TECNOLÓGICO DE MONTERREY®

Subject 6
INTERRUPTS

Interrupts

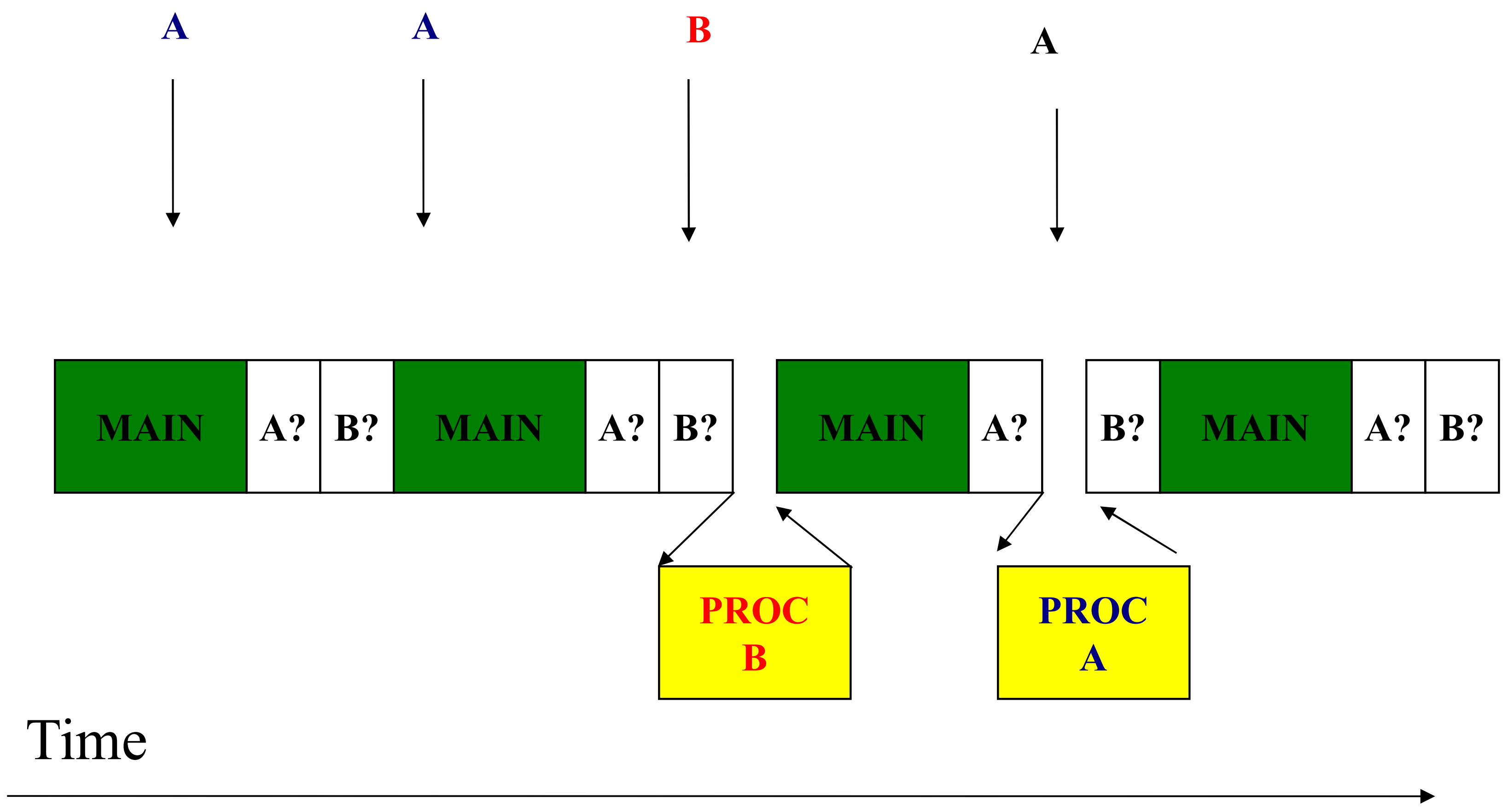
- Interrupts increase the efficiency of the system allowing a I/O device to claim for service when it requires immediate attention. Programs can be written with out continually polling a peripheral

Ronald L. Baldrige
"Interrupts and Power, Complexity to uC-System Design"
EDN Vol. 22 No 14. August 5, 1977

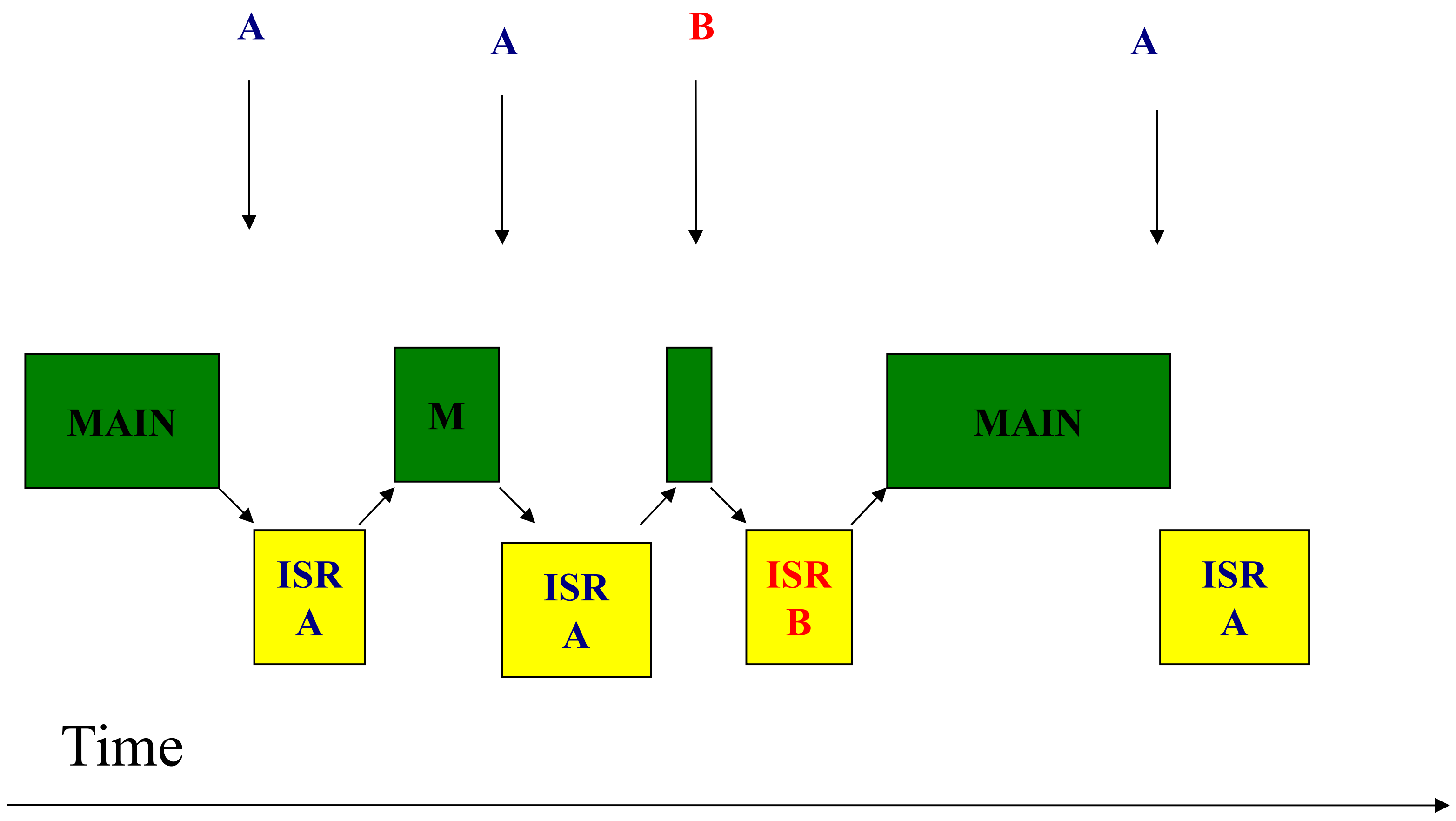
Interrupts

- An interrupt is a function that is called when a external or internal hardware event occurs.
- Exceptions are interrupts generated by te CPU to indicate an extraordinary condition like a division by 0 in an ALU process or a stack overflow in the program control section among others .

Interrupts : Advantage over polling



Interrupts : Advantage over polling



Why use interrupts ?

- It eases coordination of I/O activity
- Allows the safe storage of the system of critical processes in the presence of a “catastrophic” event, like a power outage, destructive impact, tampering, etc. Also know as “graceful shutdown”
- Allows the triggering of periodic activities like sampling, real time clock task review or conventional non critical peripheral polling.

Interrupt “masking”

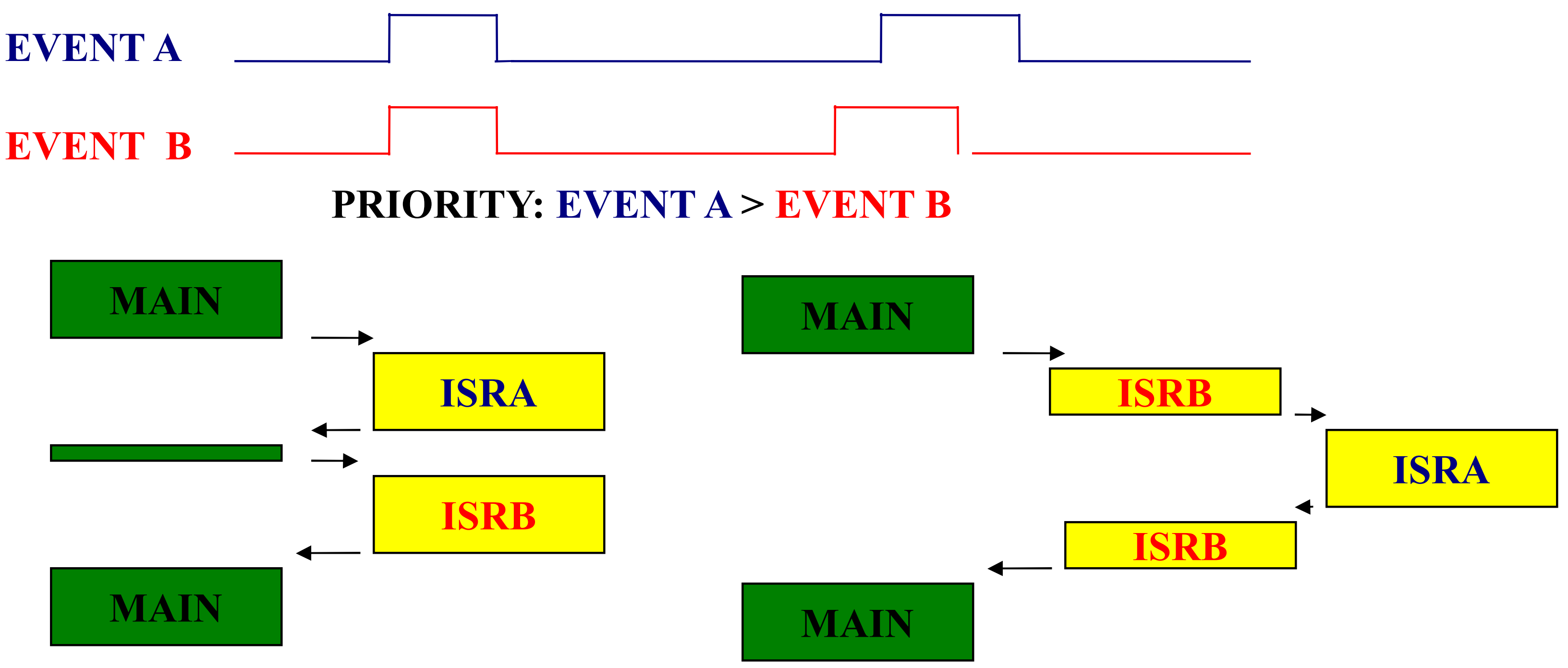
- Interrupt “masking” is a way to ignore them
- In some microcontroller, after reset, sometimes all interrupts are disabled, others maintain critical interrupts enabled.
- Usually every source of interrupt can be individually disabled, and there is a general control to disable or enable all the ones that are individually enabled.

Interrupt priority

- If more tan one interrupt happens at the same time, the priority configuration allows to define the order of execution
- The priority also defines if an interrupt can be interrupted by another.
- The PIC18 allows to define 3 levels
  - None (all have the same)
  - Low
  - High

Interrupt priority

- Simultaneous events



Interrupt service

- The CPU services the interrupt by executing a section of code that is associated to it.
- This section of code is know as the Interrupt Service Routine (ISR)
- When the ISR ends, the CPU continues to execute the main program at the point it was before the occurrence

What happens the Interrupt occurs

- The CPU waits for the current instruction to end its execution
- The program counter is saved (PC)
- Interrupts of same priority are disabled.
- Some important registers are saved in the stack (like the WREG and STATUS)
- The source of the interrupt is identified.
- The CPU assigns the address of the code where the ISR resides to the program counter (PC) register
- The ISR is executed.

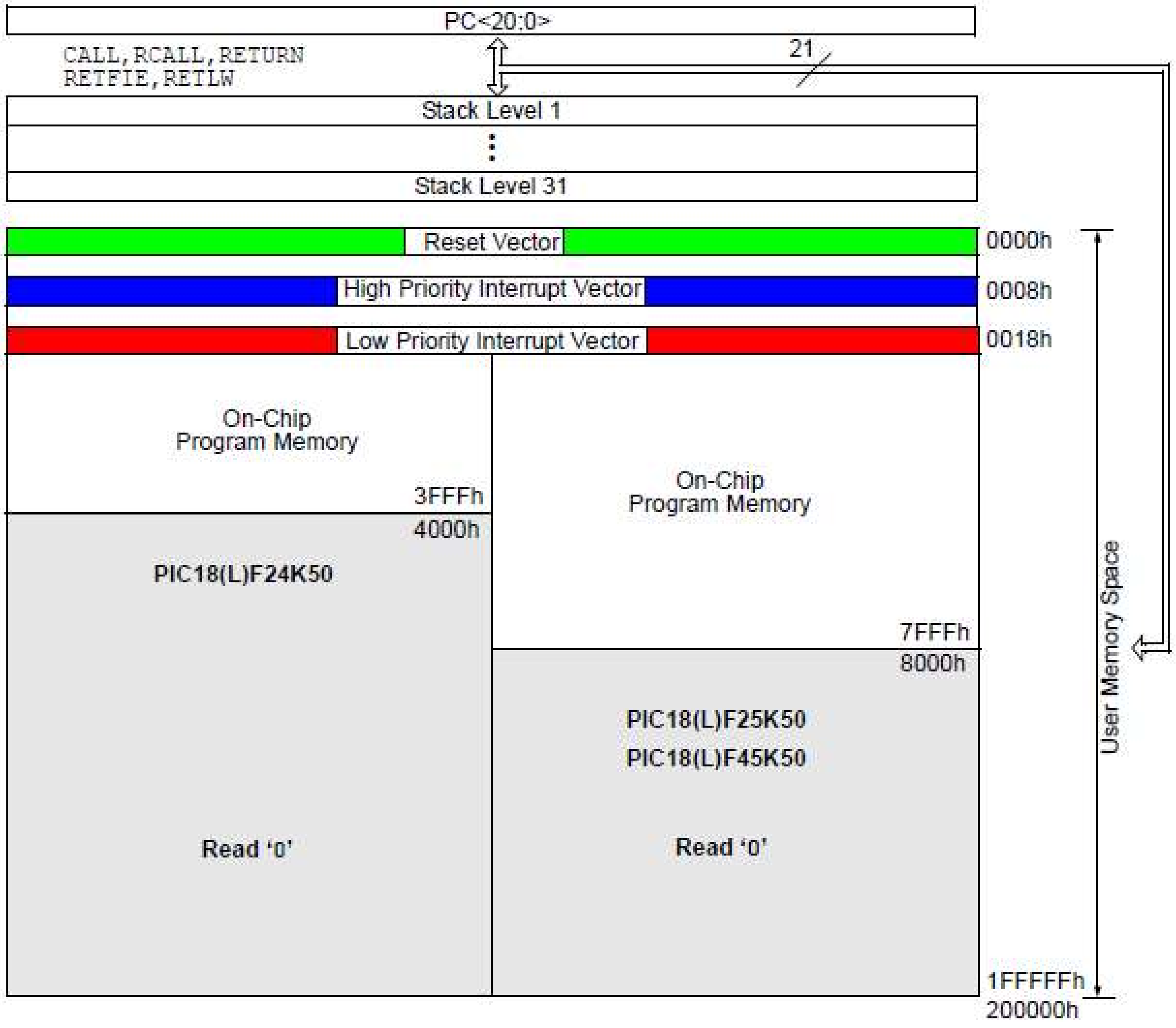
Actions when the ISR ends

- Restore the state of the CPU (return the original value to the critical registers like WREG and STATUS)
- Restore the program counter (PC) to the main program the next instruction after the last one executed.
- Re-enable the interrupt of the same priority
- Continue with the normal flow of the program

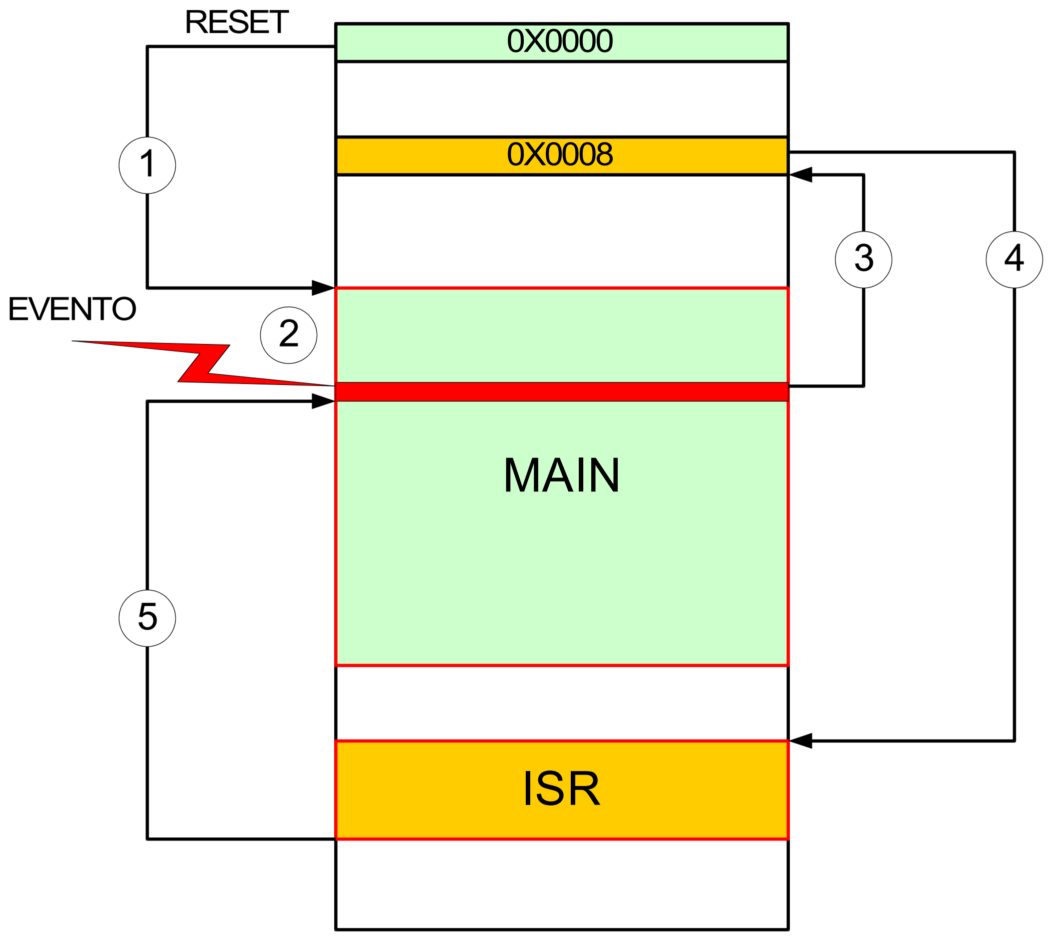
Interrupt vector

- The PIC18 uses the “predefined vector “ scheme. .
- The “vector” is an address (or a group or them) that are related to the interrupt, this address cannot be changed and is assigned by the manufacturer.
- The interrupt vector can store the entire ISR or just a jump or call to the ISR (this is the common method).

Interrupt Vectors



Interrupt Vectors



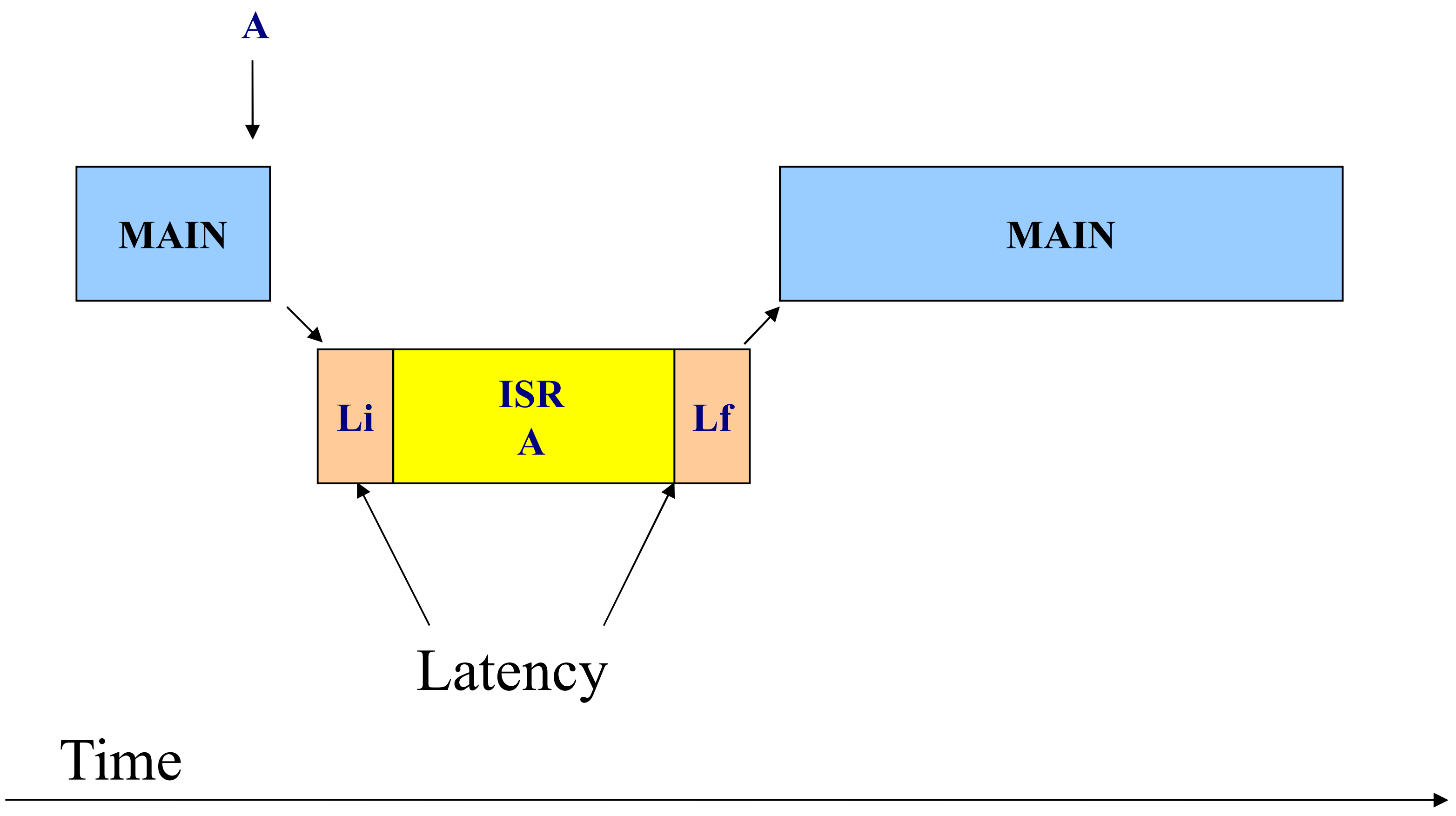
The interrupt vector

- Several interrupts can share the same interrupt vector.
- If this is the case, in the ISR the programmer must identify the source and execute the code associated to that particular interrupt.
- For the PIC this happens when several interrupts share the same priority since vectors are assigned by the priority level.

Interrupt setup in a program

- Write the ISR code.
  - Is like any other subroutine or function in the case of C language. It is recommended to be as short and as simple as possible and avoid delays since this will hang the main code execution.
- Initialize the interrupt vector with the subroutine address (the method depends of the compiler in the case of C).
- Enable the interrupt and its priority
  - There are flags or control bits to enable the interrupts individually and usually one general enable control bit (or flag)

Interrupt latency



Reset

- It can be considered as an interrupt that returns the microcontroller to a known state including the program counter (PC = 0x0000) for the PIC18
- Source of interrupts
  - External hardware reset (MCLR)
  - POR
  - Other sources

Sources of interrupts

- The interrupts on the PIC are generated by the following 2 mechanisms.
  - Default CPU peripherals.
    - Port interrupts
    - TIMER0 interrupt
  - Peripherals that are particular to the part number
    - A/D converter
    - Other timer
    - Serial ports.
    - Etc.

Interrupts

- There are 13 registers to configure the interrupts in the this particular PIC18:

  - INTCON, INTCON2, INTCON3
  - PIR1, PIR2, PIR3
  - PIE1, PIE2, PIE3
  - IPR1, IPR2, IPR3
  - RCON

$\left.\begin{array}{l}\text{PIR1, PIR2, PIR3}\\ \text{PIE1, PIE2, PIE3}\\ \text{IPR1, IPR2, IPR3}\end{array}\right\}$ PIC18F45K50 PERIPHERALS

Interrupts

- In general the interrupts have 3 bits to control the individual operation.
  - Flag Bit: Generates the interrupt and it can be used to know the source.
  - Enable Bit: Allows that the PC jumps to the vector, there are individual and global enables
  - Priority Bit: Select the interrupt priority

Interrupts with priority

- You enable this setting IPEN bit (RCON register).
- There will be a global enable control bit for the low priority and one for the high pripority interrupts.
- There will be the typical particular interrupt enable for each source
- For the high priority the vector is 0x0008
- For the low priority the vector is 0x0018

Interrupts with no priority

- For retro-compatibility with mid range PIC18 devices, setting IPEN bit to 0 (RCON register), all the interrupts will have the same priority and there will be only one interrupt vector (0x0008)

Interrupt processing

- When the CPU process an interrupt, it automatically disables the rest globally, if the user needs that a higher level interrupt interrupts the current ISR the programmer must enable it manually.
- If enabled, the high priority interrupt can interrupt a low priority one.
- Low priority interrupts are not processed while a high priority is in progress.

Interrupt processing

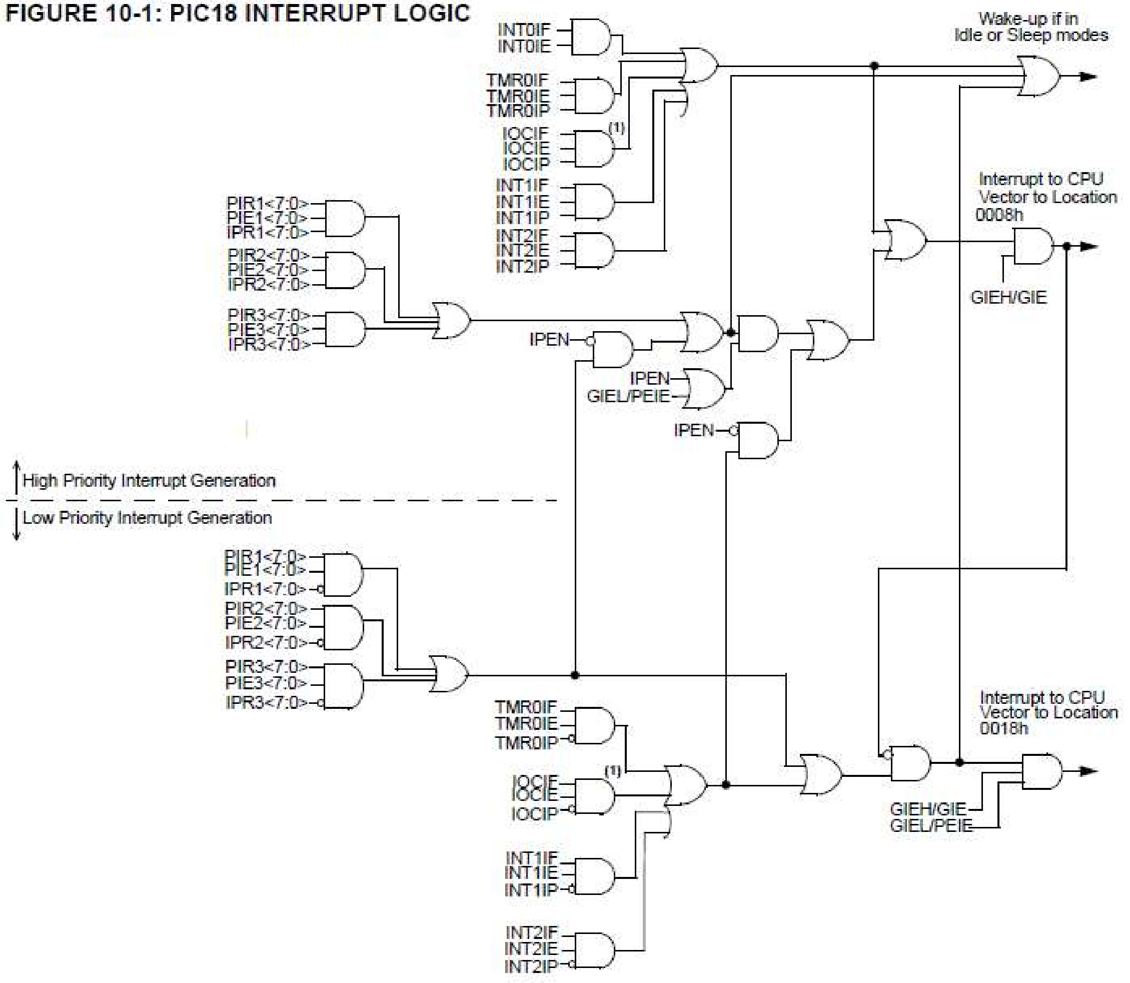
- Return address is stored in the stack
- The program counter (PC) is loaded with the vector (0x0008 or 0x0018)
- Once inside the ISR the programmer must figure out the origin using the interrupt flags (INTCONx or PIRx registers) if several possible sources can trigger the interrupt

Interrupt Operation

- The interrupt flag (INTCONx or PIRx) that generated **the flag interrupt must be cleared** if not, when the ISR ends, the interrupt will re-trigger.
- At the end of the ISR interrupts are re-enabled automatically

Interrupt logic

FIGURE 10-1: PIC18 INTERRUPT LOGIC

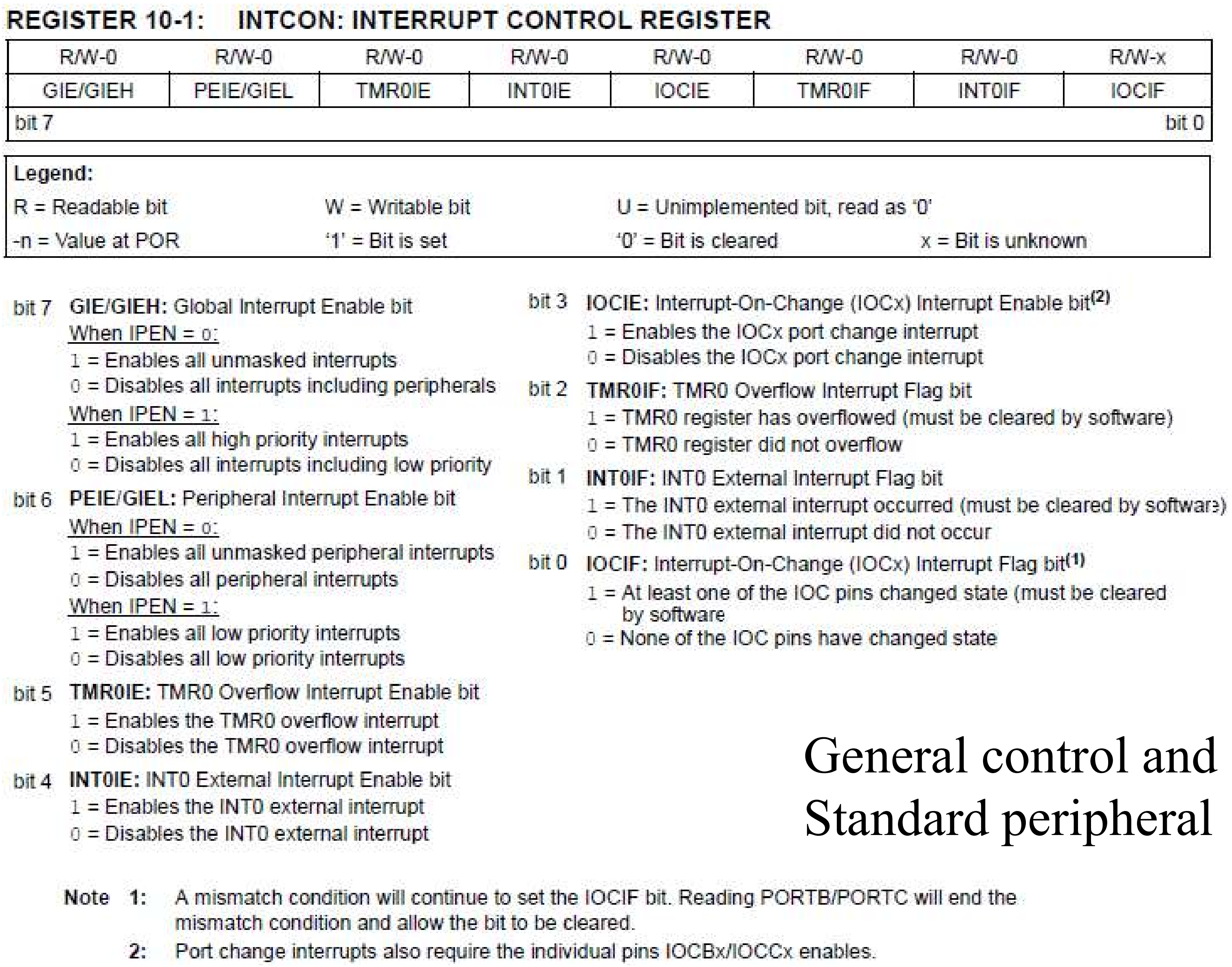


Interrupt Registers for standard peripherals

- INTCON: mixes, interrupt flag, enable flags and priority flags for:
  - Port interrupt INTx
  - Port interrupt Interupt on change
  - Timer 0 interrupts

Interrupt Registers for part specific peripherals

- PIRx: Interrupt flags (PIR1 a PIR5)
  - Peripheral Interrupt flag Register
- PIEx: Enable flags
  - Peripheral Interrupt Enable registers
- IPRx: Priority flags
  - Interrupt Priority Registers



REGISTER 10-1: INTCON: INTERRUPT CONTROL REGISTER

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	IOCIE	TMR0IF	INT0IF	IOCIF
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 7 **GIE/GIEH:** Global Interrupt Enable bit
<u>When IPEN = 0:</u>
1 = Enables all unmasked interrupts
0 = Disables all interrupts including peripherals
<u>When IPEN = 1:</u>
1 = Enables all high priority interrupts
0 = Disables all interrupts including low priority

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
<u>When IPEN = 0:</u>
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
<u>When IPEN = 1:</u>
1 = Enables all low priority interrupts
0 = Disables all low priority interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **IOCIE:** Interrupt-On-Change (IOCx) Interrupt Enable bit[2]
1 = Enables the IOCx port change interrupt
0 = Disables the IOCx port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared by software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared by software)
0 = The INT0 external interrupt did not occur

bit 0 **IOCIF:** Interrupt-On-Change (IOCx) Interrupt Flag bit[1]
1 = At least one of the IOC pins changed state (must be cleared by software
0 = None of the IOC pins have changed state

General control and Standard peripheral

Note 1: A mismatch condition will continue to set the IOCIF bit. Reading PORTB/PORTC will end the mismatch condition and allow the bit to be cleared.

2: Port change interrupts also require the individual pins IOCBx/IOCCx enables.

REGISTER 10-2: INTCON2: INTERRUPT CONTROL 2 REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	IOCIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
- 1 = All PORTB pull-ups are disabled
- 0 = PORTB pull-ups are enabled provided that the pin is an input and the corresponding WPUB bit is set.

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
- 1 = Interrupt on rising edge
- 0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
- 1 = Interrupt on rising edge
- 0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
- 1 = Interrupt on rising edge
- 0 = Interrupt on falling edge

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

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
- 1 = High priority
- 0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **IOCIP:** Port Change Interrupt Priority bit
- 1 = High priority
- 0 = Low priority

General control and Standard peripheral

REGISTER 10-3: INTCON3: INTERRUPT CONTROL 3 REGISTER

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

bit 7	**INT2IP:** INT2 External Interrupt Priority bit 1 = High priority 0 = Low priority
bit 6	**INT1IP:** INT1 External Interrupt Priority bit 1 = High priority 0 = Low priority
bit 5	**Unimplemented:** Read as '0'
bit 4	**INT2IE:** INT2 External Interrupt Enable bit 1 = Enables the INT2 external interrupt 0 = Disables the INT2 external interrupt
bit 3	**INT1IE:** INT1 External Interrupt Enable bit 1 = Enables the INT1 external interrupt 0 = Disables the INT1 external interrupt
bit 2	**Unimplemented:** Read as '0'
bit 1	**INT2IF:** INT2 External Interrupt Flag bit 1 = The INT2 external interrupt occurred (must be cleared by software) 0 = The INT2 external interrupt did not occur
bit 0	**INT1IF:** INT1 External Interrupt Flag bit 1 = The INT1 external interrupt occurred (must be cleared by software) 0 = The INT1 external interrupt did not occur

General control and Standard peripheral

REGISTER 10-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

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

bit 7	**ACTIF:** Active Clock Tuning Interrupt Flag bit 1 = An Active Clock Tuning Event generated an interrupt (must be cleared in software) 0 = No Active Clock Tuning interrupt is pending
bit 6	**ADIF:** A/D Converter Interrupt Flag bit 1 = An A/D conversion completed (must be cleared by software) 0 = The A/D conversion is not complete or has not been started
bit 5	**RCIF:** EUSART Receive Interrupt Flag bit 1 = The EUSART receive buffer, RCREG1, is full (cleared when RCREG1 is read) 0 = The EUSART receive buffer is empty
bit 4	**TXIF:** EUSART Transmit Interrupt Flag bit 1 = The EUSART transmit buffer, TXREG1, is empty (cleared when TXREG1 is written) 0 = The EUSART transmit buffer is full
bit 3	**SSPIF:** Master Synchronous Serial Port Interrupt Flag bit 1 = The transmission/reception is complete (must be cleared by software) 0 = Waiting to transmit/receive
bit 2	**CCP1IF:** CCP1 Interrupt Flag bit Capture mode: 1 = A TMR1 register capture occurred (must be cleared by software) 0 = No TMR1 register capture occurred Compare mode: 1 = A TMR1 register compare match occurred (must be cleared by software) 0 = No TMR1 register compare match occurred PWM mode: Unused in this mode
bit 1	**TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit 1 = TMR2 to PR2 match occurred (must be cleared by software) 0 = No TMR2 to PR2 match occurred
bit 0	**TMR1IF:** TMR1 Overflow Interrupt Flag bit 1 = TMR1 register overflowed (must be cleared by software) 0 = TMR1 register did not overflow

REGISTER 10-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	C1IF	C2IF	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
bit 7							bit 0

bit 7	**OSCFIF:** Oscillator Fail Interrupt Flag bit 1 = Device oscillator failed, clock input has changed to HFINTOSC (must be cleared by software) 0 = Device clock operating
bit 6	**C1IF:** Comparator C1 Interrupt Flag bit 1 = Comparator C1 output has changed (must be cleared by software) 0 = Comparator C1 output has not changed
bit 5	**C2IF:** Comparator C2 Interrupt Flag bit 1 = Comparator C2 output has changed (must be cleared by software) 0 = Comparator C2 output has not changed
bit 4	**EEIF:** Data EEPROM/Flash Write Operation Interrupt Flag bit 1 = The write operation is complete (must be cleared by software) 0 = The write operation is not complete or has not been started
bit 3	**BCLIF:** MSSP Bus Collision Interrupt Flag bit 1 = A bus collision occurred (must be cleared by software) 0 = No bus collision occurred
bit 2	**HLVDIF:** Low-Voltage Detect Interrupt Flag bit 1 = A low-voltage condition occurred (direction determined by the VDIRMAG bit of the HLVDCON register) 0 = A low-voltage condition has not occurred
bit 1	**TMR3IF:** TMR3 Overflow Interrupt Flag bit 1 = TMR3 register overflowed (must be cleared by software) 0 = TMR3 register did not overflow
bit 0	**CCP2IF:** CCP2 Interrupt Flag bit <u>Capture mode:</u> 1 = A TMR1 register capture occurred (must be cleared by software) 0 = No TMR1 register capture occurred <u>Compare mode:</u> 1 = A TMR1 register compare match occurred (must be cleared by software) 0 = No TMR1 register compare match occurred <u>PWM mode:</u> Unused in this mode.

REGISTER 10-6: PIR3: PERIPHERAL INTERRUPT (FLAG) REGISTER 3

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	CTMUIF	USBIF	TMR3GIF	TMR1GIF
bit 7							bit 0

bit 7-4	**Unimplemented:** Read as ‘0’
bit 3	**CTMUIF:** CTMU Interrupt Flag bit 1 = CTMU interrupt occurred (must be cleared in software) 0 = No CTMU interrupt occurred
bit 2	**USBIF:** USB Interrupt Flag bit 1 = USB requested an interrupt (must be cleared in software) 0 = No USB interrupt request
bit 1	**TMR3GIF:** TMR3 Gate Interrupt Flag bit 1 = TMR gate interrupt occurred (must be cleared in software) 0 = No TMR gate occurred
bit 0	**TMR1GIF:** TMR1 Gate Interrupt Flag bit 1 = TMR gate interrupt occurred (must be cleared in software) 0 = No TMR gate occurred

REGISTER 10-7: PIE1: PERIPHERAL INTERRUPT ENABLE (FLAG) 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
ACTIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7	**ACTIE:** Active Clock Tuning Interrupt Enable bit 1 = Enables Active Clock Tuning interrupt 0 = Disables Active Clock Tuning interrupt
bit 6	**ADIE:** A/D Converter Interrupt Enable bit 1 = Enables the A/D interrupt 0 = Disables the A/D interrupt
bit 5	**RCIE:** EUSART Receive Interrupt Enable bit 1 = Enables the EUSART receive interrupt 0 = Disables the EUSART receive interrupt
bit 4	**TXIE:** EUSART Transmit Interrupt Enable bit 1 = Enables the EUSART transmit interrupt 0 = Disables the EUSART transmit interrupt
bit 3	**SSPIE:** Master Synchronous Serial Port Interrupt Enable bit 1 = Enables the MSSP interrupt 0 = Disables the MSSP interrupt
bit 2	**CCP1IE:** CCP1 Interrupt Enable bit 1 = Enables the CCP1 interrupt 0 = Disables the CCP1 interrupt
bit 1	**TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit 1 = Enables the TMR2 to PR2 match interrupt 0 = Disables the TMR2 to PR2 match interrupt
bit 0	**TMR1IE:** TMR1 Overflow Interrupt Enable bit 1 = Enables the TMR1 overflow interrupt 0 = Disables the TMR1 overflow interrupt

REGISTER 10-8: PIE2: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	C1IE	C2IE	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
bit 7							bit 0

bit 7	**OSCFIE:** Oscillator Fail Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 6	**C1IE:** Comparator C1 Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 5	**C2IE:** Comparator C2 Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 4	**EEIE:** Data EEPROM/Flash Write Operation Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 3	**BCLIE:** MSSP Bus Collision Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 2	**HLVDIE:** Low-Voltage Detect Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 1	**TMR3IE:** TMR3 Overflow Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 0	**CCP2IE:** CCP2 Interrupt Enable bit 1 = Enabled 0 = Disabled

REGISTER 10-9: PIE3: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 3

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	CTMUIE	USBIE	TMR3GIE	TMR1GIE
bit 7							bit 0

bit 7-4	**Unimplemented:** Read as '0'
bit 3	**CTMUIE:** CTMU Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 2	**USBIE:** USB Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 1	**TMR3GIE:** TMR3 Gate Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 0	**TMR1GIE:** TMR1 Gate Interrupt Enable bit 1 = Enabled 0 = Disabled

REGISTER 10-10: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

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

bit 7	**ACTIP:** Active Clock Tuning Interrupt Priority bit 1 = High priority 0 = Low priority
bit 6	**ADIP:** A/D Converter Interrupt Priority bit 1 = High priority 0 = Low priority
bit 5	**RCIP:** EUSART Receive Interrupt Priority bit 1 = High priority 0 = Low priority
bit 4	**TXIP:** EUSART Transmit Interrupt Priority bit 1 = High priority 0 = Low priority
bit 3	**SSPIP:** Master Synchronous Serial Port Interrupt Priority bit 1 = High priority 0 = Low priority
bit 2	**CCP1IP:** CCP1 Interrupt Priority bit 1 = High priority 0 = Low priority
bit 1	**TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit 1 = High priority 0 = Low priority
bit 0	**TMR1IP:** TMR1 Overflow Interrupt Priority bit 1 = High priority 0 = Low priority

REGISTER 10-11: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
OSCFIP	C1IP	C2IP	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP
bit 7							bit 0

bit 7 **OSCFIP:** Oscillator Fail Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **C1IP:** Comparator C1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **C2IP:** Comparator C2 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **EEIP:** Data EEPROM/Flash Write Operation Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **BCLIP:** MSSP Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **HLVDIP:** Low-Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP:** TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP:** CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

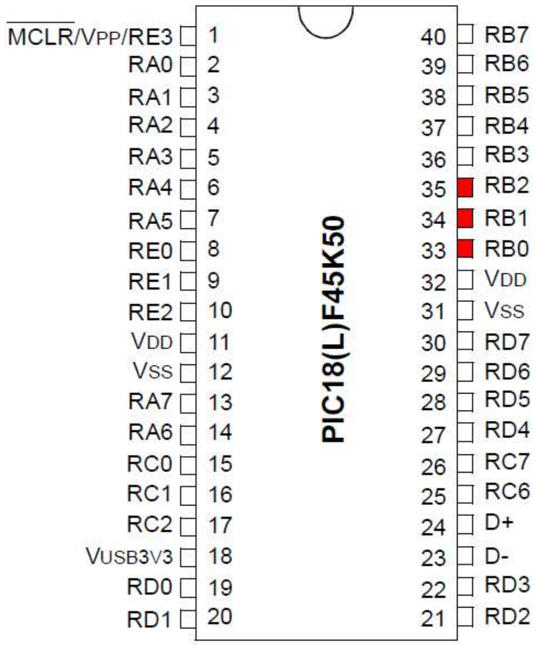
REGISTER 10-12: IPR3: PERIPHERAL INTERRUPT PRIORITY REGISTER 3

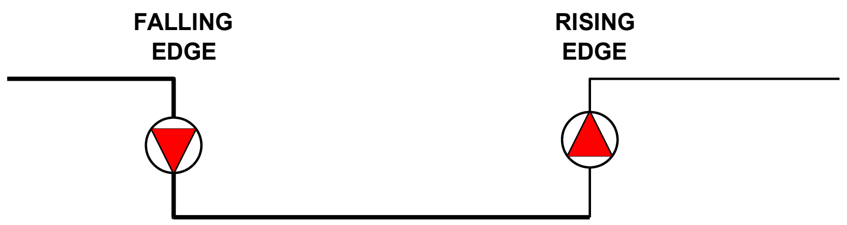
U-0	U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	—	CTMUIP	USBIP	TMR3GIP	TMR1GIP
bit 7							bit 0

bit 7-4	**Unimplemented:** Read as '0'
bit 3	**CTMUIP:** CTMU Interrupt Priority bit 1 = High priority 0 = Low priority
bit 2	**USBIP:** USB Interrupt Priority bit 1 = High priority 0 = Low priority
bit 1	**TMR3GIP:** TMR3 Gate Interrupt Priority bit 1 = High priority 0 = Low priority
bit 0	**TMR1GIP:** TMR1 Gate Interrupt Priority bit 1 = High priority 0 = Low priority

PIC18 standard peripheral interrupts

- 3 ports that are assigned to the external interrupt peripheral signals INT0 to INT2





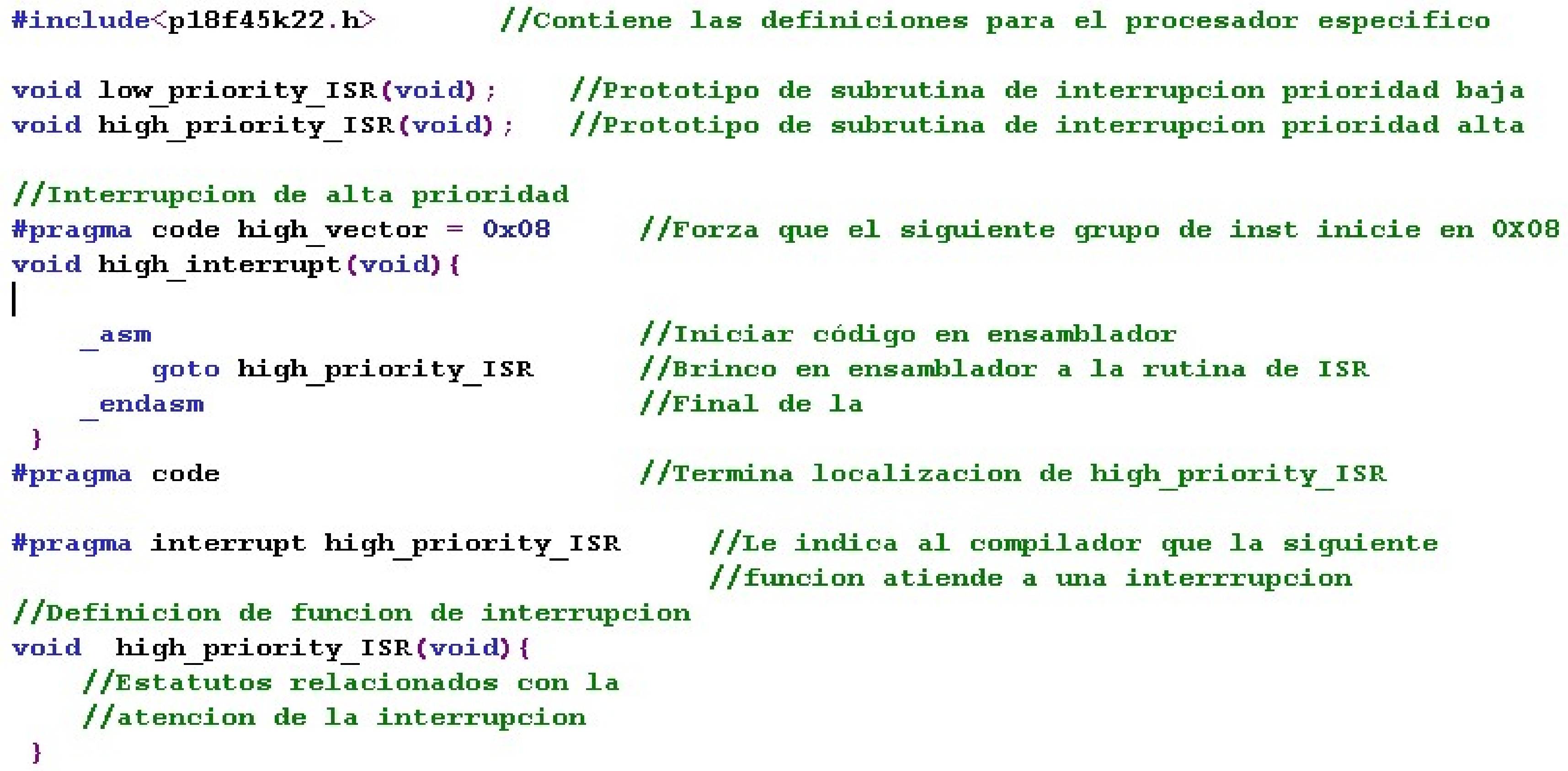
PIC18 standard peripheral interrupts

- Peripheral that detects a change of state of any or the PORTB or PORTC signals when they are configured as inputs.
- The Timer 0 peripheral

Coding the ISR in the old C compiler (C18)

- The declaration of the interrupt routines and vectors assignments are done using #pragma

In computer programming, a directive or pragma (from "pragmatic") is a language construct that specifies how a compiler (or other translator) should process its input. Directives are not part of the grammar of a programming language, and may vary from compiler to compiler. They can be processed by a preprocessor to specify compiler behavior, or function as a form of in-band parameterization……(sorce Wikipedia)



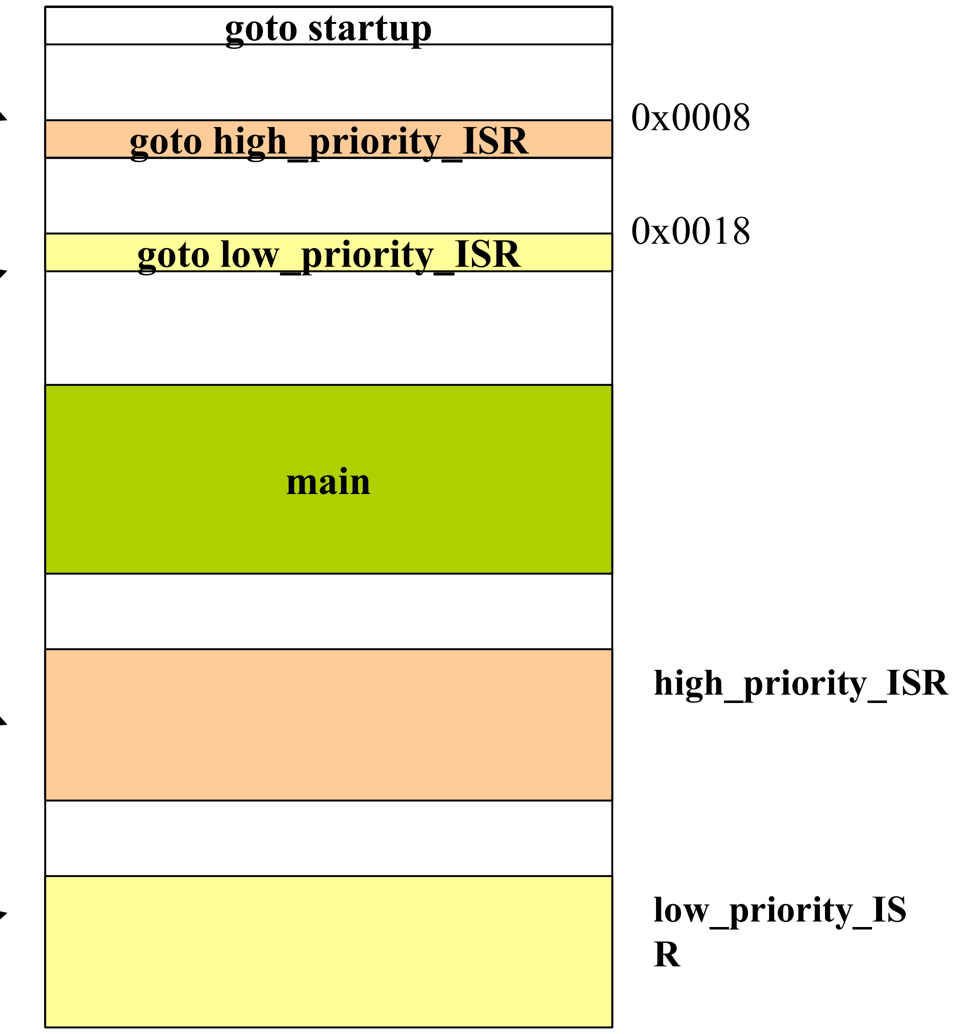
```
#include<p18f45k22.h>        //Contiene las definiciones para el procesador especifico

void low_priority_ISR(void);    //Prototipo de subrutina de interrupcion prioridad baja
void high_priority_ISR(void);   //Prototipo de subrutina de interrupcion prioridad alta

//Interrupcion de alta prioridad
#pragma code high_vector = 0x08     //Forza que el siguiente grupo de inst inicie en 0X08
void high_interrupt(void){

    _asm                            //Iniciar código en ensamblador
        goto high_priority_ISR      //Brinco en ensamblador a la rutina de ISR
    _endasm                         //Final de la
 }
#pragma code                        //Termina localizacion de high_priority_ISR

#pragma interrupt high_priority_ISR     //Le indica al compilador que la siguiente
                                        //funcion atiende a una interrrupcion
//Definicion de funcion de interrupcion
void  high_priority_ISR(void){
    //Estatutos relacionados con la
    //atencion de la interrupcion
 }
```

```
//Interrupcion de baja prioridad
#pragma code low_vector = 0x18      //Forza que el siguiente grupo de inst inicie en 0X18
void low_interrupt(void){
    _asm                            //Iniciar código en ensamblador
        goto low_priority_ISR       //Brinco en ensamblador a la rutina de ISR
    _endasm                         //Final de la
 }
#pragma code                        //Termina localizacion de high_priority_ISR

#pragma interrupt low_priority_ISR      //Le indica al compilador que la siguiente
                                        //funcion atiende a una interrrupcion
//Definicion de funcion de interrupcion
void  low_priority_ISR(void){
    //Estatutos relacionados con la
    //atencion de la interrupcion
 }

main(void){
//Codigo
}
```

```
#pragma code high_vector = 0x08      //Fc
void high_interrupt(void){
    _asm goto high_priority_ISR _endasm
 }
#pragma code                         //Te

//Interrupcion de baja prioridad
#pragma code low_vector = 0x18       //F
void low_interrupt(void){
    _asm goto low_priority_ISR _endasm
 }
#pragma code                         //T
```

```
#pragma interrupt high_priority_ISR
//Definicion de funcion de interrupcion
void  high_priority_ISR(void){
    //Estatutos relacionados con la
    //atencion de la interrupcion
 }
```

```
#pragma interrupt low_priority_ISR
//Definicion de funcion de interrupcion
void  low_priority_ISR(void){ |
    //Estatutos relacionados con la
    //atencion de la interrupcion
 }
```



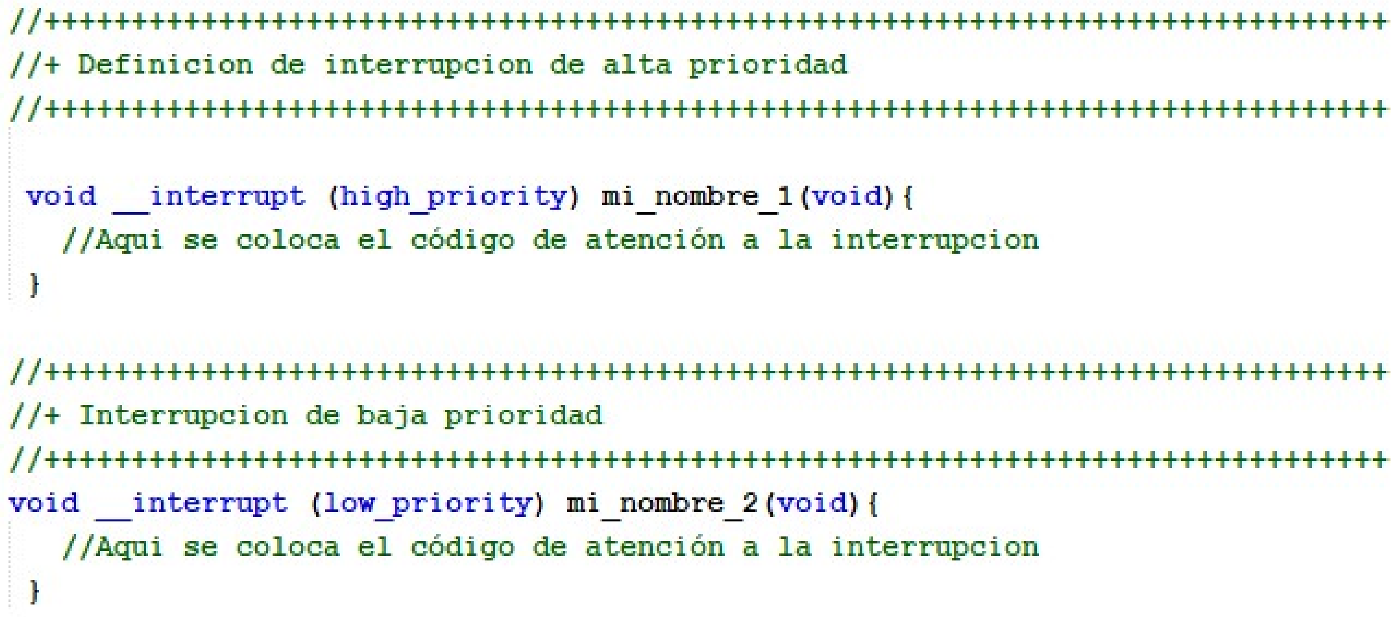
Coding an interrupt on the XC8 (before version 2.0)

- The XC8 eases the declaration of the ISR routines since it takes care of the vector assignation
- The programmer just declares two functions using the following definition

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Interrupcion de alta prioridad
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
void  interrupt mi_nombre(void){
  //Aqui va el codigo de atención de alta prioridad
 }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Interrupcion de baja prioridad
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
void  interrupt low_priority mi_nombre(void){
    //Aqui va el codigo de atención de baja prioridad
 }
```

Coding an interrupt on the XC8 (after version 2.0)

- After version 2.0 the declaration changed the syntax's to the following:



```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Definicion de interrupcion de alta prioridad
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 void __interrupt (high_priority) mi_nombre_1(void){
   //Aqui se coloca el código de atención a la interrupcion
 }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Interrupcion de baja prioridad
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void __interrupt (low_priority) mi_nombre_2(void){
   //Aqui se coloca el código de atención a la interrupcion
 }
```

Reset

a) Power-on Reset (POR)

b) $\overline{\text{MCLR}}$ Reset during normal operation

c) $\overline{\text{MCLR}}$ Reset during power-managed modes

d) Watchdog Timer (WDT) Reset (during execution)

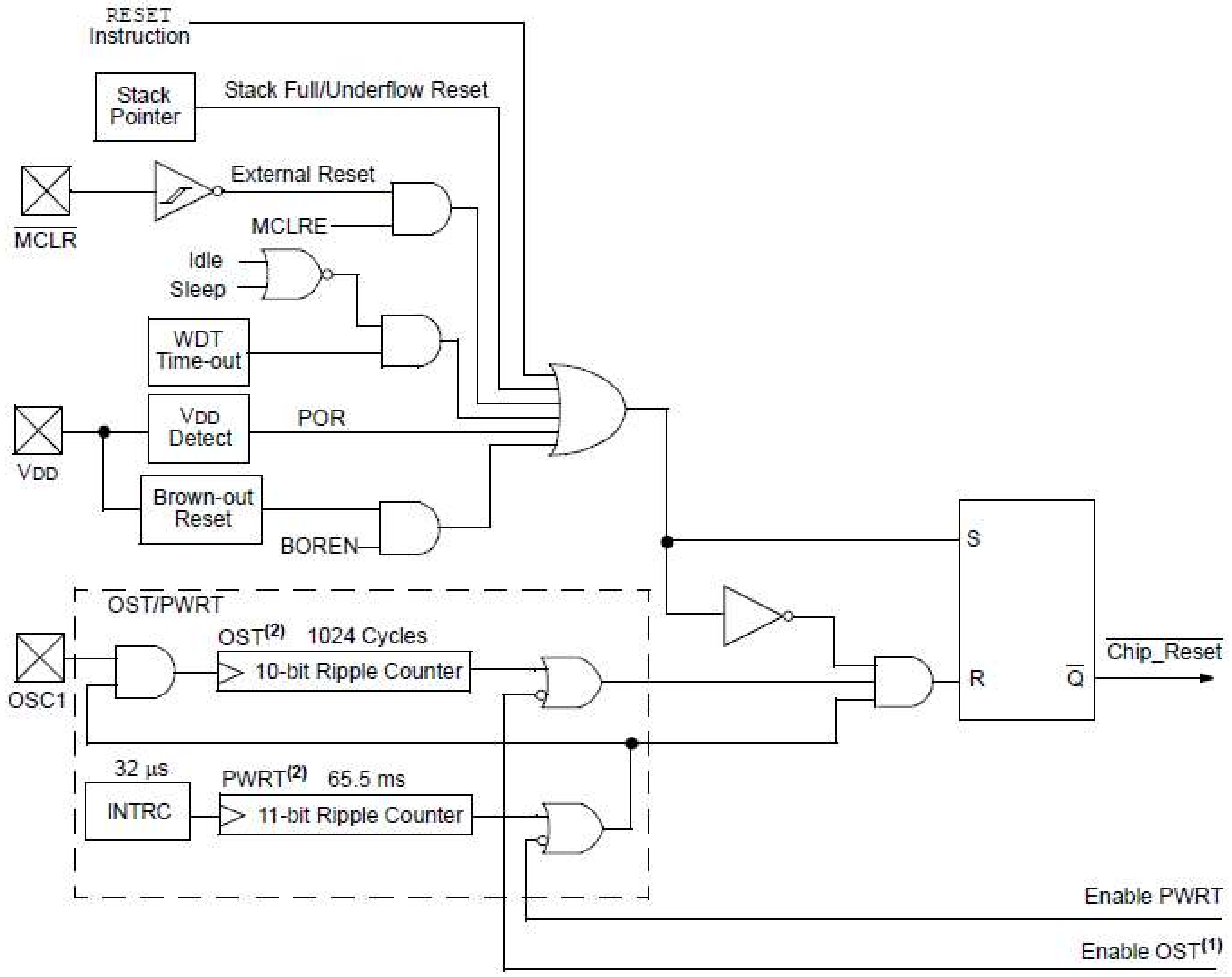
e) Programmable Brown-out Reset (BOR)

f) `RESET` Instruction

g) Stack Full Reset

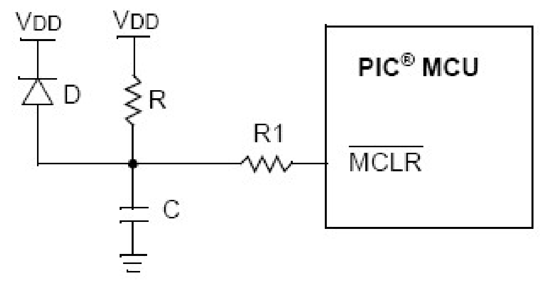
h) Stack Underflow Reset

Reset


RESET Instruction
Stack Pointer
Stack Full/Underflow Reset
External Reset
MCLR
MCLRE
Idle
Sleep
WDT Time-out
VDD
VDD Detect
POR
Brown-out Reset
BOREN
S
R
Q
Chip_Reset
OST/PWRT
OSC1
OST(2) 1024 Cycles
10-bit Ripple Counter
32 µs
INTRC
PWRT(2) 65.5 ms
11-bit Ripple Counter
Enable PWRT
Enable OST(1)

Reset

FIGURE 4-2: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)

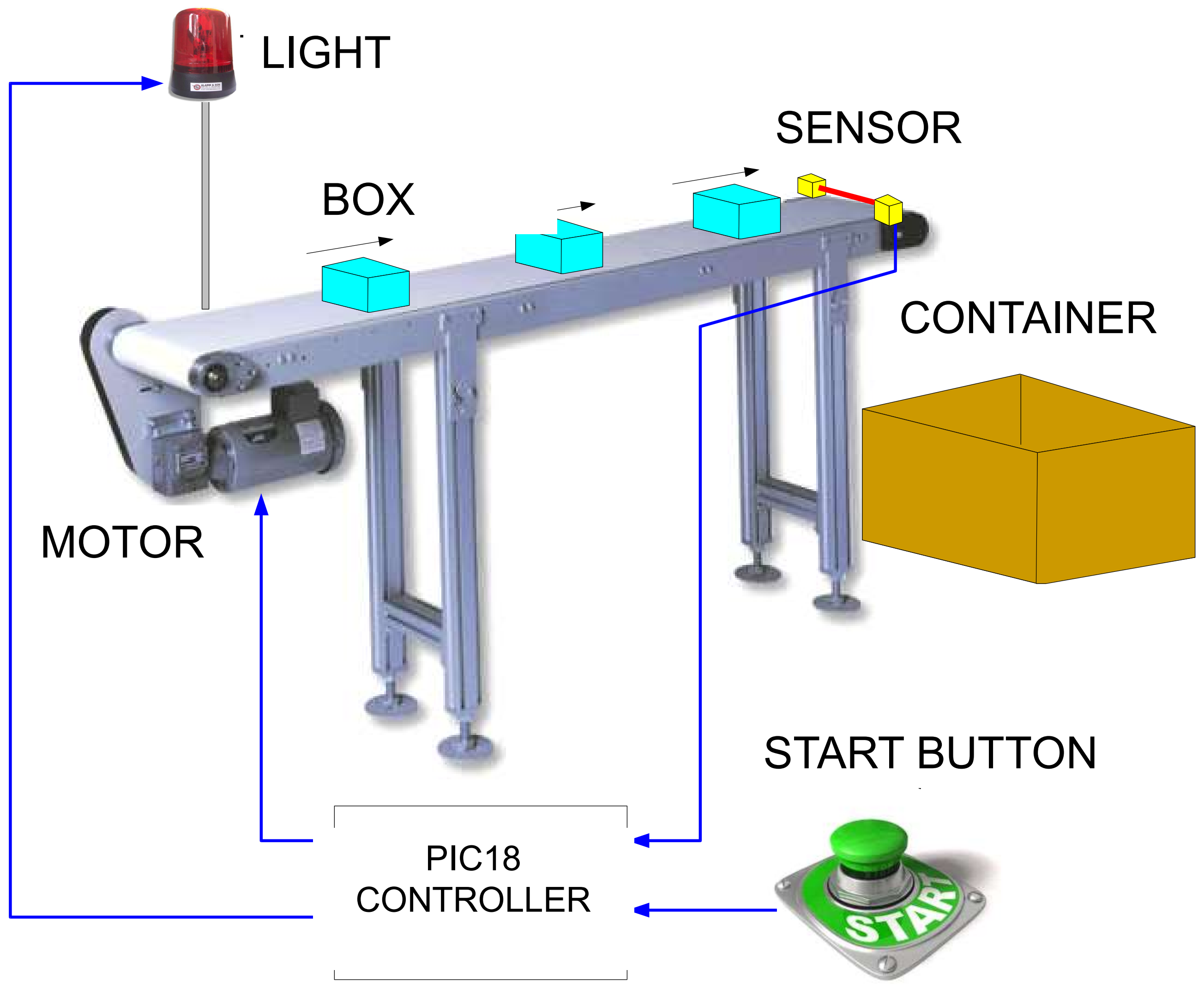


Note 1: External Power-on Reset circuit is required only if the VDD power-up slope is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: 15 kΩ < R < 40 kΩ is recommended to make sure that the voltage drop across R does not violate the device's electrical specification.

3: R1 ≥ 1 kΩ will limit any current flowing into $\overline{\text{MCLR}}$ from external capacitor C, in the event of $\overline{\text{MCLR}}$/VPP pin breakdown, due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

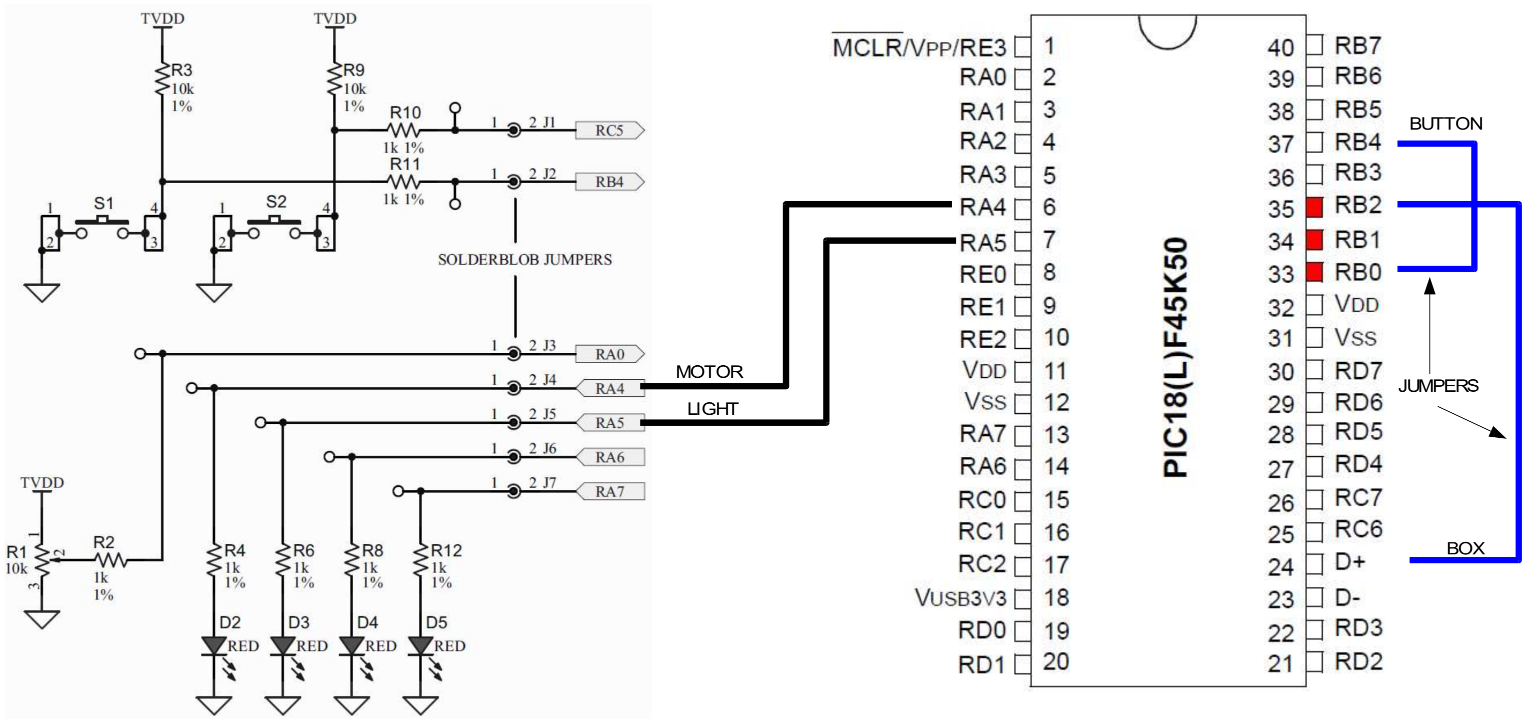
Conveyor belt controller



Conveyor belt controller

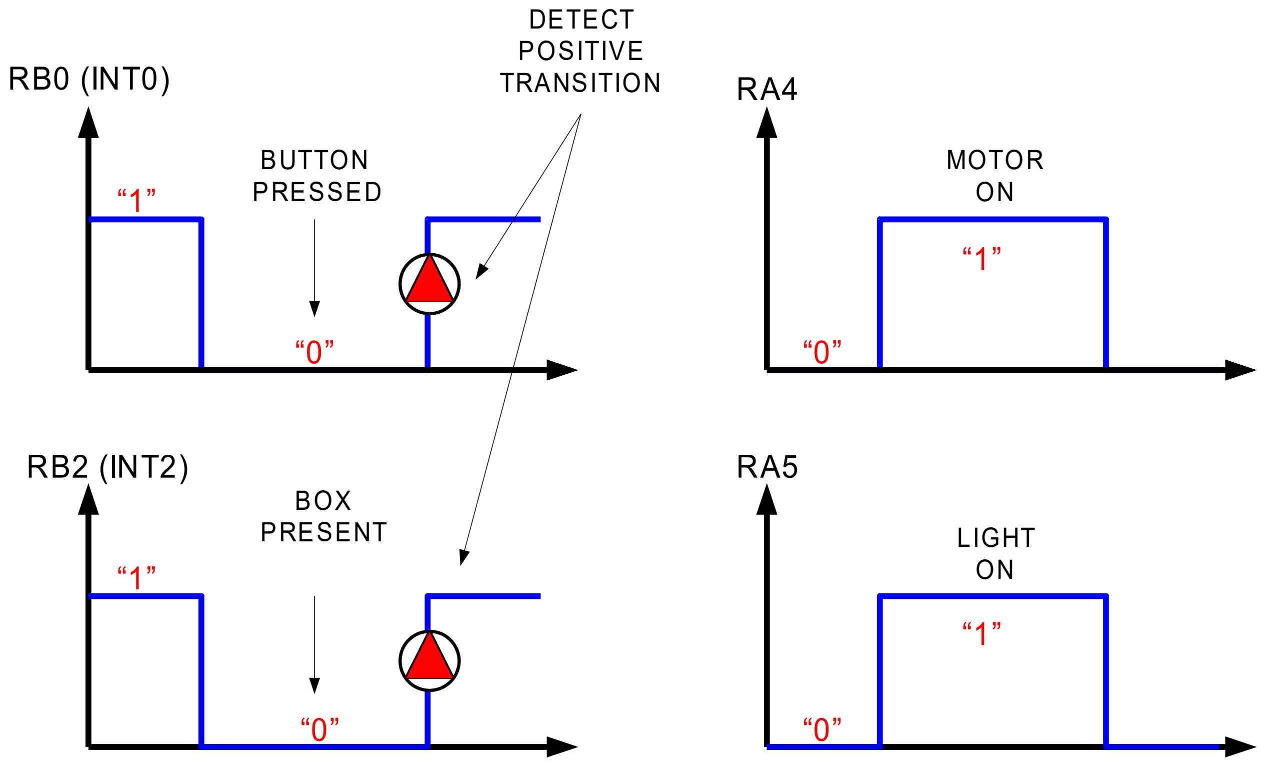
- The control program must do the following:
  - Wait for the start button to be pressed
  - Count the number of boxes
  - If the number is equal to MAXIMO
    - Stop the conveyor belt
    - Turn the light
    - Wait again for the start buton

Direct implementation in curiosity board

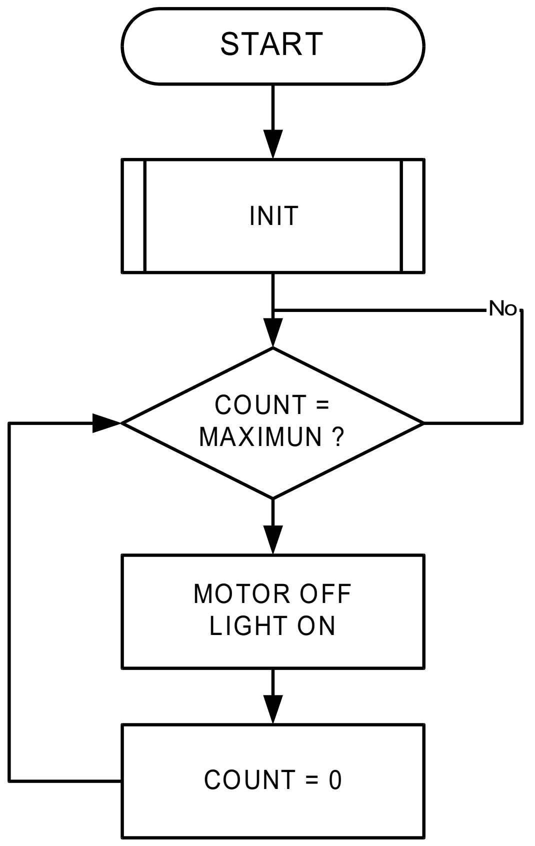


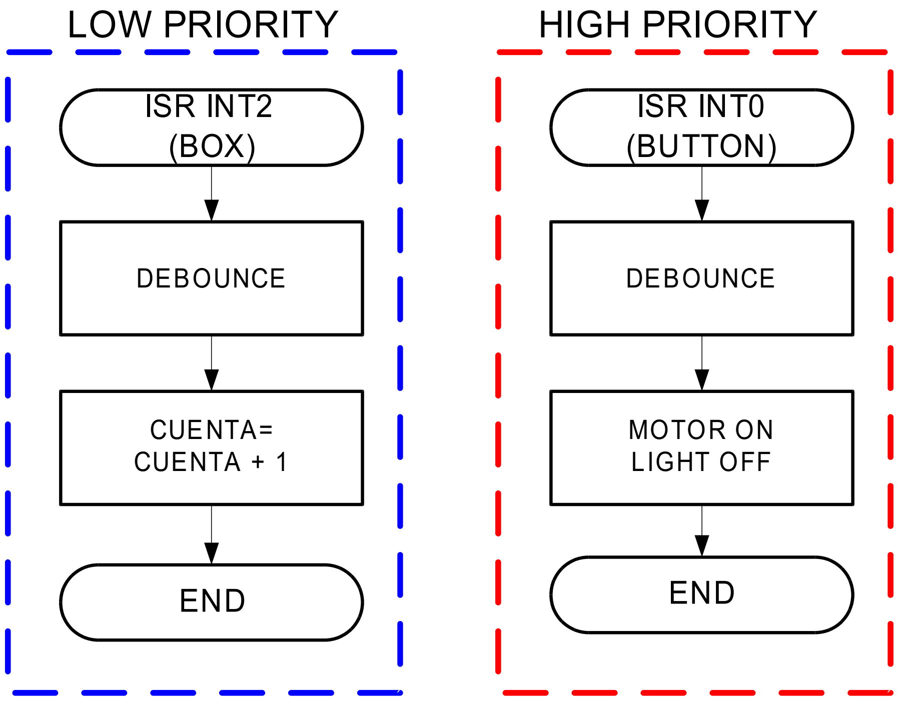
Conveyor belt controller

- IO signals to sensors (input signals will be configured as interrupts)



Flow diagram

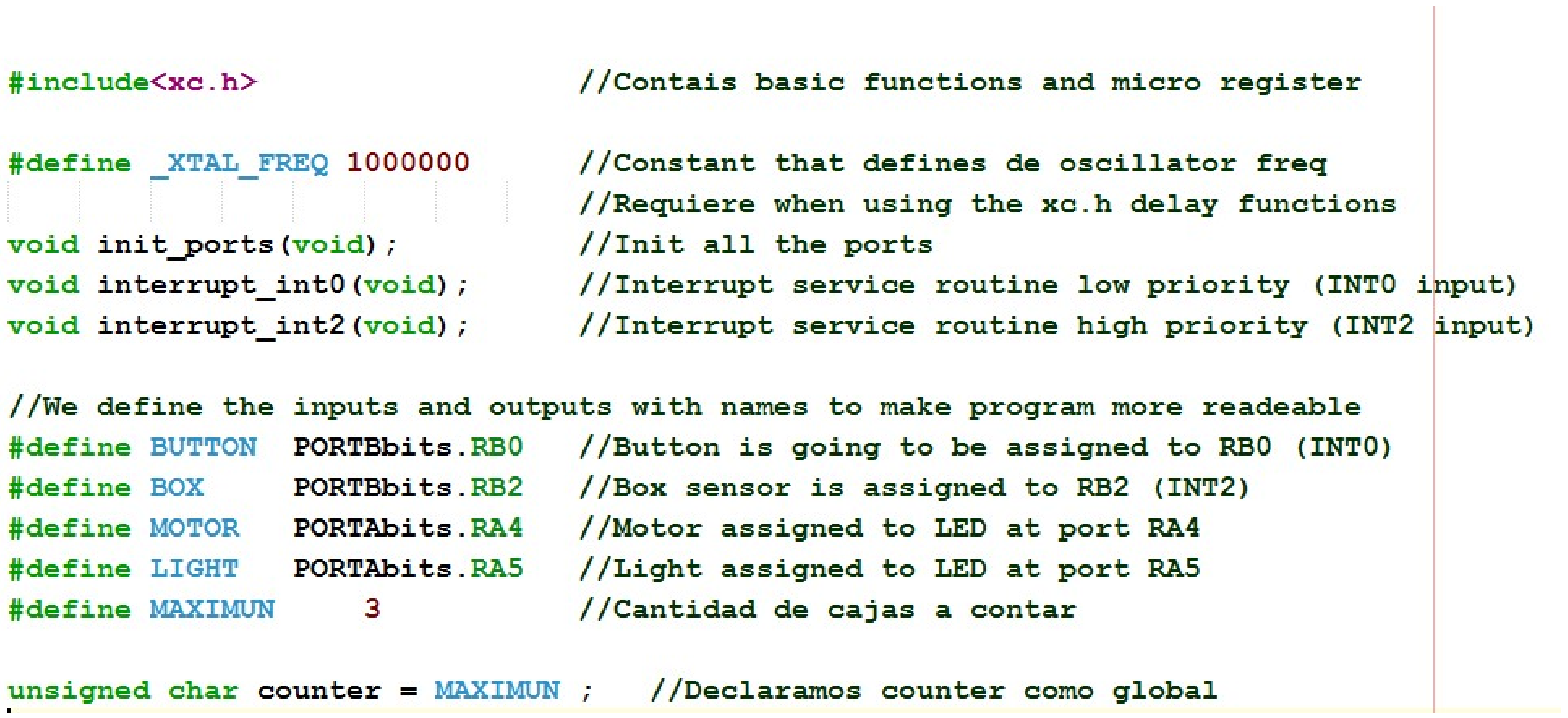




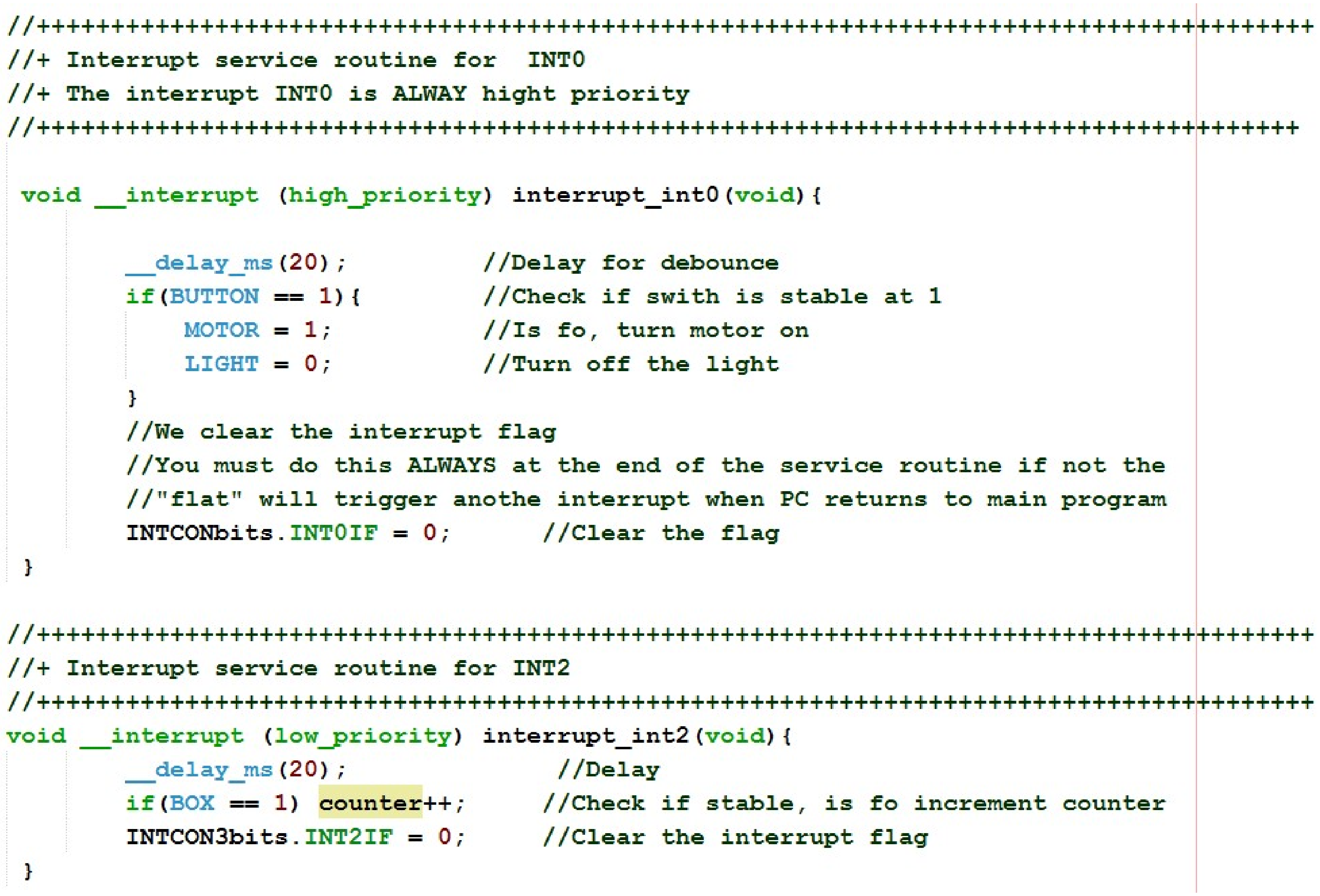
Example

- Configure INT0 to interrupt on positive transition (box just passed by)
- Configure INT2 to interrupt on positive transition (button released)
- There is no other interrupt configured

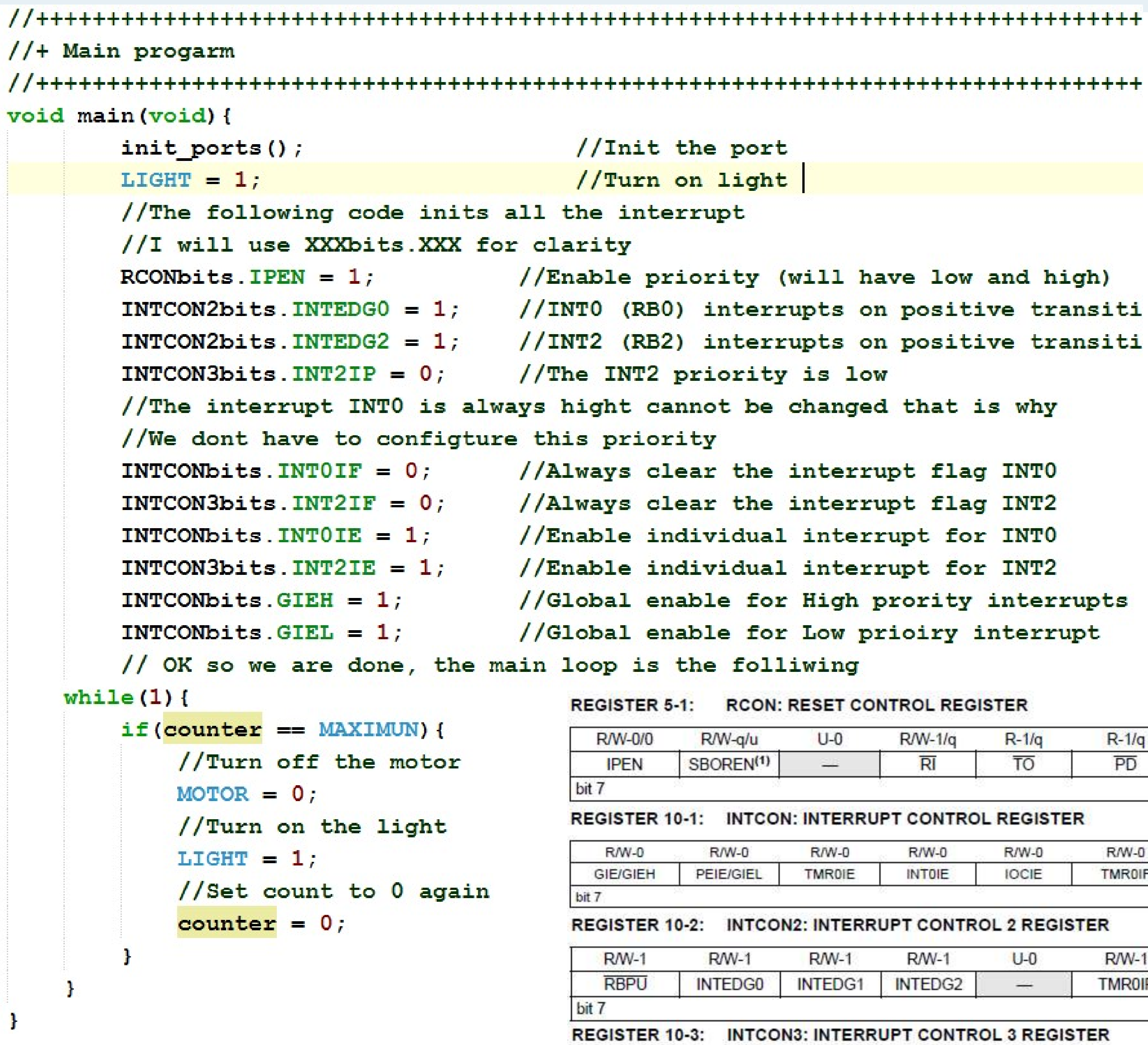
Code using XC compiler



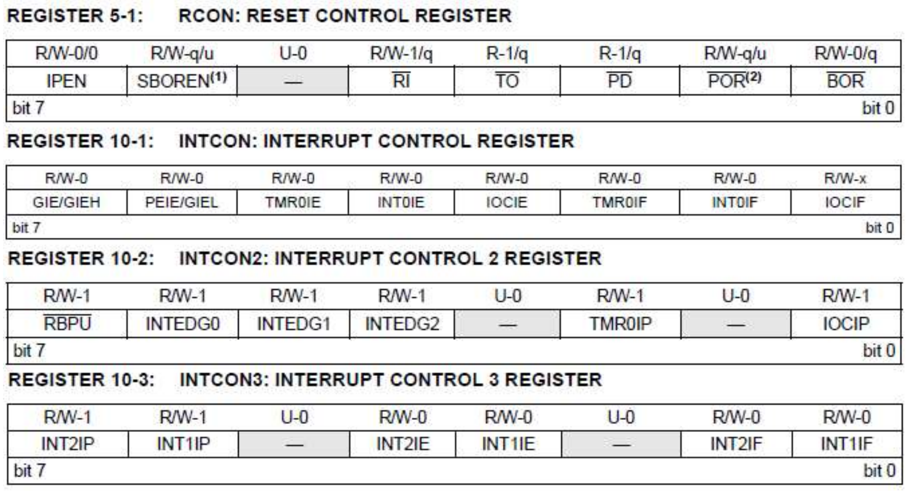
```
#include<xc.h>                   //Contais basic functions and micro register

#define _XTAL_FREQ 1000000       //Constant that defines de oscillator freq
                                 //Requiere when using the xc.h delay functions
void init_ports(void);           //Init all the ports
void interrupt_int0(void);       //Interrupt service routine low priority (INT0 input)
void interrupt_int2(void);       //Interrupt service routine high priority (INT2 input)

//We define the inputs and outputs with names to make program more readeable
#define BUTTON  PORTBbits.RB0    //Button is going to be assigned to RB0 (INT0)
#define BOX     PORTBbits.RB2    //Box sensor is assigned to RB2 (INT2)
#define MOTOR   PORTAbits.RA4    //Motor assigned to LED at port RA4
#define LIGHT   PORTAbits.RA5    //Light assigned to LED at port RA5
#define MAXIMUN     3            //Cantidad de cajas a contar

unsigned char counter = MAXIMUN ;   //Declaramos counter como global
```



```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Interrupt service routine for  INT0
//+ The interrupt INT0 is ALWAY hight priority
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 void __interrupt (high_priority) interrupt_int0(void){

        __delay_ms(20);         //Delay for debounce
        if(BUTTON == 1){        //Check if swith is stable at 1
            MOTOR = 1;          //Is fo, turn motor on
            LIGHT = 0;          //Turn off the light
        }
        //We clear the interrupt flag
        //You must do this ALWAYS at the end of the service routine if not the
        //"flat" will trigger anothe interrupt when PC returns to main program
        INTCONbits.INT0IF = 0;      //Clear the flag
 }

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Interrupt service routine for INT2
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void __interrupt (low_priority) interrupt_int2(void){
        __delay_ms(20);              //Delay
        if(BOX == 1) counter++;     //Check if stable, is fo increment counter
        INTCON3bits.INT2IF = 0;     //Clear the interrupt flag
 }
```



```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Main progarm
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void main(void){
        init_ports();                       //Init the port
        LIGHT = 1;                          //Turn on light
        //The following code inits all the interrupt
        //I will use XXXbits.XXX for clarity
        RCONbits.IPEN = 1;              //Enable priority (will have low and high)
        INTCON2bits.INTEDG0 = 1;        //INT0 (RB0) interrupts on positive transiti
        INTCON2bits.INTEDG2 = 1;        //INT2 (RB2) interrupts on positive transiti
        INTCON3bits.INT2IP = 0;         //The INT2 priority is low
        //The interrupt INT0 is always hight cannot be changed that is why
        //We dont have to configture this priority
        INTCONbits.INT0IF = 0;          //Always clear the interrupt flag INT0
        INTCON3bits.INT2IF = 0;         //Always clear the interrupt flag INT2
        INTCONbits.INT0IE = 1;          //Enable individual interrupt for INT0
        INTCON3bits.INT2IE = 1;         //Enable individual interrupt for INT2
        INTCONbits.GIEH = 1;            //Global enable for High prority interrupts
        INTCONbits.GIEL = 1;            //Global enable for Low prioiry interrupt
        // OK so we are done, the main loop is the folliwing
    while(1){
        if(counter == MAXIMUN){
            //Turn off the motor
            MOTOR = 0;
            //Turn on the light
            LIGHT = 1;
            //Set count to 0 again
            counter = 0;
        }
    }
}
```



REGISTER 5-1: RCON: RESET CONTROL REGISTER

R/W-0/0	R/W-q/u	U-0	R/W-1/q	R-1/q	R-1/q	R/W-q/u	R/W-0/q
IPEN	SBOREN[1]	—	RI	TO	PD	POR[2]	BOR
bit 7							bit 0

REGISTER 10-1: INTCON: INTERRUPT CONTROL REGISTER

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	IOCIE	TMR0IF	INT0IF	IOCIF
bit 7							bit 0

REGISTER 10-2: INTCON2: INTERRUPT CONTROL 2 REGISTER

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	—	IOCIP
bit 7							bit 0

REGISTER 10-3: INTCON3: INTERRUPT CONTROL 3 REGISTER

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

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the ports used
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_ports(void){
    //Output RA4 --> LED (motor)
    TRISAbits.TRISA4 = 0;   //Output this is always digital (see data sheet)

    //Output RA4 --> LIGHT (motor)
    TRISAbits.TRISA5 = 0;   //Output
    ANSELAbits.ANSA5 = 0;   //Digital
    //Entrada boton  RB0,
    TRISBbits.TRISB0 = 1;   //Entrada
    ANSELBbits.ANSB0 = 0;   //Digital
    //Entrada sensor RB2,
    TRISBbits.TRISB2 = 1;   //Entrada
    ANSELBbits.ANSB2 = 0;   //Digital

}
```