

**L'ELETTRONICA DI**  
***MR. A. KEER***

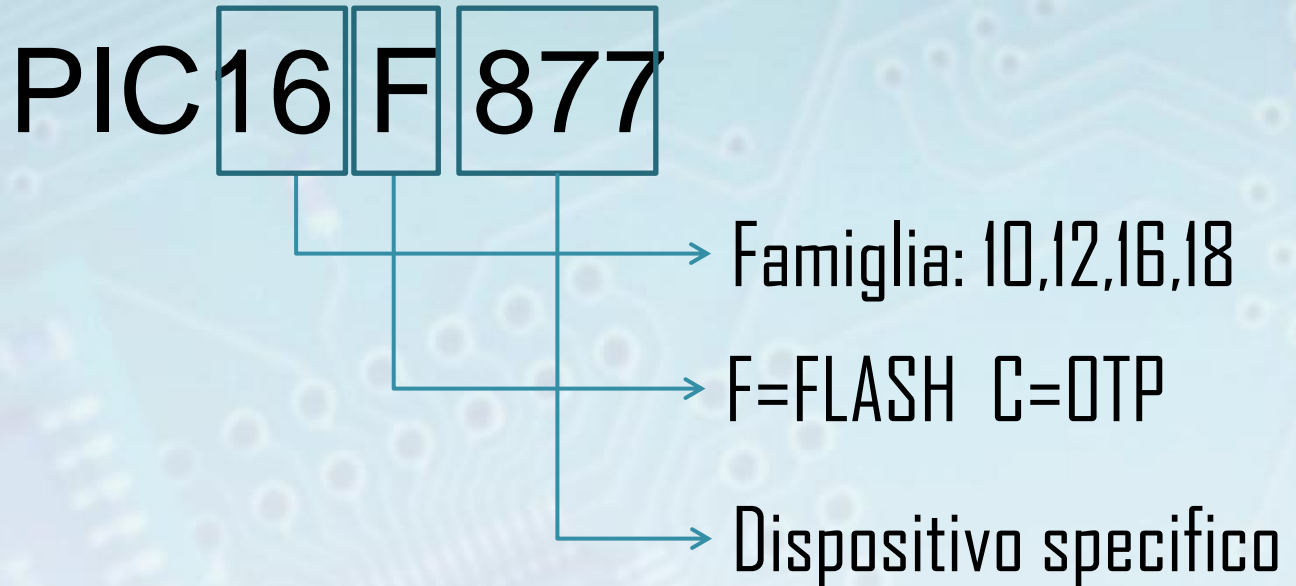


# **Il corso introduttivo per imparare a progettare con i microcontrollori PIC**

# COSA È UN MICROCONTROLLORE?

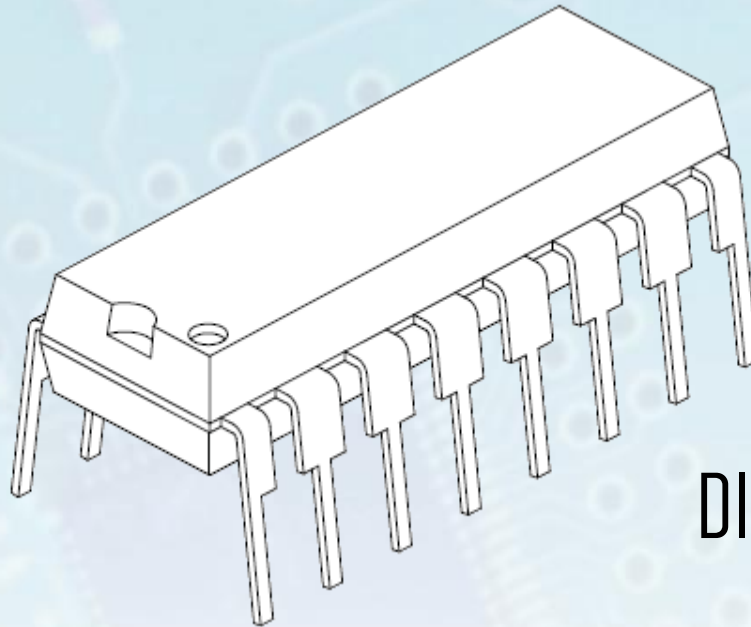
Un microcontrollore è un  
circuito integrato in grado  
di eseguire un programma

# La sigla identificativa



# I package

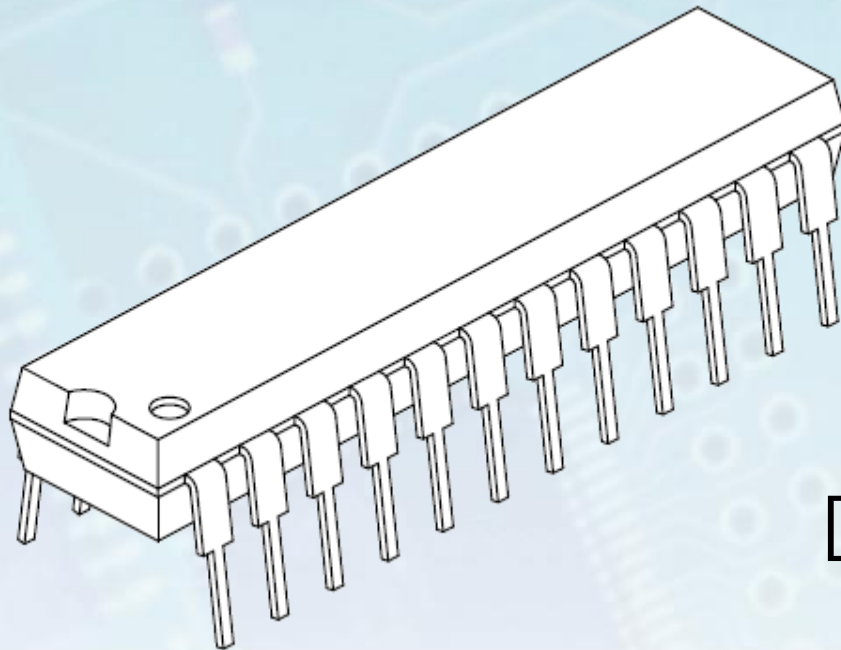
Lo stesso PIC potrebbe essere disponibile  
in diversi contenitori (package)



DIP 16pin

# I package

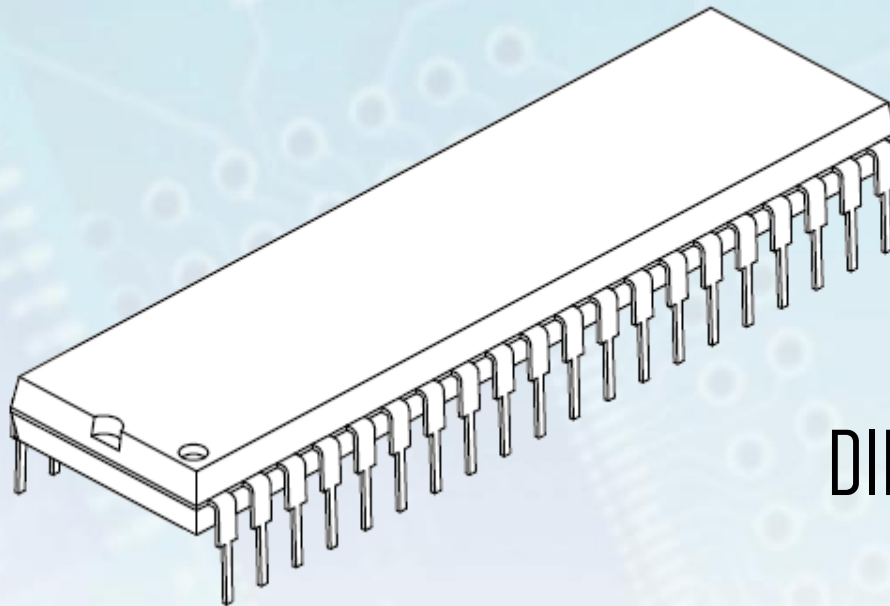
Lo stesso PIC potrebbe essere disponibile  
in diversi contenitori (package)



DIP 24pin

# I package

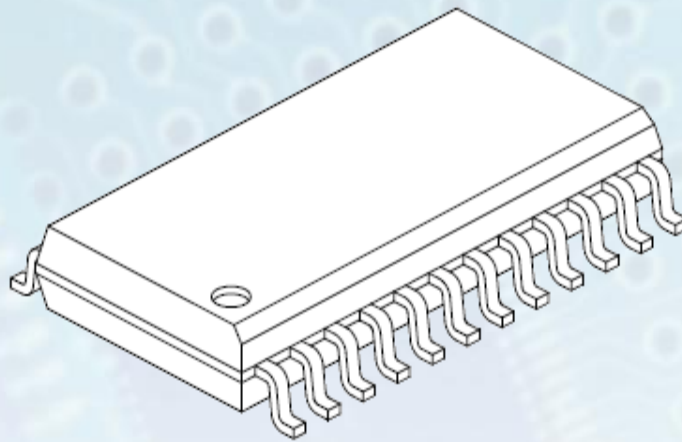
Lo stesso PIC potrebbe essere disponibile  
in diversi contenitori (package)



DIP 40pin

# I package

Lo stesso PIC potrebbe essere disponibile  
in diversi contenitori (package)

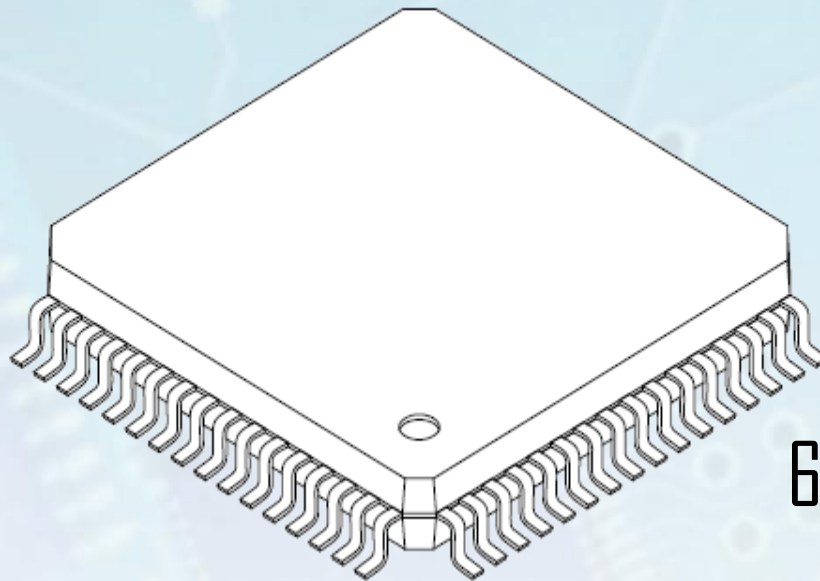


24pin SOIC



# I package

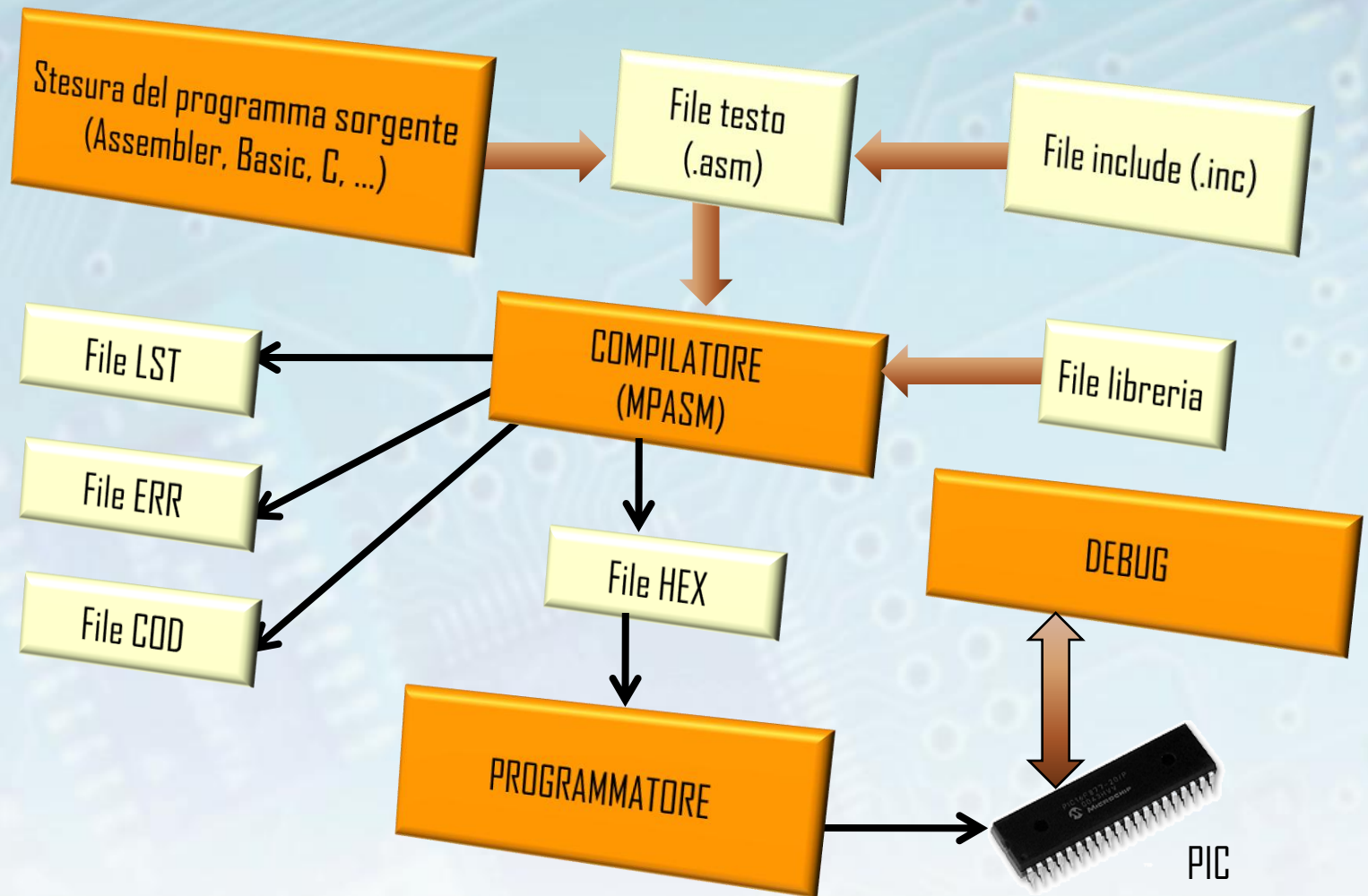
Lo stesso PIC potrebbe essere disponibile  
in diversi contenitori (package)



64pin MQFP



# Le fasi di un progetto



# Il software

Compilatore Assembler (MPASM)

+

Ambiente di sviluppo

=

MPLAB ([www.microchip.com](http://www.microchip.com))

# l'hardware

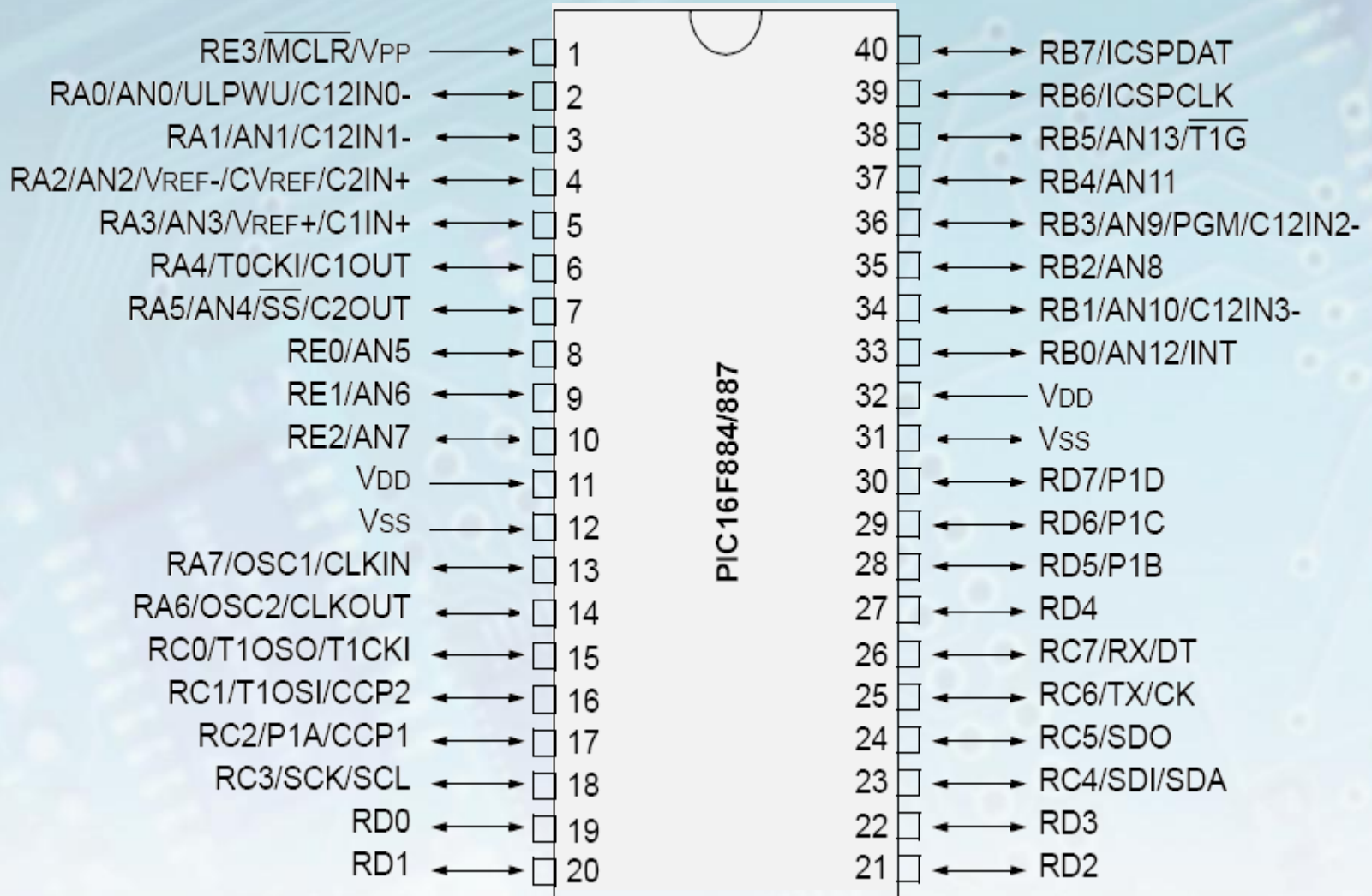
[www.ieshop.it/ep5](http://www.ieshop.it/ep5)

## easyPIC5





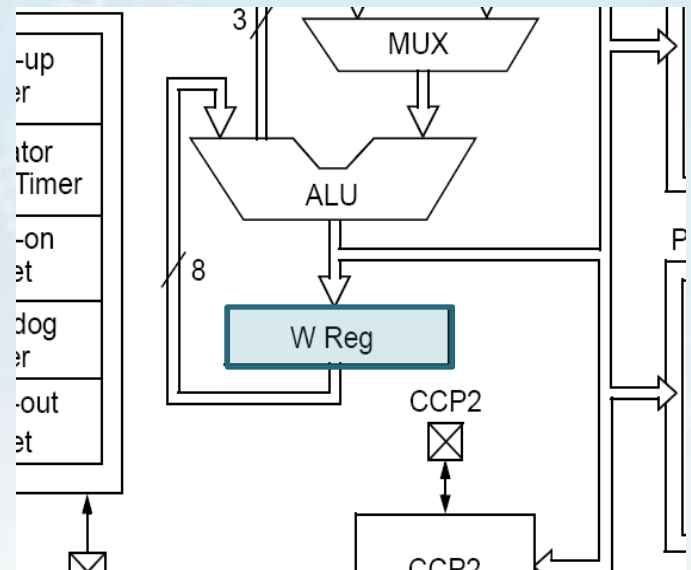
# L'architettura del PIC16F887



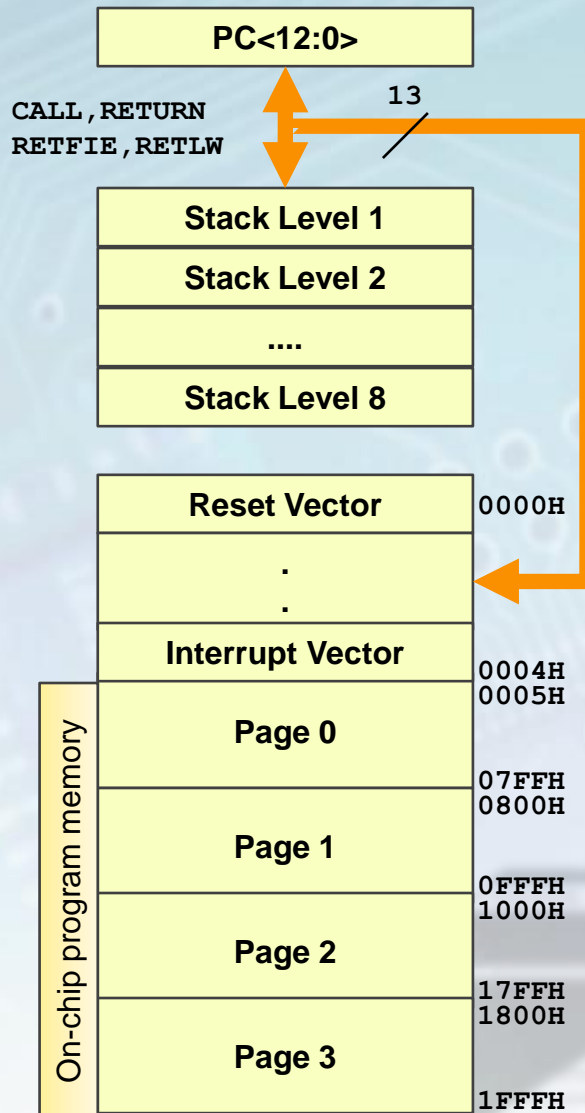
# L'architettura del PIC

- **Memoria programma**
  - Contiene il programma
  - Celle da 12/14/16bit
  - Modelli F e C (OTP)
- **Stack**
  - Memoria temporanea
  - Accesso con PUSH e POP
- **Porte I/O**
  - Diverse a seconda del PIC
- **Periferiche**
  - Diverse a seconda del PIC

- **Program counter**
  - L'indirizzo dell'istruzione in esecuzione
- **Accumulatore W**
  - Accesso alla ALU



# Memoria Programma (PIC16F887)



- Dimensione di 4Kx14bit
- Reset vector ha indirizzo 0000H  
È il valore del PC al reset
- Stack a 8 livelli
- Interrupt vector ha indirizzo 0004H  
Il PC salta a questo indirizzo quando si verifica una interruzione
- La memoria è ciclica  
Se PC contiene 1FFFH e si incrementa di 1 si salta alla locazione 0000H

- Suddivisa in Banchi selezionabili mediante i bit RPI e RPO del registro STATUS:

- | RP1:RP0 | Bank |
|---------|------|
| 00      | 0    |
| 01      | 1    |
| 10      | 2    |
| 11      | 3    |

File Address		File Address		File Address		File Address	
Indirect addr. <sup>(1)</sup>	00h	Indirect addr. <sup>(1)</sup>	80h	Indirect addr. <sup>(1)</sup>	100h	Indirect addr. <sup>(1)</sup>	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD <sup>(2)</sup>	08h	TRISD <sup>(2)</sup>	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEDADR	10Dh	EECON2 <sup>(1)</sup>	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEDADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h	General Purpose Registers  16 Bytes	110h	General Purpose Registers  16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADDD	93h		113h		193h
SSPICON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h		116h		196h
CCP1CON	17h	VRCONE	97h		117h		197h
RCSA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh	11Eh	19Eh		
ADCON0	1Fh	ADCON1	9Fh	11Fh	19Fh		
General Purpose Registers  96 Bytes	20h	General Purpose Registers  80 Bytes	A0h	General Purpose Registers  80 Bytes	120h	General Purpose Registers  80 Bytes	1A0h
	3Fh		EFh		16Fh		1EFh
	40h		F0h		170h		1F0h
	6Fh		FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	



# Registro STATUS

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

- RP1, RP0: Selezione del banco di memoria  
00=Banco 0  
01=Banco 1  
10=Banco 2  
11=Banco 3

# Registro STATUS

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

- Z: Zero Flag  
Se  $Z=1$  significa che l'operazione appena eseguita ha dato risultato nullo

# Registro STATUS

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

- C: bit CARRY/Borrow  
se C=1 significa che l'operazione di addizione ha generato un riporto  
  
se C=0 significa che l'operazione di sottrazione ha richiesto un "prestito"

# Le istruzioni Assembler

Una istruzione è composta da un codice mnemonico seguito da eventuali operandi.

Gli operandi possono essere registri o costanti numeriche.

Alcune istruzioni non richiedono operandi.

In totale vi sono 35 istruzioni comuni a tutti i PIC.

# Le istruzioni Assembler

**MOVLW k**

Carica il valore costante k nel registro accumulatore W.

**Esempio:**

**MOVLW 0x03**

Dopo questa istruzione si ha **w=00000011**

# Le istruzioni Assembler

## **MOVWF    f**

Sposta il valore contenuto in W nel registro f.

**Esempio:**

```
MOVLW    0x03
```

```
MOVWF    TRISA
```

La prima istruzione carica in W il valore 00000011

La seconda istruzione sposta questo valore nel registro TRISA quindi **TRISA=00000011**

# Le istruzioni Assembler

## **BCF    f , b**

Azzera il bit b nel registro f.

### **Esempio:**

```
MOVLW    0x03
```

```
MOVWF    TRISA
```

```
BCF       TRISA, 1
```

La prima istruzione carica in W il valore 00000011;

La seconda istruzione sposta questo valore nel registro TRISA quindi **TRISA=00000011**

La terza istruzione azzera il bit 1 (secondo bit) di TRISA quindi **TRISA=00000001**



# Le istruzioni Assembler

## BSF f, b

Pone ad 1 il bit b nel registro f.

### Esempio:

```
MOVLW 0x03
```

```
MOVWF TRISA
```

```
BSF TRISA, 2
```

La prima istruzione carica in W il valore 00000011;

La seconda istruzione sposta questo valore nel registro TRISA quindi **TRISA=00000011**

La terza istruzione mette a 1 il bit 2 (terzo bit) di TRISA quindi **TRISA=00000111**

# Le istruzioni Assembler

## BTFSC f, b

Analizza il bit b del registro f e se vale 0 salta l'istruzione successiva.

### Esempio:

```
MOVLW 0x03  
MOVWF TRISA  
BTFSC TRISA,2  
<Istruzione 1>  
<Istruzione 2>
```

La prima istruzione carica in W il valore 00000011;

La seconda istruzione sposta questo valore nel registro TRISA quindi  
**TRISA=00000011**

La terza istruzione fa sì che **<Istruzione 1>** non venga eseguita in quanto il bit 2 (terzo bit) di TRISA vale 0. Il programma salta quindi a **<Istruzione 2>**.

# Le istruzioni Assembler

## BTFSS f, b

Analizza il bit b del registro f e se vale 1 salta l'istruzione successiva.

**Esempio:**

```
MOVLW 0x03  
MOVWF TRISA  
BTFSS TRISA,1  
<Istruzione 1>  
<Istruzione 2>
```

La prima istruzione carica in W il valore 00000011;

La seconda istruzione sposta questo valore nel registro TRISA quindi  
**TRISA=00000011**

La terza istruzione fa sì che **<Istruzione 1>** non venga eseguita in quanto il bit 1 (secondo bit) di TRISA vale 1. Il programma salta quindi a **<Istruzione 2>**.

# Le istruzioni Assembler

## CLRF

Azzera il contenuto di un registro.

**Esempio:**

```
CLRF    W
```

Dopo questa istruzione W contiene il valore 00000000.

# Le istruzioni Assembler

## DECFSZ    f, d

Decrementa di 1 il contenuto del registro f e se il risultato è 0 salta l'istruzione successiva. L'argomento d specifica dove andrà posizionato il risultato del decremento: d=0 il risultato del decremento andrà in W altrimenti viene sovrascritto nel registro f.

### Esempio:

```
MOVLW 0x01  
MOVWF TRISA  
DECFSZ TRISA,1  
<Istruzione 1>  
<Istruzione 2>
```

Con le prime due istruzioni si ottiene TRISA=00000001

La terza istruzione fa sì che <Istruzione 1> non venga eseguita in quanto dopo il decremento TRISA vale 0. Il programma salta quindi a <Istruzione 2>.

# Le istruzioni Assembler

## CALL pippo

Chiama la subroutine *pippo*. Il programma salta all'etichetta *pippo* e continua da lì l'esecuzione. Torna all'origine del salto quando esegue l'istruzione **RETURN**

### Esempio:

```
CALL  accendi
```

```
.....
```

```
accendi
```

```
BSF  PORTB,1
```

```
RETURN
```

La **CALL** fa saltare il programma all'istruzione **BSF**.

La **RETURN** riporta il programma all'istruzione successiva alla **CALL**.



# Le istruzioni Assembler

## RLF f, d

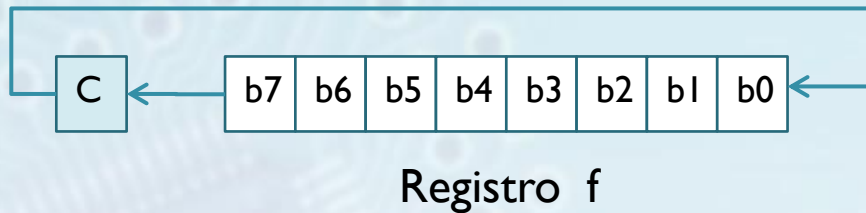
Provoca lo scorrimento del registro f, di un bit verso sinistra. Se d=0 il risultato dello scorrimento sarà messo in W altrimenti rimarrà nel registro f. Nello scorrimento viene coinvolto il bit C del registro STATUS.

### Esempio:

```
MOVLW 0x01
```

```
MOVWF TRISA
```

```
RLF TRISA,1
```



Dopo queste istruzioni il registro TRISA conterrà il valore 000000010.



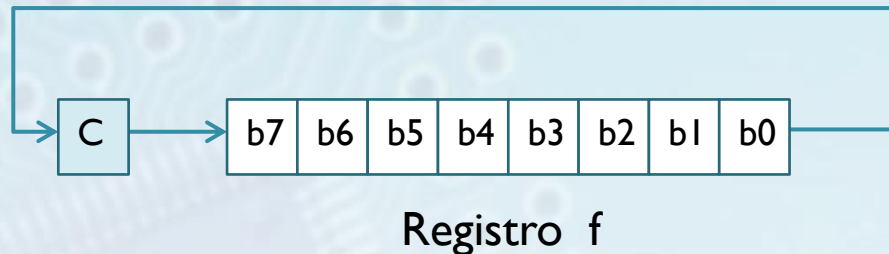
# Le istruzioni Assembler

## RRF f, d

Provoca lo scorrimento del registro f, di un bit verso destra. Se d=0 il risultato dello scorrimento sarà messo in W altrimenti rimarrà nel registro f. Nello scorrimento viene coinvolto il bit C del registro STATUS.

### Esempio:

```
MOVLW 0x02  
MOVWF TRISA  
RRF TRISA,1
```



Dopo queste istruzioni il registro TRISA conterrà il valore 00000001.

# Le porte I/O sul 16F887

- 35 porte I/O suddivise in: PORTA, B, C, D, E
- Possono essere configurate come ingresso o come uscita
- I registri TRIS determinano se la porta è un ingresso o una uscita
- I registri PORT determinano il valore di uscita o contengono il valore di ingresso
- Alcune porte hanno molteplici funzioni  
(es: RA0 può essere un ingresso del convertitore AD)

# I registri TRIS

I registri TRIS determinano la direzionalità della porta:

- 0=uscita; 1=ingresso
- Ogni bit del registro corrisponde allo stesso bit della porta
- Esempio: TRISA=00001100 significa che RA2 e RA3 sono ingressi e il resto sono uscite
- I registri TRIS si trovano nel banco1 di memoria

# I registri PORT

I registri PORT gestiscono direttamente la porta dal punto di vista dello stato della porta stessa:

- Esempio: se la porta B è impostata come uscita,  $PORTB=00000001$  mette a livello alto solo RB0
- Esempio: se la porta B è impostata come ingresso, applicando un livello alto a RB1 e leggendo la porta si ottiene  $PORTB=00000010$
- I registri PORT si trovano nel banco 0 di memoria allo stesso indirizzo dei registri TRIS. Selezionate il banco giusto!!

# Esempio: LED lampeggiante

Processore utilizzato

Notazione numerica

Definizione dei registri

Indirizzo di memoria

RP0=1 (banco 1)

Solo RC0 è una uscita

RP0=0 (banco 0)

RC0=1 (LED acceso)

Ritardo

Il LED era acceso?

Si, allora spegnilo

No, allora accendilo

Spegne il LED

Torna all'inizio

Ritardo

FINE

PROCESSOR 16F887

RADIX DEC

INCLUDE "P16F887.INC"

LED EQU 0  
ORG 20H  
Count RES 2  
ORG 00H  
bsf STATUS,RP0  
movlw 00000000B  
movwf TRISC  
bcf STATUS,RP0  
bsf PORTC,LED

MainLoop  
call Delay  
btfsc PORTC,LED  
goto SetToZero  
bsf PORTC,LED  
goto MainLoop

SetToZero  
bcf PORTC,LED  
goto MainLoop

; Subroutines

Delay  
clrf Count  
clrf Count+1

DelayLoop  
decfsz Count,1  
goto DelayLoop  
decfsz Count+1,1  
goto DelayLoop

return  
END

# Esempio: LED lampeggiante

## Direttive per il compilatore

```
PROCESSOR 16F887
RADIX      DEC

INCLUDE     "P16F887.INC"

LED         EQU          0
Count       RES          2
            ORG          00H

MainLoop    bsf           STATUS,RP0
            movlw        00000000B
            movwf        TRISC
            bcf          STATUS,RP0
            bsf          PORTC,LED

            call         Delay
            btfsc        PORTC,LED
            goto         SetToZero
            bsf          PORTC,LED
            goto         MainLoop

SetToZero   bcf          PORTC,LED
            goto         MainLoop

; Subroutines
Delay       clrfsz        Count
            clrfsz        Count+1

DelayLoop   decfsz        Count,1
            goto         DelayLoop
            decfsz        Count+1,1
            goto         DelayLoop

            return
END
```

# Esempio: LED lampeggiante

Programma  
principale

```
PROCESSOR 16F887
RADIX      DEC

INCLUDE    "P16F887.INC"

LED        EQU        0
Count      RES        2
           ORG        00H

MainLoop   bsf         STATUS,RP0
           movlw       00000000B
           movwf       TRISC
           bcf         STATUS,RP0
           bsf         PORTC,LED

           call        Delay
           btfsc       PORTC,LED
           goto        SetToZero
           bsf         PORTC,LED
           goto        MainLoop

SetToZero  bcf         PORTC,LED
           goto        MainLoop

; Subroutines
Delay      clrfsz      Count
           clrfsz      Count+1

DelayLoop  decfsz      Count,1
           goto        DelayLoop
           decfsz      Count+1,1
           goto        DelayLoop

           return
END
```



# Esempio: LED lampeggiante

```
PROCESSOR 16F887
RADIX      DEC

INCLUDE    "P16F887.INC"

LED        EQU        0
Count      RES        2
ORG        00H
bsf        STATUS,RP0
movlw      00000000B
movwf      TRISC
bcf        STATUS,RP0
bsf        PORTC,LED

MainLoop   call        Delay
           btfsc       PORTC,LED
           goto        SetToZero
           bsf         PORTC,LED
           goto        MainLoop

SetToZero   bcf        PORTC,LED
           goto        MainLoop
```

## Subroutines

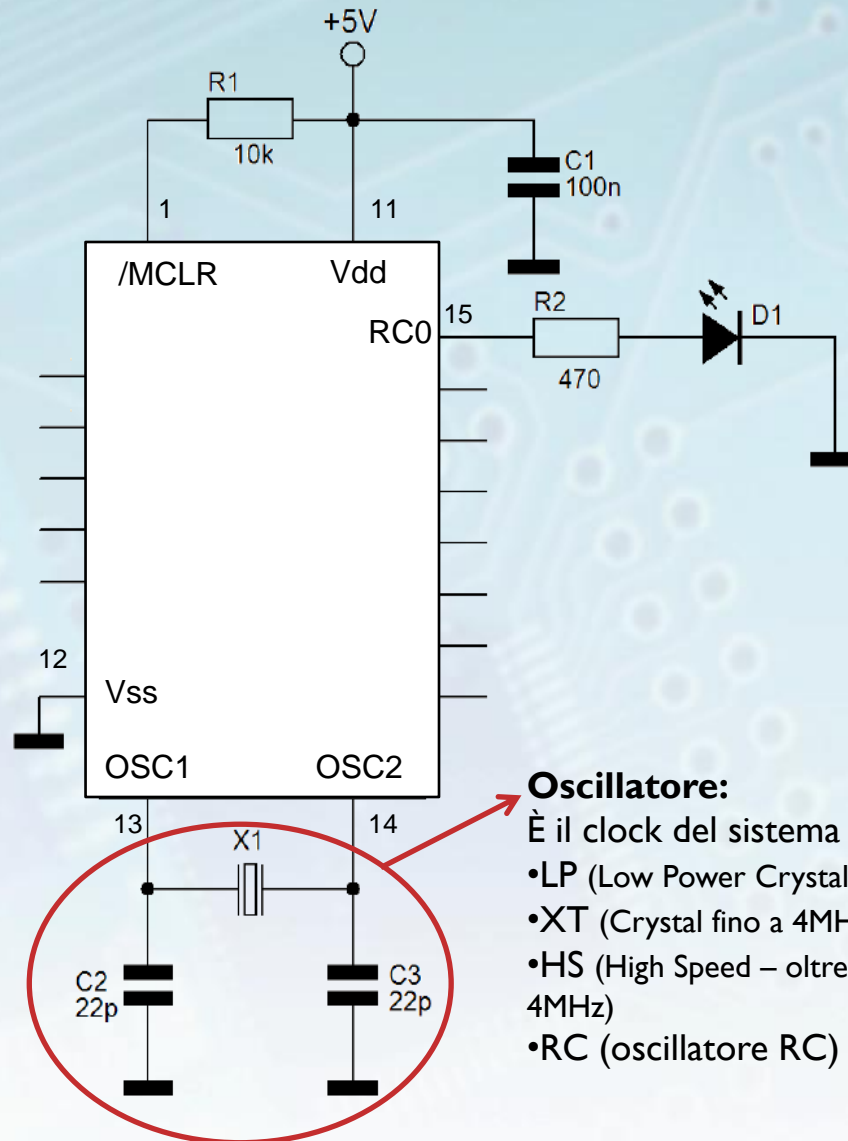
```
; Subroutines
Delay
           clrf        Count
           clrf        Count+1

DelayLoop  decfsz      Count,1
           goto        DelayLoop
           decfsz      Count+1,1
           goto        DelayLoop

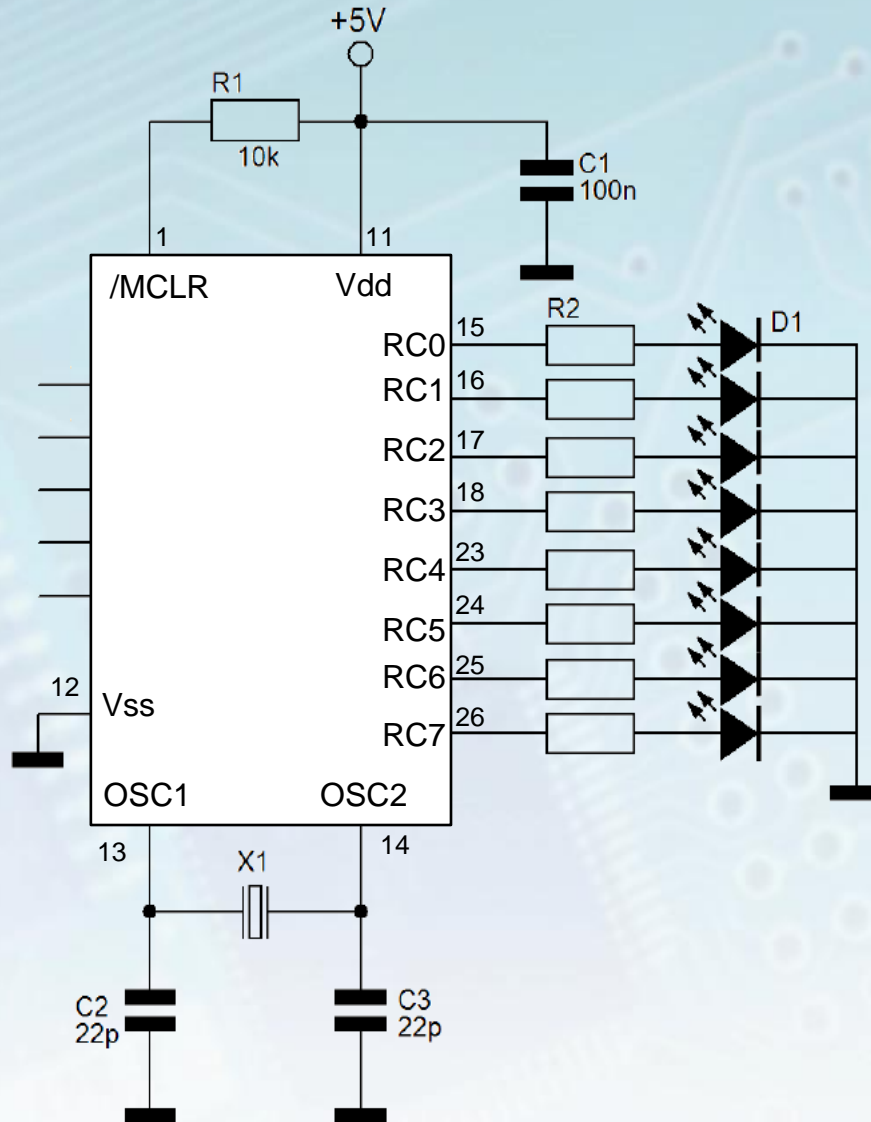
           return

END
```

# Esempio: LED lampeggiante



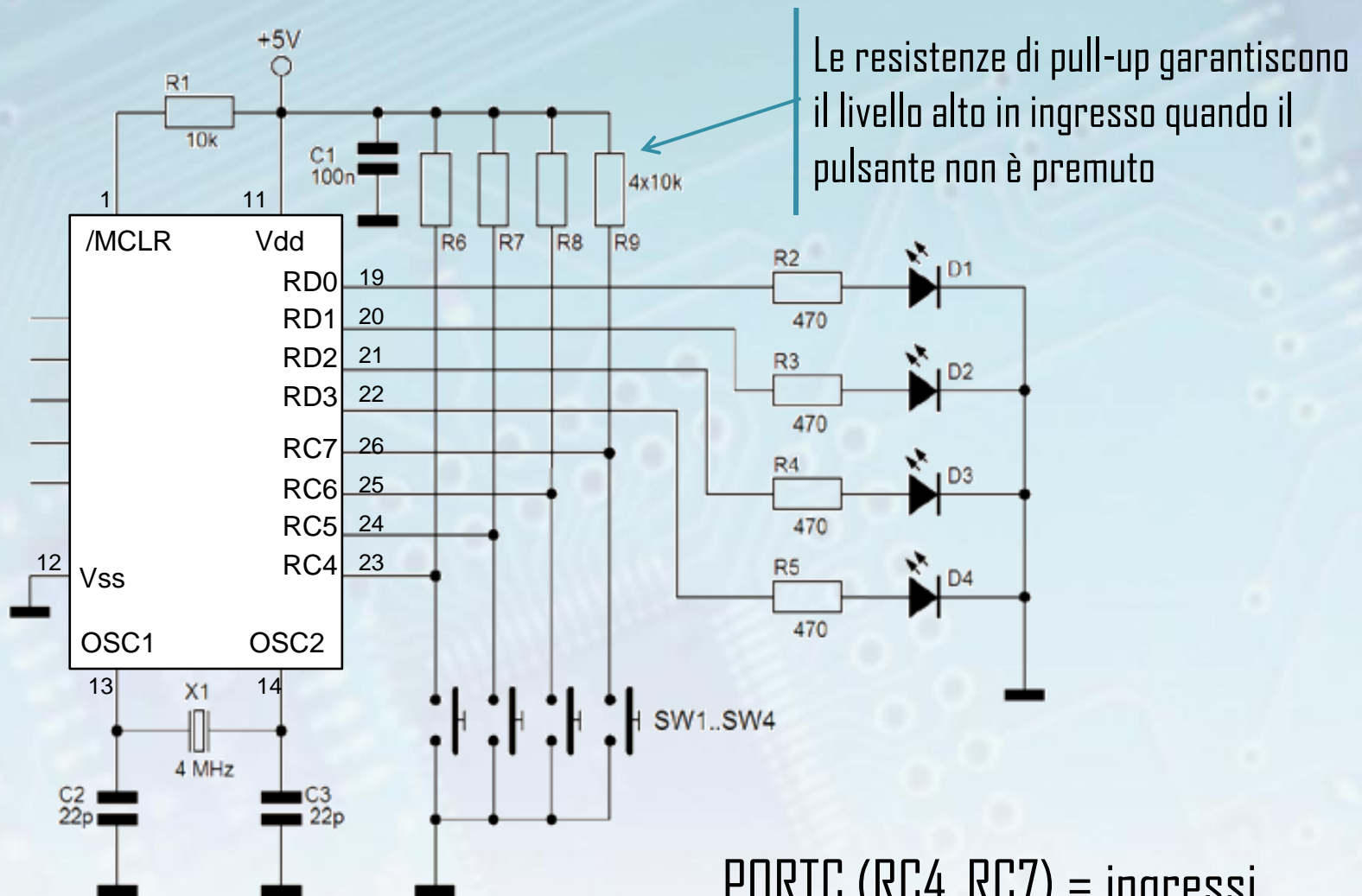
# Esempio: LED in sequenza



# Esempio: LED in sequenza

	PROCESSOR <b>16F887</b> RADIX <b>DEC</b> INCLUDE "P16F887.INC"  Count      ORG           20H Shift      RES           2 RES           1 ORG           00H
PORTC come uscita	bsf           STATUS,RP0 movlw       00000000B movwf       TRISC bcf           STATUS,RP0
W=00000001 Shift=W	movlw       00000001B movwf       Shift
PORTC=W	MainLoop movf           Shift, W movwf       PORTC
Trasla Shift di un bit a destra	rrf           Shift, F
Ritardo	call         Delay goto        MainLoop  ; Subroutines Delay clrf           Count clrf           Count+1 DelayLoop decfsz       Count,1 goto        DelayLoop decfsz       Count+1,1 goto        DelayLoop  return END

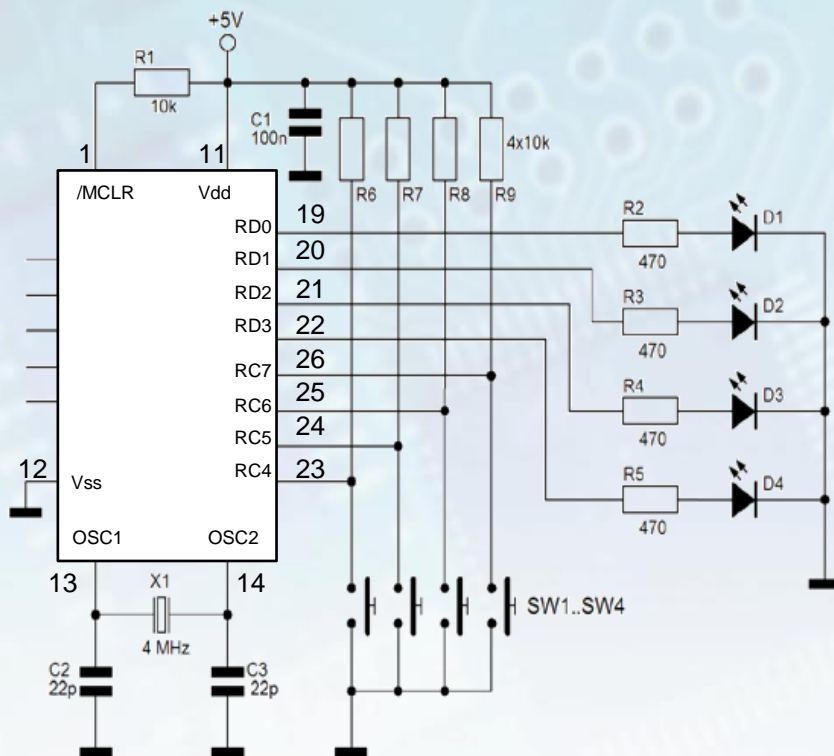
# Esempio: uso degli ingressi



PORTC (RC4..RC7) = ingressi  
PORTD (RD0..RD7) = uscite

# Esempio: uso degli ingressi

Il programma legge ciclicamente lo stato degli ingressi e accende il LED corrispondente al pulsante premuto:  
RC4-RD0, RC5-RD1, RC6-RD2, RC7-RD3



PROCESSOR **16F887**

RADIX **DEC**

INCLUDE **"P16F887.INC"**

**LED1** EQU 0

**LED2** EQU 1

**LED3** EQU 2

**LED4** EQU 3

**SW1** EQU 4

**SW2** EQU 5

**SW3** EQU 6

**SW4** EQU 7

ORG 00H

**bsf** STATUS,RP0

**movlw** 11110000B

**movwf** TRISC

**movlw** 11110000B

**movwf** TRISD

**bcf** STATUS,RP0

**MainLoop**

**clrf** PORTD

**btfss** PORTC,SW1

**bsf** PORTD,LED1

**btfss** PORTC,SW2

**bsf** PORTD,LED2

**btfss** PORTC,SW3

**bsf** PORTD,LED3

**btfss** PORTC,SW4

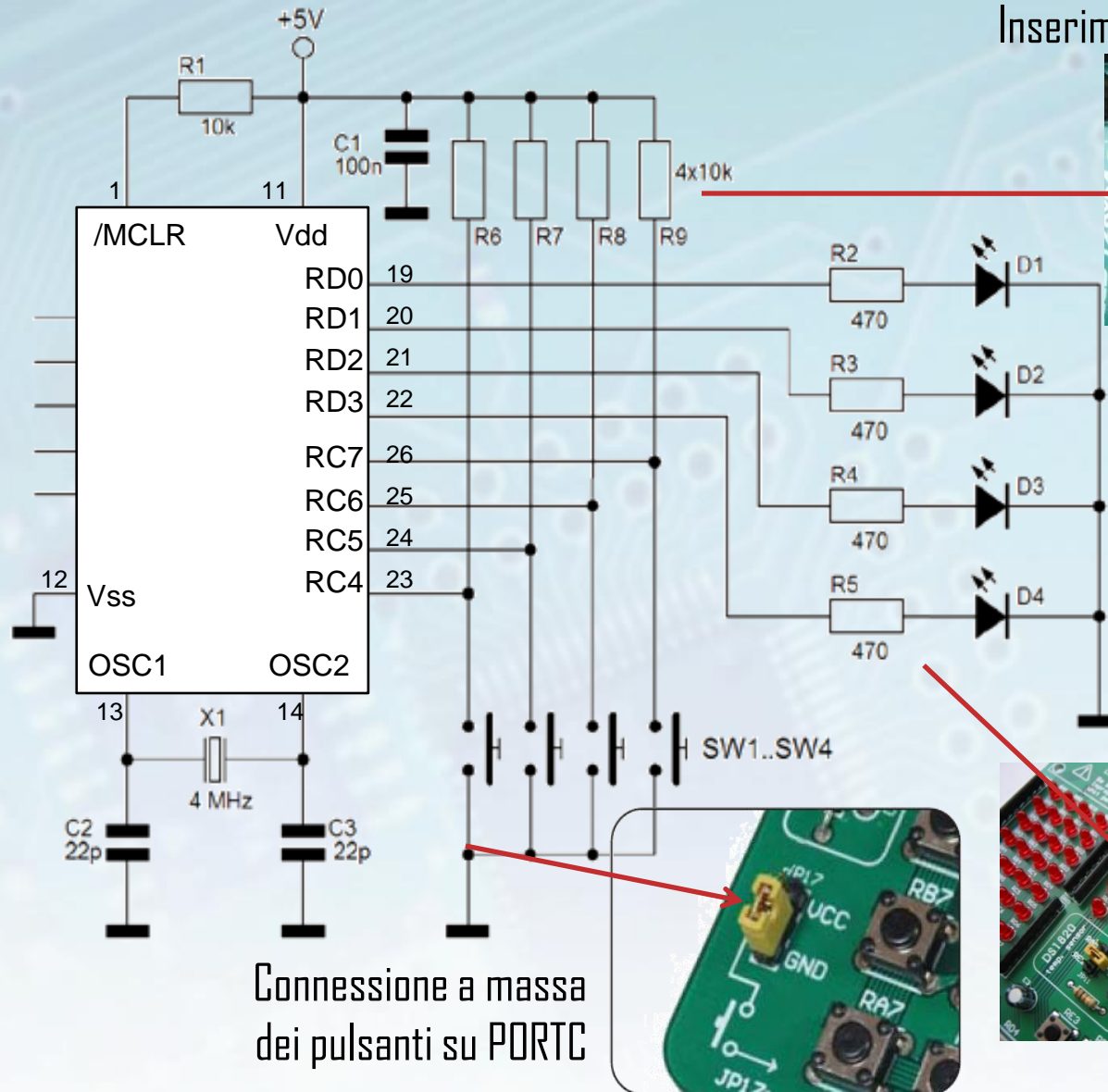
**bsf** PORTD,LED4

**goto** MainLoop

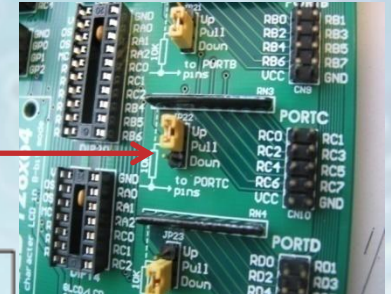
END



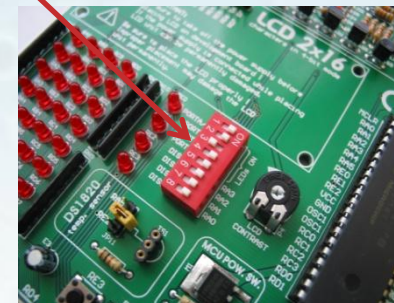
# Esempio: uso degli ingressi configurazione della scheda



Inserimento pull-up su PORTC



Disattivazione dei LED  
sulla PORTC e  
attivazione LED  
su PORTD



Connessione a massa  
dei pulsanti su PORTC

# La periferica Timer

Il PIC16F887 possiede tre periferiche TIMER:

- TIMER0: 8bit
- TIMER1: 16bit
- TIMER2: 8bit

**T0SE:** TMR0 Source Edge Select bit

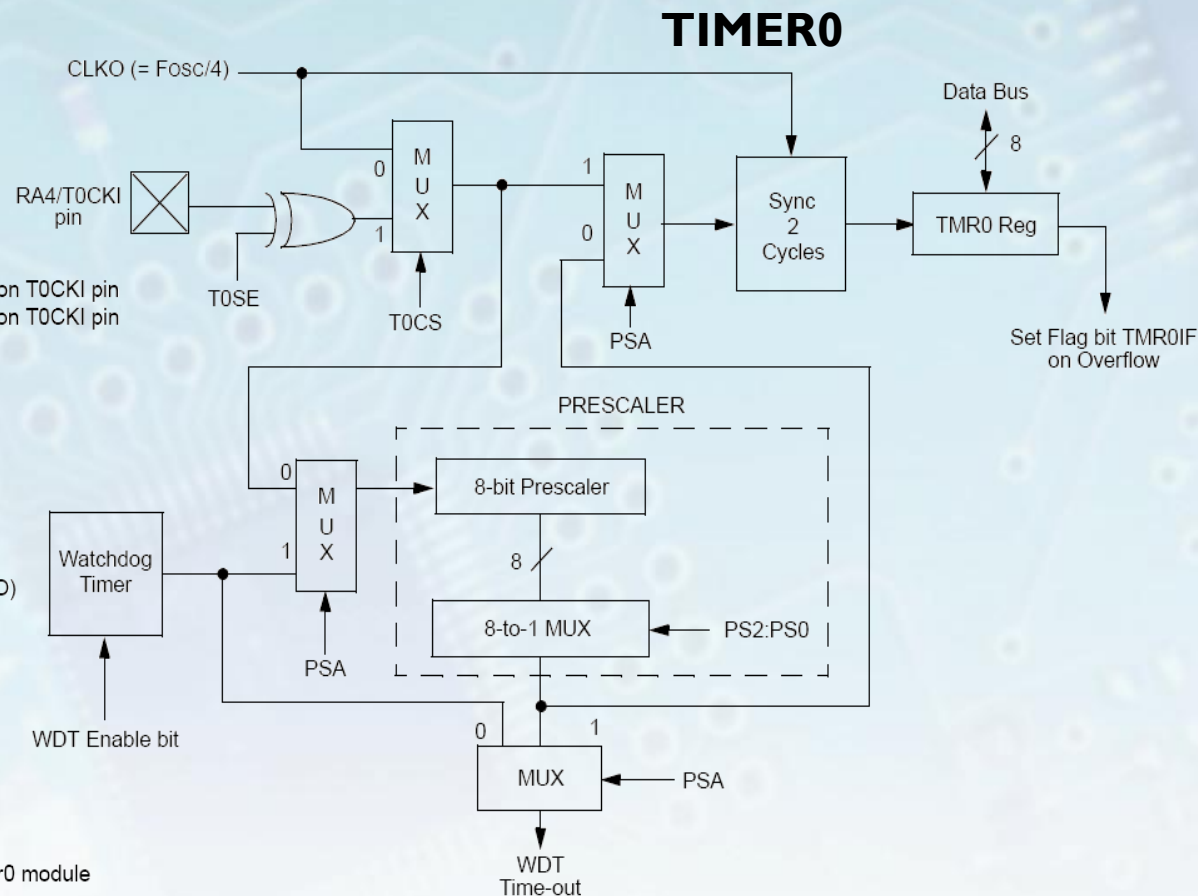
- 1 = Increment on high-to-low transition on T0CKI pin
- 0 = Increment on low-to-high transition on T0CKI pin

**T0CS:** TMR0 Clock Source Select bit

- 1 = Transition on T0CKI pin
- 0 = Internal instruction cycle clock (CLKO)

**PSA:** Prescaler Assignment bit

- 1 = Prescaler is assigned to the WDT
- 0 = Prescaler is assigned to the Timer0 module



**Note:** T0CS, T0SE, PSA, PS2:PS0 are (OPTION\_REG<5:0>).

# LA PERIFERICA TIMER

## il prescaler

- Permette di inserire un "fattore di scala" nel conteggio del timer
- È condiviso tra timer0 e watchdog ed è mutuamente esclusivo
- Si assegna mediante il bit PSA del registro OPTION\_REG
- Si imposta mediante i bit PS2:PS0 del registro OPTION\_REG

**PS2:PS0:** Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

**Esempio:**

Quarzo a 8MHz (Fosc)

PS2:PS0=110 (1:128)

TMR0 cambia valore con la frequenza di  
(8M:4):128=15,625KHz  
ovvero ogni 64 microsecondi

L'interrupt avviene ogni  
 $64 \times 256 = 16,38$  millisecondi

### OPTION\_REG REGISTER

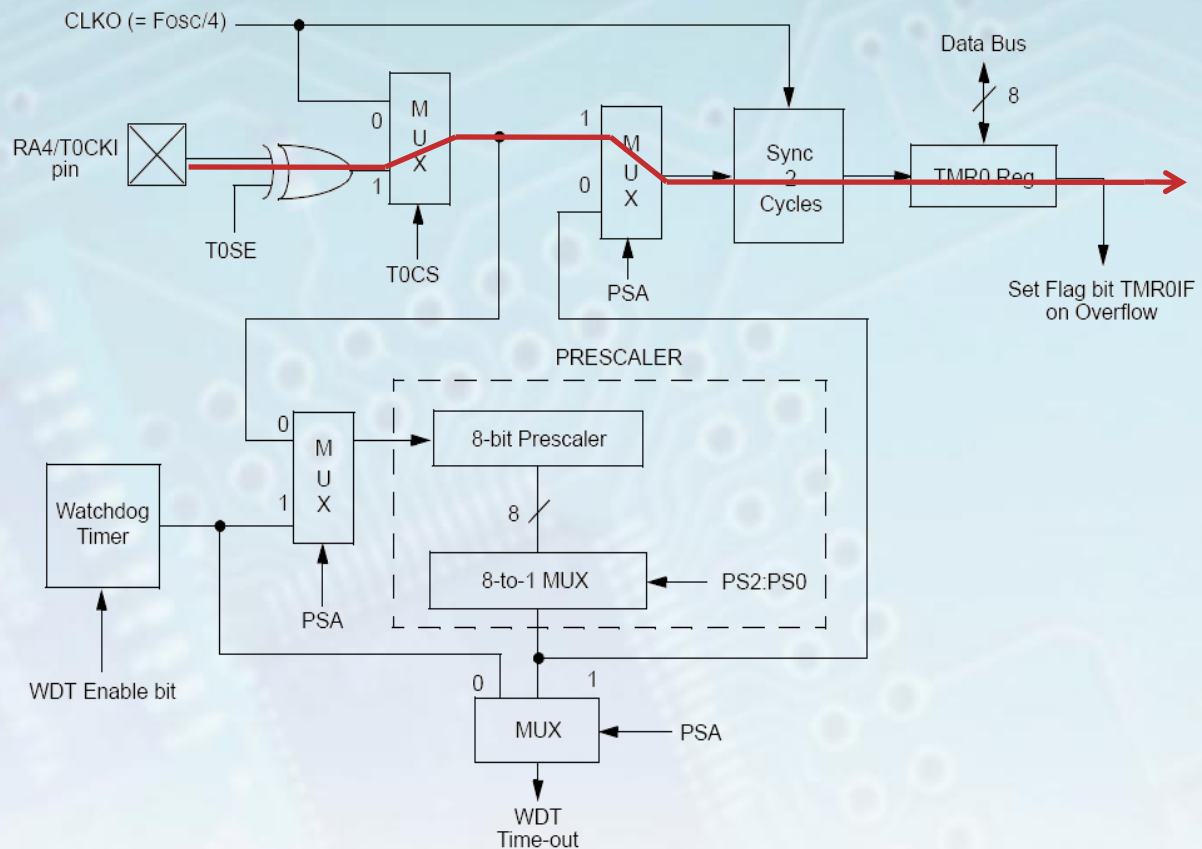
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

bit 7

bit 0

# La periferica Timer

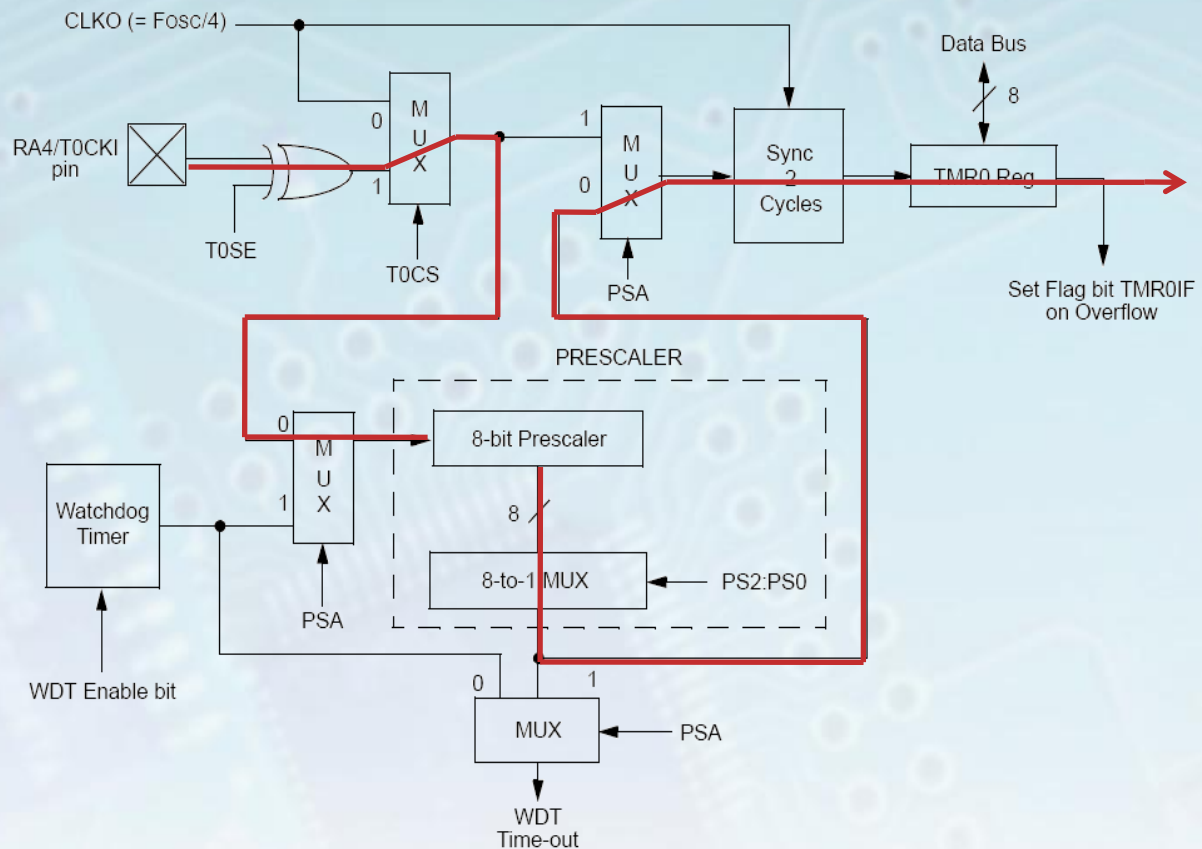
Esempio: T0CS=1 (clock esterno) PSA=1 (no prescaler)



**Note:** T0CS, T0SE, PSA, PS2:PS0 are (OPTION\_REG<5:0>).

# La periferica Timer

Esempio: T0CS=1 (clock esterno) PSA=0 (prescaler)



**Note:** T0CS, T0SE, PSA, PS2:PS0 are (OPTION\_REG<5:0>).



# Esempio: uso del Timer0

Obiettivo:

Modificare il programma delle luci in sequenza per cambiare l'uscita ogni secondo usando TMRO

Modifiche da fare:

- inizializzare il registro `OPTION_REG` per l'uso del TMRO con prescaler
- rivedere la routine `delay` per il calcolo corretto dei tempi



# Esempio: uso del Timer0

## calcolo dei ritardi

Un modo per ottenere un ritardo di 1 secondo potrebbe essere il seguente:

A partire da un quarzo da 8MHz, generiamo mediante il prescaler una frequenza di 31,25KHz:

$F_{osc} = 8\text{MHz}$

Prescaler=1:64

$$8\text{M} / 4 / 64 = 31250\text{Hz}$$

$F_{osc}/4$       Prescaler

La frequenza ottenuta la dividiamo per 250 inizializzando TMRO a 6 ( $256-6=250$ ):

$$31250\text{Hz} / 250 = 125\text{Hz}$$

Il risultato lo dividiamo per 125 usando la variabile Count:

$$125 / 125 = 1\text{Hz}$$

# Esempio: uso del Timer0

## calcolo dei ritardi

Inizializzazione di TMRO

Inizializzazione di Count

TMRO=0 ?

No, ripeti il ciclo

Sì, inizializza nuovamente TMRO

Decrementa Count e ripeti il ciclo finché Count=0

```
Delay
    movlw    6
    movwf    TMRO

    movlw    125
    movwf    Count

DelayLoop
    movf     TMRO,W
    btfss    STATUS,Z
    goto     DelayLoop

    movlw    6
    movwf    TMRO

    decfsz   Count,1
    goto     DelayLoop

    return

END
```

# Esempio: uso del Timer0

## inizializzazione di OPTION\_REG

Il registro OPTION\_REG deve essere inizializzato per usare TMRO con prescaler 1:64 e clock interno

### OPTION\_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7		↑ 0		↑ 0	↑ 1	↑ 0	↑ 1 bit 0
		Clock interno		Prescaler abilitato	Prescaler = 1:64		

# Esempio: uso del Timer0

## il programma

PORTC come uscita

Inizializzazione OPTION\_REG

```
PROCESSOR 16F887
RADIX DEC
ERRORLEVEL -302
INCLUDE "P16F887.INC"

ORG 20H

Count    RES    1
Shift    RES    1

ORG      00H

    bsf    STATUS,RPO
    movlw  00000000B
    movwf  TRISC

    movlw  00000101B
    movwf  OPTION_REG

    bcf    STATUS,RPO

    movlw  00000001B
    movwf  Shift

MainLoop
    movf   Shift,W
    movwf  PORTC
    rlf    Shift,F
    call   Delay
    goto   MainLoop
```

# Le interruzioni

Il meccanismo delle interruzioni permette di gestire gli eventi all'occorrenza.

Esempio: Il campanello che suona notifica l'arrivo di una persona

Esistono varie sorgenti di interruzione sia esterne che dovute alle periferiche interne

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- GIE (Global Interrupt Enable)  
GIE=1 Abilitazione globale delle interruzioni



# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- PEIE (Peripheral Interrupt Enable)

PEIE=1 Abilitazione globale delle interruzioni per le periferiche interne

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- TOIE (Timer0 Interrupt Enable)

TOIE=1 Abilitazione delle interruzioni  
per il Timer 0

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- INTE (INT external interrupt Enable)

INTE=1 Abilitazione delle interruzioni esterne

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- TOIF (Timer0 Interrupt Flag)

TOIF=1 l'interruzione arriva dal Timer0

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- INTF (INT external interrupt Flag)

INTF=1 l'interruzione arriva dall'esterno

# Le interruzioni

## il registro INTCON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

- RBIF (PortB change Interrupt Flag)

RBIF=1 un bit della porta B ha cambiato stato



# Le interruzioni

Ogni sorgente di interruzione ha un bit di abilitazione (E) e un flag di segnalazione (F).

Al verificarsi di una interruzione viene salvato il contesto e viene caricato l'indirizzo 04h nel PC (il che significa che il programma salta all'indirizzo 04h).

L'indirizzo 04h è detto "Interrupt Vector"

Alla fine dell'interruzione la routine deve terminare con l'istruzione IRET che ripristina il precedente valore del PC.

# Le interruzioni

## Il contesto

Il contesto è lo stato del sistema.

Prima di eseguire una interruzione lo stato deve essere salvato in modo da essere ripristinato al momento dell'uscita dell'interruzione.

Il contesto consiste in:

- registro accumulatore W
- Registro STATUS
- Registro PCLATH (tiene conto della pagina di memoria)

# Le interruzioni

## Esempio di salvataggio del contesto

```
MOVWF    W_TEMP        ;Copy W to TEMP register
SWAPF    STATUS,W       ;Swap status to be saved into W
CLRF     STATUS         ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
MOVWF    STATUS_TEMP    ;Save status to bank zero STATUS_TEMP register
MOVF     PCLATH, W       ;Only required if using pages 1, 2 and/or 3
MOVWF    PCLATH_TEMP    ;Save PCLATH into W
CLRF     PCLATH         ;Page zero, regardless of current page
:
: (ISR)                 ;(Insert user code here)
:
MOVF     PCLATH_TEMP, W  ;Restore PCLATH
MOVWF    PCLATH         ;Move W into PCLATH
SWAPF    STATUS_TEMP,W  ;Swap STATUS_TEMP register into W
                        ;(sets bank to original state)
MOVWF    STATUS         ;Move W into STATUS register
SWAPF    W_TEMP,F       ;Swap W_TEMP
SWAPF    W_TEMP,W       ;Swap W_TEMP into W
```

# Le interruzioni

## Tipi di interruzione

Come determino chi ha scatenato l'interruzione?

All'indirizzo 04h deve trovarsi un programma in grado di:

- Disabilitare le interruzioni
- Salvare il contesto
- Determinare la provenienza dell'interruzione
- Eseguire la routine opportuna per la gestione dell'interruzione
- Abilitare le interruzioni
- Ripristinare il contesto

Per determinare la provenienza dell'interruzione si deve analizzare il registro INTCON per verificare quali flag sono attivi.

Si devono quindi azzerare tali flag e saltare alla routine di gestione dell'interrupt

# Le interruzioni

## ISR

```
; ***salvataggio del contesto***
```

Si salva il contesto  
(come già visto)

```
btfs    INTCON, RBIF
```

Interruzione da PORTB?

```
goto    ServizioRB_INT
```

Si, gestiscila opportunamente

```
btfs    INTCON, T0IF
```

Interruzione da TMR0?

```
goto    ServizioTMR0_INT
```

Si, gestiscila opportunamente

```
...    ...
```

```
.
```

```
.
```

```
.
```

Verifica degli altri tipi di interruzione e salto alla routine opportuna

```
ServizioRB_INT
```

```
bcf     INTCON, RBIE
```

Gestione interruzione da PORTB

```
bcf     INTCON, RBIF
```

Disabilita interruzioni su PORTB

```
.....
```

```
; ***Ripristino del contesto***
```

Azzeramento del flag di interruzione

```
retfie
```

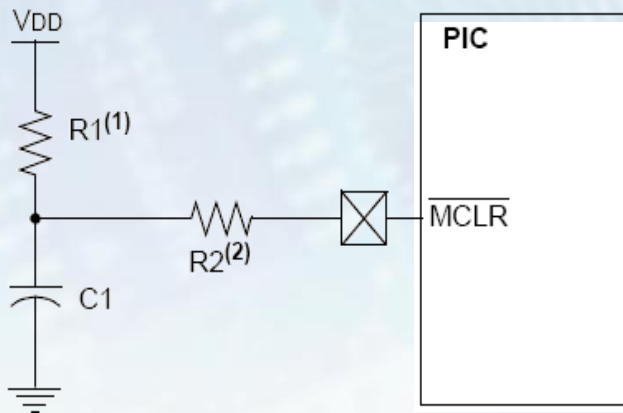
Ripristino del contesto (già visto)

Ritorno da interrupt

# II RESET

Dopo un reset tutti i registri si trovano in uno stato consistente  
**POR (Power-On-Reset)**

- Si verifica all'accensione in corrispondenza di un fronte di salita della tensione di alimentazione (tra 1,2 e 1,7V)
- è necessario mantenere /MCLR a livello alto con un pull-up
- È bene inserire una rete RC su /MCLR



- PWRT (Power-Up Timer)
  - Se è attivo introduce un ritardo di 72ms prima di iniziare l'esecuzione del programma. Durante questo tempo il micro è mantenuto nello stato di reset.
- OST (Oscillator Startup Timer)
  - Se è attivo introduce un ritardo di 1024 cicli dopo il PWRT per garantire che l'oscillatore sia a regime.



# II RESET

## **BOR (Brown-Out-Reset)**

Se questa funzionalità è abilitata, si verifica un reset se la tensione di alimentazione scende al di sotto di un valore predefinito (VBOR circa 4V) per un tempo superiore a TBOR (circa 100 micro sec.)

# II RESET

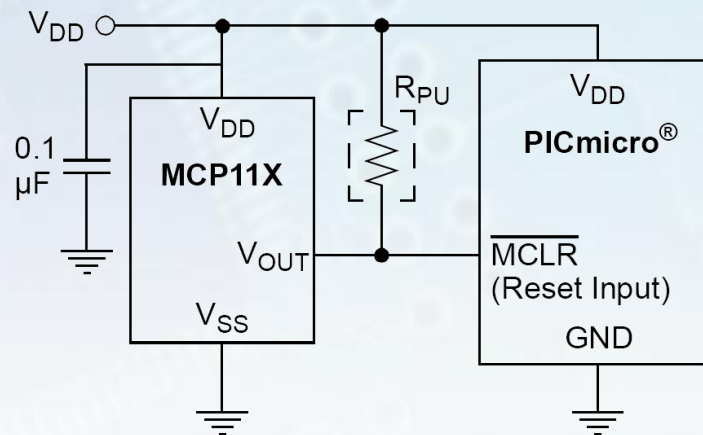
## Voltage detectors

### Dispositivo

### Soglia di commutazione

MCP1xx-195	1,900 V
MCP1xx-240	2,320 V
MCP1xx-270	2,630 V
MCP1xx-290	2,900 V
MCP1xx-300	2,930 V
MCP1xx-315	3,080 V
MCP1xx-450	4,380 V
MCP1xx-475	4,630 V

Collegamento tipico  
con un microcontrollore



# Watchdog Timer

Il reset del micro può essere effettuato anche mediante il Watchdog Timer

- Se abilitato il Watchdog Timer è un contatore che viene incrementato automaticamente da un clock interno
- Il watchdog timer deve essere azzerato manualmente dal programma prima che raggiunga la fine del conteggio (CLRWDT)
- Se raggiunge la fine del conteggio significa che non è stato azzerato quindi è probabile che il programma sia bloccato: RESET del micro.
- Anche per il watchdog timer può essere impostato un prescaler

# L'ELETTRONICA DI **MR. A. KEER**

Ora tocca a voi!

Mettete in pratica quanto avete imparato, Non scoraggiatevi e...

...Sperimentate, Sperimentate,  
Sperimentate, Sperimentate!!

