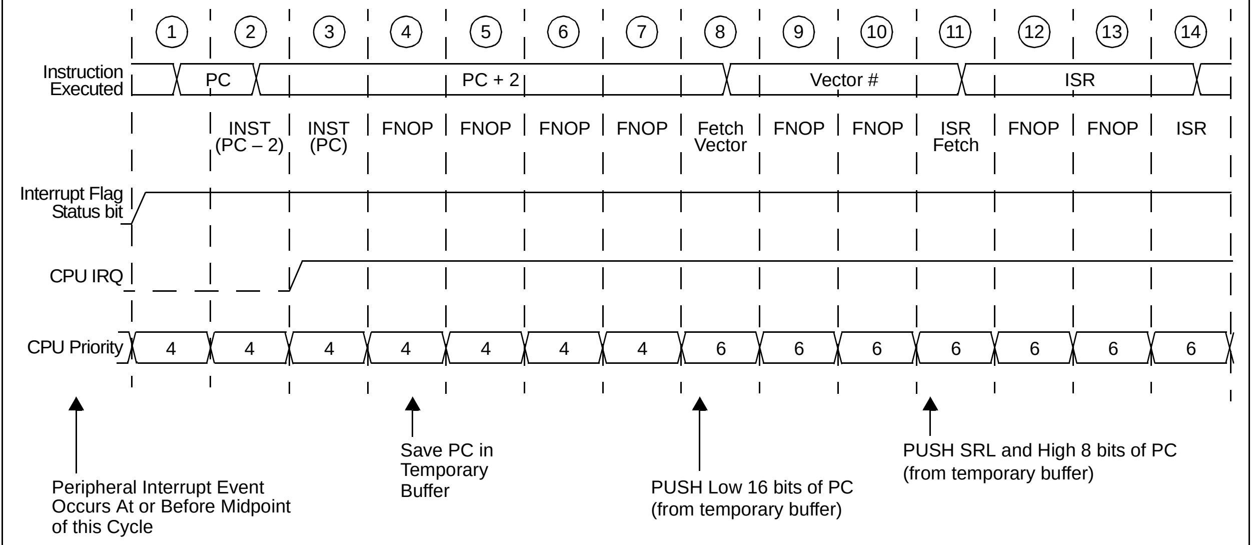
3.0 INTERRUPT PROCESSING TIMING

3.1 Interrupt Latency for One-Cycle Instructions

Figure 3-1 illustrates the sequence of events when a peripheral interrupt is asserted during a one-cycle instruction. The interrupt process takes ten instruction cycles. Each cycle is numbered in the figure for reference.

During the instruction cycle, the interrupt flag status bit is set after the peripheral interrupt occurs. The current instruction completes during this instruction cycle. In the second instruction cycle after the interrupt event, the contents of the PC and the STATUS Register Lower Byte (SRL) are saved into a temporary buffer register. The second cycle of the interrupt process is executed as a `NOP` to maintain consistency with the sequence taken during a two-cycle instruction (see **Section 3.2 "Interrupt Latency for Two-Cycle Instructions"**). In the third cycle, the PC is loaded with the vector table address for the interrupt source and the starting address of the ISR is fetched. In the fourth cycle, the PC is loaded with the ISR address. The fourth cycle is executed as a `NOP` while the first instruction in the ISR is fetched.

Figure 3-1: Interrupt Timing During a One-Cycle Instruction


1
2
3
4
5
6
7
8
9
10
11
12
13
14
Instruction Executed
PC
PC + 2
Vector #
ISR
INST (PC – 2)
INST (PC)
FNOP
FNOP
FNOP
FNOP
Fetch Vector
FNOP
FNOP
ISR Fetch
FNOP
FNOP
ISR
Interrupt Flag Status bit
CPU IRQ
CPU Priority
4
6
Peripheral Interrupt Event Occurs At or Before Midpoint of this Cycle
Save PC in Temporary Buffer
PUSH Low 16 bits of PC (from temporary buffer)
PUSH SRL and High 8 bits of PC (from temporary buffer)

3.2 Interrupt Latency for Two-Cycle Instructions

The interrupt latency during a two-cycle instruction is the same as during a one-cycle instruction. The first and second cycle of the interrupt process allow the two-cycle instruction to complete execution. The timing diagram in Figure 3-2 illustrates the peripheral interrupt event occurring in the instruction cycle prior to execution of the two-cycle instruction.

Figure 3-3 illustrates the timing when a peripheral interrupt coincides with the first cycle of a two-cycle instruction. In this case, the interrupt process completes as for a one-cycle instruction (see **Section 3.1 "Interrupt Latency for One-Cycle Instructions"**).

Figure 3-2: Interrupt Timing During a Two-Cycle Instruction

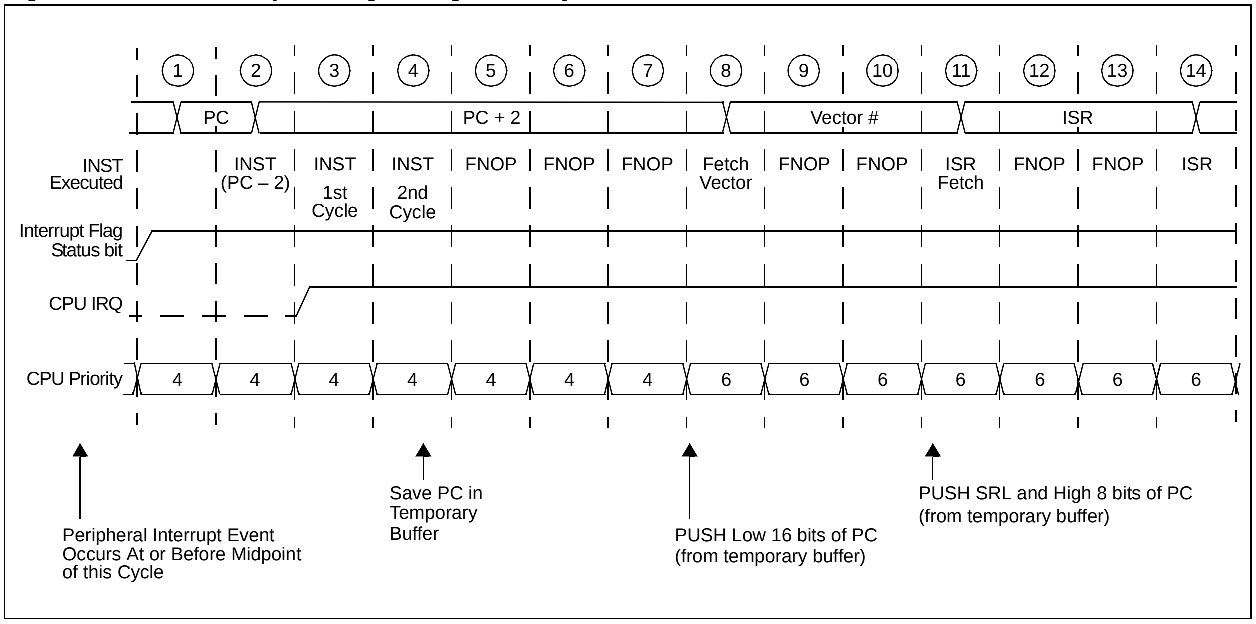
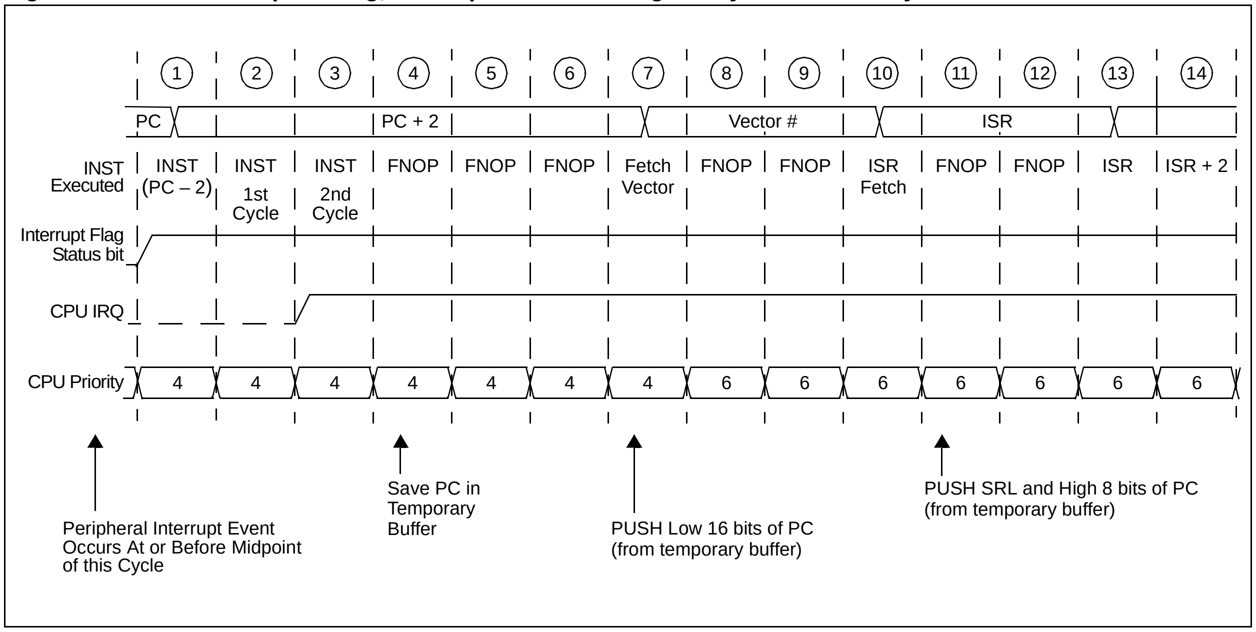

1
2
3
4
5
6
7
8
9
10
11
12
13
14
PC
PC + 2
Vector #
ISR
INST Executed
INST (PC – 2)
INST 1st Cycle
INST 2nd Cycle
FNOP
Fetch Vector
ISR Fetch
ISR
Interrupt Flag Status bit
CPU IRQ
CPU Priority
4
6
Peripheral Interrupt Event Occurs At or Before Midpoint of this Cycle
Save PC in Temporary Buffer
PUSH Low 16 bits of PC (from temporary buffer)
PUSH SRL and High 8 bits of PC (from temporary buffer)

Figure 3-3: Interrupt Timing, Interrupt Occurs During 1st Cycle of a Two-Cycle Instruction

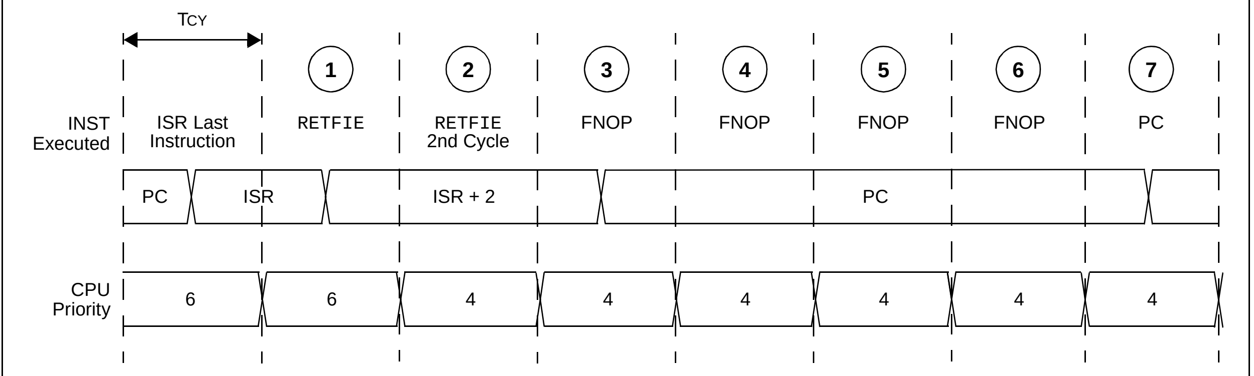

1
2
3
4
5
6
7
8
9
10
11
12
13
14
PC
PC + 2
Vector #
ISR
INST Executed
INST (PC – 2)
INST 1st Cycle
INST 2nd Cycle
FNOP
Fetch Vector
ISR Fetch
ISR
ISR + 2
Interrupt Flag Status bit
CPU IRQ
CPU Priority
4
6
Peripheral Interrupt Event Occurs At or Before Midpoint of this Cycle
Save PC in Temporary Buffer
PUSH Low 16 bits of PC (from temporary buffer)
PUSH SRL and High 8 bits of PC (from temporary buffer)

3.3 Return from Interrupt

To return from an interrupt, the program must call the `RETFIE` instruction.

During the first two cycles of a `RETFIE` instruction, the contents of the PC and the SRL register are popped from the stack. The third instruction cycle is used to fetch the instruction addressed by the updated Program Counter. This cycle executes as a `NOP` instruction. On the fourth cycle, program execution resumes at the point where the interrupt occurred.

Figure 3-4: Return from Interrupt Timing



3.4 Exception Latency

The interrupt latency can be selected as a fixed latency or variable latency. An exception process can operate in one of two modes. It is determined by the state of the VAR bit in the Core Control register (CORCON<15>).

3.4.1 FIXED LATENCY (VAR = `0` AND CPU HIGHEST PRIORITY MASTER)

If VAR = `0` (default Reset state) and the CPU is the highest priority EDS bus master (MSTRPR<2:0> = `000`), the CPU offers a deterministic fixed latency response to any highest priority exception. The interrupt latency, which is a time interval between the instance when an interrupt is recognized and the instance when the first instruction of the ISR is executed, remains unchanged for all instructions, including a `TBLRDx` instruction or instructions that require PSV access.

When the following occurs, the interrupt latency time is not fixed:

- The `MOV.D` instruction, accessing data via PSV, adds one cycle to complete the second PSV fetch
- A `TBLRDx` or PSV access instruction, associated with a stall cycle, adds one cycle
- Last iteration of a repeated PSV access adds one cycle

3.4.2 VARIABLE LATENCY (VAR = `1` AND CPU HIGHEST PRIORITY MASTER)

If VAR = `1` and the CPU is the highest priority EDS bus master (MSTRPR<2:0> = `000`), the CPU offers a variable latency response to all exceptions. If more than one interrupt is active, and the device is operating in Non-Nested mode, a higher priority interrupt request can occur when a lower priority interrupt is being processed. Therefore, the time required to execute the longest ISR must be added to the maximum latency.

If VAR = `0`, the exception processing time requires a 13-instruction cycle, Flash access time (i.e., 216 ns at a 60 MHz operating speed).

If VAR = `1`, the exception processing time is variable and it requires 9-13-instruction cycle, Flash access time (i.e., 150 ns through 216 ns at a 60 MHz operating speed).

3.5 Special Conditions for Interrupt Latency

The dsPIC33/PIC24 allows the current instruction to complete when a peripheral interrupt source becomes pending. The interrupt latency is the same for both one-cycle and two-cycle instructions. However, certain conditions can increase interrupt latency by one cycle, depending on when the interrupt occurs. If a fixed latency is critical to the application, the following conditions should be avoided:

- A `MOV.D` instruction is executed that uses PSV to access a value in program memory space
- An instruction stall cycle is appended to any two-cycle instruction
- An instruction stall cycle is appended to any one-cycle instruction that performs a PSV access
- A bit test and skip instruction (`BTSC`, `BTSS`) that uses PSV to access a value in the program memory space

4.0 INTERRUPT CONTROL AND STATUS REGISTERS

The following registers are associated with the Interrupt Controller:

- **INTCONx:** Interrupt Control Registers (Register 4-3 through Register 4-6)

  These registers control global interrupt functions, where 'x' denotes the register number:

  - **INTCON1: Interrupt Control Register 1** contains the Interrupt Nesting Disable bit (NSTDIS), as well as the control and status flags for the processor trap sources.
  - **INTCON2: Interrupt Control Register 2** controls external interrupt request signal behavior and the use of the Alternate Interrupt Vector Table.
  - **INTCON3: Interrupt Control Register 3** controls the soft trap status bits.
  - **INTCON4: Interrupt Control Register 4** controls software generated hard trap status bits.
- **IFSx: Interrupt Flag Status Registers(1)**

  All interrupt request flags are maintained in the IFSx registers, where 'x' denotes the register number. Each source of interrupt has a status bit, which is set by the respective peripherals or external signals and cleared by software.
- **IECx: Interrupt Enable Control Registers(1)**

  All Interrupt Enable Control bits are maintained in the IECx registers, where 'x' denotes the register number. These control bits are used to individually enable interrupts from the peripherals or external signals.
- **IPCx: Interrupt Priority Control Registers(1)**

  Each user interrupt source can be assigned to one of eight priority levels. The IPCx registers set the Interrupt Priority Level (IPL) for each source of interrupt.
- **INTTREG: Interrupt Control and Status Register**

  This register contains the associated interrupt vector number and the new CPU Interrupt Priority Level, which are latched into the Vector Number (VECNUM<7:0>) and Interrupt Level (ILR<3:0>) bit fields. The new Interrupt Priority Level is the priority of the pending interrupt.
- **SR: STATUS Register**

  The SR is not a specific part of the Interrupt Controller hardware; however, it contains the IPL<2:0> status bits (SR<7:5>), which indicate the current CPU priority level. The user application can change the current CPU priority level by writing to the IPLx bits.
- **CORCON: Core Control Register**

  The CORCON register is not a specific part of the Interrupt Controller hardware; however, it contains the IPL3 status bit, which indicates the current CPU priority level. The IPL3 is a read-only bit so that trap events cannot be masked by the user application.
- **FALTREG: Alternate Working Register Set Priority Level Register**

  The FALTREG register is not a specific part of the Interrupt Controller hardware; however, it contains the Interrupt Priority Level, CTXT1 and CTXT2 bit fields, for the Alternate Working register sets that can be used for dedicated Interrupt Service Routine (ISR) processing.

  Each register is described in detail in the following sections.

Note: The total number and type of interrupt sources depend on the device variant. Refer to the specific device data sheet for further details.

Register 4-1: SR: STATUS Register

R-0	R-0	R/C-0	R/C-0	R-0	R/C-0	R-0	R-0
OA	OB	SA	SB	OAB	SAB	DA	DC
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
IPL2(1,2)	IPL1(1,2)	IPL0(1,2)	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 15-8	**Not used by the Interrupt Controller** (Refer to **“CPU”** (DS70359) in the *“dsPIC33/PIC24 Family Reference Manual”* for the SR bit descriptions.)
bit 7-5	**IPL<2:0>:** CPU Interrupt Priority Level Status bits(1,2) `111` = CPU Interrupt Priority Level is 7 (15); user interrupts are disabled `110` = CPU Interrupt Priority Level is 6 (14) `101` = CPU Interrupt Priority Level is 5 (13) `100` = CPU Interrupt Priority Level is 4 (12) `011` = CPU Interrupt Priority Level is 3 (11) `010` = CPU Interrupt Priority Level is 2 (10) `001` = CPU Interrupt Priority Level is 1 (9) `000` = CPU Interrupt Priority Level is 0 (8)
bit 4-0	**Not used by the Interrupt Controller** (Refer to **“CPU”** (DS70359) in the *“dsPIC33/PIC24 Family Reference Manual”* for the SR bit descriptions.)

Note 1: 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`.

2: The IPL<2:0> status bits are read-only when NSTDIS = `1` (INTCON1<15>).

Register 4-2: CORCON: Core Control Register

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
VAR	—	US1	US0	EDT	DL2	DL1	DL0
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	SATB	SATDW	ACCSAT	IPL3[1]	SFA	RND	IF
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	**VAR:** Variable Exception Processing Latency Control bit `1` = Variable exception processing latency `0` = Fixed exception processing latency (Refer to **Section 3.4 “Exception Latency”** for more information.)
bit 14-4	**Not used by the Interrupt Controller** (Refer to **“CPU”** (DS70359) in the *“dsPIC33/PIC24 Family Reference Manual”* for the SR bit descriptions.)
bit 3	**IPL3:** CPU Interrupt Priority Level Status bit 3[1] `1` = CPU Interrupt Priority Level is greater than 7 `0` = CPU Interrupt Priority Level is 7 or less
bit 2-0	**Not used by the Interrupt Controller** (Refer to **“CPU”** (DS70359) in the *“dsPIC33/PIC24 Family Reference Manual”* for the SR bit descriptions.)

Note 1: The IPL3 bit is concatenated with the IPL<2:0> bits (SR<7:5>) to form the CPU Interrupt Priority Level.

Register 4-3: INTCON1: Interrupt Control Register 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NSTDIS	OVAERR	OVBERR	COVAERR	COVBERR	OVATE	OVBTE	COVTE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
SFTACERR	DIV0ERR	DMACERR	MATHERR	ADDRERR	STKERR	OSCFAIL	—
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	**NSTDIS:** Interrupt Nesting Disable bit 1 = Interrupt nesting is disabled 0 = Interrupt nesting is enabled
bit 14	**OVAERR:** Accumulator A Overflow Trap Flag bit 1 = Trap was caused by overflow of Accumulator A 0 = Trap was not caused by overflow of Accumulator A
bit 13	**OVBERR:** Accumulator B Overflow Trap Flag bit 1 = Trap was caused by overflow of Accumulator B 0 = Trap was not caused by overflow of Accumulator B
bit 12	**COVAERR:** Accumulator A Catastrophic Overflow Trap Flag bit 1 = Trap was caused by catastrophic overflow of Accumulator A 0 = Trap was not caused by catastrophic overflow of Accumulator A
bit 11	**COVBERR:** Accumulator B Catastrophic Overflow Trap Flag bit 1 = Trap was caused by catastrophic overflow of Accumulator B 0 = Trap was not caused by catastrophic overflow of Accumulator B
bit 10	**OVATE:** Accumulator A Overflow Trap Enable bit 1 = Trap overflow of Accumulator A 0 = Trap is disabled
bit 9	**OVBTE:** Accumulator B Overflow Trap Enable bit 1 = Trap overflow of Accumulator B 0 = Trap is disabled
bit 8	**COVTE:** Catastrophic Overflow Trap Enable bit 1 = Trap on catastrophic overflow of Accumulator A or B is enabled 0 = Trap is disabled
bit 7	**SFTACERR:** Shift Accumulator Error Status bit 1 = Math error trap was caused by an invalid accumulator shift 0 = Math error trap was not caused by an invalid accumulator shift
bit 6	**DIV0ERR:** Divide-by-zero Error Status bit 1 = Divide-by-zero error trap was caused by a divide-by-zero 0 = Divide-by-zero error trap was not caused by a divide-by-zero
bit 5	**DMACERR:** DMAC Error Trap Status bit 1 = DMAC trap has occurred 0 = DMAC trap has not occurred
bit 4	**MATHERR:** Math Error Status bit 1 = Math error trap has occurred 0 = Math error trap has not occurred

Register 4-3: INTCON1: Interrupt Control Register 1 (Continued)

bit 3	**ADDRERR:** Address Error Trap Status bit 1 = Address error trap has occurred 0 = Address error trap has not occurred
bit 2	**STKERR:** Stack Error Trap Status bit 1 = Stack error trap has occurred 0 = Stack error trap has not occurred
bit 1	**OSCFAIL:** Oscillator Failure Trap Status bit 1 = Oscillator failure trap has occurred 0 = Oscillator failure trap has not occurred
bit 0	**Unimplemented:** Read as ‘0’

Register 4-4: INTCON2: Interrupt Control Register 2

R/W-1	R/W-0	R/W-0	U-0	U-0	U-0	U-0	R/W-0
GIE	DISI	SWTRAP	—	—	—	—	AIVTEN
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP
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	**GIE:** Global Interrupt Enable bit 1 = Interrupts and associated interrupt enable bits are enabled 0 = Interrupts are disabled, but traps are still enabled
bit 14	**DISI:** `DISI` Instruction Status bit 1 = `DISI` instruction is active 0 = `DISI` instruction is not active
bit 13	**SWTRAP:** Software Trap Status bit 1 = Software trap is enabled 0 = Software trap is disabled
bit 12-9	**Unimplemented:** Read as ‘0’
bit 8	**AIVTEN:** Alternate Interrupt Vector Table Enable bit 1 = Uses Alternate Interrupt Vector Table 0 = Uses standard Interrupt Vector Table
bit 7-5	**Unimplemented:** Read as ‘0’
bit 4	**INT4EP:** External Interrupt 4 Edge Detect Polarity Select bit 1 = Interrupt on negative edge 0 = Interrupt on positive edge
bit 3	**INT3EP:** External Interrupt 3 Edge Detect Polarity Select bit 1 = Interrupt on negative edge 0 = Interrupt on positive edge
bit 2	**INT2EP:** External Interrupt 2 Edge Detect Polarity Select bit 1 = Interrupt on negative edge 0 = Interrupt on positive edge
bit 1	**INT1EP:** External Interrupt 1 Edge Detect Polarity Select bit 1 = Interrupt on negative edge 0 = Interrupt on positive edge
bit 0	**INT0EP:** External Interrupt 0 Edge Detect Polarity Select bit 1 = Interrupt on negative edge 0 = Interrupt on positive edge

Register 4-5: INTCON3: Interrupt Control Register 3

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	NAE
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0
—	UAE	DAE	DOOVR	—	USBPLL(1)	—	APLL(1)
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 15-9	**Unimplemented:** Read as ‘0’
bit 8	**NAE:** NVM Address Error Soft Trap Status bit 1 = NVM address error soft trap has occurred 0 = NVM address error soft trap has not occurred
bit 7	**Unimplemented:** Read as ‘0’
bit 6	**UAE:** USB Address Error Soft Trap Status bit 1 = USB address error (soft) trap has occurred 0 = USB address error (soft) trap has not occurred
bit 5	**DAE:** DMA Address Error Soft Trap Status bit 1 = DMA Address error soft trap has occurred 0 = DMA Address error soft trap has not occurred
bit 4	**DOOVR:** DO Stack Overflow Soft Trap Status bit 1 = DO stack overflow soft trap has occurred 0 = DO stack overflow soft trap has not occurred
bit 3	**Unimplemented:** Read as ‘0’
bit 2	**USBPLL:** USB PLL Loss of Lock Soft Trap Status bit(1) 1 = USB PLL lock fail trap has occurred 0 = USB PLL lock fail trap has not occurred
bit 1	**Unimplemented:** Read as ‘0’
bit 0	**APLL:** Auxiliary PLL Loss of Lock Soft Trap Status bit(1) 1 = APLL lock soft trap has occurred 0 = APLL lock soft trap has not occurred

Note 1: This bit field is not available on all devices. See the specific device data sheet for details.

Register 4-6: INTCON4: Interrupt Control Register 4

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	SGHT
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 15-1	**Unimplemented:** Read as ‘0’
bit 0	**SGHT:** Software Generated Hard Trap Status bit 1 = Software generated hard trap has occurred 0 = Software generated hard trap has not occurred

Register 4-7: IFSx: Interrupt Flag Status Registers(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS<7:0>							
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 15-0 **IFS<15:0>:** Interrupt Flag Status bits
`1` = Interrupt request has occurred
`0` = Interrupt request has not occurred

Note 1: This register represents a generic definition of the IFSx register. Refer to the **“Interrupt Controller”** chapter in the specific device data sheet for the exact bit definitions.

Register 4-8: IECx: Interrupt Enable Control Registers(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC<7:0>							
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 **IEC<15:0>:** Interrupt Enable Control bits
`1` = Interrupt request is enabled
`0` = Interrupt request is not enabled

Note 1: This register represents a generic definition of the IECx register. Refer to the **“Interrupt Controller”** chapter in the specific device data sheet for the exact bit definitions.

Register 4-9: IPCx: Interrupt Priority Control Registers[1]

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	IP3<2:0>			—	IP2<2:0>		
bit 15							bit 8

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	IP1<2:0>			—	IP0<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	**Unimplemented:** Read as ‘0’
bit 14-12	**IP3<2:0>:** Interrupt Priority bits 111 = Interrupt is Priority 7 (highest priority interrupt) • • • 001 = Interrupt is Priority 1 000 = Interrupt source is disabled
bit 11	**Unimplemented:** Read as ‘0’
bit 10-8	**IP2<2:0>:** Interrupt Priority bits These bits have the same definition as bits 14-12.
bit 7	**Unimplemented:** Read as ‘0’
bit 6-4	**IP1<2:0>:** Interrupt Priority bits These bits have the same definition as bits 14-12.
bit 3	**Unimplemented:** Read as ‘0’
bit 2-0	**IP0<2:0>:** Interrupt Priority bits These bits have the same definition as bits 14-12.

Note 1: This register represents a generic definition of the IPCx register. Refer to the **“Interrupt Controller”** chapter in the specific device data sheet for the exact bit definitions.

Register 4-10: INTTREG: Interrupt Control and Status Register

U-0	U-0	U-0	U-0	R-0	R-0	R-0	R-0
—	—	—	—	ILR<3:0>			
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
VECNUM<7:0>							
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 15-12	**Unimplemented:** Read as ‘0’
bit 11-8	**ILR<3:0>:** New CPU Interrupt Priority Level bits 1111 = CPU Interrupt Priority Level is 15 • • • 0001 = CPU Interrupt Priority Level is 1 0000 = CPU Interrupt Priority Level is 0
bit 7-0	**VECNUM<7:0>:** Vector Number of Pending Interrupt bits 11111111 = Interrupt vector pending is Number 263 • • • 00000001 = Interrupt vector pending is Number 9 00000000 = Interrupt vector pending is Number 8

Register 4-11: FALTREG: Alternate Working Register Set Priority Level Register

U-1	U-1	U-1	U-1	U-1	U-1	U-1	U-1
—	—	—	—	—	—	—	—
bit 15							bit 8

U-1	R/W-1	R/W-1	R/W-1	U-1	R/W-1	R/W-1	R/W-1
—	CTXT2<2:0>			—	CTXT1<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 15-7	**Unimplemented:** Read as ‘1’
bit 6-4	**CTXT2<2:0>:** Specifies the Alternate Working Register Set 2 IPL bits `111` = Not used `110` = Priority Level 7 `101` = Priority Level 6 `100` = Priority Level 5 `011` = Priority Level 4 `010` = Priority Level 3 `001` = Priority Level 2 `000` = Priority Level 1
bit 3	**Unimplemented:** Read as ‘1’
bit 2-0	**CTXT1<2:0>:** Specifies the Alternate Working Register Set 1 IPL bits `111` = Not used `110` = Priority Level 7 `101` = Priority Level 6 `100` = Priority Level 5 `011` = Priority Level 4 `010` = Priority Level 3 `001` = Priority Level 2 `000` = Priority Level 1

5.0 INTERRUPT SETUP PROCEDURES

5.1 Initialization

To configure an interrupt source, complete the following steps:

1. If nested interrupts are *not* desired, set the NSTDIS control bit (INTCON1<15>).
2. Select the user application-assigned priority level for the interrupt source by writing to the control bits in the appropriate IPCx control register. The priority level depends on the specific application and type of the interrupt source. If multiple priority levels are not desired, the IPCx register control bits for all enabled interrupt sources can be programmed to the same non-zero value.

Note: At a device Reset, the IPCx registers are initialized such that all user interrupt sources are assigned to Priority Level 4.

3. Clear the interrupt flag status bit associated with the peripheral in the associated IFSx status register.
4. Enable the interrupt source by setting the interrupt enable control bit associated with the source in the appropriate IECx control register.

5.2 Interrupt Service Routine

The method used to declare an ISR and initialize the IVT/AIVT with the correct vector address depends on the programming language (C or Assembly) and the language development toolsuite used to develop the application.

In general, the user application must clear the interrupt flag in the appropriate IFSx register for the source of the interrupt that the ISR handles. Otherwise, the application will re-enter the ISR immediately after it exits the routine. If the ISR is coded in Assembly language, it must be terminated using a `RETFIE` instruction to unstack the saved PC value, SRL value and old CPU priority level.

5.3 Trap Service Routine

A TSR is coded like an ISR, except that the appropriate trap status flag in the INTCON1 register must be cleared to avoid re-entry into the TSR.

5.4 Interrupt Disable

To enable user interrupts, set the GIE bit in the INTCON2 register. To disable interrupts, clear the GIE bit in the INTCON2 register.

Note: Only user interrupts with a priority level of 7 or less can be disabled. Trap sources (Level 8-Level 15) cannot be disabled.

The `DISI` instruction provides a convenient way to disable interrupts of Priority Levels 1-6 for a fixed period of time. Level 7 interrupt sources are not disabled by the `DISI` instruction.

5.5 Code Example

Example 5-1 illustrates the code sequence that enables nested interrupts, sets up Timer1, Timer2, Timer3, Timer4 and Change Notice (CN) peripherals to Priority Levels 2, 5, 6, 3 and 4, respectively. It also illustrates how interrupts can be enabled and disabled using the status register. Sample ISR illustrates interrupt clearing.

Example 5-1: Interrupt Setup Code Example

```
void   enableInterrupts(void)
{
    /* Enable level 1-7 interrupts */
    /* No restoring of previous CPU IPL state performed here */
    INTCON2bits.GIE = 1;

    return;
}

void   disableInterrupts(void)
{
    /* Disable level 1-7 interrupts */
    /* No saving of current CPU IPL setting performed here */

    INTCON2bits.GIE = 0;

    return;
}

void   initInterrupts(void)
{
    /* Interrupt nesting enabled here */
    INTCON1bits.NSTDIS = 0;


    /* Set Timer3 interrupt priority to 6 (level 7 is highest) */
    IPC2bits.T3IP = 6;

    /* Set Timer2 interrupt priority to 5 */
    IPC1bits.T2IP = 5;

    /* Set Change Notice interrupt priority to 4 */
    IPC4bits.CNIP = 4;

    /* Set Timer4 interrupt priority to 3 */
    IPC6bits.T4IP = 3;

    /* Set Timer1 interrupt priority to 2 */
    IPC0bits.T1IP = 2;


    /* Reset Timer1 interrupt flag */
    IFS0bits.T1IF = 0;

    /* Reset Timer2 interrupt flag */
    IFS0bits.T2IF = 0;

    /* Reset Timer3 interrupt flag */
    IFS0bits.T3IF = 0;

    /* Reset Timer4 interrupt flag */
    IFS1bits.T4IF = 0;

    /* Enable CN interrupts */
    IEC1bits.CNIE = 1;
```

Example 5-1: Interrupt Setup Code Example (Continued)

```
    /* Enable Timer1 interrupt */
    IEC0bits.T1IE = 1;

    /* Enable Timer2 interrupt (PWM time base) */
    IEC0bits.T2IE = 1;

    /* Enable Timer3 interrupt */
    IEC0bits.T3IE = 1;

    /* Enable Timer4 interrupt (replacement for Timer2 */
    IEC1bits.T4IE = 1;

    /* Reset change notice interrupt flag */
    IFS1bits.CNIF = 0;


    return;
}


void __attribute__((__interrupt__,no_auto_psv)) _T1Interrupt(void)
{
    /* Insert ISR Code Here*/

    /* Clear Timer1 interrupt */
    IFS0bits.T1IF = 0;
}

void __attribute__((__interrupt__,no_auto_psv)) _T2Interrupt(void)
{
    /* Insert ISR Code Here*/

    /* Clear Timer2 interrupt */
    IFS0bits.T2IF = 0;
}

void __attribute__((__interrupt__,no_auto_psv)) _T3Interrupt(void)
{
    /* Insert ISR Code Here*/

    /* Clear Timer3 interrupt */
    IFS0bits.T3IF = 0;
}

void __attribute__((__interrupt__,no_auto_psv)) _T4Interrupt(void)
{
    /* Insert ISR Code Here*/

    /* Clear Timer4 interrupt */
    IFS1bits.T4IF = 0;
}

void __attribute__((__interrupt__,no_auto_psv)) _CNInterrupt(void)
{
    /* Insert ISR Code Here*/

    /* Clear CN interrupt */
    IFS1bits.CNIF = 0;
}
```

6.0 REGISTER MAP

A summary of the registers associated with the dsPIC33/PIC24 family Interrupts module is provided in Table 6-1.

Table 6-1: Interrupt Controller Register Map(1)

File 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
INTCON1	NSTDIS	OVAERR	OVBERR	COVAERR	COVBERR	OVATE	OVBTE	COVTE	SFTACERR	DIV0ERR	DMACTRAP	MATHERR	ADDRERR	STKERR	OSCFAIL	—	0000
INTCON2	GIE	DISI	SWTRAP	—	—	—	—	AIVTEN	—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP	8000
INTCON3	—	—	—	—	—	—	—	NAE	—	UAE	DAE	DOOVR	—	—	—	APLL	0000
INTCON4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	SGHT	0000
IFSx	IFS15	IFS14	IFS13	IFS12	IFS11	IFS10	IFS9	IFS8	IFS7	IFS6	IFS5	IFS4	IFS3	IFS2	IFS1	IFS0	0000
IECx	IEC15	IEC14	IEC13	IEC12	IEC11	IEC10	IEC9	IEC8	IEC7	IEC6	IEC5	IEC4	IEC3	IEC2	IEC1	IEC0	0000
IPCx	—	IP3<2:0>			—	IP2<2:0>			—	IP1<2:0>			—	IP0<2:0>			4444
INTTREG	—	—	—	—	ILR3	ILR2	ILR1	ILR0	VECNUM7	VECNUM6	VECNUM5	VECNUM4	VECNUM3	VECNUM2	VECNUM1	VECNUM0	0000

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

Note 1: Not all bits are available for all devices. Please refer to the **"Interrupt Controller"** chapter in the specific device data sheet for availability.

7.0 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 product family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Interrupts module are:

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

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

8.0 REVISION HISTORY

Revision A (July 2009)

This is the initial released version of this document.

Revision B (April 2010)

This revision includes the following global updates that occur throughout the document:

- **Bit name changes (see also Table 6-2: Interrupt Controller Register Map):**
  - DMACTRAP is now DMACERR (see Register 6-3)
  - CMPIF is now CMIF (see Register 6-8)
  - PCEPIF is now PSEMIF (see Register 6-10)
  - PCESIF is now PSESMIF (see Register 6-11)
  - CMPIE is now CMIE (see Register 6-17)
  - RTCCIE is now RTCIE and PCEPIE is now PSEMIE (see Register 6-19)
  - PCESIE is now PSESMIE (see Register 6-20)
  - CMPIP is now CMIP (see Register 6-29)
  - PCEPIP is now PSEMIP (see Register 6-39)
  - RTCCIP is now RTCIP (see Register 6-40)
  - PCEPIP is now PSESMIP (see Register 6-43)

Additional updates include the following:

- **Added the following new sections:**
  - **6.2.1.4.1 "USB Address Error Soft Trap (UAE)"**
  - **6.2.1.4.2 "DMA Address Error Soft Trap (DAE)"**
  - **6.2.1.4.3 "DO Stack Overflow Soft Trap (DOOVR)"**)
- **Updated the five `void __attribute` lines of code and removed the watermark from the Interrupt Setup Code example (see Example 6-1)**
- **Updated the following Interrupt Source names in Table 6-1:**
  - CMP1 is now CM (see Interrupt Vector Number 26)
  - PCEP is now PSEM – PWM Special Event Match (see Interrupt Vector Number 65)
  - RTCC is now RTC (see Interrupt Vector Number 70)
  - PCES is now PSESM – PWM Secondary Special Event Match (see Interrupt Vector Number 81)
  - SI2C3 is now Reserved (see Interrupt Vector Number 95)
  - MI2C3 is now Reserved (see Interrupt Vector Number 96)
  - USBOIG is now USB1 (see Interrupt Vector Number 97)
  - PCG1-PCG7 are now PWM1 through PWM7 (see Interrupt Vector Numbers 105-111)
- **Register additions and deletions (see also Table 6-2: Interrupt Controller Register Map):**
  - Added a definition for the VAR bit in Register 6-2
  - Added a shaded note to clarify hard trap operation in **6.2.2.4 "Generic Hard Trap"**
  - Changed the default POR value for the GIE bit from '`0`' to '`1`' in the Interrupt Control Register 2 (see Register 6-4)
  - Removed the FLT1IF bit from IFS3 (see Register 6-10)
  - Added the CRCIF bit and removed the FLT4IF, FLT3IF, and FLT2IF bits from IFS4 (see Register 6-11)
  - Removed the MI2C3IF and SI2C3IF bits from IFS5 (see Register 6-12)
  - Removed the PWM9IF and PWM7IF bits from IFS6 (see Register 6-13)
  - Added the ICDIF and DMA12IF bits to IFS8 (see Register 6-15)
  - Removed the FLT1IE bit from IEC3 (see Register 6-19)
  - Removed the FLT4IE and FLT3IE bits from IEC4 (see Register 6-20)
  - Removed the PWM9IE and PWM8IE bits from IEC6 (see Register 6-22)

Revision B (April 2010) (Continued)

- Register additions and deletions (Continued) (see also Table 6-2: Interrupt Controller Register Map):
  - Added the ICDIE and DMA12IE bits to IEC8 (see Register 6-24)
  - Removed the FLT1IP<2:0> bits from IPC15 (see Register 6-40)
  - Removed the FLT2IP<2:0> bits from IPC16 (see Register 6-41)
  - Removed the FLT3IP<2:0> bits from IPC18 (see Register 6-43)
  - Removed IPC19 (was Register 6-44)
  - Removed the MI2C3IP<2:0> and SI2C3IP<2:0> bits from IPC21 (see Register 6-45)
  - Added the ICDIP<2:0> bits to IPC35 (see Register 6-56)
  - Updated the VECNUM bits from <6:0> to <7:0> in INTTREG (see Register 6-57)

Revision C (February 2012)

This revision includes the following updates:

- Added a paragraph describing the INTTREG register in **Section 4.0 "Interrupt Control and Status Registers"**
- Removed the Interrupt Vector Details (Table 6-1)
- Removed 6.4.1 "Assignment of Interrupts to Control Registers"
- Reformatted the IFSx, IECx, and IPCx registers (see Register 4-7 through Register 4-9)
- Added the CPU IRQ to the interrupt timing diagrams (see Figure 3-1 through Figure 3-3)
- Additional minor corrections such as language and formatting updates were incorporated throughout the document

Revision D (October 2013)

This revision includes the following updates:

- Applied new template format
- Added note to Figure 1-1
- Added Alternate Interrupt Vector Table (Figure 1-2)
- Added **Section 1.2 "Alternate Interrupt Vector Table (AIVT)"**
- Added **Figure 2-2: "Nested Interrupt Sequence"**
- Added Example 2-1
- Added AIVTEN, NAE, APPL and FALTREG bit descriptions
- Additional minor corrections such as language and formatting updates were incorporated throughout the document

ISBN: 978-1-62077-569-1

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

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

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

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

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

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

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

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

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

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

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

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

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

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

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

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

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

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

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/20/13