CP316
Interrupts

Terry Sturtevant

Wilfrid Laurier University

February 6, 2018

Introduction to Interrupts

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

where **interrupt service routines** are located.

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

where **interrupt service routines** are located.

After interrupt service, the program control returns to original.

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

where **interrupt service routines** are located.

After interrupt service, the program control returns to original.

The PIC has 2 interrupt **priority levels**,

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

where **interrupt service routines** are located.

After interrupt service, the program control returns to original.

The PIC has 2 interrupt **priority levels**,

high and low.

Introduction to Interrupts

Interrupts allow program control to change due to *events*.

Interrupt vectors are program locations

where **interrupt service routines** are located.

After interrupt service, the program control returns to original.

The PIC has 2 interrupt **priority levels**,

high and low.

A high priority interrupt can happen *during* a low priority interrupt, but not the other way around.

Terminology

Terminology

- **Enabled/Disabled**

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred
- **Flag**

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred
- **Flag**

  bit in memory indicating an interrupt is pending

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred
- **Flag**

  bit in memory indicating an interrupt is pending

  Note: Flags are usually set whether or not corresponding interrupts are enabled.

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred
- **Flag**

  bit in memory indicating an interrupt is pending

  Note: Flags are usually set whether or not corresponding interrupts are enabled.
- **Set** or **Clear**

Terminology

- **Enabled/Disabled**

  whether a specific event will generate an interrupt
- **Pending**

  whether the event has occurred
- **Flag**

  bit in memory indicating an interrupt is pending

  Note: Flags are usually set whether or not corresponding interrupts are enabled.
- **Set** or **Clear**

  state of flag indicating pending or not

Interrupts

Interrupts

overview

Interrupts

overview

→ Section 6.2.5

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

→ Section 6.2.4

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

→ Section 6.2.4

status; clear, pending

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

→ Section 6.2.4

status; clear, pending

→ Sections 6.2.3

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

→ Section 6.2.4

status; clear, pending

→ Sections 6.2.3

bits; enable, priority, flag

Interrupts

overview

→ Section 6.2.5

→ Section 8.0

vectors

→ Section 6.2.6

priority

→ Section 6.2.4

status; clear, pending

→ Sections 6.2.3

bits; enable, priority, flag

→ Sections 6.4.2

Here's a code fragment from a typical program:

Here's a code fragment from a typical program:

```
;;;  vectors
   org    0x0000     ;reset vector
   goto   start      ;beginning of program

   org    0x0008     ;high priority int. vector
   goto   high_ISR   ;program memory label
;   goto $           ;none for now

   org    0x0018     ;low priority int. vector
   goto   $          ;none for now
;   goto low_ISR
```

Interrupt Initialization

Interrupt Initialization

The *sequence* of initializing interrupts is important.

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.
4. enable (various registers)

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.
4. enable (various registers)
   for each individual source required

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.
4. enable (various registers)
   for each individual source required
5. **global** enable *all* interrupts (**INTCON**)

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.
4. enable (various registers)
   for each individual source required
5. **global** enable *all* interrupts (**INTCON**)
   after all individual sources have been initialized

Interrupt Initialization

The *sequence* of initializing interrupts is important.

1. enable priority (**RCON**)
   unless you need compatibility with older devices
2. configure individual interrupt sources (various registers)
   any needed configuration, initialization of variables, etc.
3. clear flags (various registers)
   On reset, many interrupt flags are in an unknown state, so interrupts may be incorrectly identified as pending.
4. enable (various registers)
   for each individual source required
5. **global** enable *all* interrupts (**INTCON**)
   after all individual sources have been initialized
   needed for *any* interrupt service to occur

Example: Timer 0

Example: Timer 0

1. enable priority (**RCON**)

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

   TMR0IE is in INTCON

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

   TMR0IE is in INTCON

   for each individual source required

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

   TMR0IE is in INTCON

   for each individual source required

   GIEL is in INTCON

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

   TMR0IE is in INTCON

   for each individual source required

   GIEL is in INTCON
5. **global** enable *all* interrupts (**INTCON**)

Example: Timer 0

1. enable priority (**RCON**)
2. configure individual interrupt sources (various registers)

   TMR0IP is in INTCON2
3. clear flags (various registers)

   TMR0IF is in INTCON
4. enable (various registers)

   TMR0IE is in INTCON

   for each individual source required

   GIEL is in INTCON
5. **global** enable *all* interrupts (**INTCON**)

   GIEH is in INTCON

Interrupt Service Routines

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

   STATUS, BSR, WREG

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

   STATUS, BSR, WREG

2. confirm source

   Check specific flag bit to confirm that the expected source caused the interrupt.

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

   STATUS, BSR, WREG

2. confirm source

   Check specific flag bit to confirm that the expected source caused the interrupt.

3. process

Interrupt Service Routines

The sequence of actions in *servicing* interrupts is also important.

1. save vital registers

   STATUS, BSR, WREG

2. confirm source

   Check specific flag bit to confirm that the expected source caused the interrupt.

3. process

   more on this later

Interrupt Service Routines (continued)

Interrupt Service Routines (continued)

3. process

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

   Some flags need to be cleared explicitly; others will happen as part of normal service.

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

   Some flags need to be cleared explicitly; others will happen as part of normal service.

   Always check whether a flag will be cleared automatically or not.

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

   Some flags need to be cleared explicitly; others will happen as part of normal service.

   Always check whether a flag will be cleared automatically or not.
5. restore vital registers

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

   Some flags need to be cleared explicitly; others will happen as part of normal service.

   Always check whether a flag will be cleared automatically or not.
5. restore vital registers

   STATUS, BSR, WREG

Interrupt Service Routines (continued)

3. process
4. clear flags (if required)

   Some flags need to be cleared explicitly; others will happen as part of normal service.

   Always check whether a flag will be cleared automatically or not.
5. restore vital registers

   STATUS, BSR, WREG
6. **retfie**

   like a subroutine return, but re-enables interrupts

Example: Timer 0

Example: Timer 0

1. save vital registers

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG

2. confirm source

   test TMR0IF in INTCON

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do
4. clear flags (if required)

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do
4. clear flags (if required)

   clear TMR0IF in INTCON

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do
4. clear flags (if required)

   clear TMR0IF in INTCON
5. restore vital registers

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do
4. clear flags (if required)

   clear TMR0IF in INTCON
5. restore vital registers

   STATUS, BSR, WREG

Example: Timer 0

1. save vital registers

   STATUS, BSR, WREG
2. confirm source

   test TMR0IF in INTCON
3. process

   whatever you need to do
4. clear flags (if required)

   clear TMR0IF in INTCON
5. restore vital registers

   STATUS, BSR, WREG
6. **retfie**

RETURN instruction

RETURN instruction

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged

What does *s* indicate?

RETFIE instruction

RETFIE instruction

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, 1 → GIE/GIEH or PEIE/GIEL, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged.

Note the difference from a normal *RETURN*

Interrupt Resources

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

→ Section 4.3

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

→ Section 4.3

automatically copies STATUS, BSR, WREG when an interrupt occurs

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

→ Section 4.3

automatically copies STATUS, BSR, WREG when an interrupt occurs

retfie FAST

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

→ Section 4.3

automatically copies STATUS, BSR, WREG when an interrupt occurs

retfie FAST

automatically restores STATUS, BSR, WREG on return

Interrupt Resources

The PIC has a set of internal registers to help with interrupt service.

shadow registers, aka fast register stack

→ Section 4.7.1, 4.7.5

→ Section 4.3

automatically copies STATUS, BSR, WREG when an interrupt occurs

retfie FAST

automatically restores STATUS, BSR, WREG on return

Why should you only use *retfie FAST* for *high* priority interrupts?

RETURN instruction

RETURN instruction

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged

s indicates the *FAST* option

RETFIE instruction

RETFIE instruction

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, 1 → GIE/GIEH or PEIE/GIEL, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged.

Why does it set *GIE/GIEH* **or** *PEIE/GIEL*?

RETLW instruction

RETLW instruction

RETLW	Return Literal to W
Syntax:	[*label*] RETLW k
Operands:	$0 \le k \le 255$
Operation:	$k \rightarrow W$, (TOS) $\rightarrow$ PC, PCLATU, PCLATH are unchanged

Why is *FAST* not an option?

Rules for Interrupts

Rules for Interrupts

- Never wait in interrupts.

Rules for Interrupts

- Never wait in interrupts.

  set flags, have waiting in main program

Rules for Interrupts

- Never wait in interrupts.

  set flags, have waiting in main program
- On PIC, make timer high priority.

Rules for Interrupts

- Never wait in interrupts.

  set flags, have waiting in main program
- On PIC, make timer high priority.

  see timer overflow code

Rules for Interrupts

- Never wait in interrupts.

  set flags, have waiting in main program
- On PIC, make timer high priority.

  see timer overflow code
- Use interrupts for events of very short duration

Rules for Interrupts

- Never wait in interrupts.

  set flags, have waiting in main program
- On PIC, make timer high priority.

  see timer overflow code
- Use interrupts for events of very short duration

  i.e. if polling may miss them entirely

Interrupt Sources

Interrupt Sources

internal, external

Interrupt Sources

internal, external

→ Chapter 8 (Figure 8-1)

Interrupt Sources

internal, external

→ Chapter 8 (Figure 8-1)

portB, timers, serial, ADC, etc.

Interrupt Sources

internal, external

→ Chapter 8 (Figure 8-1)

portB, timers, serial, ADC, etc.

→ Section 6.5.6

Interrupt Sources

internal, external

→ Chapter 8 (Figure 8-1)

portB, timers, serial, ADC, etc.

→ Section 6.5.6

special case: INT0

Interrupt Sources

internal, external

→ Chapter 8 (Figure 8-1)

portB, timers, serial, ADC, etc.

→ Section 6.5.6

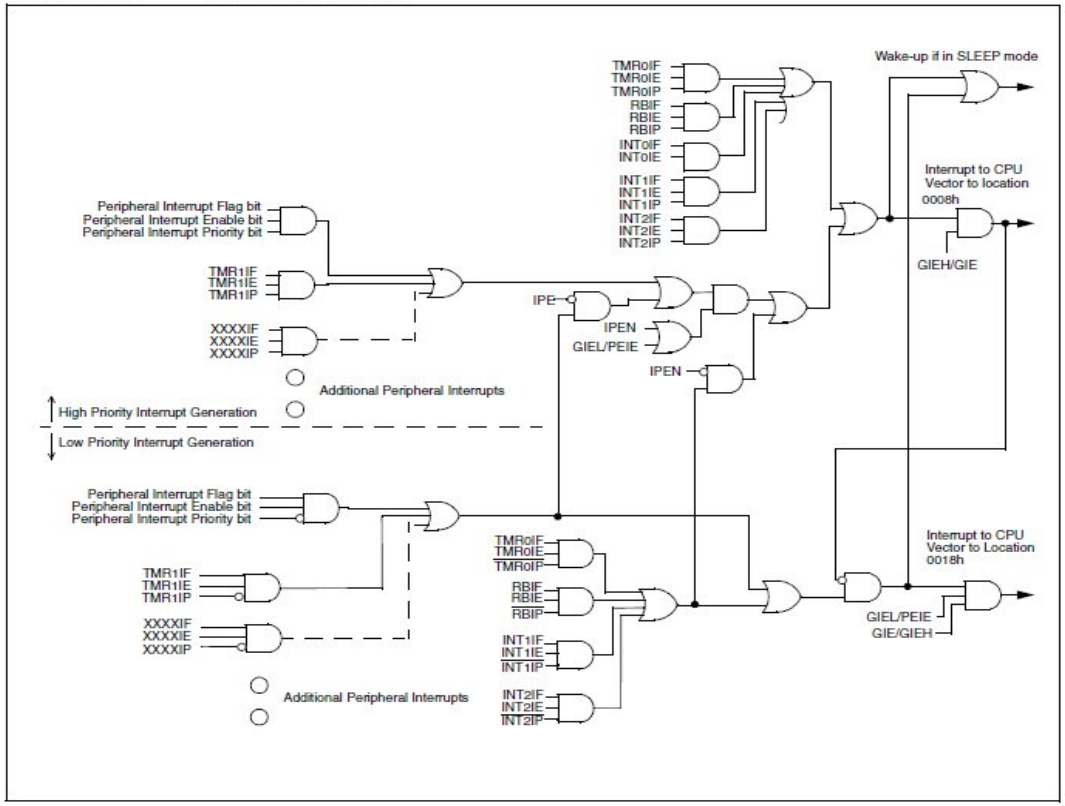
special case: INT0

→ Section 8.6

PIC interrupts

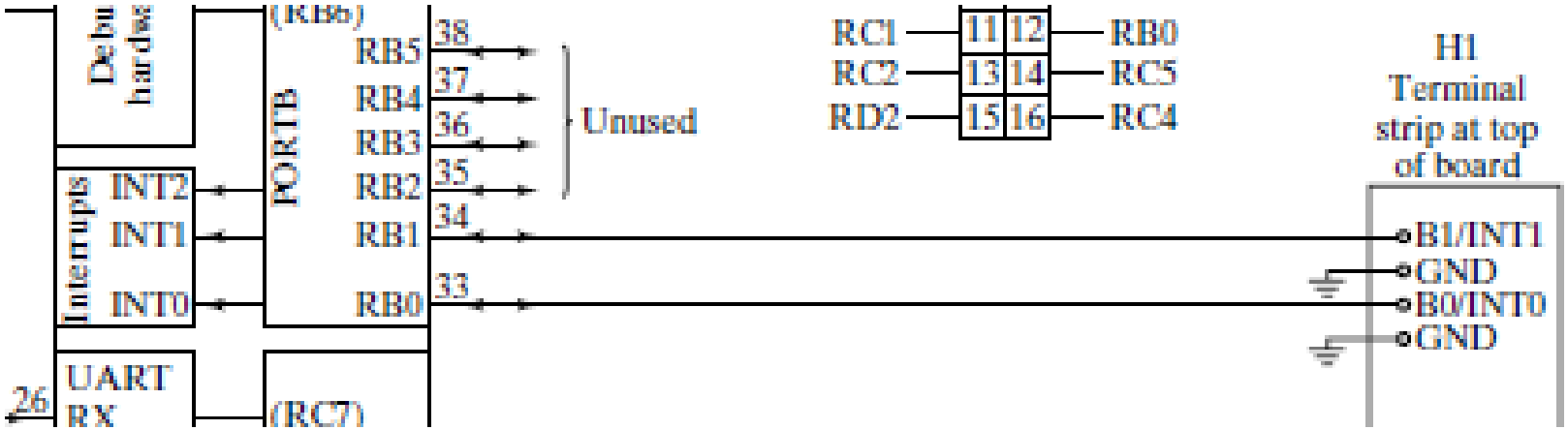
PIC interrupts

FIGURE 8-1: INTERRUPT LOGIC



Qwikflash interrupt connections

Qwikflash interrupt connections



Interrupt Registers

Interrupt Registers

RCON

Interrupt Registers

RCON

→ Section 6.4.3

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

→ Section 6.4.5

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

→ Section 6.4.5

PIE1, **PIE2**

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

→ Section 6.4.5

PIE1, **PIE2**

→ Section 6.4.6

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

→ Section 6.4.5

PIE1, **PIE2**

→ Section 6.4.6

IPR1, **IPR2**

Interrupt Registers

RCON

→ Section 6.4.3

INTCON, **INTCON2**, **INTCON3**

→ Section 6.4.4

PIR1, **PIR2**

→ Section 6.4.5

PIE1, **PIE2**

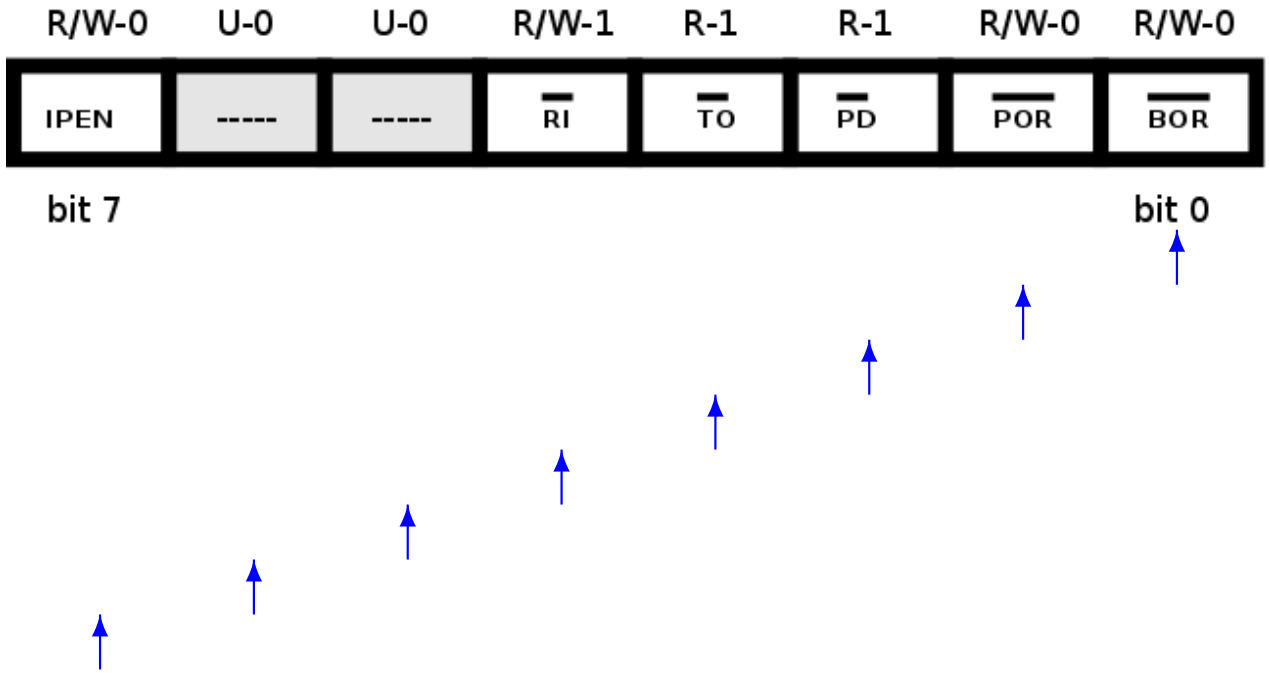
→ Section 6.4.6

IPR1, **IPR2**

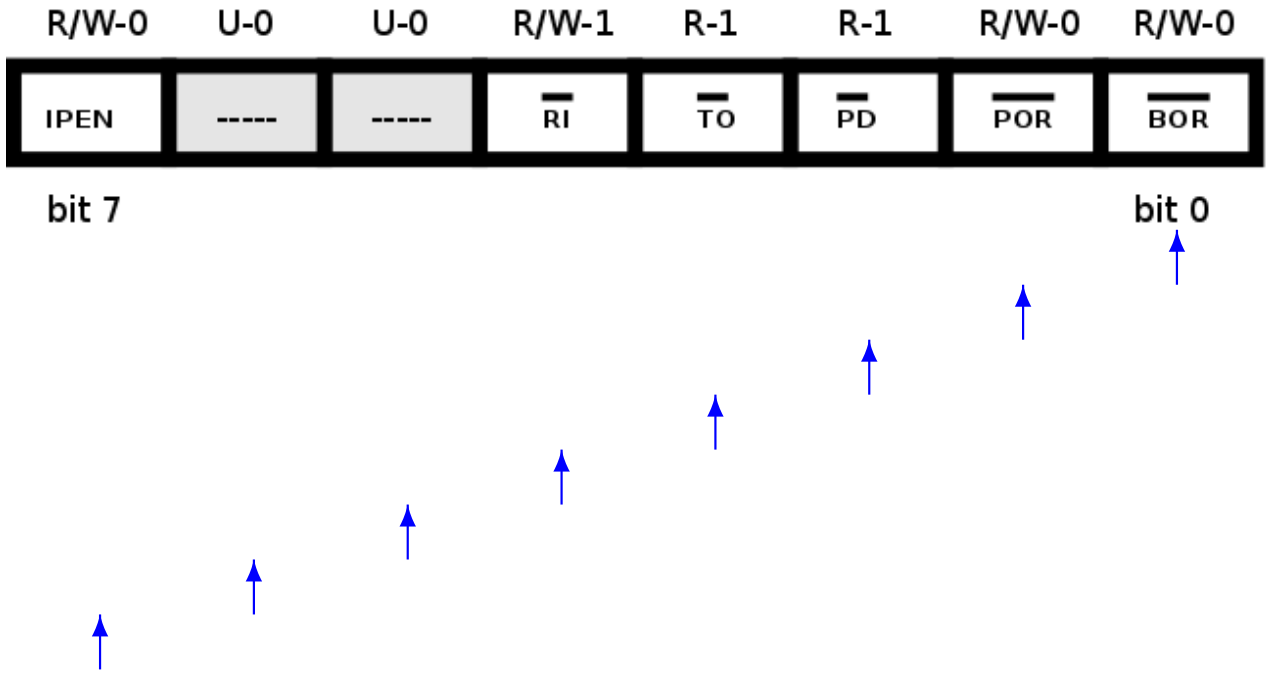
→ Section 6.4.7

RCON

RCON

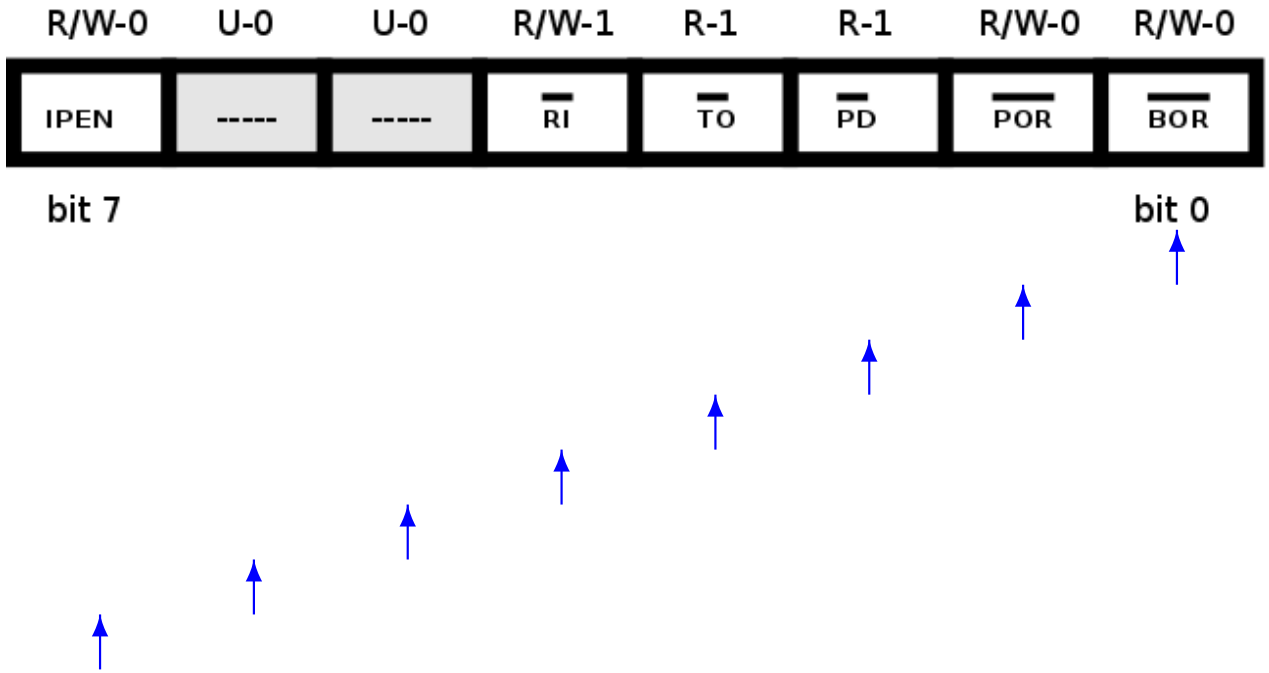


RCON



Bits in RCON register

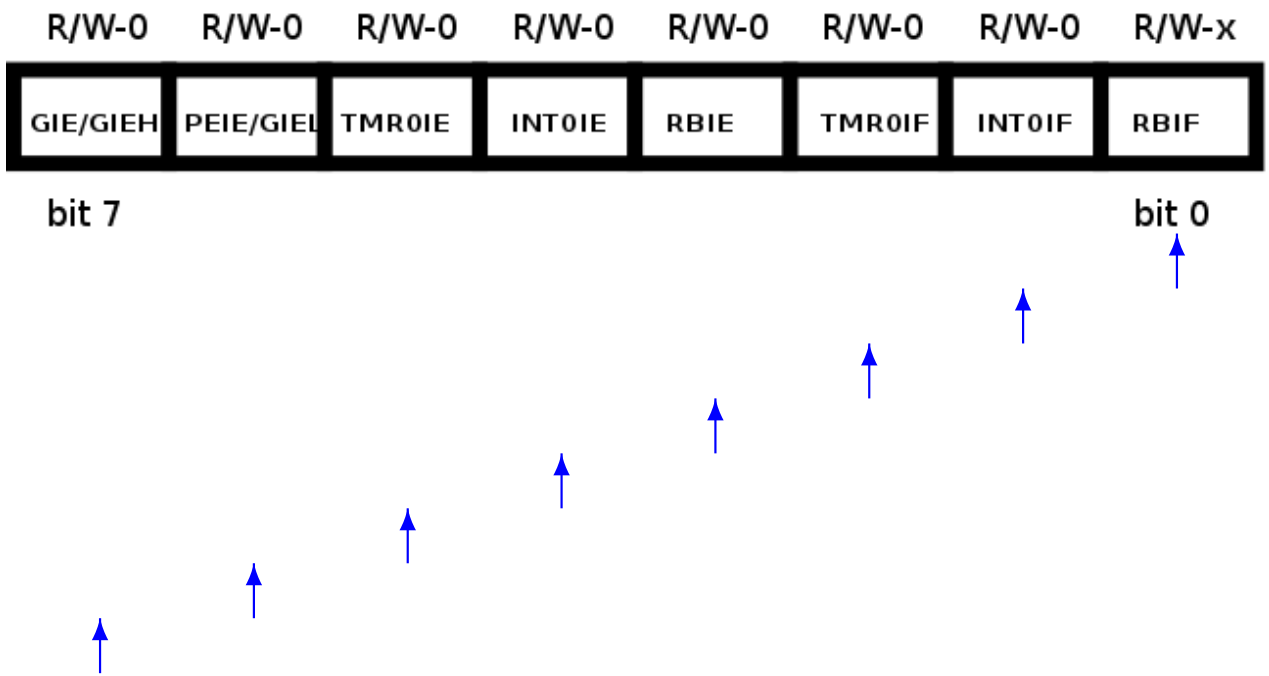
RCON



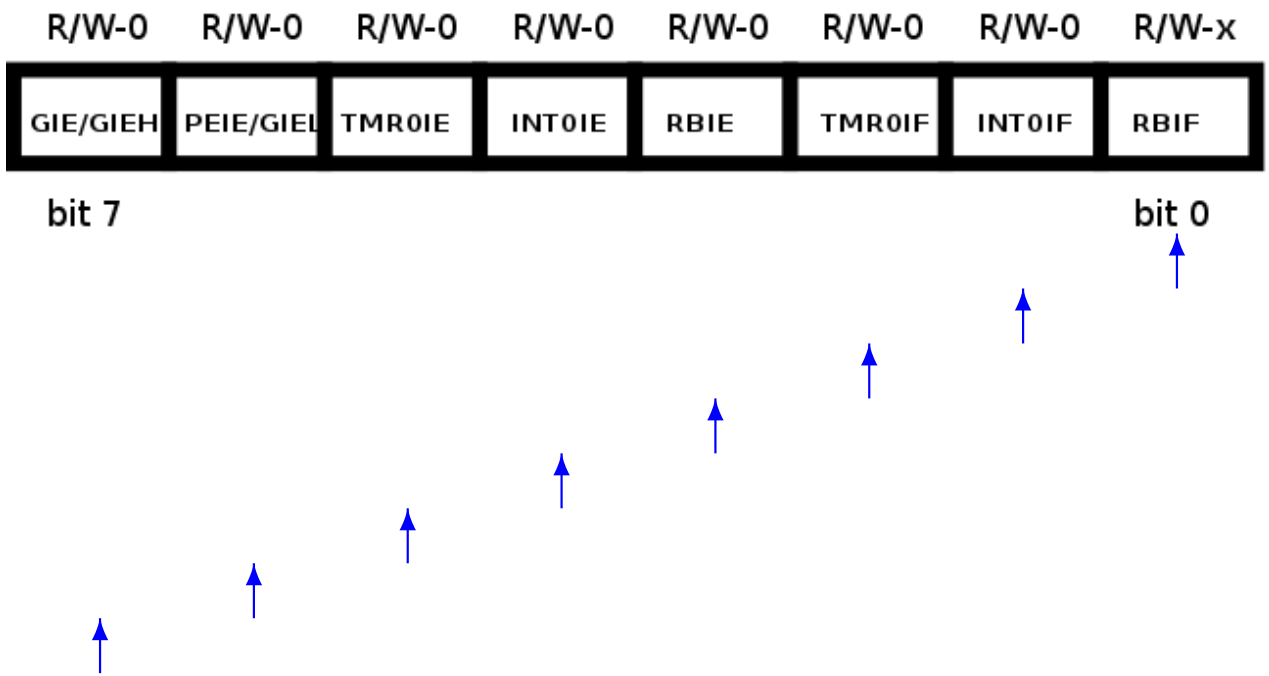
Bits in RCON register -Set IPEN for priority

INTCON

INTCON

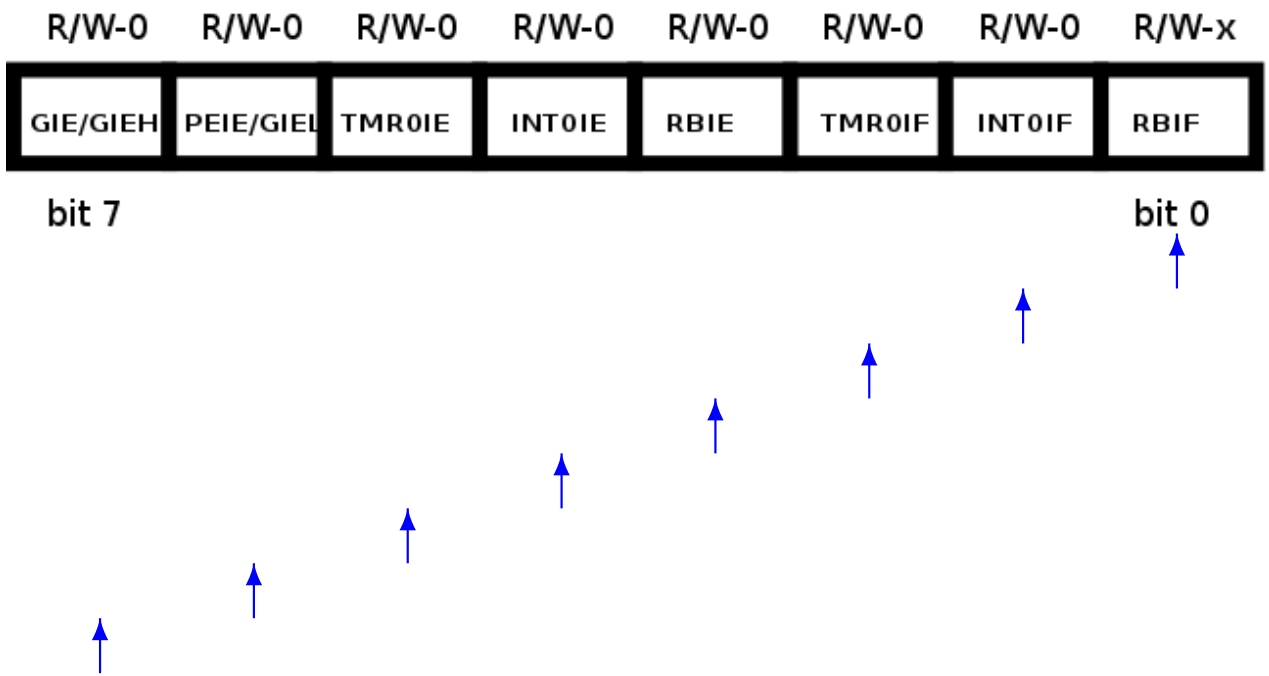

R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-x
GIE/GIEH
PEIE/GIEL
TMR0IE
INT0IE
RBIE
TMR0IF
INT0IF
RBIF
bit 7
bit 0

INTCON



Bits in INTCON register

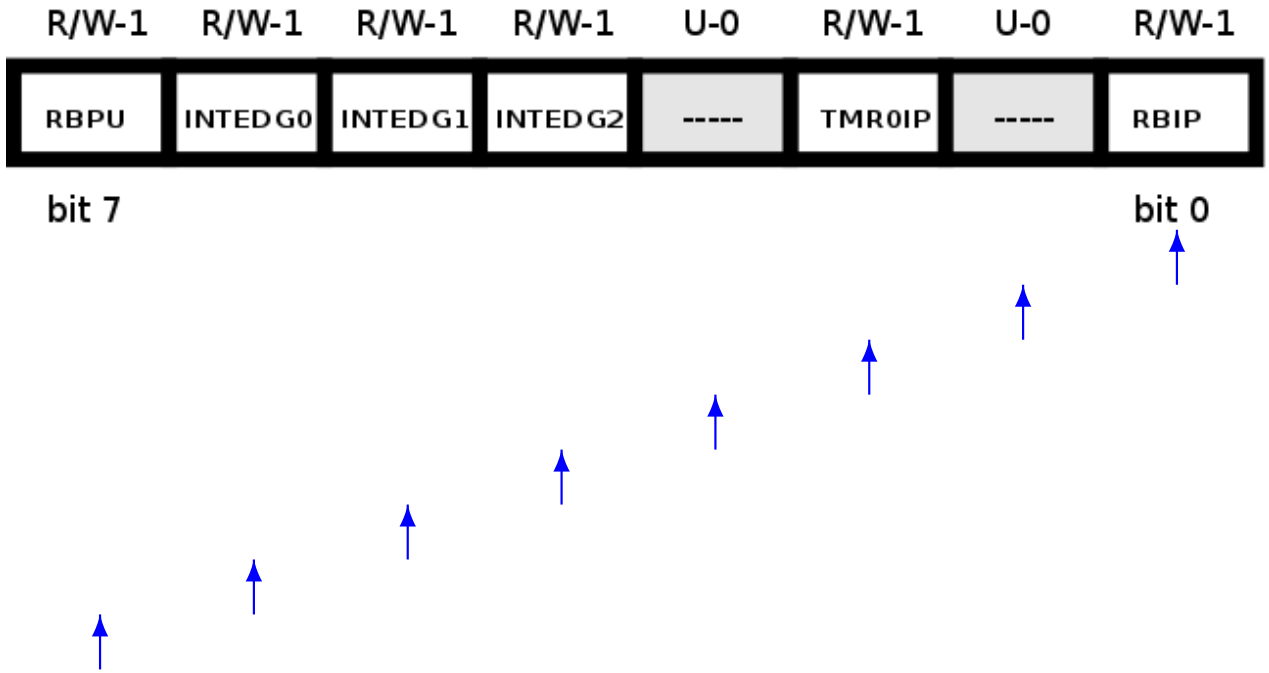
INTCON



Bits in INTCON register -Set GIEH to enable high priority, GIEL to enable low priority (if GIEH set)

INTCON2

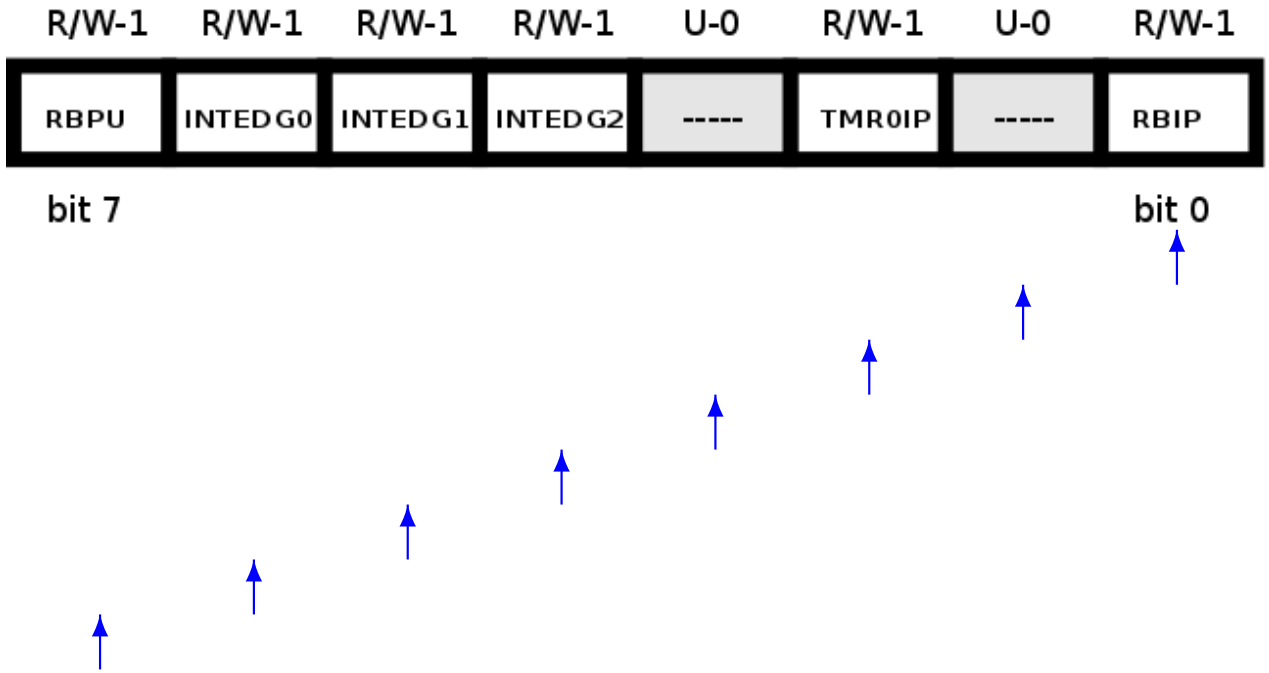
INTCON2


R/W-1
R/W-1
R/W-1
R/W-1
U-0
R/W-1
U-0
R/W-1
RBPU
INTEDG0
INTEDG1
INTEDG2

TMR0IP

RBIP
bit 7
bit 0

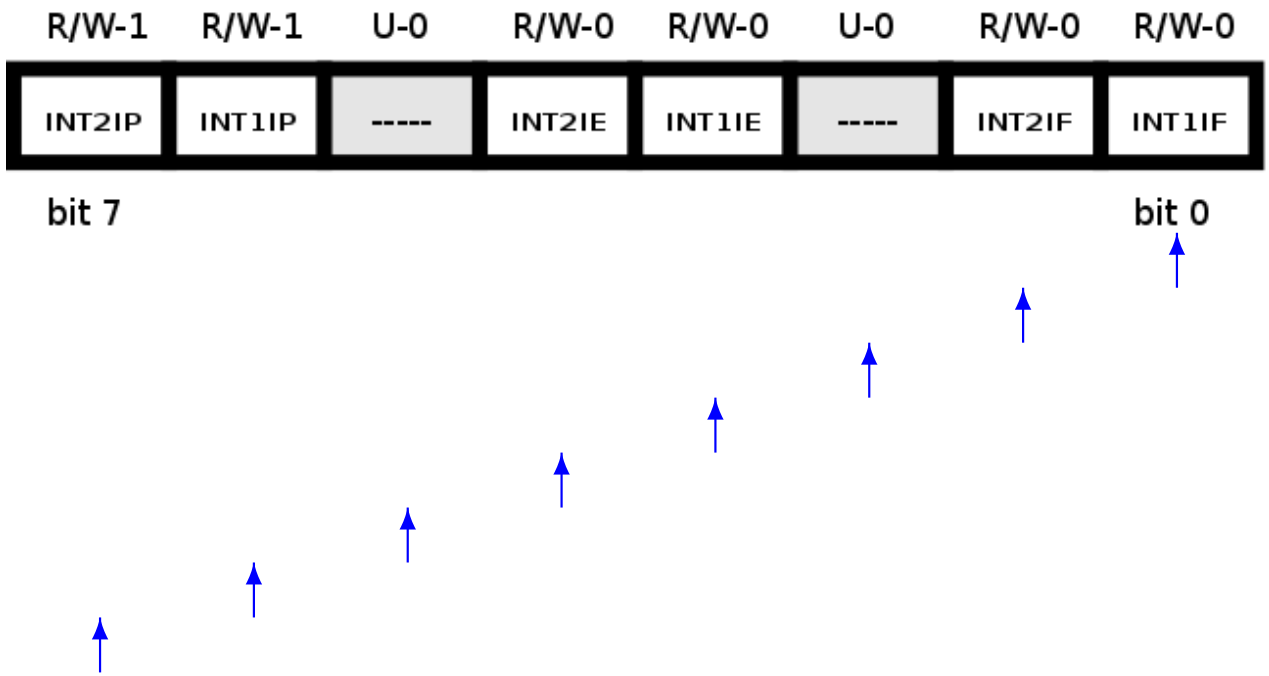
INTCON2



Bits in INTCON2 register

INTCON3

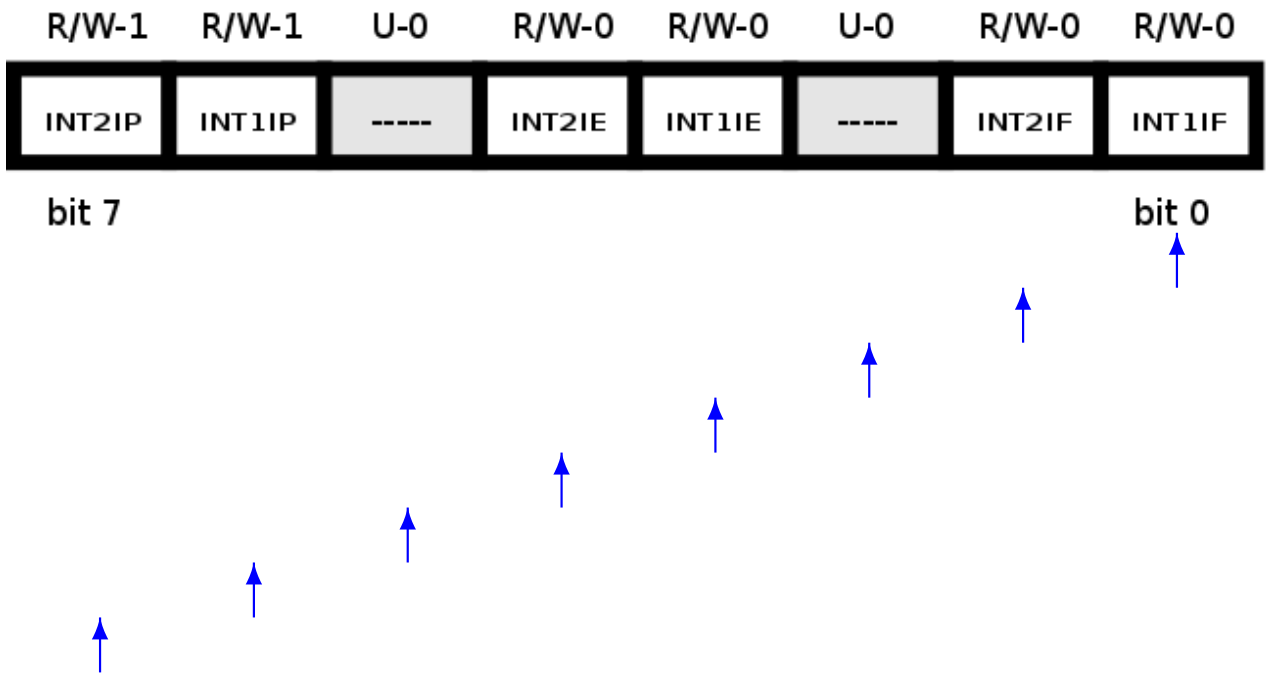
INTCON3


R/W-1
R/W-1
U-0
R/W-0
R/W-0
U-0
R/W-0
R/W-0
INT2IP
INT1IP

INT2IE
INT1IE

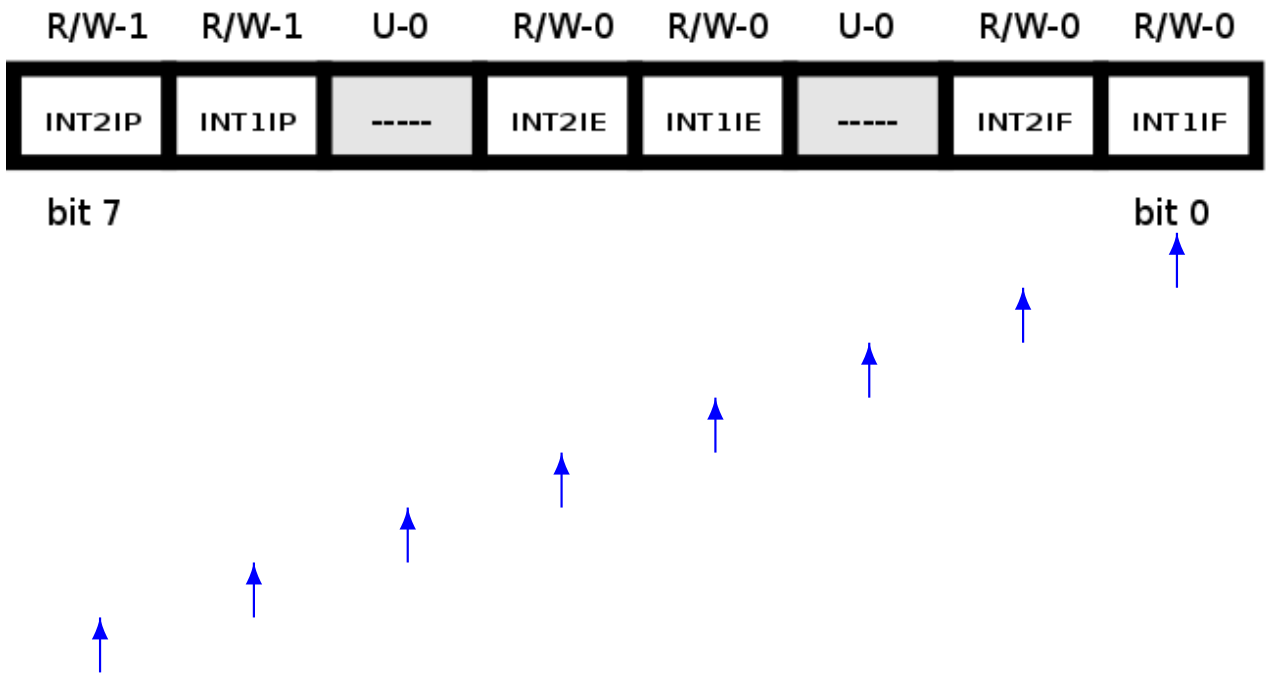
INT2IF
INT1IF
bit 7
bit 0

INTCON3



Bits in INTCON3 register

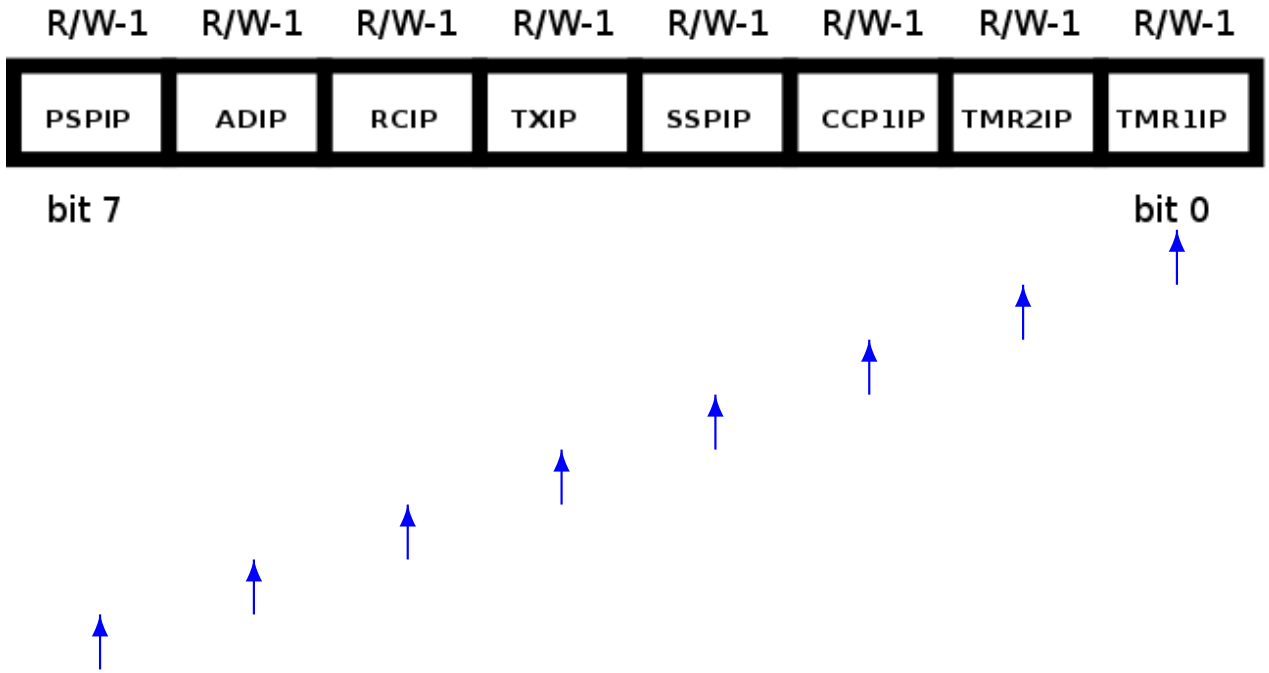
INTCON3



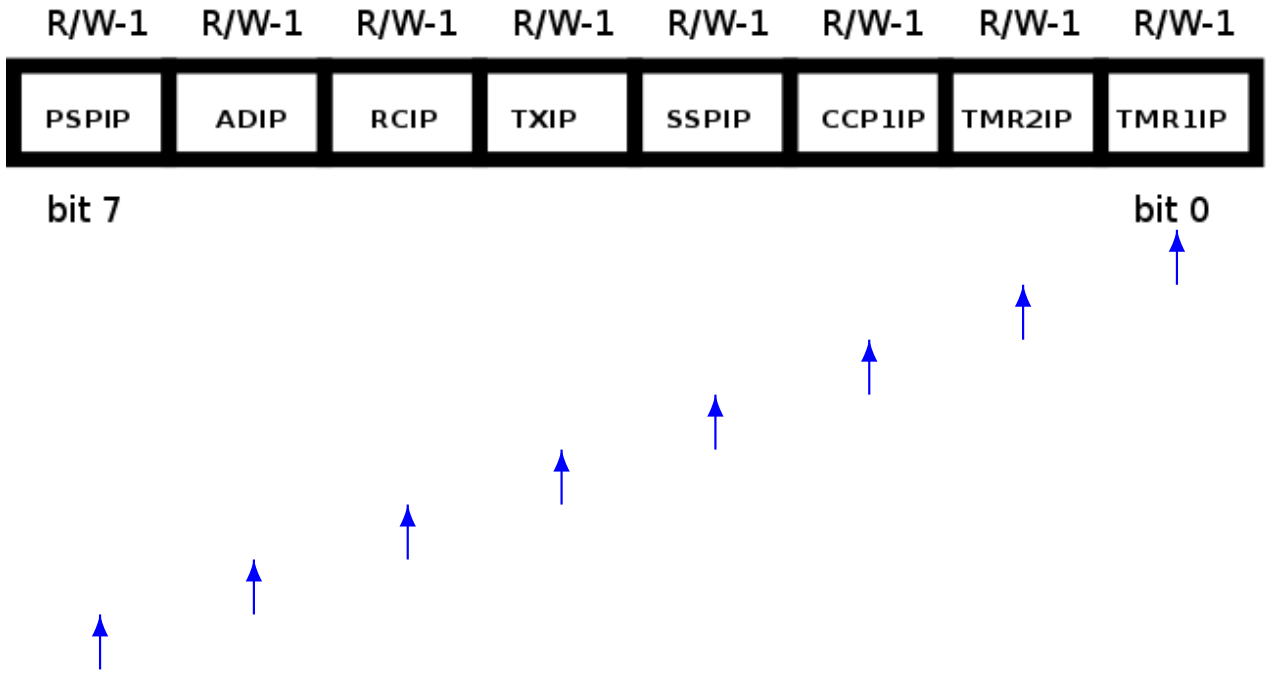
Bits in INTCON3 register -Note INT0 is not mentioned

IPR1

IPR1


R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
PSPIP
ADIP
RCIP
TXIP
SSPIP
CCP1IP
TMR2IP
TMR1IP
bit 7
bit 0

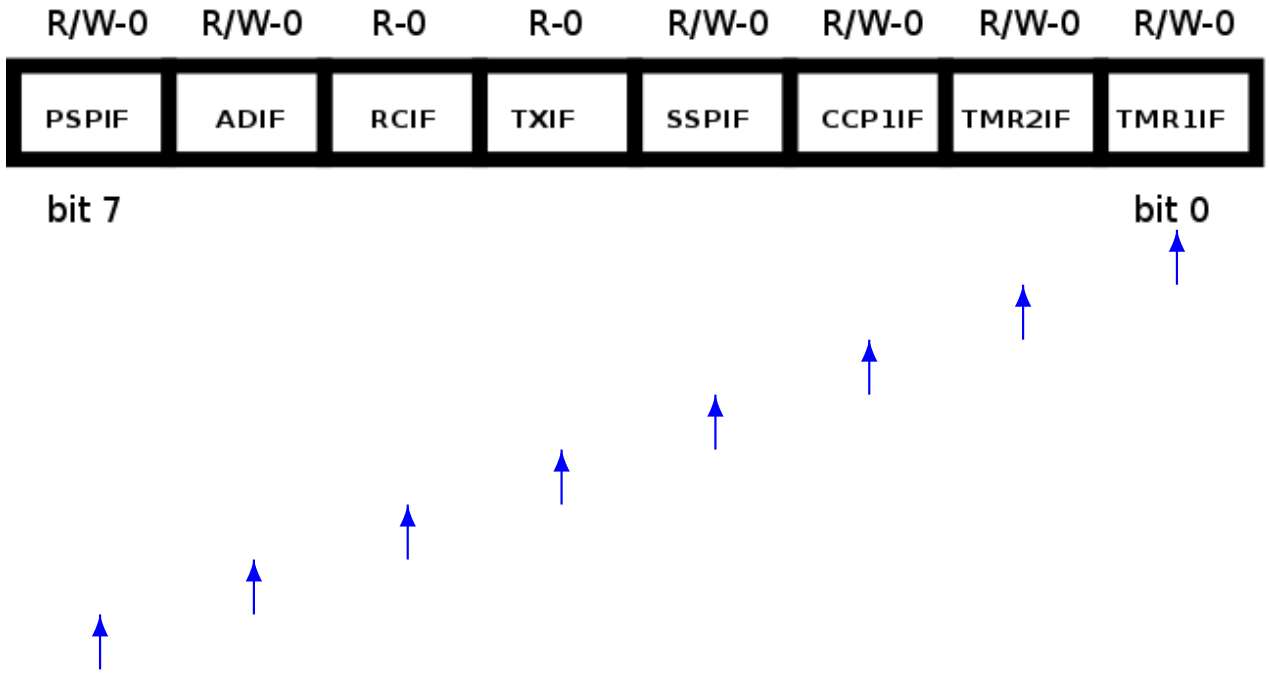
IPR1



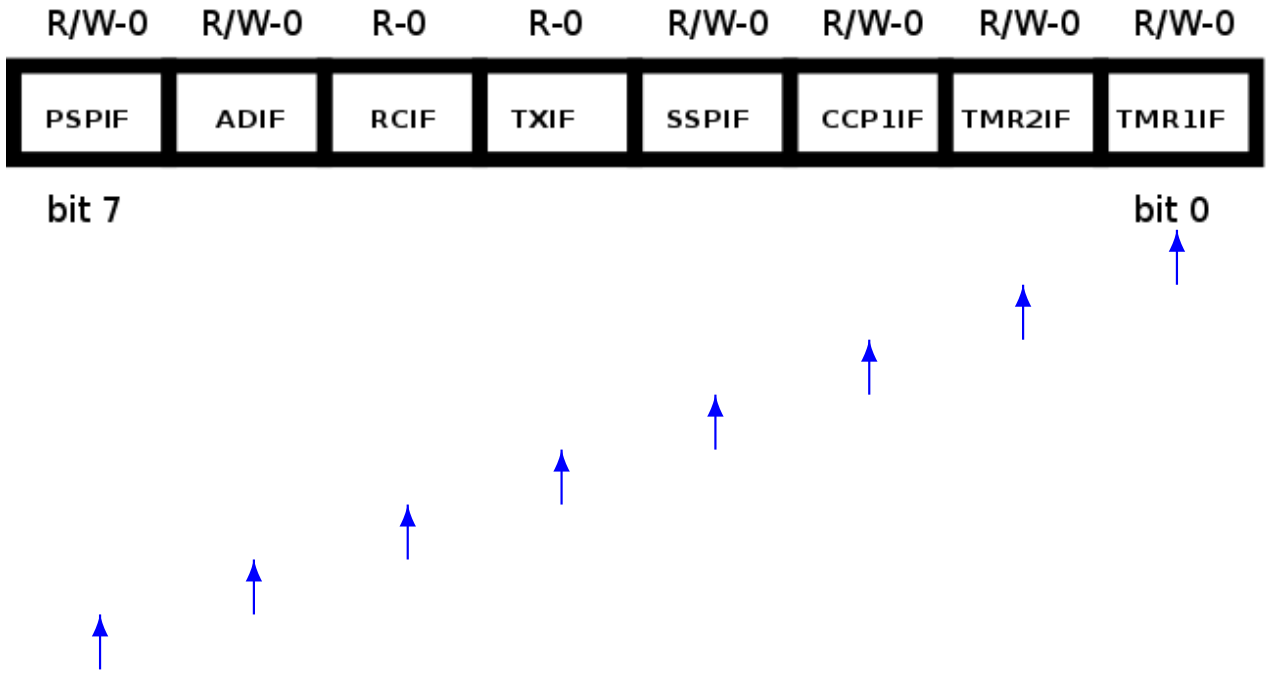
Bits in IPR1 register

PIR1

PIR1


R/W-0
R/W-0
R-0
R-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIF
ADIF
RCIF
TXIF
SSPIF
CCP1IF
TMR2IF
TMR1IF
bit 7
bit 0

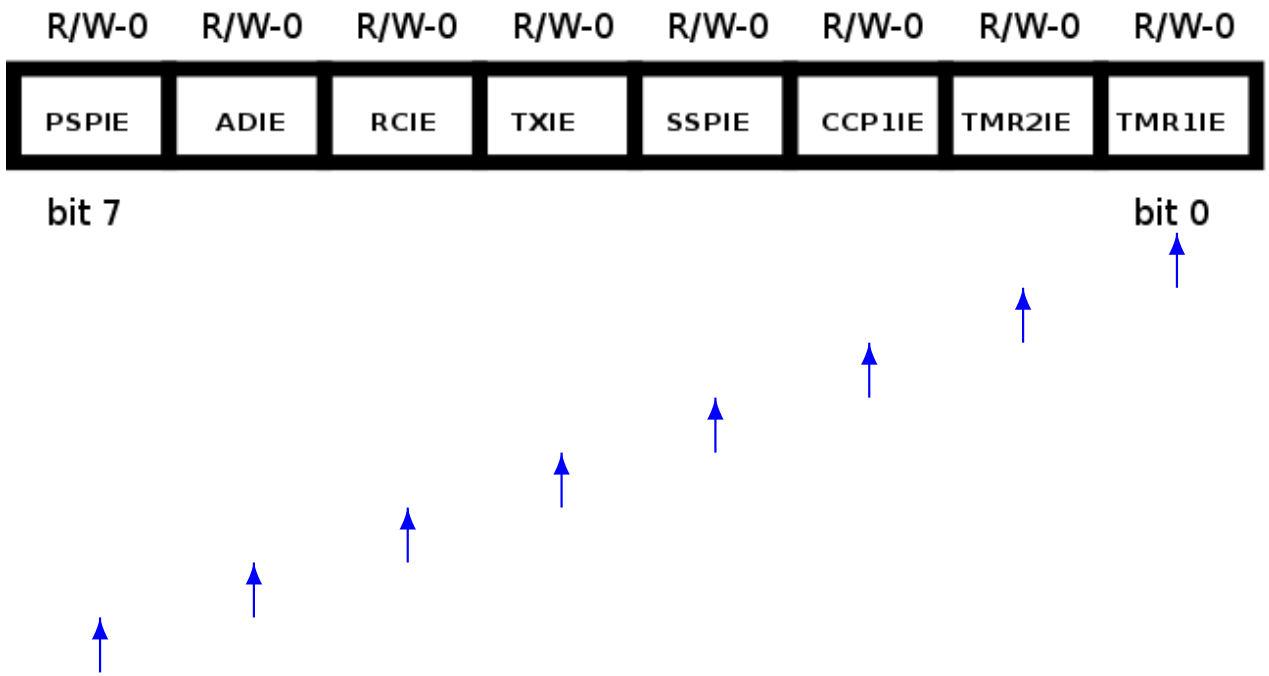
PIR1



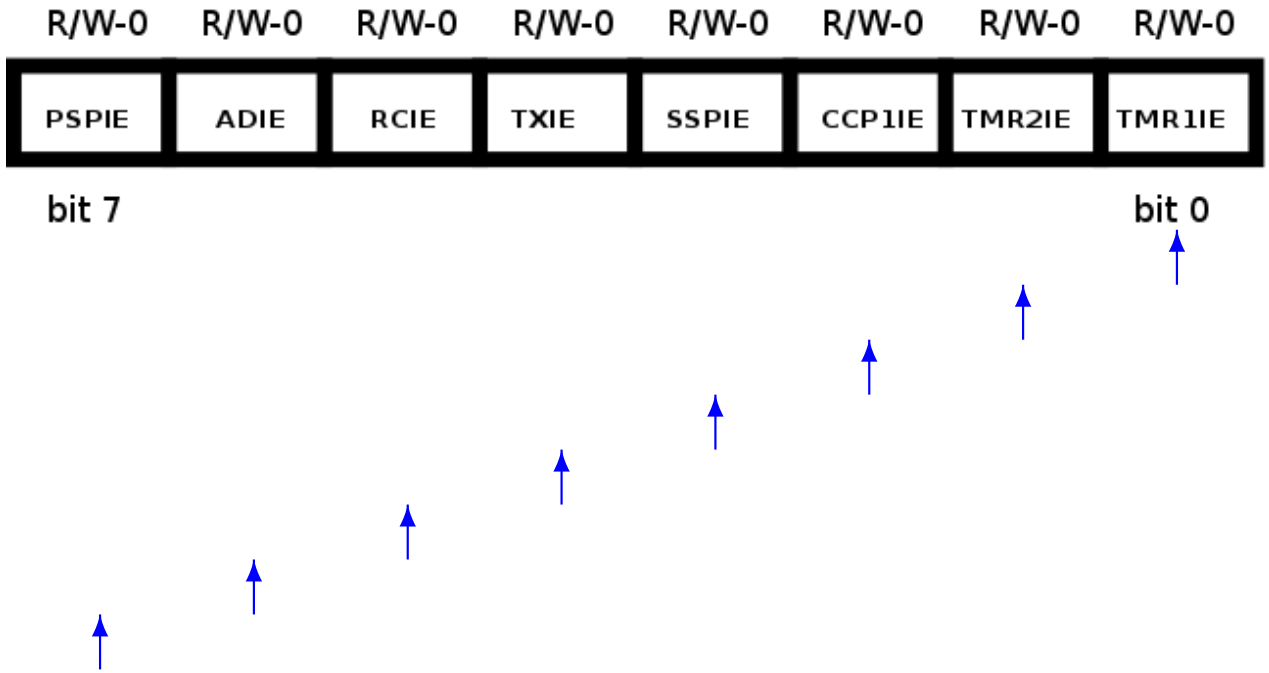
Bits in PIR1 register

PIE1

PIE1


R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIE
ADIE
RCIE
TXIE
SSPIE
CCP1IE
TMR2IE
TMR1IE
bit 7
bit 0

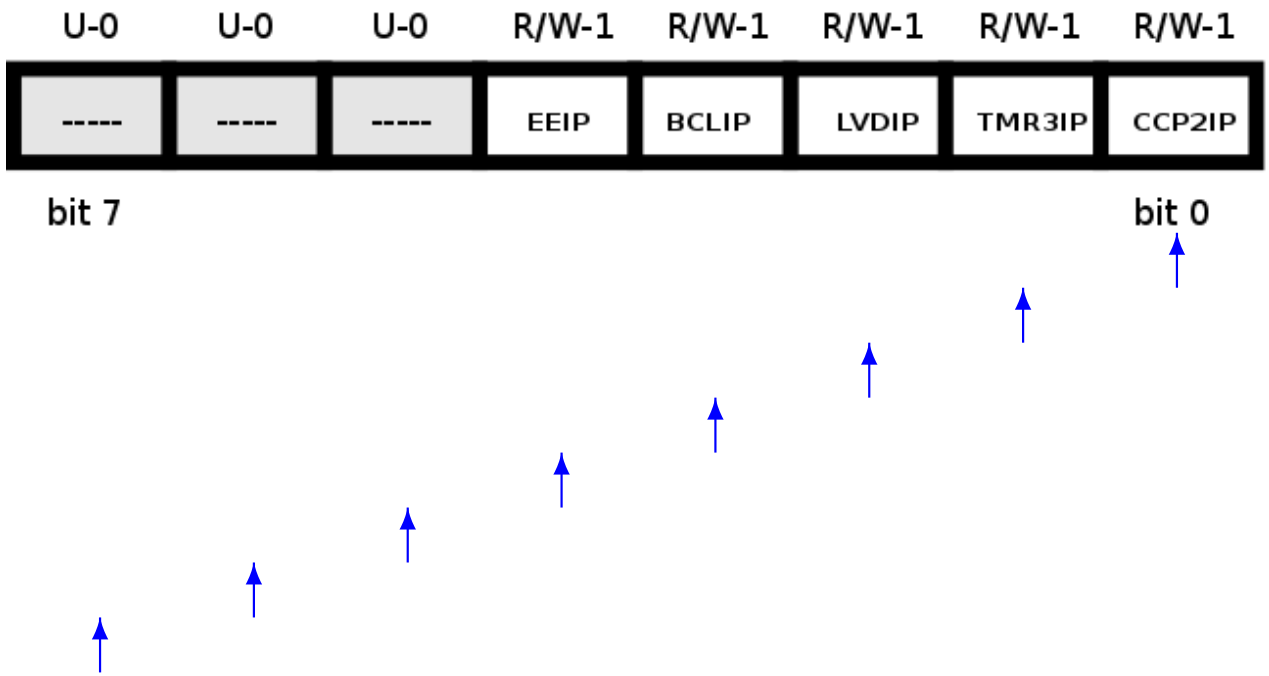
PIE1



Bits in PIE1 register

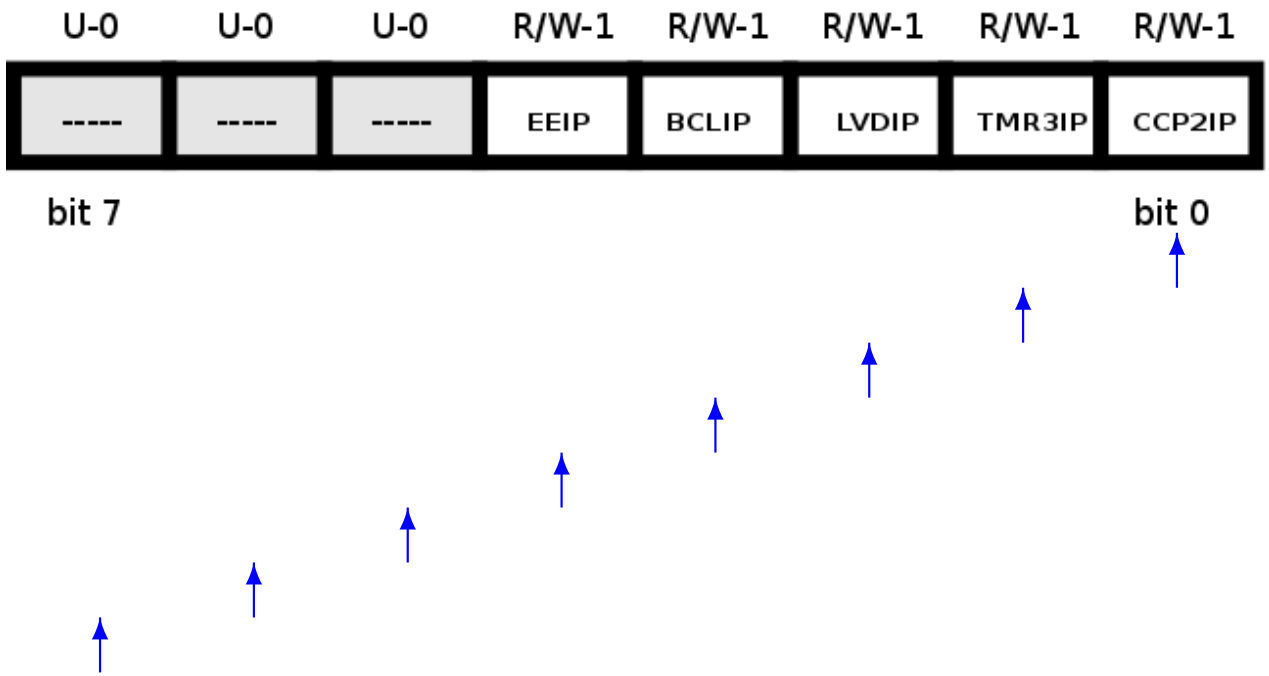
IPR2

IPR2


U-0
U-0
U-0
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1

EEIP
BCLIP
LVDIP
TMR3IP
CCP2IP
bit 7
bit 0

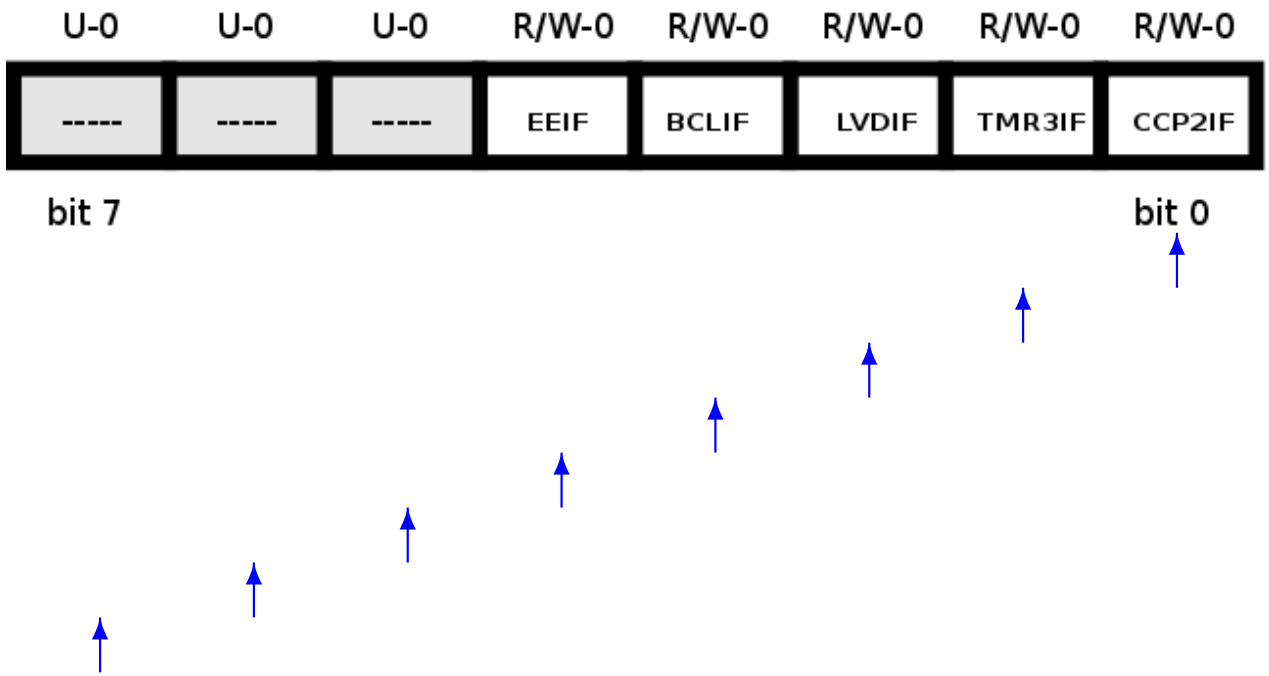
IPR2



Bits in IPR2 register

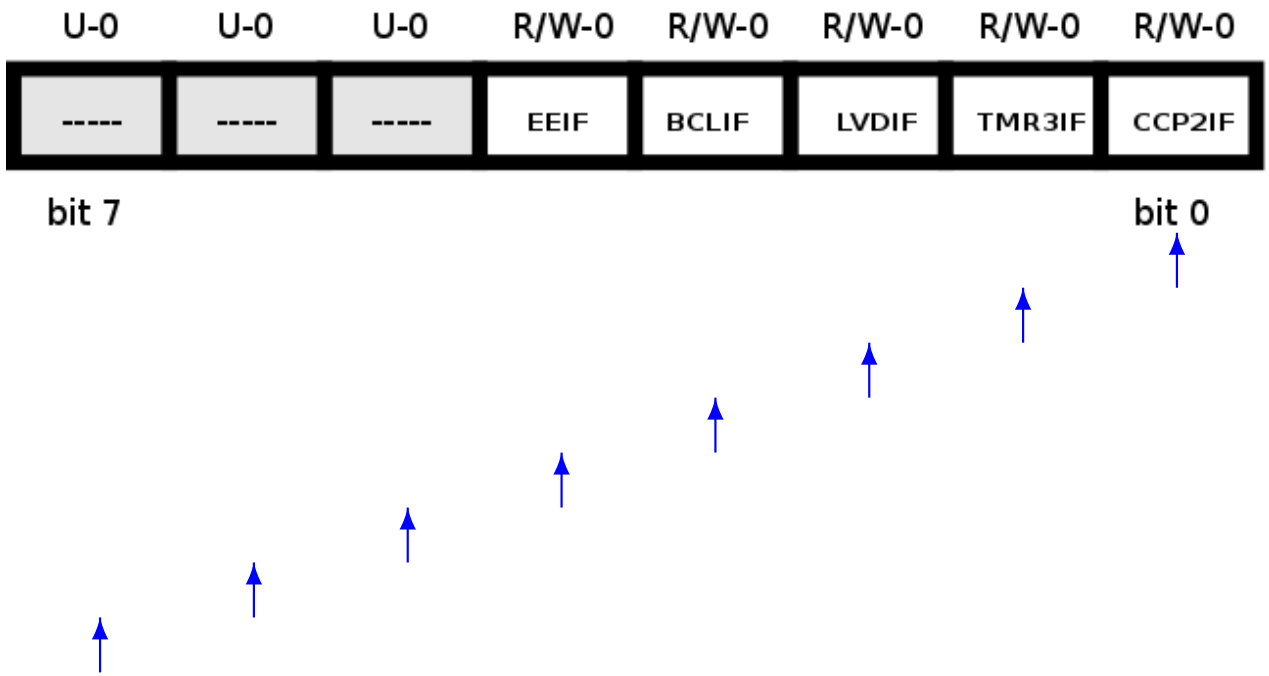
PIR2

PIR2


U-0
U-0
U-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0

EEIF
BCLIF
LVDIF
TMR3IF
CCP2IF
bit 7
bit 0

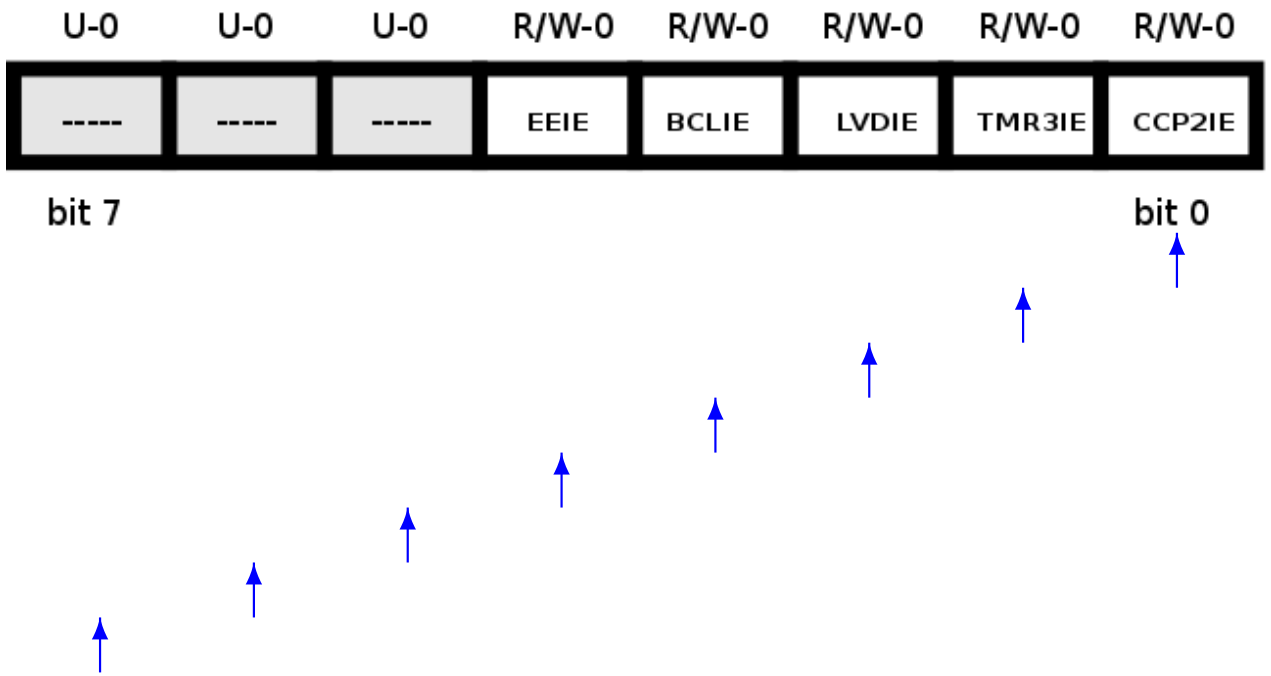
PIR2



Bits in PIR2 register

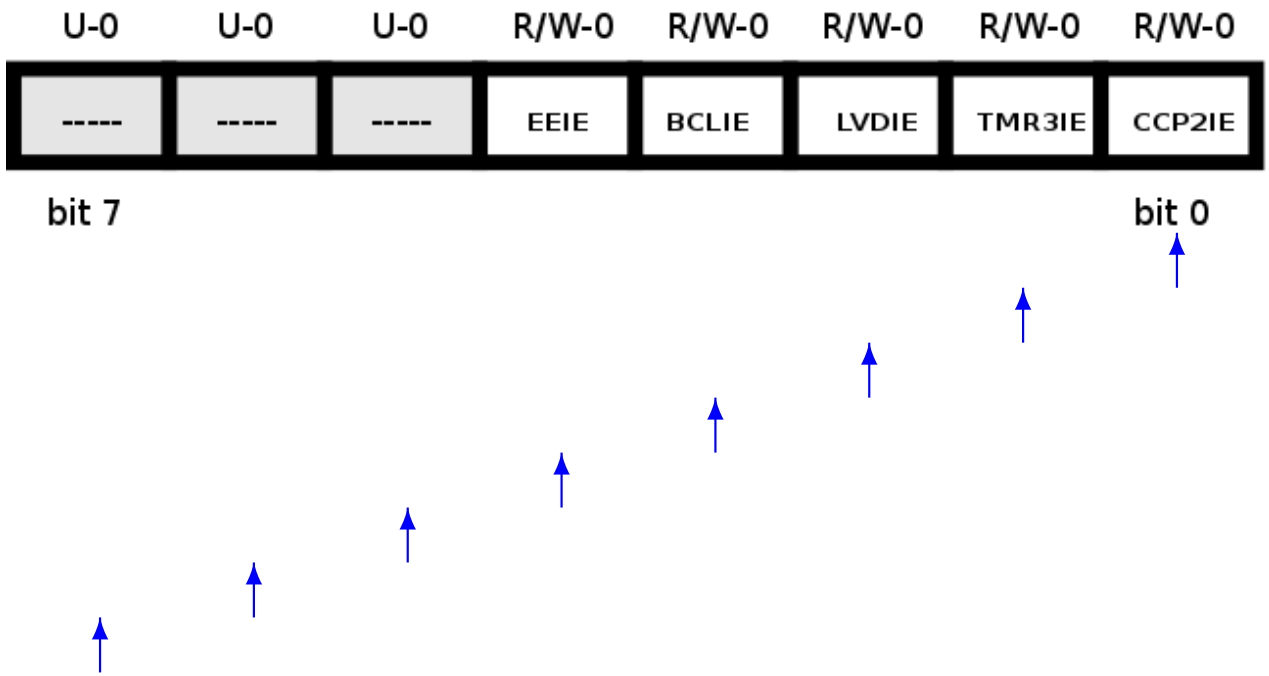
PIE2

PIE2


U-0
U-0
U-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0

EEIE
BCLIE
LVDIE
TMR3IE
CCP2IE
bit 7
bit 0

PIE2



Bits in PIE2 register