SISTEMA DI SVILUPPO

MC-16

per microcontrollori

PIC16F84 - PIC16F876

- **MANUALE DI PROGRAMMAZIONE**

Il sistema "MC-16" contiene:

• Scheda di sviluppo a microcontrollore con PIC16F84 e PIC16F876.

• Unità di programmazione
• Alimentatore da rete
• Cavo RS232 per il collegamento al PC
• Dischetto con programmi dimostrativi in linguaggio assembler

• **Manuale di programmazione**
• Manuale di istruzioni con esempi di programmi applicativi
• Due campioni di microcontrollori: uno di PIC16F84 ed uno di PIC16F876

• Scheda di sviluppo a microcontrollore con:

• Zoccoli text-tool per l'impiego dei microcontrollori PIC16F876 e PIC16F84
• Quattro linee di ingresso analogiche
• Un convertitore D/A
• Una memoria EEPROM (24LC256)
• Una sorgente di tensioni potenziometrica (0-5V.)
• Un modulo per il controllo della luminosità
• Un modulo per il controllo della velocità di un motore c.c.
• Un modulo per il controllo della temperatura
• Un visualizzatore con display a sette segmenti
• Un display LCD intelligente (16x2 5mm.)
• Una tastiera a matrice
• Un buzzer piezoelettrico
• Un visualizzatore con led (1x8 bit)
• Un relè attuatore da 1A
• Un 1-wire
• Quattro pulsanti di ingresso
• Una interfaccia seriale RS-232
• Due sorgenti di alimentazione : 12Vcc e 5Vcc

• Unità di programmazione :

La presenza di un deviatore permette di commutare la fase di simulazione con quella di programmazione dei microcontrollori senza toglierli dal circuiti di simulazione. Le fasi di predisposizione alla programmazione e quella di trasferimento dei dati sono monitorate da due led (rispettivamente rosso e verde).

• Alimentatore da rete:
L'alimentazione del sistema è affidata ad un alimentatore switching, con protezione dai corto circuiti, in grado di fornire una tensione a 15V @ 1,7A dalla rete 220 Vca @ 50H

• Cavo di interfacciamento:

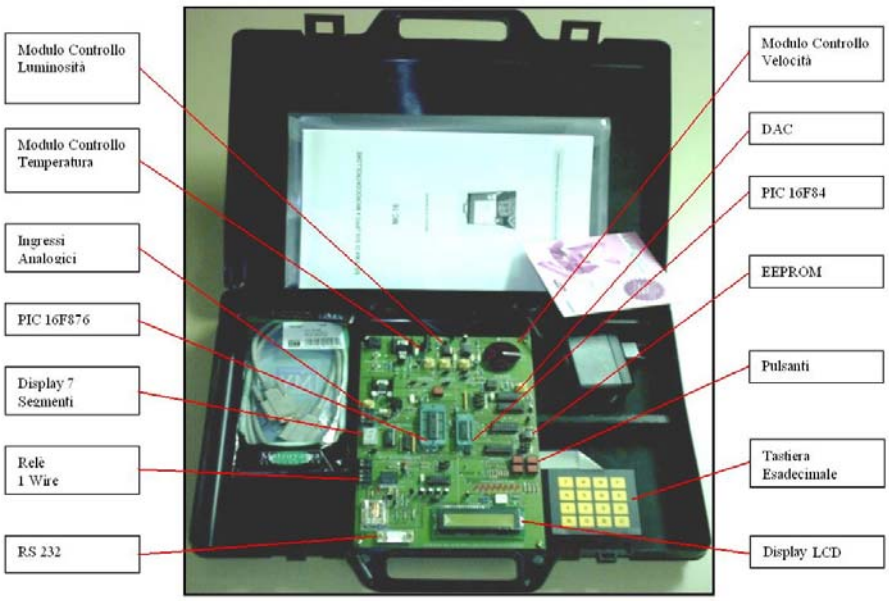
• Un cavo di interfacciamento RS232 mt.1,2

• Software di gestione costituito da:

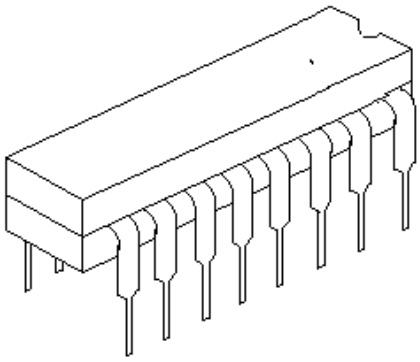
• software per la programmazione dei PIC
• esempi di programmazione e simulazione

• Manuale di programmazione con:

• Introduzione al microcontrollore
• Set di istruzioni con esempi applicativi



Introduzione



Per microcontrollore s'intende comunemente un sistema a microprocessore integrato su di un unico chip, che comprende, oltre alla *CPU*, una memoria di programma che può essere di diversi tipi *EPROM* (Erasable Programmable Read Only Memory), *EEPROM* (Electrically Erasable Programmable Read Only Memory), una memoria *Ram*, generalmente di dimensioni ridotte, per i risultati intermedi dell'elaborazione e per lo *stack* e i periferici di *I/O* vari.
Il microcontrollore è un dispositivo fondamentalmente programmabile in grado di svolgere diverse funzioni in modo autonomo, in relazione al programma implementato.
Un microcontrollore è fornito almeno di un timer, all'interno dei più evoluti vi sono anche più timer, dei comparatori (in grado di confrontare livelli di tensione), dei convertitori analogici-digitale, delle interfacce seriali per il collegamento a dispositivi esterni.
Le differenze sostanziali tra microcontrollore e microprocessore(CPU) sono le seguenti :

- Nel microcontrollore il programma di gestione risiede al suo interno, collocato in un'area di memoria apposita, non volatile. Nel microprocessore il programma che viene svolto, risiede nella memoria esterna.
- Nei microcontrollori è presente anche una zona di memoria di tipo RAM utilizzata per l'elaborazione dei dati. Alcuni microcontrollore hanno al loro interno una memoria per i dati di tipo EEPROM.
- Nei microcontrollori a loro interno vi sono delle linee di I/O le quali permettono di pilotare direttamente le periferiche esterne con segnali digitali o acquisire livelli logici provenienti da dispositivi esterni
- I microcontrollori hanno un ridottissimo set di istruzioni (sono dei dispositivi RISC) mentre i microprocessori sono spesso di tipo CISC o CISC/RISC.

I microcontrollori appartenenti alla famiglia della Microchip Tecnology si distinguono in base al loro tipo ed estensione di memoria, per la frequenza di lavoro, per il tipo e numero di periferiche integrati in essi. In base a queste caratteristiche si distinguono delle famiglie base, distinte con particolari sigle.

Tab. 1.1

PIC micro	MEMORIA				TIMER	PORTE				
	PROGRAMMA		DATI			I/O	A/D	PWM	CMP	SERIALE
	TIPO	WORD (14 bit)	RAM (byte)	EEPROM						
FAMIGLIA PICI2C5XX — con contenitore a 8 pin CLOCK 4 MHz – programmabile *in circuit*										
PIC12C508A	EPROM	512x12	25		1+WDT	6				
PIC12C5O9A	EPROM	1024 x 12	41		1+WDT	6				
FAMIGLIA PICI2CE5XX — con contenitore a 8 pin – programmabile *in circuit*										
PIC12CE518	EPROM	512x12	25	16	1+WDT	6				
PIC12CE519	EPROM	1024x12	41	16	1+WDT	6				
FAMIGLIA PICI2C67X — con contenitore a 8 pin CLOCK 10 MHz – programmabile *in circuit*										
P1C12C671	EPROM	1024	128		1+WDT	6	4 (8 bit)			
PIC12C672	EPROM	2048	128		1+WDT	6	4 (8 bit)			
FAMIGLIA PIC12CE67X — con contenitore a 8 pin CLOCK 10 MHz – programmabile *in circuit*										
PIC12CE673	EPROM	1024	128	16	1+WDT	6	4 (8 bit)			
PIC12CE674	EPROM	2048	128	16	1+WDT	6	4 (8 bit)			
FAMIGLIA PIC16C5X - CLOCK 20 MHz										
PIC16C54A	EPROM	512	25		1+WDT	12				
PIC16C555	EPROM	512	24		1+WDT	20				
PIC16C56	EPROM	1024	25		1+WDT	12				
PIC16C57	EPROM	2048	72		1+WDT	20				
PI016C58A	EPROM	2048	73		1+WDT	12				
FAMIGLIA PIC16C55X - CLOCK 20 MHz - P1C16C554 programmabile *in circuit*										
P1C16C554	EPROM	512	80		1+WDT	13				
P1C16C558	EPRQM	2048	128		1+WDT	13				
FAMIGLIA PICI6C6X - CLOCK 20 MHz – programmabile *in circuit*										
P1C16C61	EPROM	1024	36		1+WDT	3				
PIC16C62	EPROM	2048	128		3+WDT	2		1		I²/SPI
PIC16C64	EPROM	2048	128		3+WDT	3		1		I²/SPI
PIC16C65	EPROM	4096	192		3+WDT	3		2		I²/SPI /USART
FAMIGLIA PICI6C64X/ PIC16C66X – CLOCK 20 MHz – programmabile *in circuit*										
PIC16C642	EPROM	4096	176		1+WDT	2			2	
PIC16C662	EPROM	4096	176		1+WDT	3			2	
FAMIGLIA PICI6CE62X - CLOCK 20 MHz - programmabile *in circuit*										
PIC16CE623	EPRQM	512	96	128	1+WDT	3				
PIC16CE624	EPROM	1024	96	128	1+WDT	3				
PIC16CE625	EPROM	2048	128	128	1+WDT	3				
FAMIGLIA PICI6X62X - CLOCK 20 MHz - programmabile *in circuit*										
PIC16C620A	EPROM	512	96		1+WDT	13			2	
PIC16C621A	EPROM	1024	96		1+WDT	13			2	
PIC16C622A	EPROM	2048	128		1+WDT	13			2	
PIC16F627	EPRQM	1024	224		3+WDT	15		1	2	USART/SCI
PIC16F628	EPROM	2048	224		3+WDT	15		1	2	USART/SCI

Tab 1.1

PIC micro	MEMORIA				TIMER	PORTE				
	PROGRAMMA		DATI			I/O	A/D	PWM	CMP	SERIALE
	TIPO	WORD (14 bit)	RAM (byte)	EEPROM						
FAMIGLIA PIC16C7XX - CLOCK 24 MHz – programmabile *in circuit*										
PIC16C745	EPROM	8182	256		3+WDT	19	5 (8 bit)	2		USB/USART/SCI
PIC16C765	EPROM	8192	256		3+WDT	30	8 (8 bit)	2		USB/USART/SCI
FAMIGLIA PIC16C71X - CLOCK 20 MHz programmabile *in circuit*										
PIC16C710	EPROM	512	36		1+WDT	13	4 (8 bit)			
PIC16C711	EPROM	1024	68		1+WDT	13	4 (8 bit)			
P1C16C712	EPROM	1024	128		3+WDT	13	4 (8 bit)	1		
PIC16C715	EPROM	2048	128		1+WDT	13	4 (8 bit)			
PIC16C716	EPROM	2048	128		3+WDT	13	4 (8 bit)	1		
PIC16C717	EPROM	2048	256		3+WDT	16	10 (8 bit)	1		
FAMIGLIA PIC16C77X - CLOCK 20 MHz programmabile *in circuit*										
PIC16C770	EPROM	2048	256		3+WDT	16	12 (8 bit)	1		I2C/SPI
PIC16C771	EPROM	2048	256		3+WDT	16	12 (8 bit)	1		I2C/SPI
PIC16C773	EPROM	4096	256		3+WDT	22	6 (12 bit)	2		I2C/SPI/USART
PIC16C774	EPROM	4096	256		3+WDT	33	10 (12 bit)	2		I2C/SPI/USART
FAMIGLIA PIC16C7X - CLOCK 20 MHz – programmabile *in circuit*										
PIC16C70	EPROM	512	36		1+WDT	13	4 (8 bit)			
PIC16C71A	EPROM	1024	68		1+WDT	13	4 (8 bit)			
PIC16C72	EPRQM	2048	128		3+WDT	22	5 (8 bit)	1		I2C/SPI
PIC16C73A	EPROM	4096	192		3+WDT	22	5 (8 bit)	2		I2C/SPI/USART
PIC16C74A	EPROM	4096	192		3+WDT	33	8 (8 bit)	3		I2C/SPI/USART
FAMIGLIA PIC16F87X - CLOCK 20 MHz - programmabile *in circuit*										
PIC16F873	FLASH	4096	192	128	3+WDT	3	5 (10 bit)	2		USART/MSSP
PIC16F874	FLASH	4096	192	128	3+WDT	5	8 (10 bit)	2		USART/MSSP
PIC16F876	FLASH	8192	368	256	3+WDT	3	5 (10 bit)	2		USART/MSSP
PIC16F877	FLASH	8192	368	256	3+WDT	5	8 (10 bit)	2		USART/MSSP
FAMIGLIA PIC16X8X CLOCK 10 MHz (20 MHz PIC16F84A) – programmabile *in circuit*										
PIC16C84	FLASH	512	36	64	1+WDT	13				
PIC16F83	FLASH	512	36	64	1+WDT	13				
PIC16F84	FLASH	1024	68	64	1+WDT	13				
PIC16F84A	FLASH	1024	68	64	1+WDT	13				
FAMIGLIA PIC17C4X – CLOCK 33 MHz										
PIC17C42A	EPROM	2048x16	232		4+WDT	33		2		USART
P1C17C43	EPROM	4096x16	454		4+WDT	33		2		USART
PIC17C44	EPROM	8192x16	454		4+WDT	33		2		USART
FAMIGLIA PIC17C7XX CLOCK 33 MHz - PIC17C756A e PIC17C762 programmabili *in circuit*										
P1C17C752	EPROM	8192x16	454		4+WDT	50	12 (8 bit)	3		I2C/SPI/USART
PIC17C756A	EPRQM	16384x16	902		4+WDT	50	12 (8 bit)	3		I2C/SPI/USART
PIC17C762	EPROM	8192x16	678		4+WDT	66	16 (8 bit)	3		I2C/SPI/USART

In tabella 1.1 vi sono elencate le fondamentali caratteritiche per ogni famiglia, il tipo ed estensione della memoria sia per il programma che per i dati, la frequenza massima di clock, la presenza integrata nel chip di timer,del WDT (Watch Dog Timer), di A/D (Convertitore Analogico Digitale), di uscita PWM (Pulse Width Modulation), di CMP (Comparatori di tensione) infine il numero di ingressi digitali I/O. Nei microcontrollori Microchip abbiamo tre tipi di memoria : EPROM, EEPROM(Flash), OTP (Eprom non finestrati). I modelli con memoria EPROM sono dotati di una piccola finestra tramite la quale, utilizzando una comune lampada a raggi ultravioletti, è possibile cancellare il programma contenuto al loro interno. Di solito vengono impiegati in fase di sviluppo per la creazione di prototipi, infatti è possibile cancellarmi e riprogrammarli un centinaio di volte. I modelli con memoria EEPROM hanno la caratteristica di poter cancellare il programma al loro interno, elettricamente (con impulsi elettrici). Anch'essi vengono usati in fase di sviluppo con la particolarità di essere veloci nella cancellazione e riprogrammazione con la possibilità di cancellarli e ripogrammarli più di un centinaio di volte. I modelli con memoria OTP sono identici ai modelli con memoria EPROM ma possono essere programmati una sola volta (TP = One Time Programmable – programmabile una sola volta). Costano meno dei modelli EPROM o EEPROM ed offrono la possibilità di proteggere da lettura il programma contenuto al loro interno, ecco perché, dopo aver testato un programma su modello EPROM o EEPROM si utilizzerà il modello OTP sul circuito finale. La programmazione in volti microcontrollori avviene serialmente. Il set di istruzioni di base per tutti i microcontrollori è lo stesso, tranne le famiglie PIC17C4X e PIC 17C7xx le quali hanno un set di istruzioni leggermente più ampio rispetto alle altre famiglie. I microcontrollori che il nostro sistema di sviluppo utilizza sono : PIC16F84 e PIC16F876.

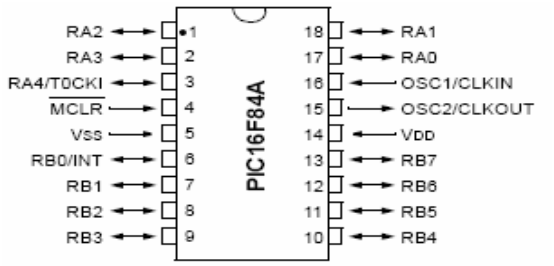
PIC 16F84

PIC 16F876

1. Microcontrollore PIC 16F84

1.1. Caratteristiche principali del microcontrollore PIC16F84

- CPU 8 bit
- Frequenza di clock massima 10 MHz (20 MHz PIC16F84A)
- Alimentazione compresa tra 2 V ÷ 6 V
- Programmazione in Circuit
- 13 linee di I/O (5 Linee PORTA A e 8 Linee PORTA B)
- Memoria tipo FLASH 1024 Word(14 bit)
- 68 bytes di memoria RAM
- 64 bytes di memoria EEPROM
- 1 Timer e un WDT
- Interrupt
- 8 Livelli di stack



1.2. Descrizione Piedini del PIC 16F84

VDD e VSS

Ingressi di alimentazione (VDD positivo e VSS negativo), su VDD va applicata una tensione compresa tra 2 / 6 V e VSS viene collegato a massa.

RA0 – RA4

Sono 5 linee della *PORTA*,ogni linea può essere programmata in modo indipendente dalle altre come input o output. Quando vengono programmate in uscita effettuano il *latch* del dato posto in uscita.La linea RA4 può essere utilizzata anche come ingresso del clock esterno per il timer TMR0. In più la linea RA4, programmata come ingresso, è del tipo *trigger di Schmitt* invece programmata come uscita è di tipo *open drain* (questa ultima configurazione richiede un resistore di pull-up esterno).

RB0 – RB7

Sono 8 linee della *PORT*B, anche queste linee possono essere configurate indipendente come input o output, quando vengono programmate come uscita effettuano il *latch* del dato posto in uscita. La linea RB0 è utilizzata anche come ingresso per segnali d'*interrupt* esterni. Programmando le linee RB4 – RB7 come ingressi, possono generare un'interrupt quando cambia lo stato del segnale presente su una o più linee.

MCLR /VPP

Linea di reset attiva al livello basso. Normalmente posta ad un livello alto collegandola con un resistore di *pull-up* a VDD. La linea MCLR è utilizzata anche per la tensione di programmazione seriale.

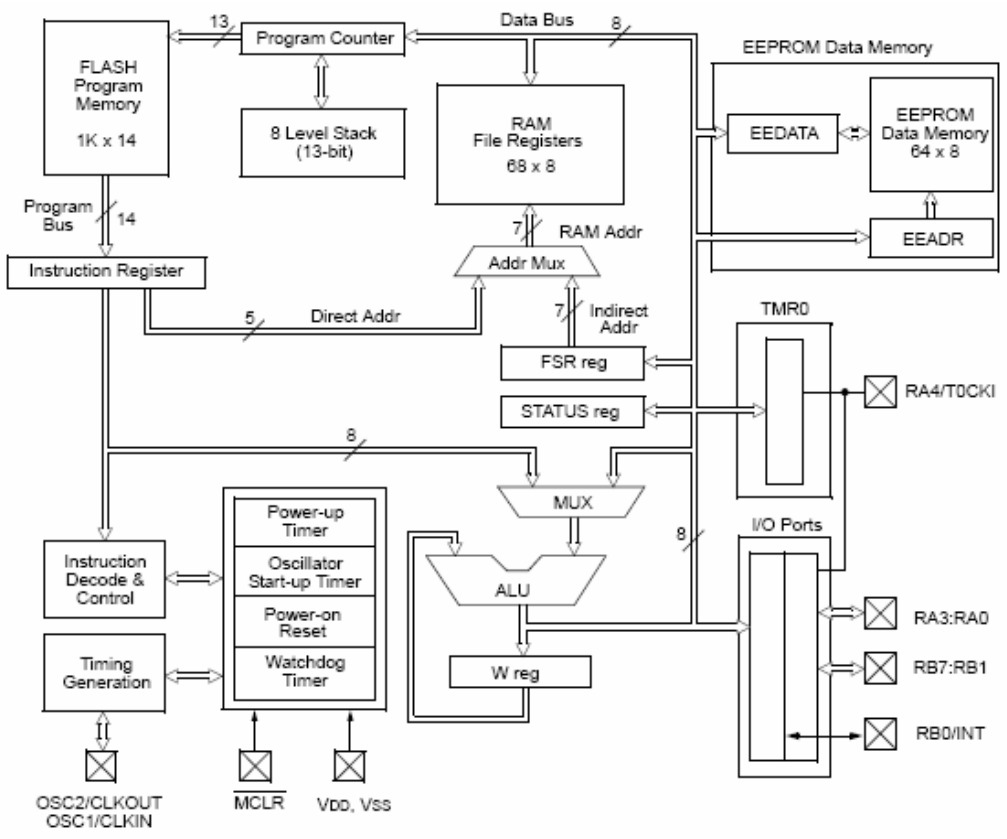
OSC1/CLKIN e OSC2/CLKOUT

Il PIC 16F84A a differenza del PIC 16C84 può operare ad una frequenza fino i 20 MHz .

2. Struttura Interna del microcontrollore PIC 16F84

Come possiamo notare dallo schema riportato più sotto il Microcontrollore si distingue dal microprocessore per le seguenti caratteristiche:

- Una CPU a 8 bit (denominata CORE)
- Una memoria programmata di tipo ROM,EPROM o EEPROM
- Una memoria dati di tipo RAM
- Alcune porte di Ingresso/Uscita
- Un Timer a 8 bit completo di prescaler a sette bit
- Un watchdog digitale
- Un oscillatore di Clock pilotato da un quarzo esterno
- Un ingresso di interrupt



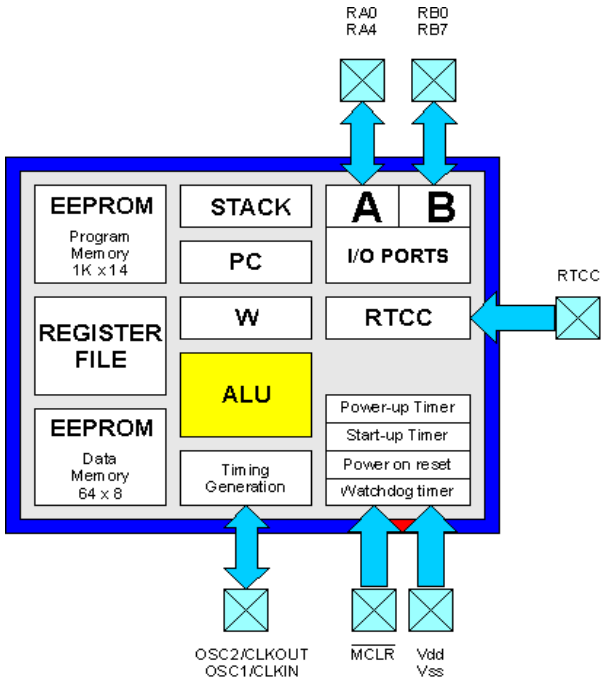
Tutte queste unità, in realtà, rendono una MCU molto simile ad un vero e proprio computer in miniatura racchiuso all'interno di un unico circuito integrato, ecco perché è più corretto definirlo come *microcontrollore*.

2.1. CPU

La periferica più importante contenuta all'interno di un PIC è la CPU (detta anche "Core"), essa è in pratica il cervello di tutto il sistema,è quella che comunica con tutte le varie periferiche interne attraverso dei canali di comunicazione chiamati "Bus". La CPU ha il compito di svolgere il programma ed elaborare i dati.

2.2. L'ALU

L'*ALU* (acronimo di **A**rithmetic and **L**ogic **U**nit ovvero unità aritmetica e logica) è la componente più complessa del PIC in quanto contiene tutta la circuiteria delegata a svolgere le funzioni di calcolo e manipolazione dei dati durante l'esecuzione di un programma.
L'*ALU* è una componente presente in tutti i microcontrollore e da essa dipende direttamente la potenza di calcolo del microcontrollore stesso.
L'*ALU* del PIC16F84 è in grado di operare su valori ad 8 bit, ovvero valori numerici non più grandi di 255. Esistono microprocessori con *ALU* a 16, 32, 64 bit e oltre. La famiglia Intel 80386, 486 e Pentium ad esempio dispone di un'*ALU* a 32 bit. Le potenze di calcolo raggiunte da questi processori sono notevolmente superiori a scapito della complessità della circuiteria interna ed accessoria e conseguentemente dello spazio occupato.



2.3. MEMORIA

La memoria del microcontrollore PIC si divide in 2 tipi : memoria programma e memoria dati

2.3.1. Memoria Programma

La memoria programma è di tipo EEPROM, cancellabile elettricamente tramite un programmatore idoneo e quindi può essere riscritta. Ha una estensione di 1024 locazioni a 14 bit (i codici relativi alle istruzioni sono lunghi proprio 14 bit). Le locazioni sono distinte da un indirizzo,normalmente espresso in esadecimale, parte da 000h fino ad arrivare a 3FFh. A queste locazioni di memoria si devono aggiungere 8 locazioni riservate allo stack,nei quali vengono memorizzati gli indirizzi di ritorno della subroutine. Quando avviamo il dispositivo, il Program Counter punta alla locazione con indirizzo 000h (locazione di reset), questa locazione contiene sempre il primo codice del programma. La locazione con indirizzo 004h è utilizzata come interupt vector,ovvero se viene attivato l'interupt questo registro punta alla routine dell'interupt.

2.3.2. Memoria Dati

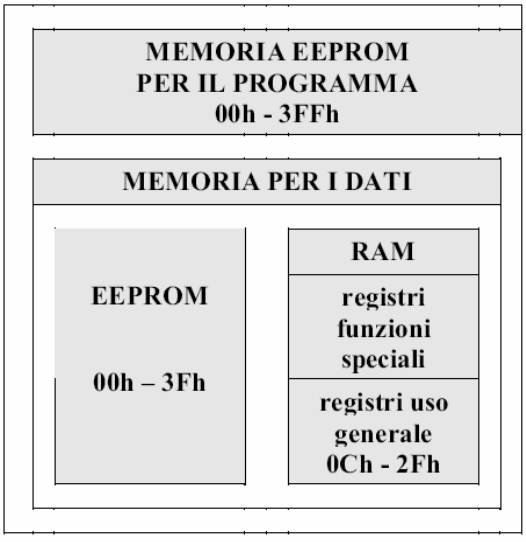
- EEPROM ha una estensione di 64 locazioni di un byte ciascuno con indirizzamento 00h ÷ 3Fh.
- RAM, è suddivisa in 2 banchi con un'estensione di 47 locazioni di un byte ciascuna, con indirizzamento 00h ÷ 3Fh per il primo banco (banco 0) e 80h ÷ 8Bh per il secondo banco (banco 1). Di queste locazioni, alcune sono riservate ai registri speciali con esattezza quelle con indirizzo 00h ÷ 0Bh sono riservate per il banco 0 quelle con indirizzo 80h ÷ 80Bh, le rimanenti vengono usate per registri di uso generale disponibile per il programmatore, sono 68 locazioni di 1 byte ciascuna con indirizzamento 0Ch ÷ 4Fh

Tab. 1.1

MEMORIA PROGRAMMA (EEPROM)		MAPPA FILE REGISTER (RAM DATI)			
STACK LIVELLO 1		00h	INDF	INDF	80h
STACK LIVELLO 2		01h	TMR0	OPTION	81h
STACK		02h	PCL	PCL	82h
		03h	STATUS	STATUS	83h
		04h	FSR	FSR	84h
STACK LIVELLO 8		05h	PORTA	TRISA	85h
		06h	PORTB	TRISB	86h
VETTORE DI RESET	000h	07h			87h
	001h	08h	EEDATA	EECON1	88h
	002h	09h	EEADR	EECON2	89h
	003h	0Ah	PCLATH	PVLATH	8Ah
VETTORE DI INTERRUPT	004h	0Bh	INTCON	INTCON	8Bh
	005h	0Ch			
EEPROM PROGRAMMA UTENTE			RAM USO GENERALE		
	3FEh	2Eh			
	3FFh	2Fh			

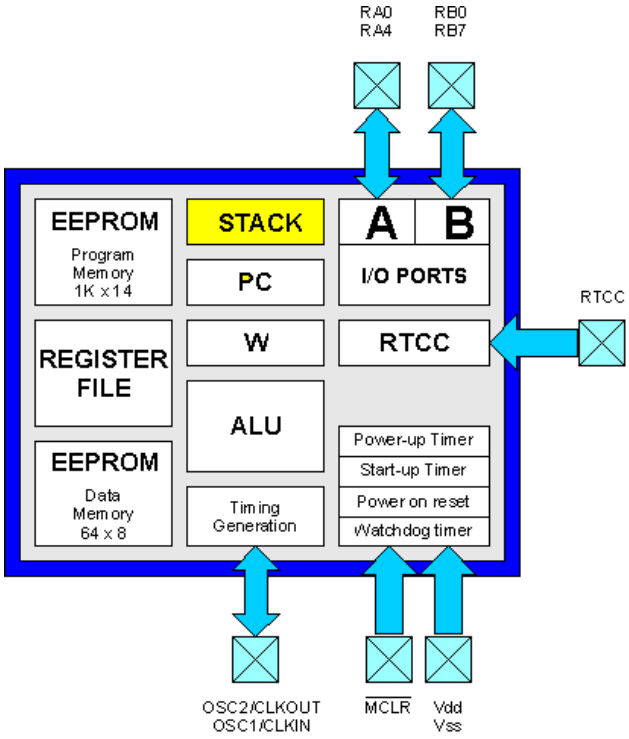
TABELLA SINTETIZZATA

Tab. 1.2



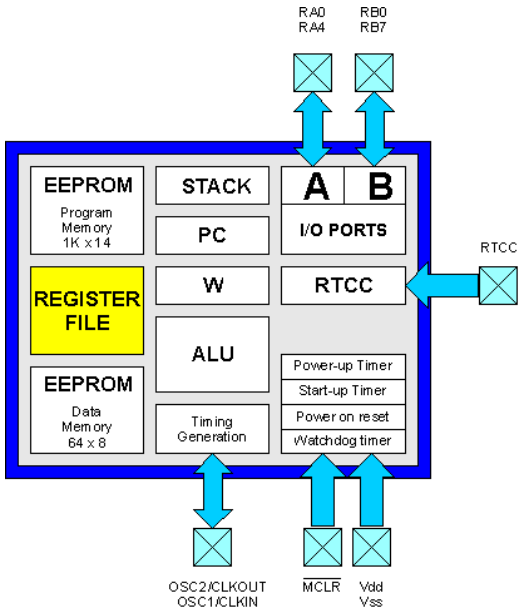
2.4. STACK

La parola *STACK* significa *"catasta"* ed, infatti, su questa catasta è possibile depositare, uno sull'altro, più indirizzi per recuperarli quando servono. Questo tipo di memorizzazione viene anche denominata *LIFO* dall'inglese ***Last In First Out***, in cui l'ultimo elemento inserito (last in) deve necessariamente essere il primo ad uscire (last out). E' spesso utilizzata quando si esegue una *CALL*, ossia quando il programma salta ad un particolare indirizzo per eseguire una subroutine. Terminata quest'esecuzione si dà il comando *RETURN* che preleva dallo STACK l'indirizzo di ritorno, ossia l'indirizzo dell'istruzione successiva alla *CALL*.



2.5. REGISTER FILE

Il *REGISTER FILE* è un'insieme di locazioni di memoria ram denominate registri. Contrariamente alla memoria *EEPROM* destinata a contenere il programma, l'area di memoria RAM è direttamente visibile dal programma stesso.



Quindi potremo scrivere, leggere e modificare tranquillamente ogni locazione del *REGISTER FILE* nel nostro programma ogni volta che se ne presenti la necessità.

L'unica limitazione consiste nel fatto che alcuni di questi registri svolgono una funzione speciale per il PIC e non possono essere utilizzati per scopi diversi da quelli per cui sono stati riservati. Questi registri speciali si trovano nelle locazioni più basse dell'area di memoria *RAM* secondo quanto illustrato di seguito.

Le locazioni di memoria presenti nel *REGISTER FILE* sono indirizzabili direttamente in uno spazio di memoria che va da *00H a 2FH* per un totale di 48 byte, denominato pagina 0. Un secondo spazio di indirizzamento denominato pagina 1 va da *80H* a *AFH*. Per accedere a questo secondo spazio è necessario ricorrere ai due bit ausiliari *RP0 e RP1*.

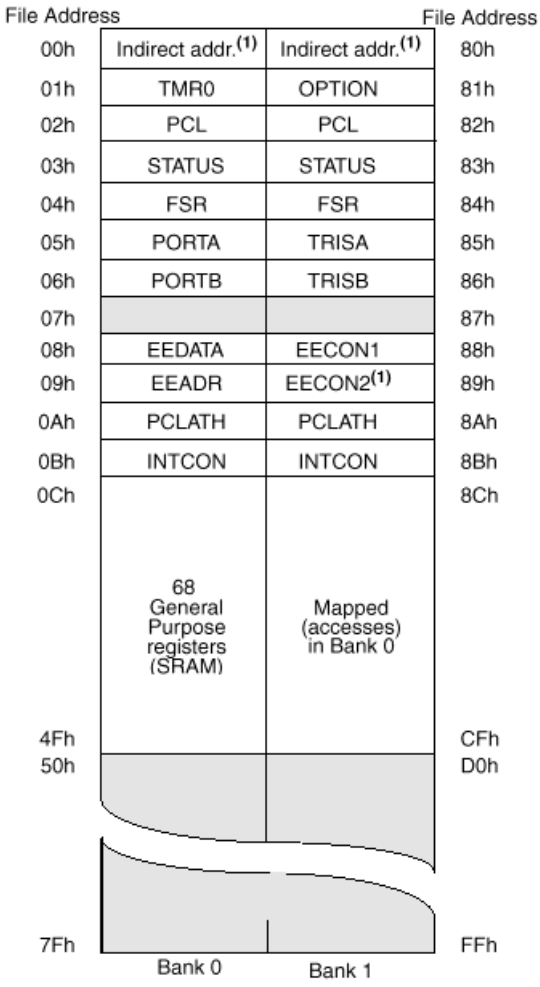
Le prime 12 locazioni della pagina 0 (da 00H a 0BH**)** e della pagina 1 (da 80H a 8BH) sono quelle riservate alle funzioni speciali per il funzionamento del PIC e non possono essere utilizzate per altri scopi.

Le *36 locazioni* in *pagina 0* indirizzate da *0CH* a *2FH* possono essere utilizzate liberamente dai nostri programmi per memorizzare variabili, contatori, ecc.

I registri specializzati del PIC sono utilizzati molto di frequente nei programmi.

Ad esempio, si ricorre alla coppia di registri specializzati *TRISA* e *TRISB*, per definire quali linee di I/O sono in ingresso e quali in uscita. Lo stesso stato logico delle linee di I/O dipende dal valore dei due registri *PORTA* e *PORTB*.

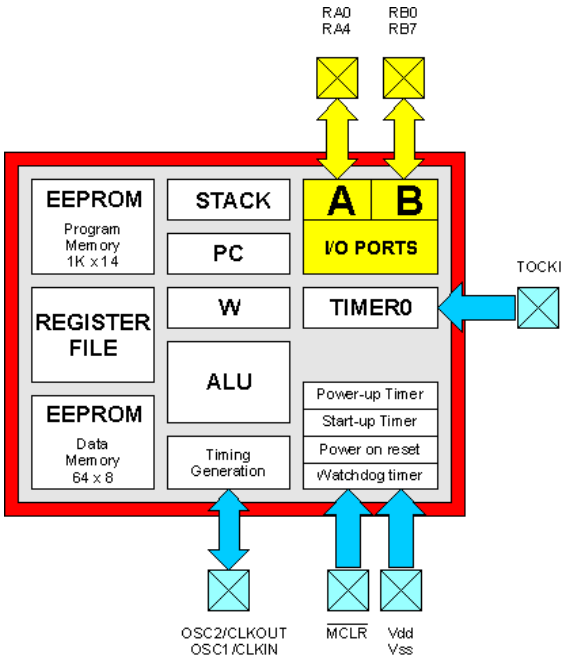
Alcuni registri riportano lo stato di funzionamento dei dispositivi interni al PIC o il risultato di operazioni aritmetiche e logiche. E' necessario conoscere quindi esattamente quale funzione svolge ciascun registro specializzato e quali effetti si ottengono nel manipolarne il contenuto.



2.6. PORTE I/O

Il PIC16F84 dispone di un totale di 13 linee di I/O organizzate in due porte denominate *PORTA A* e *PORTA B*. La *PORTA A* dispone di 5 linee configurabili sia in ingresso che in uscita identificate dalle sigle *RA0, RA1, RA2, RA3 e RA4*. La *PORTA B* dispone di 8 linee anch'esse configurabili sia in ingresso che in uscita identificate dalle sigle *RB0, RB1, RB2, RB3, RB4, RB5, RB6 E RB7*.

La suddivisione delle linee in due porte distinte è dettata dai vincoli dell'architettura interna del PIC16F84 che prevede la gestione di dati di lunghezza massima pari a 8 bit.



Per la gestione delle linee di I/O da programma, il PIC dispone di due registri interni per ogni porta denominati *TRISA* e *PORTA* per la porta **A** e *TRISB* e *PORTB* per la porta **B**.

I registri *TRISA* e *TRISB*, determinano il funzionamento in ingresso o in uscita di ogni singola linea, i registri *PORTA* e *PORTB* in essi vengono posti i dati che debbono essere inviati alle singole linee della porta A e delle porta B, oppure in essi vengono posti i dat che provengono dall'esterno in base alla configurazione assegnata alle singole linee delle porte con i registri *TRISA* e *TRISB*.

Ad esempio il bit 0 del registro *PORTA* e del registro *TRISA* corrispondono alla linea *RA0*, il bit 1 alla linea *RA1* e così via.

Se il bit 0 del registro *TRISA* è messo a zero, la linea *RA0* sarà configurata come linea in uscita, quindi il valore a cui sarà messo il bit 0 del registro *PORTA* determinerà lo stato logico di tale linea ("0" = 0 volt, "1" = 5 volt).

Se il bit 0 del registro *TRISA* è messo a 1 la linea *RA0* sarà configurata come linea in ingresso, quindi lo stato logico in cui sarà posta dalla circuiteria esterna la linea *RA0* si rifletterà sullo stato del bit 0 del registro *PORTA*.

Facciamo un esempio:

TRISA

			IN	IN	OUT	OUT	IN
X	X	X	1	1	0	0	1
-	-	-	RA4	RA3	RA2	RA1	RA0

```
MOVLW  00011001b   ;carica nell'accumulatore (W) la configurazione 00011001b
TRIS       05h          ;carica nel registro TRISA il contenuto dell'accumulatore
```

Notiamo come il dato di configurazione deve essere posto prima nell'accumulatore e poi spostato nel registro TRISA. L'istruzione TRIS utilizza l'indirizzo 05h (quello della porta A) e non 85h, la logica interna del microcontrollore provvederà a porre il dato nel giusto banco di memoria (banco 1) all'indirizzo 85h.

Dopo aver configurato la porta A come nell'esempio precedente, per porre a livello alto tutte le linee configurate come uscite, si deve inviare sulla porta A la configurazione 00000110b

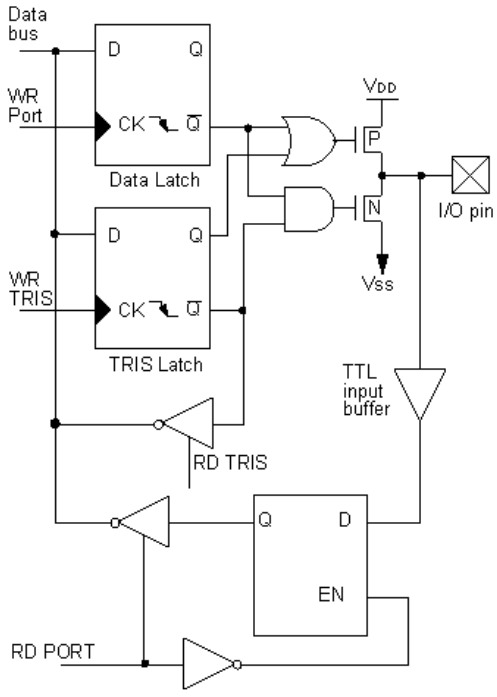
PORT_A

			IN	IN	OUT	OUT	IN
X	X	X	X	X	1	1	X
-	-	-	RA4	RA3	RA2	RA1	RA0

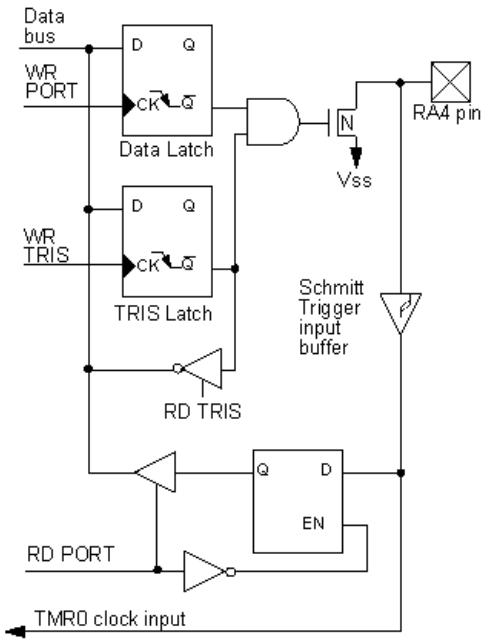
```
MOVLW  00000110b        ;carica in accumulatore la configurazione 00000110b
MOVWF  05h              ;sposta il contenuto dell'accumulatore nel file register
                        ;della porta A
```

In questo modo le linee RA1 e RA2 assumono un livello alto.

La Microchip per meglio adattare i dispositivi alle esigenze del pubblico ha differenziato gli stadi d'uscita di alcuni pin di I/O.
Difatti i pin da *RA0* a *RA3* avranno uno stadio d'uscita che corrisponde al primo schema riportato sotto.
Il pin *RA4* è diverso dagli altri in quanto quando la linea *RA4* è programmata in uscita e messa a 1, in realtà non è connessa al positivo ma rimane scollegata.
Tale tipo di circuiteria d'uscita è denominata a *"collettore aperto"* e se vogliamo essere sicuri che la linea *RA4* vada a 1 dovremo collegare esternamente una resistenza di *pull-up*, ovvero una resistenza collegata al positivo di alimentazione.
Lo stadio d'uscita del pin *RA4* è visibile nel secondo schema.



(Schema stadio d'uscita pin RA0, RA1, RA2, RA3)

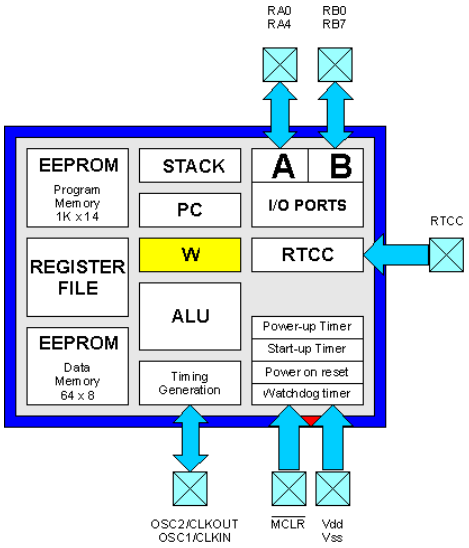


(Schema stadio d'uscita pin RA4)

Il funzionamento dei pin da *RB1* a *RB7* è del tutto analogo a quello delle linee da *RA0* a *RA3*. L'unica differenza è data dalla linea *RB0* che quando è configurata come linea di ingresso, può generare, in corrispondenza di un cambio di stato logico, un *interrupt*, ovvero un'interruzione immediata del programma in esecuzione ed una chiamata ad una *subroutine* speciale denominata *interrupt handler* di cui si parlerà più avanti.

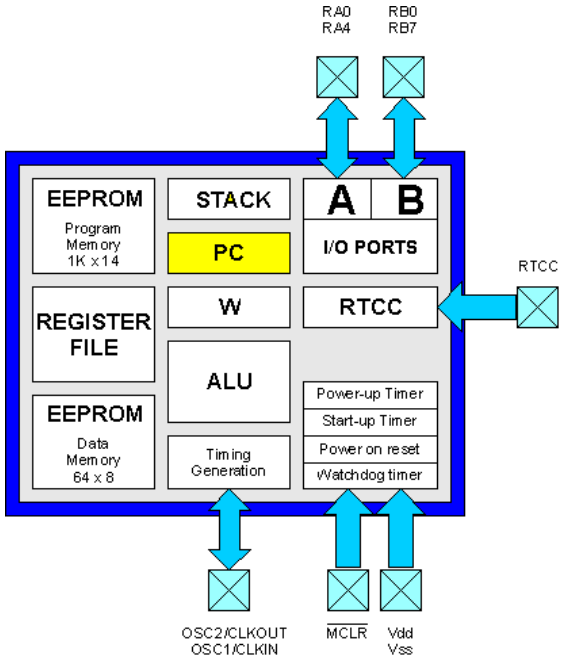
2.7. REGISTRO ACCUMULATORE W

Direttamente connesso con l'*ALU* c'è il registro *W* denominato anche accumulatore. Questo registro consiste di una semplice locazione di memoria in grado di contenere un solo valore a 8 bit.
La differenza sostanziale tra il registro *W* e le altre locazioni di memoria consiste proprio nel fatto che, per referenziare il registro *W*, l'*ALU* non deve fornire nessun indirizzo di memoria, ma può accedere direttamente.



2.8. PROGRAM COUNTER

Il PIC16F84 inizia l'esecuzione del programma dal vettore di reset (Reset Vector) ovvero dall'istruzione memorizzata nella prima locazione di memoria (indirizzo 0000H).
Dopo aver eseguito questa prima istruzione passa quindi all'istruzione successiva memorizzata nella locazione 0001H e così via. Se non esistesse nessuna istruzione in grado di influenzare in qualche modo l'esecuzione del programma, il PIC arriverebbe presto ad eseguire tutte le istruzione presenti nella sua memoria fino all'ultima locazione disponibile.
Sappiamo ovviamente che non è così e che qualsiasi microprocessore o linguaggio di programmazione dispone di istruzioni di salto, ovvero di istruzioni in grado di modificare il flusso di esecuzione del programma in base alle esigenze del programmatore. Una di queste istruzioni è la *GOTO* (*GO TO*, vai a) che ci permette di cambiare la sequenza di esecuzione e di "saltare" direttamente ad un qualsiasi punto, all'interno della memoria programma, e di continuare quindi l'esecuzione da quel punto. Per determinare quale sarà l'istruzione successiva da eseguire, il PIC utilizza uno speciale registro denominato *PROGRAM COUNTER* (contatore di programma) la cui funzione è proprio quella di mantenere traccia dell'indirizzo che contiene la prossima istruzione da eseguire. Questo registro è incrementato automaticamente.



3. Funzionamento altri dispositivi Hardware & Software

3.1 Power down mode (sleep)

Il *Power Down Mode* o *Sleep Mode* è un particolare stato di funzionamento del PIC utilizzato per ridurre il consumo di corrente nei momenti in cui il PIC non è utilizzato perché in attesa di un evento esterno.
L'istruzione *SLEEP* è utilizzate per mettere il PIC in Power Down Mode e ridurre di conseguenza la corrente assorbita che passerà da circa 2mA (a 5 volt con clock di funzionamento a 4Mhz) a circa 2µA.
Per entrare in *Power Down Mode* basta inserire questa istruzione in un punto qualsiasi del nostro programma:

```
SLEEP
```

Qualsiasi istruzione presente dopo la *SLEEP* non sarà eseguita dal PIC che terminerà in questo punto la sua esecuzione, spegnerà tutti i circuiti interni, tranne quelli necessari a mantenere lo stato delle porte di I/O (stato logico alto, basso o alta impedenza) ed a rilevare le condizioni di "risveglio". Per risvegliare il PIC dal suo sonno possono essere utilizzate diverse tecniche:

1. *Reset* del PIC mettendo a 0 il pin *MCLR* (pin 4)
2. *Timeout del timer del Watchdog* (se abilitato)
3. Verificarsi di una situazione di interrupt (*interrupt dal pin RB0/INT*, cambio di stato sulla porta B, termine delle operazioni di scrittura su EEPROM)

Nei casi 1 e 2 il PIC è resettato e l'esecuzione ripresa dalla locazione 0. Nel caso 3 il PIC si comporta come nella normale gestione di un interrupt eseguendo per primo l'interrupt handler e quindi riprendendo l'esecuzione dopo l'istruzione *SLEEP*. Perché il PIC sia risvegliato da un *interrupt* devono essere abilitati opportunamente i *flag* del registro *INTCON*.

3.2 Interrupt

L'interrupt è una particolare caratteristica dei PIC (e dei microprocessori in generale) che consente di intercettare un evento esterno, interrompere momentaneamente il programma in corso, eseguire una porzione di programma specializzata per la gestione dell'evento verificatosi e riprendere l'esecuzione del programma principale.

Il PIC16F84 è in grado di gestire in interrupt quattro eventi diversi, vediamo quali sono:

1. Il cambiamento di stato sulla linea *RB0* (*External interrupt RB0/INT pin*).
2. La fine del conteggio del registro *TMR0* (*TMR0 overflow interrupt*).
3. Il cambiamento di stato su una delle linee da *RB4* a *RB7* (*PORTB* change interrupts).
4. La fine della scrittura su una locazione *EEPROM* (*EEPROM* write complete interrupt).

L'interrupt su ognuno di questi eventi può essere abilitato o disabilitato indipendentemente dagli altri agendo su alcuni bit del registro *INTCON*. La configurazione dei bit del registro INTCON la vedremo più avanti quando parleremo dei registri.

3.3 Watch Dog Timer (WDT)

Il *Watch Dog Timer* è in pratica un oscillatore interno al PIC, ma completamente indipendente dal resto della circuiteria, il cui scopo è di rilevare eventuali blocchi della CPU del micro e resettare il PIC per riprendere la normale esecuzione del programma. Per rilevare un eventuale blocco della CPU durante l'esecuzione del programma principale, è inserita all'interno di questo, un'istruzione speciale, la:

CLRWDT (CLeaR Watch Dog Timer)

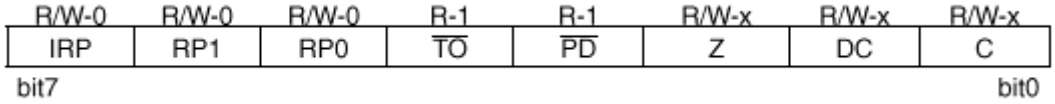
la quale azzera ad intervalli regolari il *Watch Dog Timer* non consentendogli di terminare il suo conteggio. Se la CPU non effettua questa istruzione prima del termine del conteggio allora si assume che il programma si è bloccato per qualche motivo e si effettua il Reset della CPU.
Il periodo minimo raggiunto il quale la CPU viene resettata è di circa 18ms (dipende dalla temperatura e dalla tensione di alimentazione). E' possibile però assegnare il *PRESCALER* al *Watch Dog Timer* per ottenere ritardi più lunghi fino a 2.3 secondi.
Agendo sul bit *PSA* del registro *OPTION* è possibile assegnare il *prescaler* al *Watch Dog Timer* per ottenere dei tempi di ritardo di intervento maggiori

PS2	PS1	PS0	Divisore	Periodo di ritardo del WDT
0	0	0	1	18ms
0	0	1	2	36ms
0	1	0	4	72ms
0	1	1	8	144ms
1	0	0	16	288ms
1	0	1	32	576ms
1	1	0	64	1.152s
1	1	1	128	2.304s

4. REGISTRI

4.1 REGISTRO STATUS

Il registro *STATUS* è un registro in cui alcuni bit sono impostati alla sola lettura e altri possono essere sia letti che scritti. Questo registro contiene una serie di *flag* che indicano lo stato aritmetico dell'unità *ALU* (**A**rithmetic **L**ogic **U**nit, Unità Logica Aritmetica), lo stato dell'hardware del PIC al RESET ed i *flag* che consentono l'indirizzamento ai diversi banchi di registri. Questo registro è localizzato all'indirizzo 03h ed è composto di 8 bit ognuno dei quali ha un significato preciso e una propria sigla identificativa.
La disposizione dei flag all'interno del registro STATUS è la seguente:



La funzione svolta da ogni singolo flag è descritta nella tabella 1.3:

Tab. 1.3

Posizione Flag	Funzione
Bit 7	**IRP: Register Bank Selected bit** Questo registro non viene utilizzato nei PIC16C8X e dovrebbe essere mantenuto a 0
Bit 6-5	**RP1:RP0:Register Bank Select bits** Questi due bit servono per selezionare il banco di registri che si vuole utilizzare. nei PIC16C8X solo RP0 (bit 5) viene usato realmente per selezionare uno dei due banchi registri disponibili. RP1 dovrebbe essere sempre mantenuto a 0: RP0 = **0** *Selezione il BANCO 0 (indirizzi da 00h a 7Fh)* RP0 = **1** *Selezione il BANCO 1 (indirizzi da 80h a FFh)*
Bit 4	**TO: Time-Out bit** Questo bit viene posto a 0 quando il WDT dà un time out.
Bit 3	**PD: Power-Down bit** Quando il micro viene posto nella condizione di riposo, dalla relativa istruzione, questo bit viene posto a 1 .
Bit 2	**Z: Zero bit** Questo bit viene posto a 1 logico se il risultato di un'operazione matematica è uguale a zero.
Bit 1	**DC: Digit Carry/borrow bit** Questo bit viene settato quando un'istruzione di somma o di sottrazione dà un riporto sul quarto bit; viene principalmente usato nelle conversioni da binario a BCD.
Bit 0	**C: Carry/borrow bit** Con le operazioni di addizione e sottrazione questo bit viene settato se l'operazione dà luogo ad un riporto. Durante le operazioni di rotazione di un registro, questo bit viene invece caricato con il valore del bit più alto o più basso del registro.

4.2 REGISTRO EECON1

Il registro *EECON1* è un registro di controllo usato nelle operazioni di lettura e scrittura sulla memoria *EEPROM DATI*. Esso contiene una serie di *flag* con cui è possibile controllare ogni singola operazione effettuata sull'*EEPROM dati*. La disposizione dei *flag* è la seguente:

U	U	U	R/W-0	R/W-x	R/W-0	R/S-0	R/S-x
—	—	—	EEIF	WRERR	WREN	WR	RD
bit7							bit0

La funzione svolta da ogni singolo *flag* è descritta nella tabella 1.4:

Tab. 1.4

Posizione Flag	Funzione
Bit 7-5	**Non utilizzati**
Bit 4	**EEIF: EEPROM Write Operation Interrupt Flag bit** Questo flag indica se la scrittura su EEPROM è stata completata dall'hardware del PIC e se l'interrupt è stato generato per questo motivo. **1** = *Operazione completata* **0** = *Operazione di scrittura non completata oppure non iniziata* Una volta generato l'interrupt questo flag deve essere resettato via software altrimenti la circuiteria interna del PIC non sarà più in grado di generare interrupt al termine delle successive scritture.
Bit 3	**WRERR: EEPROM Error Flag bit** Questo flag indica se l'operazione di scrittura è stata interrotta prematuramente a causa, ad esempio di un reset del PIC o un reset dal Watch Dog Timer **1** = *Operazione di scrittura interrotta prematuramente* **0** = *Operazione di scrittura completata correttamente*
Bit 2	**WREN: EEPROM Write Enable bit** Questo flag abilita le successive operazioni di scrittura su una cella EEPROM. Deve essere messo a uno prima di iniziare qualsiasi operazione di scrittura su EEPROM. Se messo a zero l'EEPROM si comporta come una memoria a sola lettura. **1** = *Scrittura su EEPROM abilitata* **0** = *Scrittura su EEPROM disabilitata*
Bit 1	**WR: Write Control bit** Questo flag serve ad attivare il ciclo di scrittura su EEPROM. Per attivare la scrittura occorre mettere a 1 questo flag. Lo stesso flag verrà messo automaticamente a zero dall'hardware del PIC una volta completata la scrittura sulla cella. **1** = *Comanda l'inizio della scrittura su EEPROM. Viene rimesso a 0 a fine ciclo di scrittura. Può essere solo settato dal nostro programma ma non resettato.* **0** = *Ciclo di scrittura completato*
Bit 0	**RD: Read Control bit** Questo flag serve ad attivare il ciclo di lettura da EEPROM. Per attivare la lettura occorre mettere a 1 questo flag. **1** = *Comanda l'inizio della lettura da EEPROM. Viene rimesso a 0 a fine ciclo di lettura. Può essere solo settato dal nostro programma ma non resettato.* **0** = *Non inizia la lettura su EEPROM*

4.3 REGISTRO INTCON

Questo è il registro grazie alla quale è possibile controllare e abilitare gli *interrupt*.
L'*interrupt* su ognuno di questi eventi può essere abilitato o disabilitato indipendentemente dagli altri agendo sui bit del seguente registro *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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Posizione Flag	Funzione
Bit 7	**GIE: Global Interrupt Enable bit** Questo bit deve essere messo ad 1 per l'abilitazione generale degli interrupt
Bit 6	**EEIE: EEPROM write complete Interrupt Enable bit** Se questo bit viene messo ad 1 viene abilitato l'interrupt alla fine della scrittura su una locazione EEPROM
Bit 5	**TOIE: TMR0 Overflow Interrupt Enable bit** Se questo bit viene messo ad 1 viene abilitato l'interrupt sulla fine del conteggio del registro TMR0
Bit 4	**INTE: Interrupt RB0/INT Enable bit** Se questo bit viene messo ad 1 viene abilitato l'interrupt sul cambiamento di stato di RB0.
Bit 3	**RBIE: RB chance Interrupt Enable bit** Se questo bit viene messo ad 1 viene abilitato l'interrupt sul cambiamento di stato su una delle linee da RB4 a RB7
Bit 2	**TOIF: TMR0 Overflow Interrupt Flag** Se questo flag vale 1 l'interrupt è stato generato al termine del conteggio del timer TMR0
Bit 1	**INTF: Interrupt RB0/INT Flag** Se questo flag vale 1 l'interrupt è stato generato dal cambiamento di stato sulla linea RB0
Bit 0	**RBIF: RB Interrupt Flag** Se questo flag vale 1 l'interrupt è stato generato dal cambiamento di stato di una delle linee da RB4 a RB7

Esiste un bit d'abilitazione generale degli interrupt che deve essere settato anch'esso ad 1 ovvero il bit ***GIE*** (Global Interrupt Enable bit) posto sul *bit 7* del registro *INTCON*. Qualunque sia l'evento abilitato, al suo manifestarsi il PIC interrompe l'esecuzione del programma in corso, memorizza automaticamente nello *STACK* il valore corrente del *PROGRAM COUNTER* e salta all'istruzione presente nella locazione di memoria 0004H denominata *Interrupt vector* (vettore d'interrupt). E' da questo punto che dobbiamo inserire la nostra subroutine di gestione dell'interrupt denominata *Interrupt Handler*. Dato che qualunque interrupt genera una chiamata alla locazione 0004H, nel registro *INTCON* sono presenti dei *flag* che indicano qual è l'evento che ha generato l'interrupt, vediamoli:

- ***INTF*** (bit 1)
- ***T0IF*** (bit 2)
- ***RBIF*** (bit 0)

Quando è generato un interrupt il PIC disabilita automaticamente il bit *GIE* (Global Interrupt Enable) del registro *INTCON* in modo da disabilitare tutti gli interrupt mentre è già in esecuzione un *interrupt handler*. Per ritornare al programma principale e reinizializzare a 1 questo bit occorre utilizzare l'istruzione: ***RETFIE***

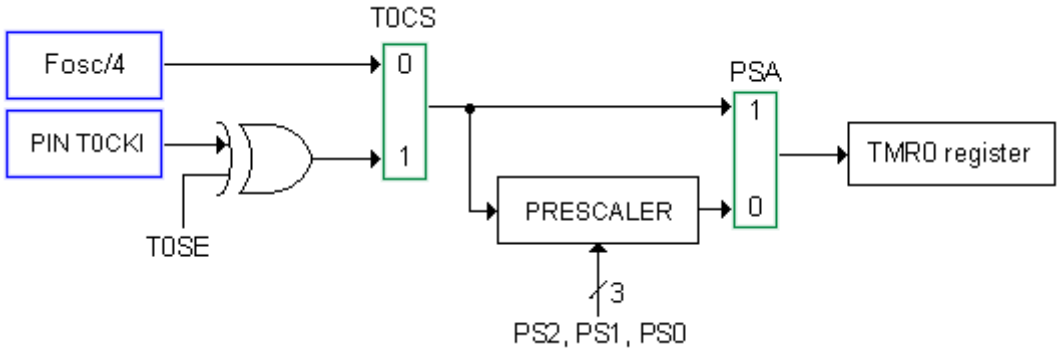
4.4 TIMER

4.4.1 REGISTRO TMR0

All'interno del microcontrollore PIC 16F84 è presente anche un timer ad 8 bit che permette di eseguire precise temporizzazioni.
Il registro *TMR0* è un contatore, ovvero un particolare tipo di registro il cui contenuto è incrementato con cadenza regolare e programmabile direttamente dall'hardware del PIC. In pratica, a differenza di altri registri, il *TMR0* non mantiene inalterato il valore che gli è memorizzato, ma lo incrementa continuamente, se ad esempio scriviamo in esso il valore 10 con le seguenti istruzioni:

```
movlw   10
movwf   TMR0
```

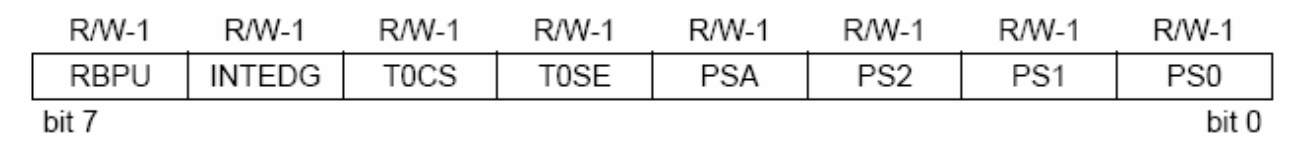
dopo un tempo pari a quattro cicli macchina, il contenuto del registro comincia ad essere incrementato a 11, 12, 13 e così via con cadenza costante e del tutto indipendente dall'esecuzione del resto del programma. Una volta raggiunto il valore 255 il registro *TMR0* è azzerato automaticamente riprendendo quindi il conteggio non dal valore originariamente impostato ma da zero. La frequenza di conteggio è direttamente proporzionale alla frequenza di clock applicata al chip e può essere modificata programmando opportunamente alcuni bit di configurazione.



I blocchi *Fosc/4* e *T0CKI* riportati in blu rappresentano le due possibili sorgenti di segnale per il contatore *TMR0*. *Fosc/4* è un segnale generato internamente al PIC dal circuito di clock ed è pari alla frequenza di clock divisa per quattro. *T0CKI* è un segnale generato da un eventuale circuito esterno ed applicato al pin *T0CKI* corrispondente al pin 3 nel PIC16F84. I blocchi *T0CS* e *PSA* riportati in verde sono due commutatori di segnale sulla cui uscita è presentato uno dei due segnali in ingresso in base al valore dei bit *T0CS* e *PSA* del registro *OPTION*. Il blocco *PRESCALER* è un divisore programmabile.

Come abbiamo accennato prima il registro *Option* serve per gestire il *timer interno del nostro PIC*. Difatti, tramite questo registro, si può impostare il moltiplicatore del *prescaler*, decidere se assegnarlo al *timer* o al *watch dog* e impostare altre funzioni come da schema sotto riportato.

4.4.2 REGISTRO OPTION

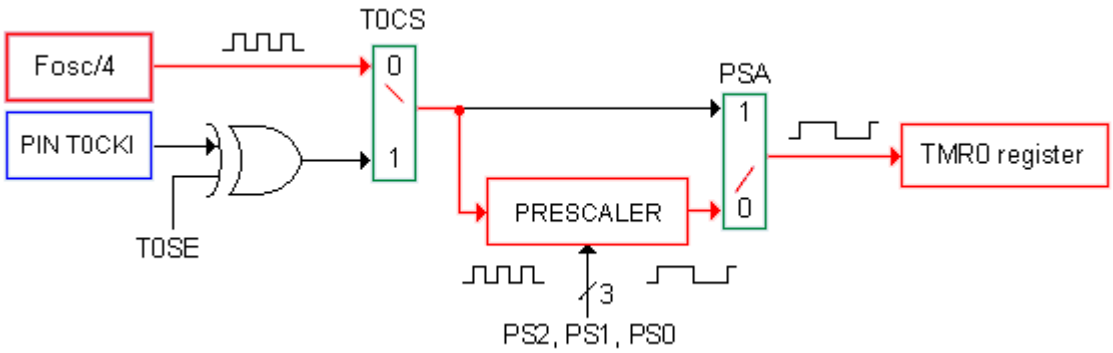


Bit	Nome	Funzione
D0	**PS0**	Rapporto di divisione del Prescaler
D1	**PS1**	Rapporto di divisione del Prescaler
D2	**PS2**	Rapporto di divisione del Prescaler
D3	**PSA**	Se il valore di questo bit è a 0 il prescaler è assegnato al TMR0, se è 1 è assegnato al WDT
D4	**TOSE**	Se è posto a 0 l'incremento avviene sul fronte di salita, se è 1 sul fronte di discesa
D5	**TOCS**	Modalità del Timer: Se è posto a 0 è usato il timer interno (f = Frequenza clock/4) Se è posto a 1 il contatore incrementa sul fronte di salita/discesa del pin RA4
D6	**Intedg**	Se è posto a 1 il conteggio avviene su commutazione sul fronte di salita di RB0 (linea di interrupt), se invece è posto a 0 avviene sul fronte di discesa
D7	**RPBU**	Abilita il Pull-Up della Porta B **1** = I Pull-Up della Porta B sono disabilitati **0** = I Pull-Up sono abilitati dai valori configurati

Il segnale di clock che pilota il timer può essere selezionato tra un segnale esterno (con ingresso sul pin RA4/TOCKI) e l'oscillatore interno. Nel caso venga scelto l'oscillatore interno sul timer viene applicato un segnale con frequenza pari ad un quarto di quella generata dall'oscillatore (fosc / 4). Se per esempio è utilizzato un quarzo da 4MHz la frequenza di eccitazione dell'oscillatore sarà di 1 MHz. E' possibile far variare la frequenza del segnale di clock applicato al timer inserendo un prescaler (divisore di frequenza) con valore di divisione selezionabile via software. Il timer genera un interupt ogni volta che il conteggio passa da FFh a 00h.

4.4.3 PRESCALER

Se configuriamo il bit *PSA* del registro *OPTION* a 0 inviamo al registro *TMR0* il segnale in uscita dal *PRESCALER* come visibile nella seguente figura:



Con l'uso del *PRESCALER* possiamo dividere ulteriormente la frequenza *Fosc/4* configurando opportunamente i bit *PS0, PS1 e PS2* del registro *OPTION* secondo la seguente tabella.

PS2	PS1	PS0	Divisore	Frequenza in uscita al prescaler (Hz)
0	0	0	2	500.000
0	0	1	4	250.000
0	1	0	8	125.000
0	1	1	16	62.500
1	0	0	32	31.250
1	0	1	64	15.625
1	1	0	128	7.813
1	1	1	256	3.906

Per utilizzare il timer ed il prescaler bisogna procedere come segue :

1. scrivere la routine d'interrupt del timer con inizio nella locazione 04h (vettore d'interrupt) come nel caso senza prescaler.
2. nel programma principale
   - assegnare il prescaler al timer (porre a 0 il bit 3 del registro OPTION; se il bit è posto ad 1 il prescaler è assegnato al Watchdog Timer)
   - selezionare il fattore di divisione del prescaler (impostare il bit 2, il bit 1 e il bit 0 del registro OPTION secondo la tabella riportata sotto)
   - inizializzare il timer caricando in esso un valore iniziale di conteggio (per esempio 00h)
   - abilitare l'interrupt del timer (bit 5 di INTCON = 1)
   - abilitare gli interupt (bit 7 di INTCON = 1)

Per utilizzare il timer senza prescaler bisogna procedere come segue :

1. Scrivere la routine d'interrupt del timer con inizio nella locazione 04h (vettore d'interrupt). In tale routine deve essere tra l'altro :
   - Ricaricato nel timer il valore iniziale del conteggio (per esempio 00h)
   - Azzerato il flag di avvenuto interrupt (bit 2 del registro INTCON)
   - Porre al termine della routine l'istruzione di ritorno dell'interrupt (RETFILE)
2. nel programma principale
   - impostare il modo timer (sorgente del clock del timer interna-porre a 0 il bit 5 del registro OPTION)
   - inizializzare il timer caricando in esso un valore iniziale di conteggio (per esempio 00h)
   - abilitare l'interrupt del timer (bit 5 di INTCON = 1)
   - abilitare gli interrupt (bit 7 di INTCON = 1)

Nei precedenti casi si è presupposto che la sorgente del clock del timer sia interna (modo timer). Se deve essere utilizzata una sorgente esterna (modo counter) bisogna porre ad 1 il bit 5 del registro OPTION. In tal caso può essere scelto per l'incremento del contatore il fronte di salita del clock (ponendo a 0 il bit 4 del registro OPTION o il fronte di discesa ponendo a 1 il bit 4 del registro OPTION).

Esempio senza prescaler :

frequenza del quarzo 4 MHz. Timer impostato a 00h.
Il timer è incrementato con una frequenza

$$ft=fosc/4 = 4000000/4 = 1000000\ Hz$$

Poiché l'interrupt viene generato ogni volta che il contatore passa da FFh a 00h si deve incrementare il timer da 00h a FFh (255 volte) e poi ancora una volta per tornare a 00h (in totale 256 volte). Quindi la routine d'interrupt viene chiamata con una frequenza

$$fi = ft/256 = 1000000/256 = 3906{,}25\ Hz$$

ovvero ogni 256 ms (1/fi). Con il quarzo assegnato (4 MHz) questo è il più grande periodo di temporizzazione che si può ottenere. Variando il valore impostato nel timer (maggiore di 00h) si possono ottenere valori di periodo più piccoli (il timer impiega meno tempo a raggiungere FFh).
Dalla osservazioni precedenti si può trovare una formula che stabilisce la frequenza e il periodo con cui si attiva a routine d'interrupt.

$$fi = (fosc/4) / (256 - Nt)$$
$$ti = (256 - Nt) / (fosc/4)$$

dove con Nt si è indicato il valore (in decimale) con cui è inizializzato il timer.

Esempio con prescaler :

Frequenza del quarzo 4 MHz. Timer impostato a 00h. Prescaler con fattore di divisione 256. Il timer è incrementato con una frequenza :

$$ft = (fosc/4) / 256 = (4000000 / 4) / 256 = 3906,25 \text{ Hz}$$

Essendo 256 il fattore di divisione impostato con il prescaler. Quindi la routine d'interrupt viene chiamata con una frequenza

$$fi = ft / 256 = 396,25 / 256 = 15,258$$

ovvero ogni 65,5 ms (1/fi). Con il quarzo assegnato (4 MHz) questo è il più grande periodo di temporizzazione che si può ottenere utilizzando anche il prescaler.
Le formule che stabiliscono la frequenza e il periodo con cui si attiva la routine d'interrupt sono:

$$fi = (fosc/4) / Np \times (256 - Nt)$$
$$ti = (Np \times (256 - Nt) / (fosc/4)$$

dove con Nt si è indicato il valore (in decimale) con cui è inizializzato il timer e con Np il fattore di divisione impostato per il prescaler. Per aumentare i periodi di temporizzazione si deve usare un contatore che viene decrementato ogni volta che viene attivata la routine d'interrupt.

5. SET DI ISTRUZIONI PER MICROCONTROLLORI PIC 16F84 e PIC16F876

I PIC 16F84 e 16F876 hanno un set 35 istruzioni di base. A queste possono essere aggiunte una serie di 28 istruzioni speciali valide solo per alcune famiglie di controllori.
queste istruzioni si dividono in tre gruppi che sono:

- Byte-oriented
- Bit-oriented
- Literal and control

Byte-oriented: questo gruppo d'istruzioni opera su un registro di 8 bit, quindi tutte le operazioni fatte vanno a modificare il contenuto di un registro.
Bit-oriented: questo gruppo comprende 4 istruzioni che operano su un singolo bit di un registro.
Literal and control: questo gruppo d'istruzioni opera su una costante di 8 bit, questa costante (literal) va a modificare un registro in base all'operazione che si deve eseguire.
Questi gruppi d'istruzione hanno in aggiunta una serie di parametri elencati in tabella 5.1

Tab. 5.1

f	Indirizzo del file register
d	Destinazione : d= 0 risultato in W; d= 1 risultato nel file register
k	Campo letterale, costante o etichetta
b	Ordine del bit da **7** (MSB) a **0** (LSB)

f : Questo parametro rappresenta un registro, un registro può essere personalizzato in modo da facilitare l'utente poiché al registro possiamo dare un nome qualunque, penserà poi il programma in fase di compilazione a dare un vero indirizzo al registro.

w : Questo registro è un accumulatore che usa il PIC per memorizzarci un dato temporaneo.

d : Questo parametro può assumere solo i valori 0 1 e indica dove il dato verrà salvato se d = 1 allora il dato verrà salvato nel registro f, se d = 0 il dato verrà salvato nel registro w.

b : Questo parametro definisce il bit su cui deve essere portata a termine l'operazione, poiché i bit di un registro sono 8 il valore di b varia tra 0 e 7.

k : Questo dato è una costante di 8 bit e lavora solo con le istruzioni del terzo gruppo.

Nella tabella 6.3 è elencato il set completo delle istruzioni suddiviso in quattro categorie:

- **Operazioni orientate al byte con i file register**
- **Operazioni orientate al bit con i file register**
- **Operazioni di controllo e con letterali**
- **Operazioni speciali**

Per ciascuna istruzione sono riportati anche i flag influenzati durante l'esecuzione dell'istruzione stessa. (Tabella 5.2)

Tab. 5.2

↕	Flag modificato
Z	Flag di zero
C	Flag di carry
DC	Digit carry (Half flag)

Tab. 5.3

OPERAZIONI ORIENTATE AL **BYTE** CON I FILE REGISTER		FLAG		
		Z	**C**	**DC**
ADDWF f,d	Somma *W* e *f* e pone il risultato in *W* (d=0) o in *f* (d=1)	↕	↕	↕
ANDW f,d	AND tra *W* e *f* e pone il risultato in *W* (d=0) o in *f* (d=1)	↕		
CLRF f	Azzera *f*	↕		
CLRW	Azzera l'accumulatore *W*	↕		
COMF f,d	Complementa *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1)	↕		
DECF f,d	Decrementa *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1)	↕		
DECFSZ f,d	Decrementa *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1); salta l'istruzione successiva se il risultato dell'operazione è zero			
INCF f,d	Incrementa *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1)	↕		
INCFSZ f,d	Incrementa *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1); salta l'istruzione successiva se il risultato dell'operazione è zero			
IORWF f,d	EX NOR tra *W* e *f* e pone il risultato in *W* (d=0) o in *f* (d=1)	↕		
MOVF f,d	Sposta f in *W* (d = 0) o in *f* (d = 1)	↕		
MOVWF	Sposta *W* in *f*			
NOP	Nessuna operazione			
RLF f,d	Ruota a sinistra, attraverso il carry, il contenuto di f e pone il risultato in *W* (d=0) o in *f* (d = 1)		↕	
RRF f,d	Ruota a destra, attraverso il carry, il contenuto di f e pone il risultato in *W* (d = 0) o in *f* (d = 1)		↕	
SUBWF f,d	sottrae *W* e *f* e pone il risultato in *W* (d = 0) o in *f* (d = 1)	↕	↕	↕
SWAPF f,d	Scambia i semi byte (*nibble*) di *f* e pone il risultato in *W* (d = 0) o in *f* (d=1)			
XORWF f,d	EX OR tra *W* e *f* e pone il risultato in *W* (d = 0) o in *f* (d=1)	↕		
OPERAZIONI ORIENTATE AL **BIT** CON I FILE REGISTER		**Z**	**C**	**DC**
BCF f,b	Azzera il bit *b* di *f* (b = 0÷7)			
BSF f,b	Pone a uno il bit *b* di *f* (b = 0÷7)			
BTFSC f,b	Testa il bit *b* di *f* e salta l'istruzione successiva se esso è zero			
BTFSS f,b	Testa il bit *b* di *f* e salta l'istruzione successiva se esso è uno			

Tab. 5.3

OPERAZIONI DI CONTROLLO E CON LETTERALI		Z	C	DC
ADDLW k	Somma il valore *k* all'accumulatore (*W*)	↕	↕	↕
ANDLW k	Fai la AND tra l'accumulatore (*W*) e il valore *k*	↕		
CALL k	Chiama la subroutine all'indirizzo *k*			
CLRWDT	Azzera il *Watchdog*			
GOTO k	Salta all'indirizzo *k*.			
IORLW k	Fai la EX NOR tra l'accumulatore (*W*) e il valore *k*	↕		
MOVLW k	Carica il valore *k* nell'accumulatore (*W*)			
RETFIE	Ritorna dalla routine di servizio dell' *interrupt*			
RETLW k	Ritorna dalla *subroutine* ponendo il valore k nell'accumulatore (*W*)			
RETURN	Ritorna dalla *subroutine*			
SLEEP	Poni il microcontroller in *standby*.			
SUBLW k	Sottrai il valore *k* dall'accumulatore (*W*)	↕	↕	↕
XORLW k	Fai la EX OR tra l'accumulatore (*W*) e il valore *k*	↕		

OPERAZIONI SPECIALI		OPERAZIONI EQUIVALENTI	Z	C	DC
ADDCF f,d	Somma *f* con il *carry* e pone il risultato in *W* (d=0) o in *f* (d=1)	BTFSC 3,0 INCF f,d	↕		
ADDDCF f,d	Somma *f* con il *digit carry* e pone il risultato in *W* (d=0) o in *f* (d=1)	BTFSC 3,1 INCF f,d	↕		
B k	Salta all'indirizzo *k*	GOTO k			
BC k	Salta se c'è carry all'indirizzo *k*	BTFSC 3,0 GOTO k			
BDC k	Salta se c'è *digit carry* all'indirizzo *k*	BTFSC 3,1 GOTO k			
BNC k	Salta se non c'è *carry* all'indirizzo *k*	BTFSS 3,0 GOTO k			
BNDC k	Salta se non c'è *digit carry* all'indirizzo *k*	BTFSS 3,1 GOTO k			
BNZ k	Salta se non c'è *zero* all'indirizzo *k*	BTFSS 3,2 GOTO k			
BZ k	Salta se c'è *zero* all'indirizzo *k*	BTFSC 3,2 GOTO k			

Le operazioni speciali sono anche dotate da set di istruzioni semplici ed equivalenti Tab. 5.3 e Tab.5.4

Tab.5.4

OPERAZIONI SPECIALI		OPERAZIONI EQUIVALENTI	Z	C	DC
CLRC	Azzera il *flag* di *carry*	BCF 3,0			
CLRDC	Azzera il *flag* di *digit carry*	BCF 3,1			
CLRZ	Azzera il *flag* di *zero*	BCF 3,2			
LCALL k	Chiamata a *sub* lunga	BSF 0Ah,3 BSF 0Ah,4 CALL k			
LGOTO k	Salto lungo	BSF 0Ah,3 BSF 0Ah,4 GOTO k			
MOVFW f	Carica *f* in *W*	MOVF f,0	↕		
NEGF f,d	Complementa *f* e pone il risultato in *W* (d=0) o in *f* (d=1)	COM f,1 INCF f,d	↕		
SETC	Pone a UNO il *flag* di *carry*	BSF 3,0			
SETDC	Pone a UNO il *flag* di digit *carry*	BSF 3,1			
SETZ	Pone a UNO il *flag* di *zero*	BSF 3,2			
SKPC	Salta la successiva istruzione se c'è *carry*	BTFSS 3,0			
SKPDC	Salta la successiva istruzione se c'è digit *carry*	BTFSS 3,1			
SKPNC	Salta la successiva istruzione se non c'è *carry*	BTFSC 3,0			
SKPNZ	Salta la successiva istruzione se non c'è *zero*	BTFSC 3,1			
SKPZ	Salta la successiva istruzione se c'è *zero*	BTFSC 3,2			
SUBCF f,d	Sottrai *f* con il *carry* e pone il risultato in *W* (d=0) o in *f* (d=1)	BTFSS 3,2	↕		
SUBDCF f,d	Sottrai *f* con il digit carry e pone il risultato in *W* (d=0) o in *f* (d=1)	BTFSC 3,0 DECF f,d	↕		
TSTF f	Testa il *file register f*	MOVF f,1	↕		
TRIS f	Pone contenuto di *W* nel registro di configurazione delle porte				

Per poter scrivere programmi in assembler per i microcontrollori della MICROCHIP è possibile utilizzare l'ambiente grafico di sviluppo MPLAB, il quale è fornito da un editore di testo, un'assemblatore (MPASM), un debugger ed un simulatore.
Le istruzioni riconosciute dall'assemblatore MPASM sono strutturate nel seguente modo:

Mnemonico Operando (o Opernandi)

Se nelle istruzioni ci sono 2 operandi, essi vengono separati da una virgola. Le istruzioni sono dei mnemonici che utilizzano le lettere dell'alfabeto ed illustrano sinteticamente il tipo di istruzione che deve essere svolta, nella maggior parte dei casi indicano anche il registro con cui si operare (un ***file register***, l'***accumulatore*** o entrambi). L'accumulatore è individuato dalla lettera W.

Vediamo ora le singole istruzioni in dettaglio…

ADDLW ADD Literal and W

Somma la costante k a W

Sintassi: **addlw k**

Operazione equivalente: W = W + k

Descrizione: Somma la costante **k** al valore memorizzato nell'accumulatore **W** e mette il risultato nell'accumulatore.

Esempio:

```
      org   00H
start
      movlw 10
      addlw 12
      ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà 22.

Note: Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 1 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

ADDWF ADD W and F

Somma il valore contenuto in W con il valore contenuto nel registro F

Sintassi: **addwf f,d**

Operazione equivalente: d = W + f (dove d può essere W o f)

Descrizione: Questa istruzione somma il valore contenuto nell'accumulatore **W** con il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Esempio:

Vediamo un esempio di somma tra due registri:

```
add1   equ    0CH
add2   equ    0DH

       org     00H
       movlw   10          ;Primo addendo = 10
       movwf   add1
       movlw   15          ;Secondo addendo = 15
       movwf   add2
       movf    add1,W      ;W = add1
       addwf   add2,W      ;W = W + add2
```

Note: Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 1 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

ANDLW — AND Literal with W

Effettua l'AND tra W ed una costante k

Sintassi: **andlw k**

Operazione equivalente: W = W AND k

Descrizione: Effettua l'AND tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**. Il risultato viene memorizzato nell'accumulatore.

Esempio:

```
       org    00H
start
       movlw  10101010B
       andlw  11110000B
       …
```

Dopo aver eseguito questo programma l'accumulatore W varrà 10100000B.

Note: Questa istruzione influenza il bit **Z** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

ANDWF AND W with F

Effettua l'AND logico tra il valore contenuto in W ed il valore contenuto nel registro F

Sintassi: **andwf f,d**

Operazione equivalente: d = W AND f (dove d può essere W o f)

Descrizione: Questa istruzione effettua l'AND logico tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d** = **W** il risultato viene memorizzato nel **registro W**
Per **d** = **F** il risultato viene memorizzato nel **registro f**

Esempio:

Spesso l'AND logico viene utilizzato per mascherare il valore di alcuni bit all'interno di un registro. Se ad esempio volessimo estrarre dal numero binario 01010101B i quattro bit meno significativi al fine di ottenere il seguente valore 00000101B, basterà preparare una maschera del tipo 00001111B e farne l'AND con il nostro valore di partenza, vediamo come:.

```
movlw   01010101B       ;Memorizza nel registro
movwf   0CH             ; all'indirizzo 0CH il valore iniziale da  mascherare
movlw   00001111B       ;Prepara la maschera di bit
andwf   0CH,W           ;Effettua l'AND e memorizza il risultato in W
```

Il risultato in W sarà 00000101B come richiesto.

```
W = 00001111 AND
f  = 01010101 =
---------------
W = 00000101
```

La **ANDWF** influenza il bit **Z** del registro **STATUS** che varrà 1 se il risultato dell'operazione è 0.

BCF Bit Clear F

Azzera un bit nel registro F

Sintassi: **bcf f,b**

Operazione equivalente: f(b) = 0

Descrizione: Questa istruzione azzera il **bit b** del registro all'indirizzo **f**.

Esempio:

```
parm1   equ    0CH
        org    00H
        movlw  11111111B      ;Valore iniziale
        movwf  parm1
        bcf    parm1,0        ;D0=0
```

Al termine del programma il registro parm1 varrà **11111110B**.

Note: Questa istruzione non influenza alcun bit di stato

BSF Bit Set F

Mette a uno un bit nel registro F

Sintassi: **bsf f,b**

Operazione equivalente: f(b) = 1

Descrizione: Questa istruzione mette a uno il **bit b** del registro all'indirizzo **f**.

Esempio:

```
parm1   equ    0CH
        org    00H
        movlw  00000000B  ;Valore iniziale
        movwf  parm1
        bsf    parm1,0    ;D0=1
```

Al termine del programma il registro parm1 varrà **00000001B**.

Note: Questa istruzione non influenza alcun bit di stato.

BTFSC Bit Test F, Skip if Clear

Salta l'istruzione successiva se un bit nel registro F vale 0

Sintassi: **btfsc f,b**

Operazione equivalente: f(b) = 0 ? Si, salta una istruzione

Descrizione: Testa il bit b contenuto nel registro all'indirizzo f e salta l'istruzione successiva se questo vale 0.

Esempio:

```
parm1   equ     0CH
        org     00H
        movlw   11111110B       ;Valore iniziale
        movwf   parm1
loop
        btfsc   parm1,0         ;D0 = 0 ? Si, esce
        goto    loop            ;No, esegue il loop
```

Questa programma esegue un loop infinito lo stesso programma non esegue il loop se sostituiamo l'istruzione:

```
movlw  11111110B    ;Valore iniziale
```

con l'istruzione:

```
movlw  11111111B    ;Valore iniziale
```

Note: Questa istruzione non influenza alcun bit di stato

BTFSS Bit Test F, Skip if Set

Salta l'istruzione successiva se un bit nel registro F vale 1

Sintassi: **btfss f,b**

Operazione equivalente: f(b) = 1 ? Si, salta una istruzione

Descrizione: Testa il bit b contenuto nel registro all'indirizzo f e salta l'istruzione successiva se questo vale 1.

Esempio:

```
parm1   equ    0CH
```

```
        org     00H
        movlw   11111110B       ;Valore iniziale
        movwf   parm1
loop
        btfss   parm1,0         ;D0 = 1 ? Si, esce
        goto    loop            ;No, esegue il loop
```

Se invece del valore 11111110b si caricava un il valore 11111111b l'istruzione btfss avrebbe generato un loop.

Note: Questa istruzione non influenza alcun bit di stato

CALL Subroutine CALL

Chiamata a subroutine

Sintassi: **call k**

Descrizione: Richiama in esecuzione una subroutine memorizzata all'indirizzo **k**.Il parametro k può essere specificato utilizzando direttamente il valore numerico dell'indirizzo oppure la relativa label.

Esempio:

```
        org     00H
        call    ledOn
        ...

                ;Subroutine di accensione di un led
ledOn
        bsf     PORTB,LED1
        return
```

Quando la CPU del PIC incontra una istruzione CALL, memorizza nello STACK il valore del registro PC + 1 in modo da poter riprendere l'esecuzione dall'istruzione successiva alla CALL, quindi scrive nel PC l'indirizzo della subroutine saltando all'esecuzione di quest'ultima. Il valore originale del PC viene ripristinato all'uscita della subroutine con l'esecuzione dell'istruzione di ritorno RETURN o RETLW. Nel PIC16C84 sono disponibili 8 livelli di stack, per cui il numero massimo di CALL rientranti, ovvero di istruzioni CALL all'interno di subroutine che a loro volta contengono altre CALL, è limitato ad 8 livelli.

Note: Questa istruzione non influenza nessun bit di stato.

CLRF CLeaR F register

Azzera il registro F

Sintassi: clrf f

Operazione equivalente: f = 0

Descrizione: Questa istruzione azzera il valore contenuto nel registro indirizzato dal parametro **f**.

Esempio:

Ipotizziamo di voler azzerare il registro **TMR0** il cui indirizzo è 01H esadecimale, l'istruzione da eseguire sara:

```
clrf  01H
```

Dopo l'esecuzione di questa istruzione il bit **Z** del registro **STATUS** viene messo a **1**.

CLRW CLeaR W register

Azzera il registro W

Sintassi: clrw

Operazione equivalente: W = 0

Descrizione: Azzera il valore contenuto nel registro **W**.

Note: Dopo l''esecuzione di questa istruzione il bit **Z** del registro **STATUS** viene messo a **1**.

CLRWDT CLeaR WatchDog Timer

Reset del timer del watchdog

Sintassi: clrwdt

Descrizione: Questa istruzione deve essere utilizzata quando programmiamo il PIC con l'opzione Watchdog abilitata (fusibile WDTE). In questa modalità il PIC abilita un timer che, una volta trascorso un determinato tempo, effettua il reset del PIC. Per evitare il reset, il nostro programma dovrà eseguire ciclicamente l'istruzione CLRWDT per azzerare il timer. Se non azzeriamo il timer in tempo, la circuiteria del watchdog (dall'inglese cane da guardia) interpreterà questo come un blocco del programma in esecuzione ed effettuerà il reset al fine di sbloccarlo.

Esempio:

```
        org     00H
loop
        clrwdt
        goto    loop
```

Note: Questa istruzione non influenza nessun bit di stato

COMF COMplement F

Effettua il complemento del registro F

Sintassi: **comf f,d**

Operazione equivalente: d = NOT f (dove d può essere W o f)

Descrizione: Questa istruzione effettua il complemento del valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** determina la destinazione del valore ottenuto.

Esempio:

```
parm1   equ   0CH
        org   00H
        movlw 01010101B
        movwf parm1
        comf  parm1,F
              ...
```

Al termine dell'esecuzione del programma il valore del registro **parm1** sarà **10101010B**.

Note: Questa istruzione influenza il bit **Z** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

DECF DECrement F register

Azzera il contenuto del registro F

Sintassi: **decf f,d**

Operazione equivalente: d = f -1 (dove d può essere W o f)

Descrizione: Questa istruzione decrementa il contenuto del registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Esempio:

Con il seguente programma scriviamo il valore 23H nel registro all'indirizzo 0CH e quindi lo decrementiamo di uno. Al termine dell'esecuzione il registro alla locazione 0CH conterrà il valore 22H.

```
movlw   23H         ;Scrive in W il valore 23H
movwf   0CH         ;Copia nel registro 0CH il valore di W
decf    0CH,F       ;Decrementa il valore Contenuto nel registro 0CH
```

Questa istruzione influenza il bit **Z** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

DECFSZ DECrement F, Skip if Zero

Decrementa il valore del registro f e salta l'istruzione successiva se il risultato vale zero

Sintassi: **decfsz f,d**

Operazione equivalente: d = f -1 (dove d può essere W o f) se d = 0 salta

Descrizione: Decrementa il valore del registro all'indirizzo **f** e se il risultato vale zero salta l'istruzione successiva. Il risultato del decremento può essere memorizzato nello stesso registro **f** oppure nell'accumulatore **W** in base al valore del flag **d**.

Esempio:

```
counter equ     0CH
        org     00H
        movlw   10              ;counter = 10
        movwf   counter
loop
        decfsz  counter,F       ;counter = counter -1;counter = 0 ? Si esce,
        goto    loop            ;No, continua
        ……..
```

Questa programma esegue per 10 volte l'istruzione decfsz finchè esce per counter = 0.

Note: Questa istruzione non influenza alcun bit di stato.

GOTO GO TO address

Vai in esecuzione all'indirizzo k

Sintassi: goto k

Descrizione: Determina un salto del programma in esecuzione all'indirizzo k. Il parametro k può essere specificato utilizzando direttamente il valore numerico dell'indirizzo oppure la relativa label.

Esempio:

```
        org   00H
loop
        goto  loop
```

Questro programma esegue un ciclo (loop) infinito.

Note: Questa istruzione non influenza nessun bit di stato.

INCF INCrement F

Incrementa il valore del registro all'indirizzo F

Sintassi: incf f,d

Operazione equivalente: d = f +1 (dove d può essere W o f)

Descrizione: Incrementa il contenuto del registro all'indirizzo **f** e memorizza il risultato nello stesso registro o nell'accumulatore **W** in base al valore del flag **d**

Note: Questa istruzione influenza il bit **Z** del registro **STATUS**..

- **Z** vale 1 se il risultato dell'operazione vale 0.

INCFSZ INCrement F, Skip if Zero

Incrementa il valore del registro f e salta l'istruzione successiva se il risultato vale zero

Sintassi: incfsz f,d

Operazione equivalente: d = f +1 (dove d può essere W o f) se d = 0 salta

Descrizione: Incrementa il valore del registro all'indirizzo **f** e se il risultato vale zero salta l'istruzione successiva. Il risultato dell'incremento può essere memorizzato nello stesso registro **f** oppure nell'accumulatore **W** in base al valore del flag **d**.

Esempio:

```
counter equ     0CH
        org     00H
        movlw   250         ;counter = 250
        movwf   counter
   loop
        incfsz  counter,F   ;counter = counter + 1; counter = 0 ? Si esce
        goto    loop        ;No, continua
        ..........
```

Questa programma esegue per 6 volte l'istruzione incfsz finché esce per counter = 0. Essendo counter un registro a 8 bit quando viene incrementato dal valore 255 assume nuovamente valore 0.

Note: Questa istruzione non influenza alcun bit di stato.

IORLW Inclusive OR Literal with W

Effettua l'OR inclusivo tra W ed una costante k

Sintassi: iorlw k

Operazione equivalente: W = W OR k

Descrizione: Effettua l'OR inclusivo tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**.

Esempio:

```
        org     00H
start
        movlw   00001111B
        iorlw   11110000B
        ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà **11111111B**.

Note: Questa istruzione influenza il bit **Z** dEl registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

IORWF Inclusive OR W with F

Effettua l'OR inclusivo tra il valore contenuto in W ed il valore contenuto nel registro F

Sintassi: **iorwf f,d**

Operazione equivalente: d = f OR W (dove d può essere W o f)

Descrizione: Questa istruzione effettua l'OR inclusivo tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** determina dove viene memorizzato il risultato dell'operazione:

Esempio:

```
parm1   equ     0CH
        org     00H
        movlw   00001111B
        movwf   parm1
        movlw   11111111B
        iorwf   parm1,F
```

Al termine dell'esecuzione il valore del registro parm1 sarà **11111111B**.

Note: Questa istruzione influenza il bit **Z** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

MOVLW MOVe Literal to W

Assegna a W un valore costante

Sintassi: **movlw k**

Operazione equivalente: W = k

Descrizione: Assegna all'accumulatore **W** il valore costante **k**.

Esempio:

```
      org  00H
start
      movlw  20
      ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà 20.

Note: Questa istruzione non influenza nessun bit di stato.

MOVF MOVe F

Muove il contenuto del registro F

Sintassi: **movf f,d**

Operazione equivalente: d = f (dove d può essere W o f)

Descrizione: Questa istruzione copia il contenuto del registro indirizzato dal parametro **f** o nell'accumulatore **W** o nello stesso **registro F**. Il parametro **d** determina la destinazione. In questo caso l'utilità dell'istruzione sta nel fatto che viene alterato il bit Z del flag STATUS in base al valore contenuto nel **registro f**.

Esempio:

L'esempio seguente copia il valore contenuto nel registro all'indirizzo 0CH nell'accumulatore W:

```
      movf  0CH,W
```

MOVWF MOVe W to F

Muove il contenuto del registro W nel registro F

Sintassi: **movwf f**

Operazione equivalente: f = W

Descrizione: Questa istruzione copia il contenuto del **registro W** nel registro indirizzato dal parametro **f**.

Esempio:

Ipotizziamo di voler scrivere il valore **10H** (esadecimale) nel registro **TMR0**. Le istruzioni da eseguire sono le seguenti.

```
movlw   10H     ;Scrive nel registro W il valore 10H
movwf   01H     ;e lo memorizza nel registro TMR0
```

Note: L'esecuzione della **MOVWF** non influenza nessun bit di stato.

NOP No OPeration

Nessuna operazione

Sintassi: **nop**

Descrizione: Questa istruzione non esegue nessuna operazione ma è utile per inserire ritardi pari ad un ciclo macchina .

Esempio:

Utilizzando un quarzo da **4MHz** potremo ottenere un ritardo pari ad **1µs** per ogni istruzione **NOP** inserita nel nostro source..

```
nop     ;Esegue un ritardo pari ad 1µs
```

Note: La **NOP** non influenza nessun bit di stato.

OPTION load OPTION register

Assegna il valore in W al registro OPTION

Sintassi: **option**

Operazione equivalente: OPTION = W

Descrizione: Questa istruzione memorizza nel registro speciale OPTION il valore contenuto nell'accumulatore W.

Esempio:

```
    org     00H
start
      movlw   01000100B
      option
      ...
```

Note: Questa istruzione esiste per mantenere la compatibilità con i PIC prodotti finora, la Microchip ne sconsiglia l'uso. In alternativa è consigliabile usare le seguenti istruzioni.

```
    org     00H
start
      bsf     STATUS,RP0      ;Attiva il banco registri 1
      movlw   01000100B
      movwf   OPTION_REG
      ...
```

In pratica si consiglia di scrivere direttamente nel registro OPTION presente nel banco 1 dei registri del PIC utilizzando la MOVWF anziché l'istruzione OPTION che in futuro potrebbe non essere più implementata.

Questa istruzione non influenza nessun bit di stato.

RETFIE RET From Interrupt

Ritorna da una subroutine

Sintassi: **retfie**

Descrizione: Questa istruzione deve essere inserita al termine di ogni subroutine di gestione degli interrupt per ridare il controllo al programma principale.

Esempio:

```
        org    00H
loop
        goto   loop   ;Loop infinito
        org    04H    ;Interrupt vector

intHandler
……..
        retfi         ;Ritorna dall'interrupt
```

In questo source il programma principale esegue un loop infinito. Se abilitiamo uno degli interrupt del 16C84 non appena esso si verificherà il controllo verrà dato automaticamente al programma allocato dall'indirizzo 04H (nell'esempio intHandler), l'istruzione RETFI determinerà quindi il ritorno al loop principale.

Note: Questa istruzione non influenza alcun bit di stato.

RETURN RETURN from subroutine

Ritorna da una subroutine

Sintassi: **return**

Descrizione: Questa istruzione deve essere inserita al termine di ogni subroutine per riprendere l'esecuzione del programma principale.

Esempio:

```
        org     00H
        call    mySub1
        ....
mySub1
        nop
        return
```

Note: Nel PIC16F84 possono essere annidate fino ad 8 chiamate a subroutine del tipo:

```
        org     00H
        call    mySub1
        ....
mySub1
        call    mySub2
        return
mySub2
        call    mySub3
        return
mySub3
        return
```

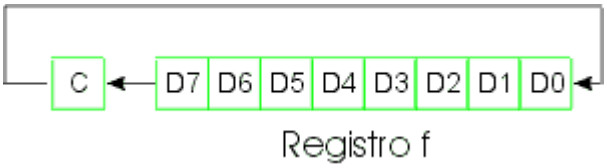
RLF Rotate Left F through carry

Ruota a sinistra il contenuto del registro f passando per il Carry

Sintassi: **rlf f,d**

Operazione equivalente: d = f << 1 (dove d può essere W o f)

Descrizione: Ruota i bit contenuti nel registro all'indirizzo **f** verso sinistra (ovvero dai bit meno significativi verso quelli più significativi) passando per il bit **CARRY** del registro **STATUS** come illustrato in figura:



Registro f

Il contenuto del bit CARRY del registro status viene spostato nel bit D0 mentre il valore in uscita dal bit D7 viene spostato nel CARRY.

Il valore del parametro **d** determina la destinazione del risultato ottenuto al termine della rotazione:

Per **d** = **W** il risultato viene memorizzato nel **registro W**
lasciando il registro f invariato.
Per **d** = **F** il risultato viene memorizzato nello stesso **registro f**

Esempio:

```
parm1   equ    0CH
        org    00H
        clrf   C,STATUS    ;Azzera il CARRY
        movlw  01010101B   ;Valore iniziale
        movwf  parm1
        rlf    parm1,F
```

Al termine del programma il registro **parm1** varrà **10101010B** mentre il CARRY varrà 0.

Note: Questa istruzione non influenza nessun altro bit di stato oltre al CARRY.

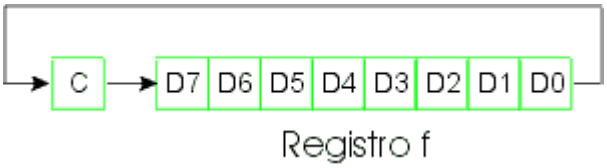
RRF Rotate Right F through carry

Ruota a destra il contenuto del registro f passando per il Carry

Sintassi: `rrf f,d`

Operazione equivalente: d = f >> 1 (dove d può essere W o f)

Descrizione: Ruota i bit contenuti nel registro all'indirizzo **f** verso destra (ovvero dai bit più significativi verso quelli meno significativi) passando per il bit **CARRY** del registro **STATUS** come illustrato in figura:



Il contenuto del bit CARRY del registro status viene spostato nel bit D7 mentre il valore in uscita dal bit D0 viene spostato nel CARRY.

Il valore del parametro **d** determina la destinazione del risultato ottenuto al termine della rotazione:

Per **d** = **W** il risultato viene memorizzato nel **registro W** lasciando il registro f invariato.
Per **d** = **F** il risultato viene memorizzato nello stesso **registro f**

Esempio:

```
parm1  equ   0CH
       org    00H
       clrf    C,STATUS    ;Azzera il CARRY
       movlw   01010101B   ;Valore iniziale
       movwf   parm1
       rrf     parm1,F
```

Al termine del programma il registro **parm1** varrà **00101010B** mentre il CARRY varrà 1.

Note: Questa istruzione non influenza nessun altro bit di stato oltre al CARRY.

SLEEP go into standby mode

Mette il PIC in standby

Sintassi: **sleep**

Descrizione: Questa istruzione blocca l'esecuzione del programma in corso e mette il PIC in stato di standby (sleep dall'inglese to sleep, dormire).

Esempio:

```
        org   00H
start
        sleep
```

Note: Questa istruzione non influenza nessun bit di stato

SUBLW SUBtract W from Literal

Sottrae a k il valore in W

Sintassi: **sublw k**

Operazione equivalente: W = k - W

Descrizione: Sottra alla costante **k** il valore memorizzato nell'accumulatore **W**.

Esempio:

```
        org   00H
start
        movlw  10
        sublw  12
        ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà 2.

Note: Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 1 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

SUBWF SUBstract W from F

Sottrae il valore contenuto in W dal valore contenuto nel registro F

Sintassi: subwf f,d

Operazione equivalente: d = f - W (dove d può essere W o f)

Descrizione: Questa istruzione sottrae il valore contenuto nel **registro W** dal valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Esempio:

Analizziamo un esempio estratto dal datasheet della Microchip:

Se inseriamo l'istruzione:

```
subwf   REG1,F
```

Dove REG**1** è l'indirizzo di un qualsiasi registro specificato con la direttiva:

```
REG1    RES     1
```

Per valori iniziali di REG1=3 e W=2, dopo l'esecuzione avremo REG1=1 e C=1 in quanto il risultato è positivo.

Per valori iniziali di REG1=2 e W=2 dopo l'esecuzione avremo REG1=0 e C=1 perché il risultato è sempre positivo.

Per valori iniziali di REG1=1 e W=2, avremo REG1=FFH ovvero -1 quindi C=0 perchè il risultato è negativo.

Note: Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 1 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

TRIS load TRIS register

Assegna il valore in W al registro TRIS

Sintassi: tris f

Operazione equivalente: TRIS di f = W

Descrizione: Questa istruzione memorizza in uno dei registri speciale TRIS il valore contenuto nell'accumulatore W. I registri TRIS determinano il funzionamento in ingresso e uscita delle linea di I/O del PIC. Esiste un registro TRIS per ogni porta di I/O denominato TRISA, TRISB, ecc.

Esempio:

```
    org     00H
start
    movlw   11111111B
    tris    PORTA
    ...
```

Note: Questa istruzione esiste per mantenere la compatibilità con i PIC prodotti finora, la Microchip ne sconsiglia l'uso. In alternativa è consigliabile usare le seguenti istruzioni.

```
    org     00H
start
    bsf     STATUS,RP0      ;Attiva il banco registri 1
    movlw   11111111B
    movwf   TRISA
    ...
```

In pratica si consiglia di scrivere direttamente nei registri TRIS presenti nel banco 1 dei registri del PIC utilizzando la MOVWF anziché l'istruzione TRIS che in futuro potrebbe non essere più implementata.

Note: Questa istruzione non influenza nessun bit di stato.

XORLW Exclusive OR Literal with W

Effettua l'OR esclusivo tra W ed una costante k

Sintassi: xorlw k

Operazione equivalente: W = W XOR k

Descrizione: Effettua l'OR esclusivo tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**.

Esempio:

```
  org     00H
start
  movlw   00000000B
  xorlw   11110000B
  ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà 11110000B.

Note: Questa istruzione influenza il bit **Z** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.

XORWF eXclusive OR W with F

Effettua l'OR esclusivo tra il valore contenuto in W ed il valore contenuto nel registro F

Sintassi: xorwf f,d

Operazione equivalente: d = f XOR W (dove d può essere W o f)

Descrizione: Questa istruzione effettua l'OR esclusivo (XOR) tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato. Questa istruzione influenza i bit **Z** del registro **STATUS** che varrà 1 se il risultato dell'operazione è 0.

Esempio:

Ipotizziamo di dover effettuare lo XOR tra il registro W ed il registro **REG1** da noi definito all'indirizzo **0CH** con la direttiva:

```
REG1   EQU   0CH
```

possiamo utilizzare l'istruzione **IORWF** in due forme a seconda di dove vogliamo mettere il risultato, ovvero:

```
xorwf  COUNTER,F    ;COUNTER = COUNTER XOR W
```

oppure:

```
xorwf  COUNTER,W    ;W = COUNTER XOR W
```

Note:

L'OR esclusivo (XOR) è un'operazione tra due bit in cui il bit risultante vale 0 se i due bit sono uguali.

Spesso lo XOR viene utilizzato nell'assembler del PIC per effettuare la comparazione tra due valori in mancanza di un'istruzione specifica.

Vediamo come:

ipotizziamo di avere un valore nel registro REG1 e di voler verificare se è uguale a 57H. Le istruzioni da eseguire sono le seguenti:

```
movlw  57H          ;W = Valore da comparare = 57H; isultato. W = 57H
xorwf  REG1,W       ;W = W XOR REG1 Effettua lo XOR con il valore in REG1
btfss  STATUS,Z     ;Salta l'istruzione seguente se il risultato dello XOR vale 0,
                    ;ovvero se il valore di REG1 e' pari a 57H
goto   diverso      ;Salta se diverso da 57H
goto   uguale       ;Salta se uguale da 57H
```

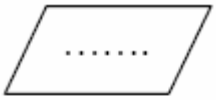
6. Principi base per la programmazione

6.1 Introduzione alla programmazione

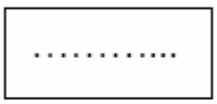
Per creare un programma bisogna seguire quattro fasi:

1. Analisi
2. Realizzazione di uno schema a blocchi o flow-chart
3. Scrittura del programma
4. Compilazione

Analisi: è una fase molto importante perché si descrive cosa deve fare il programma; si considerano le variabili da utilizzare e le funzioni che si devono eseguire.
Realizzazione di uno schema a blocchi o flow-chart: In questa fase invece si realizza uno schema primitivo del programma che ci permetterà di individuare eventuali errori nel terzo passaggio. Questi flow-chart hanno dei blocchi standard che rappresentano delle operazioni particolari, dentro questi blocchi vanno inserite delle istruzioni o dei valori. I principali blocchi per realizzare uno schema sono:

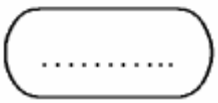


Questo blocco attribuisce i valori alle variabili, inizializza (crea) le variabili nella ram dell'elaboratore e visualizza il valore di una variabile.



Questo blocco ha invece ha la funzione di eseguire una istruzione, per intenderci le operazioni aritmetiche e altre funzioni.

Questo blocco invece esegue una scelta vero o falso, se è vero il programma esegue una certa serie d'istruzioni, mentre se è falso ne esegue una serie diversa. Questo comando si usa quando si deve uscire da un ciclo o si deve fare una scelta.



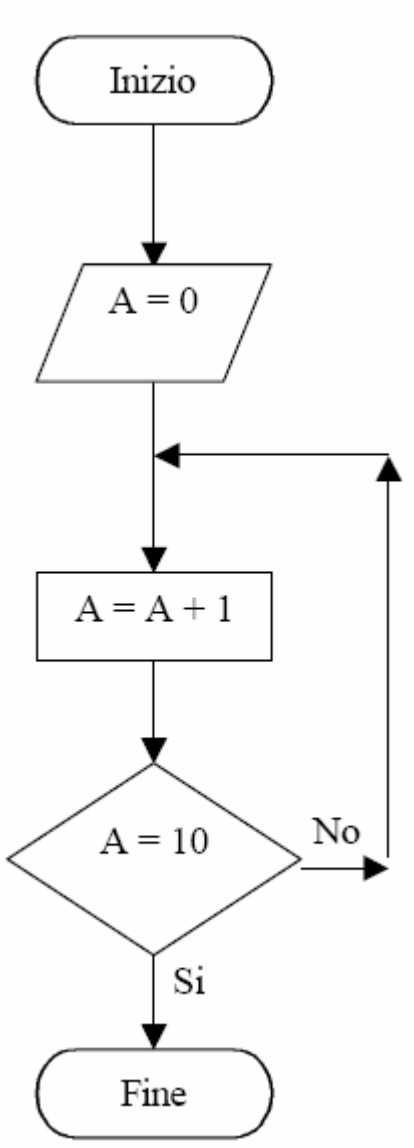
Questo blocco viene posto all'inizio e alla fine dello schema a blocchi e rappresenta soltanto dove il programma inizia e dove finisce.

Questi blocchi sono collegati da rami orientati che definiscono la sequenza delle operazioni da svolgere.

Esempio su come si esegue l'analisi e lo schema a blocchi.
Questo programma conta da 0 fino a 10.

Analisi:
1. Il programma ha bisogno di una variabile per contare fino a 10
2. Il programma deve eseguire un ciclo in modo da poter incrementare la variabile.

Diagramma a blocchi:



Con questo blocco si indica l'inizio del programma.

Con questo blocco si crea la variabile A e le si attribuisce il valore 0.

Questo blocco incrementa A di uno e salva il nuovo valore in A.

Con questo blocco si effettua una scelta se A è uguale a 10 allora esce dal ciclo se no esegue l'istruzione che che la precede fino a che non si verifica la situazione.

Con questo blocco si indica la fine del programma.

Scrittura del programma: In questa fase non si fa altro che tradurre lo schema a blocchi nel linguaggio di programmazione in cui si scrive (C/C++, Assembler, Basic, Pascal, ecc.).
Compilazione: Questa fase solitamente non è influenzabile dall'utente, poiché è il programma stesso che esegue questa operazione che traduce il programma scritto in Alto Livello (linguaggio umano) in linguaggio macchina ovvero un linguaggio a Basso Livello (comprensibile all'elaboratore).

6.2 Strutture basi di programmazioni

A prescindere dal linguaggio che si utilizza, vi sono strutture canoniche che si devono conoscere quali:
Il ciclo o loop, controllo in testa o in coda e scelta vero o falso

6.2.1 Il ciclo o loop, controllo in testa o in coda e scelta vero o falso

Queste strutture sono fondamentali perché i cicli e le scelte sono la parte più consistente di qualsiasi programma. Nel programmare in Assembler si farà grande uso di queste strutture visto che il set d'istruzioni del PIC 16F84 comprende solamente 35 istruzioni.

Il ciclo o loop:

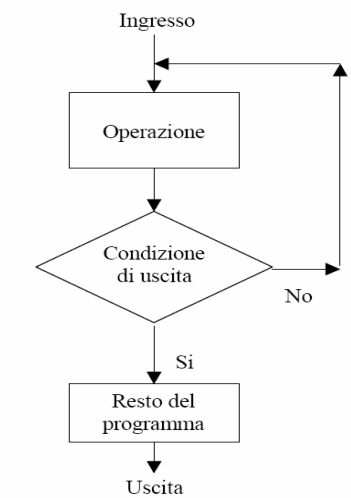
Questa struttura è quasi sempre accoppiato con un controllo in testa o in coda, con questa operazione si esegue un certo numero di volte una serie di funzioni fino a che sia soddisfatta una condizione. Questa condizione è verificata dal controllo in testa o in coda, la differenza tra queste due opzioni è che nel controllo in testa la condizione è controllata subito, senza quindi eseguire ciò che sta all'interno del ciclo, mentre nel controllo in coda prima di verificare la condizione si esegue ciò che sta all'interno del ciclo.

Esempio di ciclo con controllo in testa o in coda.

Controllo in testa

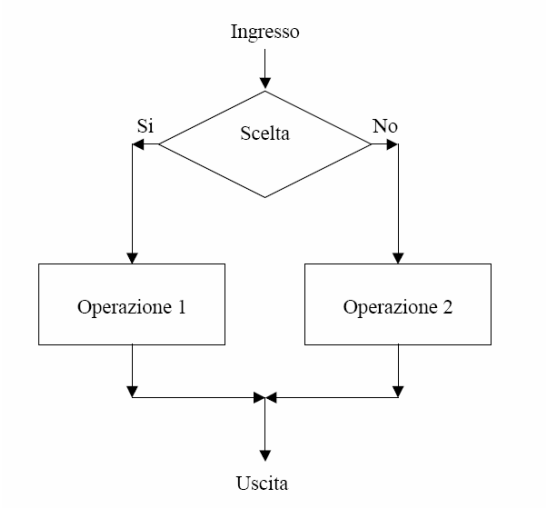
Questo tipo di ciclo ha un controllo in testa ovvero il dato viene prima controllato nel blocco di scelta, se la condizione è verificata il ciclo viene saltato, se la condizione invece non è verificata il programma esegue il ciclo fino a che si crea la condizione di uscita.

Controllo in Coda



Questo ciclo esegue prima l'operazione e poi controlla se la condizione è soddisfatta.
Nel caso positivo esce dal ciclo altrimenti continua l'iterazione.

La scelta vero o falso: Questa struttura esegue una scelta e in base al risultato, vero o falso, esegue una operazione anziché un'altra.



6.3 Linguaggio assembler

Il PIC 16F84 è un vero e proprio piccolo computer, dotato di una CPU, una RAM, una ROM e una memoria programma; il suo linguaggio di programmazione è l'assembler. Questo linguaggio è a basso livello, quindi è molto complicato e poco elastico, perciò, per poter eseguire anche solo una semplice operazione, come dare un valore a una variabile, bisogna usare due istruzioni.
Per programmare il PIC non basta conoscere il set d'istruzioni ma bisogna saper usare anche le direttive, quest'ultime non vengono eseguite durante il programma ma servono al programma **compilatore** per creare il file hex 1.

Il source

Per creare un programma bisogna seguire due fasi, la prima è la scrittura del source e la seconda è la compilazione.
Per creare quindi un listato bisogna generare un file txt; questo file può essere generato da Notepad, all'interno del quale si scriverà il programma e solo in seguito lo si trasformerà in un file hex.

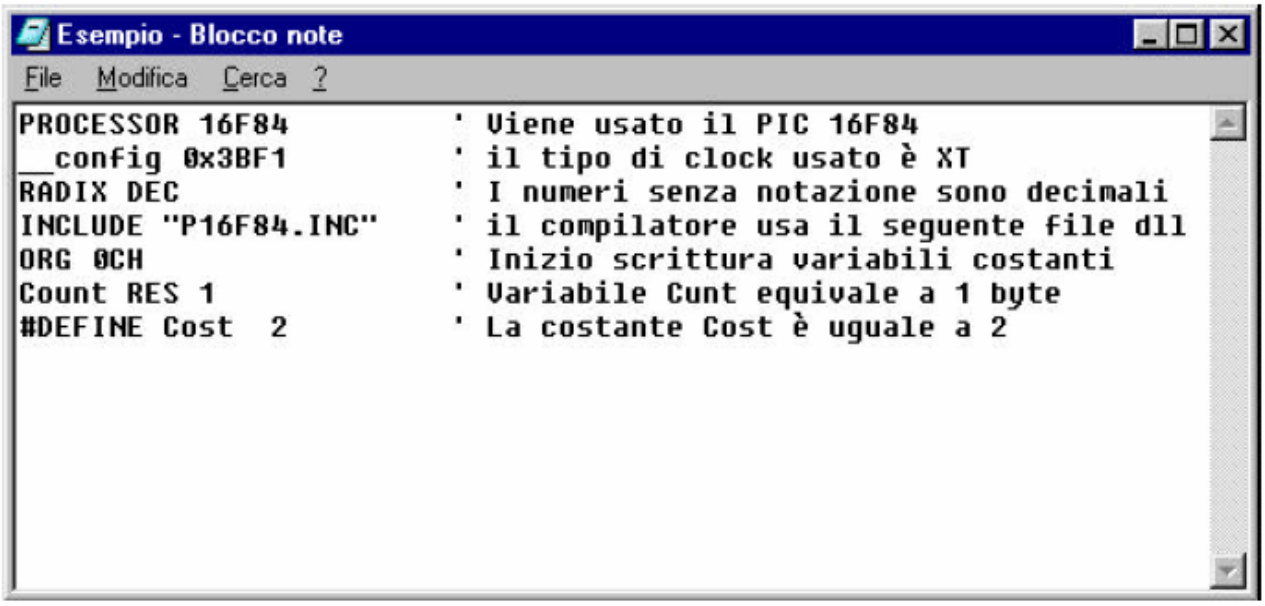
Le direttive

Come si è già detto per programmare il PIC bisogna prima impostare le direttive. Queste vanno scritte all'inizio del listato e definiscono: il tipo d'integrato, le variabili, le costanti e il tipo clock (direttiva opzionale).
PROCESSOR 16F84 = Questa direttiva definisce il tipo d'integrato usato.
__config = 0xFFF = Questa direttiva definisce il tipo di clock, il parametro da inserire dopo l'uguale è definito dal checksum del programma "programmatore ".
RADIX notazione = Questa direttiva definisce che tutti i numeri definiti nel programma senza la notazione definita nella direttiva sono da considerarsi :
decimale = DEC, esadecimale = HEX o binario = BIN.
INCLUDE "P16F84.INC" = Questa direttiva dice al programma compilatore quale libreria usare per creare il file hex.
RES … = Questa direttiva dice quanto è lunga una variabile se è uguale a 1 sarà di un byte se 2 sarà di due etc.
#DEFINE variabile valore = definisce una costante.
variabile EQU valore = definisce una costante.

6.4 Realizzazione e compilazione del source per il PIC

6.4.1 Realizzazione di un listato

Per creare un programma hex bisogna avere un file sorgente, questo file si può ottenere con un semplice programma di scrittura come Notepad. Le istruzioni vanno scritte incolonnate poiché il programma compilatore ammette una sola istruzione per riga.
La prima cosa da scrivere sono le direttive, queste dovranno specificare il tipo di integrato, le variabili, le costanti e eventualmente il clock. Bisogna per le variabili e le costanti specificare l'indirizzo dell'area ram in cui verranno create che corrisponde a ORG 0CH.
A questo punto bisogna impostare le porte, per far ciò bisogna modificare i registri Trisa e Trisb che corrispondono rispettivamente a PortA e PortB. Il registro TrisA è un registro composto da 5 bit, tale è infatti il numero di pin che corrispondono a questa porta di comunicazione; partendo con RA0 per il bit meno significativo fino a RA4 per il bit più significativo. Con 0 si definisce un pin di uscita mentre con 1 un pin di ingressoi. La stessa cosa si fa per la PortB con l'unica differenza che il suo registro TrisB è composto da 8 bit, tale infatti è il numero di piedini che la compongono.



A questo punto bisogna impostare le porte, per far ciò bisogna modificare i registri Trisa e Trisb che corrispondono rispettivamente a PortA e PortB. Il registro TrisA è un registro composto da 5 bit, tale è infatti il numero di pin che corrispondono a questa porta di comunicazione; partendo con RA0 per il bit meno significativo fino a RA4 per il bit più significativo. Impostando i bit o a 0 o a 1 abbiamo rispettivamente delle porte di output e input. La stessa cosa si fa per la PortB con l'unica differenza che il suo registro TrisB è composto da 8 bit, tale infatti è il numero di piedini che la compongono.
Prima d'impostare le porte di comunicazione bisogna dire al programma compilatore che ora si sta scrivendo nella memoria di programma, ciò si ottiene scrivendo ORG 00h.

Esempio - Blocco note

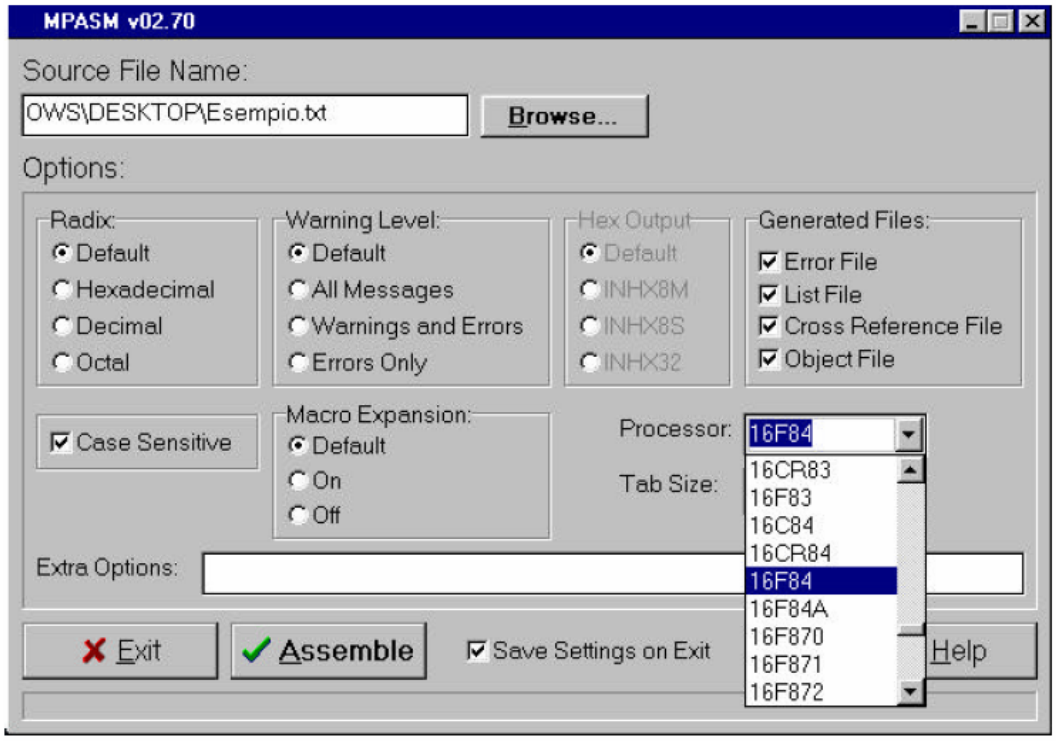
File Modifica Cerca ?

```
#DEFINE Cost  2          ' La costante Cost è uguale a 2
ORG 00H                  ' Inizio scrittura programma
bsf STATUS,RP0           ' Setta a uno il bit RB0 del registro Status
clrf PORTA               ' Setta a zero PORTA
clrf PORTB               ' Setta a zero PORTB
movlw B'01011'           ' Muovi la costante 01011 nel registro W
movwf TRISA              ' Muovi dall'accumulatore W al registro TrisA
movlw B'01010101'        ' Muovi la costante 01010101 nel registro W
movwf TRISB              ' Muovi dall'accumulatore W al registro TrisB
bcf STATUS,RP0           ' Setta a uno il bit RB0 del registro Status

listato programma

end
```

Una volta specificata l'area in cui si scrive dobbiamo settare a livello alto il bit RP0 del registro STATUS, questa operazione serve ad impedire che, per errore, durante l'esecuzione del programma, questi parametri vengano modificati. Ciò non toglie la possibilità di modificare, in seguito, l'impostazione delle porte.
A questo punto si muove il valore con notazione binaria dentro i due registri Tris, si setta a zero il bit RB0 e si comincia a scrivere il programma. Stilato il programma bisogna concludere con **end** per dire al programma compilatore che il listato è terminato.

6.4.2 Programma compilatore

Per generare il file hex bisogna disporre di un programma assemblatore: si consiglia il programma **Mpasm** della Microchip che viene distribuito in versione " freeware " e disponibile in dotazione al sistema di sviluppo MC-16. A questo punto, una volta aperto il programma, si imposti il tipo d'integrato (in questo caso il PIC 16F84) ed il percorso del file sorgente.



Una volta stabilito il tipo d'integrato e il percorso del file sorgente bisogna impostare il Generated Files cliccando su Error file – List file – Cross Ref. File e Object File come riportato in figura. La abilitazione del Error File è importante poiché segnala la presenza di un eventuale errore nel file.asm e genera un file.err.
Questo file può essere aperto con Notepad per individuare la riga interessata ed il tipo di errore.
A questo punto cliccare su " Assemble " per avviare la compilazione.
Se l'operazione è andata a buon fine procedere con il trasferimento del file .hex nel microcontrollore. Per fare ciò si dovrà utilizzare un appropriato softawre di programmazione.

6.4.3 Software per programmare il PIC

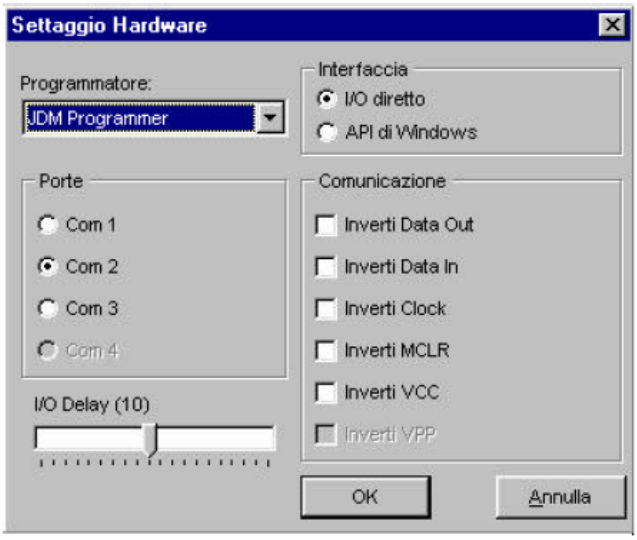
Introduzione

Il software per la programmazione del PIC non fa altro che gestire la porta COM; questi programmi si trovano disponibile in versione "freeware" in rete . Questi programmi sia che lavorino in ambiente Dos o in ambiente Window hanno le stesse opzioni, si consiglia l'uso del programmatore **Icprog** reperibile in rete. Questo programma (in dotazione al sistema MC-16 in versione italiana) è molto intuitivo e di facile applicazione. **Prima di procedere collegare il Sistema MC-16 al PC tramite il cavo R232 e commutare il deviatore S7 nella posizione PROGRAMMAZIONE**

Impostazioni di Icprog

Avviato il programma e richiamato il file.hex comparirà la seguente videata..
Prima di poter programmare un PIC bisogna impostare la porta di comunicazione ed alcuni parametri importanti. Per impostare la porta bisogna cliccare su settaggi :

Una volta selezionato **Hadware** si aprirà una seconda finestra **Settaggio Hardware** nella quale si dovranno effettuare seguenti impostazioni:

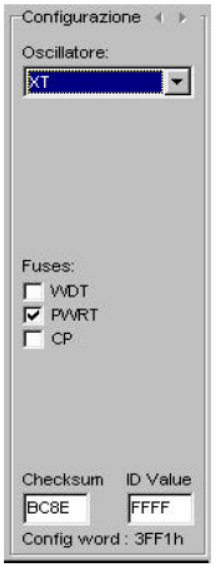


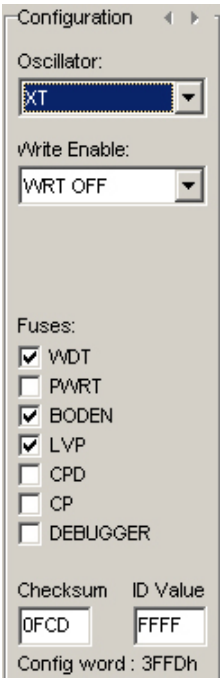
Cliccare su **Programmatore** e scegliere il JDM Programmer, quindi su **Porte** e scegliere la Com disponibile , su **Interfaccia** e scegliere I/O diretto. Premere **OK**
Selezionare il PIC dal menù a tendina che si trova in alto a destra.
NOTA BENE : Ci sono 2 tipi di PIC per le 2 famiglie 16F84 e 16F876, rispettivamente : Il PIC 16F84 e il PIC 16F84A; il PIC 16F876 ed il PIC 16F876A. Selezionare il PIC corretto, in base alla sigla riportata sul PIC in dotazione, affinché la programmazione vada a buon fine.
Un'altra serie di parametri importanti da impostare sono, l'**oscillatore** e i **fuses**. L'oscillatore deve essere impostato in base al tipo di clock che si ha (**vedi Tabella**).

Parametro oscillatore	Tipo di oscillatore
RC	Resistenza condensatore
XT	Standard quarzo
LP	Quarzo con basso assorbimento
HS	Quarzo alta frequenza

Stabilito l'oscillatore bisogna definire i **Fuses**. Questi parametri si riferiscono all'alimentazione, l'unico da selezionare è **PWRT**.
Il **Checksum**, come si nota nella figura a lato è definito da quattro cifre o lettere; questo parametro è da abbinare alla direttiva config, in questo modo si potrà evitare di dover impostare ogni volta l'oscillatore e i fuses. Una volta impostati tutti i parametri non rimane che cliccare con il mouse sul pulsante con il segno del **fulmine**.

Dopo aver programmato il PIC si consiglia di effettuare la verifica.





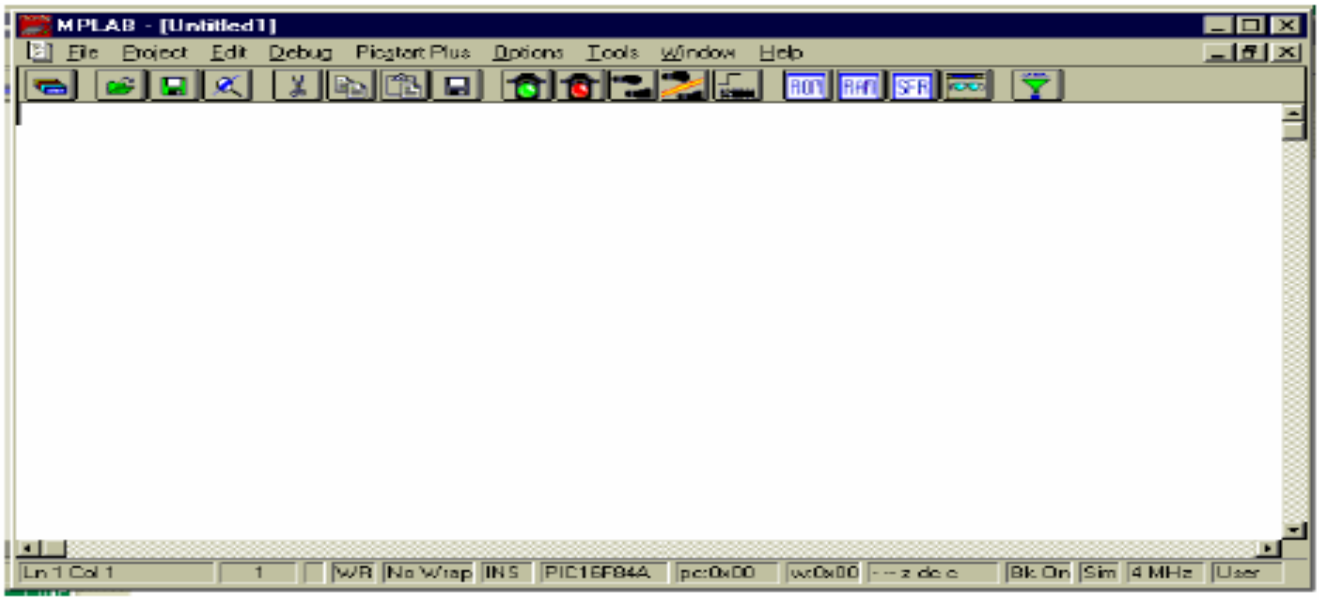
NOTA BENE:

Quando si programmerà il PIC 16F876 la configurazione dei **fuses** per default, è quella dell'immagine riportata a sinistra. **Per una corretta programmazione bisogna deselezionare la sigla LVP**

7. L'AMBIENTE INTEGRATO MPLAB

7.1. Introduzione

In questa unità saranno illustrati i metodi per editare ed assemblare un programma e creare il file eseguibile, lavorando con l'ambiente integrato di MPLAB della MICROCHIP TECHNOLOGY. Verranno brevemente illustrati anche i metodi per eseguire la simulazione del programma.
Una volta programmato il dispositivo, il programma realizzato, potrà essere messo in esecuzione nella stesso sistema di sviluppo MC-16 . Avviato il programma MPLAB l'ambiente che si presenta all'operatore è rappresentato nella figura riportata di seguito. In base alla versione del prodotto installato potranno aversi alcune differenze rispetto a quanto raffigurato.



7.2 Creazione di un progetto

Per procedere all'editazione di un programma bisogna innanzitutto creare un nuovo progetto usando il comando *New Project* del menu a tendina che si apre facendo clic su *Project*.

Menu Project

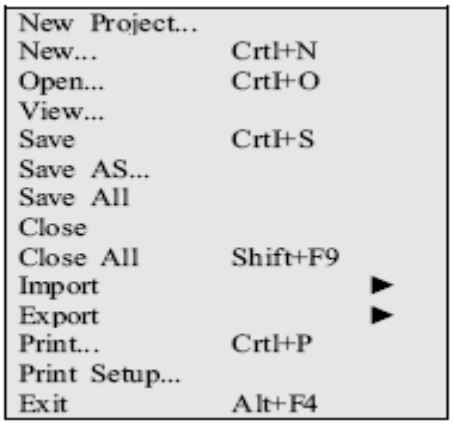
New Project...	
Open Project...	Crtl+F2
Close Project	
Save Project	
Edit Project	Crtl+F3
Make	F10
Build All	Crtl+F10
Build Node	Alt+F1O
Install Language Tool...	

Nella finestra ***New Project*** che si apre si sceglie un nome per il progetto (per esempio *led.prj*) inserendo nel riquadro *File Name* e si sceglie poi l'unità e la cartella su cui salvarlo dando infine OK. Nella finestra di *Edit Project* che si apre premere il pulsante *Change* e nella nuova finestra scegliere il

microcontrollore 16F84 ed il modo simulazione come illustrato nella figura riportata qui di seguito e premere poi il pulsante **OK**.
Ritornati nella finestra *Edit Project* dare **OK**.

Bisogna ora scrivere il *file sorgente* rispettando le regole dell'assemblatore. Si utilizzi il comando *File* e poi nel menu a tendina il comando *New*. Si apre la finestra dell'editore su cui va scritto il programma. Se la finestra è di formato ridotto allargarla usando il quadratino ☐ in alto a destra.

Menu File



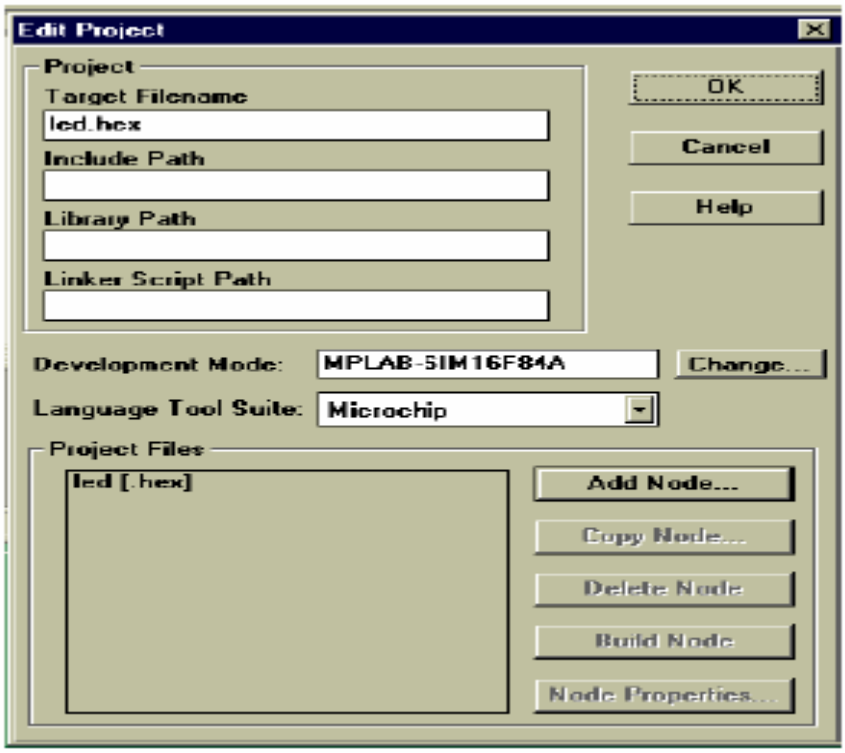
Si editi, per esempio il seguente programma, rispettando gli incolonnamenti (si può usare a tale proposito il tasto TAB della tastiera per un più facile allineamento del testo).

Per comodità il listato del programma è qui riportato:

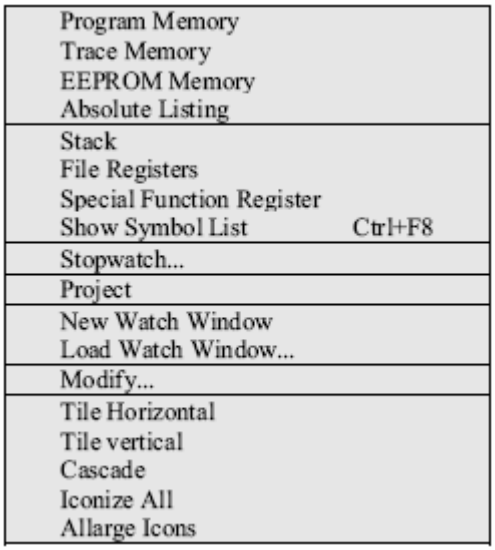
```
        portA   EQU     5           ;indirizzo porta A
portB   EQU     6           ;indirizzo porta B
        ORG     0
inizio: MOVLW   0FFh        ;dato di configurazione della porta B
        TRIS    port_B      ;pin porta B tutti d'ingresso
        MOVLW   00h         ;dato di configurazione della porta A
        TRIS    port_A      ;pin porta A tutti d'uscita
ancora: MOVF    port_B, W   ;legge porta B - suo contenuto in accumulatore
        MOVWF   port_A      ;pone contenuto dell'accumulatore sulla porta A
        GOTO    ancora
        END
```

Si effettui il salvataggio del programma con *File => Save As.....* attribuendo ad esso il nome *led.asm*.
Dal menu *Project* si faccia clic su *edit Project*, si riapre di nuovo la finestra di Edit Project mostrata in figura. Fare clic su *led [.hex]* nel riquadro ***Project File*** e poi premere il pulsante *Node Properties*. Nella

finestra che i apre accettarsi che nel riquadro *Language Tool* ci sia scritto MPASM e dare **OK**. Si torna alla finestra **Edit Project**. Fare clic sul pulsante *Add Node* e selezionare nella finestra che si apre il file *led.asm*. Esso verrà aggiunto nella finestra *Project Files*. Chiudere con OK.
Si può ora assemblare il file sorgente usando nel menu *Project => Make Project*. Inizia il procedimento di assemblaggio e se non ci sono errori nel *file* sorgente viene visualizzata la dicitura: *Buil Completed successfully.*
In caso di errori questi vanno corretti prima di procedere ad un nuovo assemblaggio.



Dal menù di Mplab cliccare su Window per aprire la seguente videata:



Facendo clic su *Absolute Listing* si visualizza il file eseguibile in formato listabile contenente indirizzi di memoria, codice eseguibile numeri di linea:

```
                 00001                    LED1.ASM
                 00002                    LIST      P=16C84,      F=INHX8M
00000005         00003                    port_A    EQU 5         ;indirizzo porta A
00000006         00004                    port_B    EQU 6         ;indirizzo porta B
0000             00005                    ORG       0
0000 30FF        00006    inizio:         MOVLW     0FFh          ;dato di configurazione della porta B
Warning[224]: Use of this instruction is not recommended.
0001 0066        00007                    TRIS      port_B        ;pin porta B tutti d'ingresso
0002 3000        00008                    MOVLW     00h           ;dato di configurazione della porta A
Warning[224]: Use of this instruction is not recommended.
0003 0065        00009                    TRIS      port_A        ;pin porta A tutti d'uscita
0004 0806        00010    ancora:         MOVF      port_B, 0     ;legge porta B - suo contenuto in accumulatore
                                                                  W
0005 0085        00011                    MOVWF     port_A        ;pone contenuto dell'accumulatore sulla porta A
0006 2804        00012                    GOTO      ancora
                 00013                    END

SYMBOL TABLE
LABEL                             VALUE
_16C84                            00000001
ancora                            00000004
inizio                            00000000
port_A                            00000005
port_B                            00000006

MEMORY USAGE MAP ('X' = Used, '_'= Unused)
0000: XXXXXXX ———— ———————— ———————————— ———————————— ————————————
All other memory blocks unused.
Program Memory Words Used:                7
Program Memory Words Free:                1017
Errors:          0
Warnings:        2 reported, 0 suppressed
Messages:        0 reported, 0 suppressed
```

Con *Program Memory* vengono visualizzate gli indirizzi di memoria e il codice eseguibile con accanto il file sorgente; è possibile osservare dal listato come le etichette siano state convertite in indirizzi di memoria.

```
0000 30FF    inizio    movlw    0xFF
0001 0066              tris     0x6
0002 3000              movlw    0x0
0003 0065              tris     0x5
0004 0806    ancora    movf     0x6,W
0005 0085              movwf    0x5
0006 2804              goto     ancora
0007 3FFF              addlw    0xff
```

7.3 Simulazione di un Programma

Il simulatore di MPLAB permette di eseguire anche una simulazione di tipo interattivo. Infatti possono essere assegnati ad alcuni pulsanti (disegnati sul video) il collegamento con le linee di input del microcontrollore. Alla pressione dei pulsanti corrispondono un cambiamento di stato delle linee in base alle specifiche impostate. E' inoltre possibile seguire sul video, in apposite finestre, l'evoluzione del programma e il cambiamento che subiscono variabili e *File Register*. Per eseguire la simulazione, dopo aver realizzato il progetto e creato il file eseguibile, si deve aprire il menu Debug.

Menu Debug

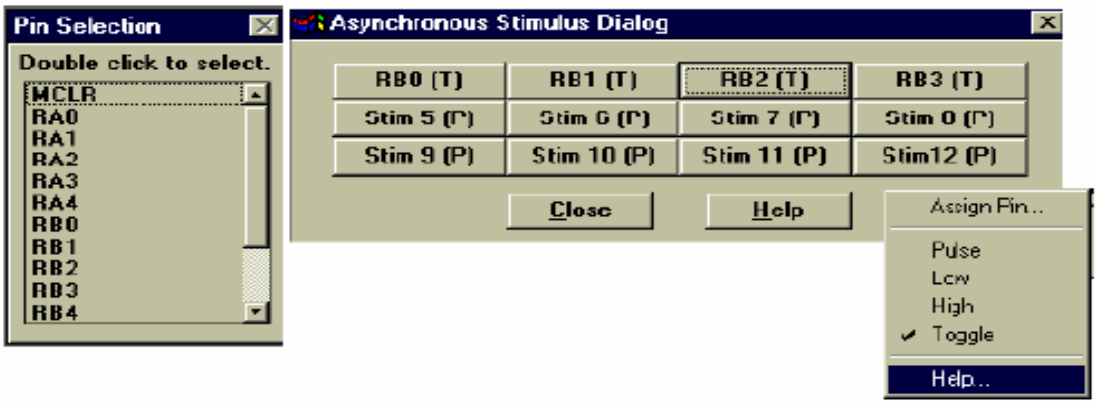
Run ▶	
Execute ▶	
Simulator Stimulus ▶	
Center Debug Locatio	
Bresk Settings...	F2
Trace Settings...	
Trigger In/Out Settings...	
Clear All Points...	
Complex Trigger Settings...	
Enable code Coverage	
Clear Program Memory...	Ctrl+Shift+F2
System Reset	Ctrl+Shift+F3
Power On Reset	Ctrl+Shift+F5

7.4 Creazione dello Stimulus

Poiché il programma da simulare (ci si riferisce a LED1.ASM), prevede la lettura delle linee RB0, RB1, RB2 e RB3 (alle quali sono collegati 4 interruttori), per la simulazione si collegheranno 4 pulsanti a queste linee.
Dal menu *Debug* scegliere *Simular Stimulus => Asynchronous Stimulus*; si apre la finestra con i pulsanti (Asynchronous Stimulus Dialog), visualizzata nella figura riportata di seguito. Con il tasto destro del mouse fare clic sul primo pulsante a sinistra. Si apre il menu *Assign Pin*. Facendo clic su *Assign Pin* si apre la finestra *Pin Selection*. Si faccia doppio clic su *Assign Selection*. Si faccia doppio clic su RB0; la finestra si chiude e sul pulsante in alto a sinistra compare scritto RB0 [P]. Si proceda in questo modo per i quattro pulsanti della prima riga assegnando loro le linee RB1, RB2 e RB3.
Si deve ora scegliere per ogni pulsante assegnato, il tipo di segnale che viene generato quando esso è premuto. Sono disponibili 4 segnali diversi:

1. *Pulse* (impulso) porta il livello del pin ad un livello opposto a quello in cui si trova e poi lo riporta automaticamente al livello originario (basso => alto => basso o alto => basso => alto)
2. *Low* (Basso) porta il livello del pin a livello basso e lo lascia a tale livello
3. *High* (Alto) porta il livello del pin a livello alto e lo lascia a tale livello
4. *Toggle* inverte il livello ogni volta che viene premuto il pulsante

Nella simulazione in oggetto deve essere selezionato *Toggle* per tutti i pulsanti. Si deve quindi selezionare un pulsante per volta con il tasto destro



e scegliere per ognuno di essi *Toggle*. Nella figura riportato qui sopra *Asynchronous Stimulus* è rappresentata la situazione finale dei quattro pulsanti.

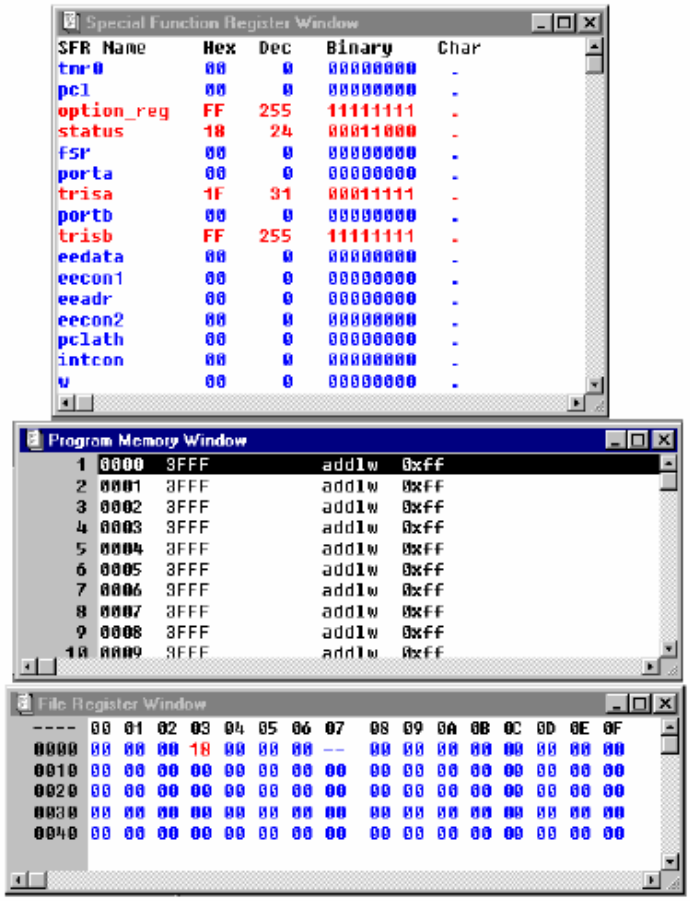
7.5 Apertura delle finestre di visualizzazione

Per poter seguire le evoluzioni del programma mentre è in esecuzione, è possibile aprire con il menu *Windows* una serie di finestre per visualizzare i *files register di tipo generale*, i *files register di tipo speciale*, lo *stack* e la memoria di programma.

Nella figura riportata in seguito sono visualizzate le finestre:

- *Special Function Register* (tra l'altro sono visibili in questa finestra l'accumulatore (W), il TMR0, i registri OPTION, STATUS TRISA e TRISB, PORTA e PORTB;
- *File Register*
- *Program memory* contenente le locazioni di memoria, i codici macchina e i mnemonici.

Aperte le finestre necessarie conviene ridimensionare e sistemare in modo che non ci sia sovrapposizione. Si tenga presente che deve essere visibile anche la finestra con i pulsanti.



7.6 Avvio della Simulazione

Dal menu *Debug => Run* è possibile scegliere tra diverse modalità operative per l'esecuzione del programma.

Menu Debug Run

Run	F9
Reset	F6
Halt	F5
Halt Trace	Shift+F5
Animate	Ctrl+F9
Step	F7
Step Over	F8
Update All Register	
Change Program Counter	

- *Run:* la simulazione del programma viene avviata e continua fin quando non si dà Halt o non viene incontrato un punto di arresto precedentemente impostato. Solo quando l'esecuzione del programma si ferma, è possibile osservare le variazioni avvenute nei registri e nella memoria.
- *Step :* viene eseguito un passo alla volta (con la pressione del tasto funzione F7).
- *Step Over :* si esegue ancora un passo alla volta (tasto funzione F7) senza però entrare all'interno di *routine* o *loop*.
- *Animate :* Il programma viene eseguito lentamente e si possono osservare immediatamente i cambiamenti che avvengono nei registri e nella memoria.

Si scelga per la simulazione del programma assemblato precedentemente la modalità *Animate*. Avviato il programma attraverso il menu (*Debug => Run => Animate*) o con Ctrl + F9, si osservi il cambiamento che avviene nei registri PORTA e PORTB quando si premono i pulsanti precedentemente impostati. Ogni volta che si preme (con il mouse) uno dei pulsanti cambia lo stato della linea RBn corrispondente (interruttore) e quindi quello della linea Rn (LED). Per arrestare la simulazione entrare nel menu *Debug => Run* e fare clic su *Halt*.

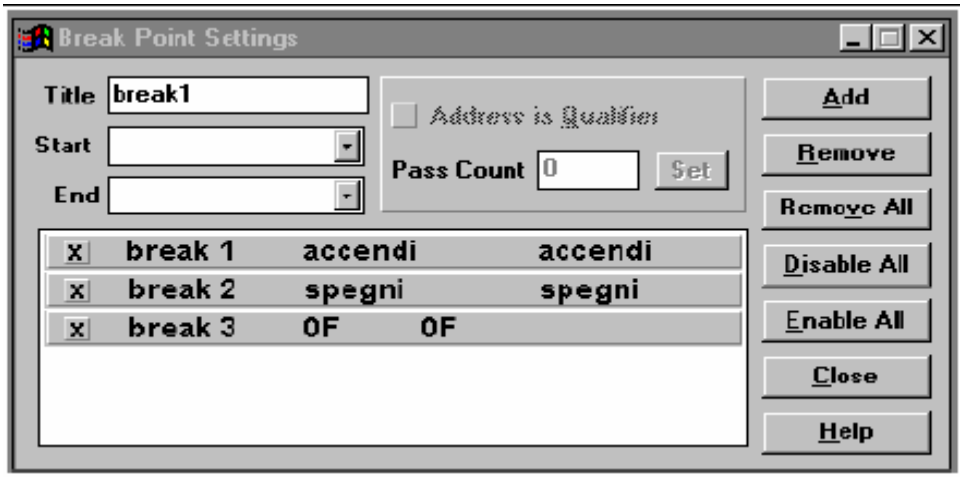
7.7 Debug di un Programma

In genere per eseguire il *debug* di un programma si utilizzano le fnzioni del menu *Debug => Run Step* e *Step Over* o anche direttamente la funzione *Run*. Quando si esegue il *Debug* usando direttamente la funzione *Run* è necessario inserire nel programma dei punti di accesso (*break point*) in modo che l'esecuzione si fermi nei punti stabili e sia possibile visualizzare lo stato dei registri.

7.8 Inserimento di BREAK POINT

L'inserimento dei *break point* è effettuato con il menu *Debug => Break Settings*. Si apre la finestra visualizzata nella figura riportata di seguito.
I *break point* possono essere inseriti sia utilizzando le eventuali *label* presenti nel programma che direttamente gli indirizzi di memoria espressi in esadecimale (senza h e se iniziano con una lettera devono essere preseduti da 0). Per inserire punti di arresto contrassegnati da un *label* fare clic sul triangolino (▼) della casella Start; si apre un menu a tendina contenente tutte le *label* e il programma. Fare clic poi sul segno di spunta (√) . Il punto di arresto verrà visualizzato nella finestra sottostante. L'indirizzo si scrive nella casella *Start* e poi si fa clic sul seno di spunta. Nella figura sottostante sono stati inseriti 3 break point: due relativi alle etichette *accendi* e *spegni* ed il terzo all'indirizzo *0Fh*.

Qualora si desideri inserire una serie di *break point* consecutivi nella casella *End* può essere messo l'indirizzo o la *label* finale del gruppo contiguo di punti di arresto. In questo modo saranno predisposti tutti i *break point* inclusi tra l'etichetta (o l'indirizzo) di partenza e l'etichetta (o l'indirizzo finale).



Esempio di debug di un programma

Si voglia eseguire il debug del programma (GEN1.ASM) per verificarne il corretto funzionamento utilizzando la funzione *Run* ed inserendo dei *break point*.

```
;GEN1.ASM
        LIST    P=16C84, F=INHX8M
port_A  EQU     5
conta1  EQU     0Ch
conta2  EQU     0Db
inizio: MOVLW   00h
        TRIS    port_A          ;pin porta A tutti      d'uscita
        BCF     port_A, 0       ;spegne LED
ancora: CALL    ritardo
accendi: BSF    port_A, 0       ;porta alto bit 0 porta A - accende il LED
        CALL    ritardo
spegni: BCF     port_A, 0       ;porta basso bit 0 porta A - spegne LED
        GOTO    ancora
ritardo: MOVLW  0Fh
        MOVWF   conta2
loop2:  MOVLW   0Fh
        MOVWF   conta1
loop1:  DECFSZ  conta1
        GOTO    loop1
        DECFSZ  conta2
        GOTO    loop2
        RETURN
        END
```

La procedura da seguire è qui sintetizzata:

1. aprire un nuovo progetto (Project => New) assegnargli il nome gen1.pjt. Dare OK nella finestra *New Project*.
2. Nella finestra Edit Project che si apre scegliere in Development Mode: MPLAB-SIM 16F84 e dare OK.
3. Con *File => New* aprire la finestra per editare il sorgente.
4. Salvare il sorgente con *File => Save As* ..... attribuendogli il nome gen1.asm
5. Con *Project => Edit Project...* riaprire la finestra *Edit Project*. Fare clic su gen1[.hex] nel riquadro *Project Files*. Fare clic su *Node Properties*. Dare OK nella finestra che si apre ritornando quella di *Edit Project*.
6. Fare clic sul pulsante *Add Node*. Scegliere nella finestra che si apre il programma sorgente salvato (gen1.asm) e dare OK.
7. Con Project =>Save Project salvare il progetto appena creato.
8. Con Project => Make Project assemblare il file sorgente. Se ci sono errori correggerlo e riassemblarlo.
9. Con Windows => Special Functio Register aprire la finestra dei registri speciali.
10. Con Debug => Break Setting inserire due punti di arresto in corrispondenza delle etichette accendi e spegni.
11. Avviare l'esecuzione del programma con *Debug => Run => Run* (o con F9). Il programma si avvia e si arresta in corrispondenza del primo *break point* (accendi). Osservare il *file register* PORTA: il bit corrispondente alla linea RA0 (quello meno significativo) deve essere passato da 0 ad 1.
12. Riavviare l'esecuzione del programma con *Debug => Run => Run* (o con F9). Il programma si riavvia e si arresta in corrispondenza del secondo *break point* (spegni). Osservare il *file register* PORTA: il bit corrispondente alla linea RA0 (quello meno significativo) deve essere passato da 1 ad 0.
13. Se si continua a riavviare il programma si hanno variazioni di livello tra 0 e 1 della linea RA0. Il programma funziona correttamente.
14. Arrestare l'esecuzione con *Debug = Run => Halt* (o con F5).
15. Rimuovere tutti i *break point* con *Debug => Break Settings => Remove All*.

8. Il Microcontrollore PIC 16F876

8.1 Introduzione

Il 16F876 è un microcontrollore con memoria di programma di tipo flash che risulta particolarmente indicato per la messa a punto e la prova di programmi che fanno uso di dispositivi periferici non implementati nel PIC 16F84. Sono integrati infatti nel chip i seguenti moduli periferici:

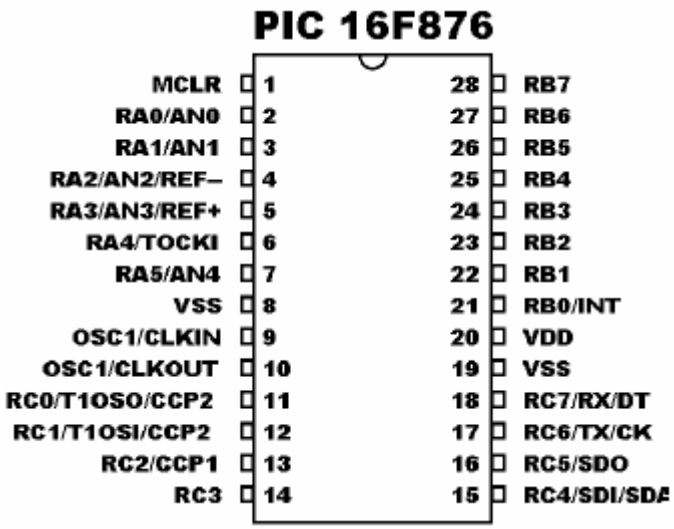
- 1 convertitore Analogico-Digitale a 10bit con 5 canali d'ingresso
- 3 timer
- 2 moduli Capture, Compare, PWM
- porte seriali sincrone ed asincrone.

Rimangono valide per il 16F876 le regole di programmazione ed i codici già esposti precedentemente per il PIC 16F84. Si presuppone che il lettore abbia già acquisito una buona conoscenza delle caratteristiche tecniche e dei principi di programmazione del microcontrollore 16F84 e dell'ambiente MPLAB.

8.2 Caratteristiche di base del PIC16F876

Il PIC 16F876 è un dispositivo a 28 pin che lavora con frequenza massima di clock pari a 20 MHz e dispone di un set di 35 istruzioni .

- 8 Kword (di 14 bit) di memoria FLASH di programma
- 256 byte di memoria EEPROM per i dati
- 368 byte di RAM (per i dati)
- 3 porte I/O : Port A con cinque linee di I/O, Port B e PortC con 8 linee di I/O



Nella tabella 9.1 riportata di seguito sono descritti i principali segnali del dispositivo. Si tenga presente che molti dei pin svolgono più di una funzione.

Tab. 9.1

PIN	FUNZIONE			DESCRIZIONE
1	MCLR	Vpp		*Reset* - ingresso per la tensione di programmazione
2	RA0	AN0		*I/O* digitale Port A - ingresso canale 0 analogico
3	RA1	AN1		*I/O* digitale Port A - ingresso canale 1 analogico
4	RA2	AN2	V_{REF}-	*I/O* digitale Port A - ingresso canale 2 analogico
5	RA3	AN3	V_{REF}+	*I/O* digitale Port A - ingresso canale 3 analogico
6	RA4	TOCKI		*I/O* digitale Port A - ingresso *clock TMR0*
7	RA5	AN4	SS	*I/O* digitale Port A - ingresso canale 4 analogico
8	Vss			GND
9	OSC1	CLKIN		Ingresso oscillatore a cristallo - ingresso *clock* esterno
10	OSC2	CLKOUT		Uscita oscillatore a cristallo - uscita *clock* (1/4 frequenza di OSC1)
11	RC0	T1CKI	T1OSO	*I/O* digitale Port C - ingresso *clock* TMR1 - *out* oscillatore TMR1
12	RC1	CCP2	T1OSI	*I/O* digitale Port C - *input Capture* 2 - *input* oscillatore TMR1
13	RC2	CCP1		*I/O* digitale Port C - *input Capture* 1
14	RC3	SCK	SCL	*I/O* digitale Port C
15	RC4	SDI	SDA	*I/O* digitale Port C
16	RC5	SDO		*I/O* digitale Port C
17	RC6	TX	CK	*I/O* digitale Port C - Tx per trasmiss. Asincrona – *clock* sincrona
18	RC7	RX	DT	*I/O* digitale Port C - Rx per ricezione Asincrona - *dati* sincrona
19	V_{SS}			GND
20	V_{DD}			Positivo alimentazione
21	RB0	INT		*I/O* digitale Port B - ingresso *interrupt*
22	RB1			*I/O* digitale Port B
23	RB2			*I/O* digitale Port B
24	RB3	PGM		*I/O* digitale Port B - ingresso per la programmazione
25	RB4			*I/O* digitale Port B
26	RB5			*I/O* digitale Port B
27	RB6	PGC		*I/O* digitale Port B - *clock* per la programmazione seriale
28	RB7	PGD		*I/O* digitale Port B - *dati* per la programmazione seriale

Nella tabella 9.2 viene riportata la mappa di memoria dei *file register* con i relativi indirizzi, suddivisi per banchi. Sono presenti quattro banchi di memoria, selezionabili con i bit RP1 (bit6) e RP0 (bit5) del registro STATUS. Sono mostrate inoltre le aree di memoria RAM per dati (registri di uso generale).

Tab. 9.2

BANK 0		BANK 1		BANK 2		BANK 3	
00h	Indirect addr.	80h	Indirect addr.	100h	Indirect addr.	180h	Indirect addr.
01h	TMR0	81h	OPTION REG	101h	TMR0	181h	OPTION REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h		88h		108h		188h	
09h		89h		109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	
10h	T1CON	90h		110h	REGISTRI USO GENERALE (16 BYTE)	190h	REGISTRI USO GENERALE (16 BYTE)
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPRIL	95h		115h		195h	
16h	CCPRIH	96h		116h		196h	
17h	CCP1CON	97h		117h		197h	
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	90h		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	REGISTRI USO GENERALE (96 BYTE)	A0h	REGISTRI USO GENERALE (80 BYTE)	120h	REGISTRI USO GENERALE (80 BYTE)	1A0h	REGISTRI USO GENERALE (80 BYTE)
. . .		. . .		. . .		. . .	
		EFh		16Fh		16Fh	
7Fh							

Nella tabella 9.3 vengono rappresentate le configurazioni dei bit di alcuni dei principali registri. Vengono anche rappresentate, in maniera dettagliata, le funzioni svolte dai bit dei registri STATUS, OPTION_REG e INTCON.

Tab. 9.3

BANK 0									
Nome	Ind.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bitl	Bit0
INDF	00h	utilizza il contenuto di FSR per indirizzare la m emoria dati							
TMR0	0lh	registro del timer 0							
STATUS	03h	IRP	RP1	RP0	TO	PD	Z	DC	C
FSR	04h	puntatore per l'indirizzamento indiretto della mem oria							
PORTA	05h			RA5	RA4	RA3	RA2	RA1	RA0
PORTB	06h	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
PORTC	07h	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
INTCON	0Bh	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIR1	0Ch		ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIR2	0Dh				EEIF	BCLIF			CCP2IF
TMR1L	0Eh	byte basso del registro del TIMER 1							
TMR1H	0Fh	byte alto del registro del TIMER 1							
T1CON	10h			T1 CKPS1	T1 CKPS0	T1 OSCEN	T1 SYNC	TMR1 CS	TMR1 ON
TMR2	11h	registro del TIMER 2							
T2CON	12h		TOUT PS3	TOUT PS2	TOUT PS1	TOUT PS0	TMR2 ON	T2 CKPS1	T2 CKPS0
CCPR1L	15h	byte basso del registro del modulo 1 CCP (*Capture/Compare/PWM*)							
CCPR1H	16h	byte alto del registro del modulo 1 CCP *(Capture/Compare/PWM)*							
CCP1CON	17h			CCP 1X	CCP 1Y	CCP 1M3	CCP 1M2	CCP 1M1	CCP 1M0
CCPR2L	1Bh	byte basso del registro del modulo 2 CCP *(Capture/Compare/PWM)*							
CCPR2H	1Ch	byte alto del registro del modulo 2 CCP *(Capture/Compare/PWM)*							
CCP2CON	1Dh			CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0
ADRESH	1Eh	byte alto del risultato del convertitore A/D							
ADCON0	1Fh	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO!/ DONE		ADON
BANK 1									
OPTION	81h	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
TRISA	85h			configurazione INGRESSO/USCITA della PORTA A					
TRISB	86h	configurazione INGRESSO/USCITA della PORTA B							
TRISC	87h	configurazione INGRESSO/USCITA della PORTA C							
PIE1	8Ch		ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMT1IE
PIE2	8Dh				EEIE	BCLIE			CCP2IE
PCON	8Eh							POR	BOR
PR2	92h	registro del periodo del TIMER 2							
ADRESL		byte basso del risultato del con vertitore A/D							
ADCON1		ADFM				PCFG3	PCFG2	PCFG1	PCFG0
BANK 2									
EEDATA		registro dei dati della EEPROM							
EEADR		registro degli indirizzi della EEPROM							
EEDATH				parte alta registro dati della EEPROM					
EEADRH					parte alta registro indirizzi della EEPROM				

BANK 3									
EECON1	l8Ch	EEPGD				WRERR	WREN	WR	RD

STATUS							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

bit 7	IRP	Utilizzato per l'indirizzamento indiretto: **0** banchi 0 e 1 - **1** banchi 2 e 3.
bit 6	RP1	Selezionano i banchi:
bit5	RP0	**0 0:** banco **0**; **0 1**: banco **1**; **1 0:** banco **2**; **1 1**: banco **3**
bit 4	TO	È posto a **zero** quando il *Watchdog Timer* va in *Time-out*
bit 3	PD	è posto a **zero** quando viene eseguita un'istruzione di SLEEP
bit 2	Z	*flag* di zero posto a **uno** se risultato di un'operazione logica o aritmetica è **zero**
bit 1	DC	*flag* di *digit carry* posto a **uno** se c'è **riporto** dal semi-byte meno significativo
bit 0	C	*flag* di *carry* posto a **uno** quando c'è **riporto** dal bit più significativo

OPTION_REG							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPU	**INTEDG**	**TOIE**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**

bit 7	**RBPU**	disabilita o abilita i resistori interni di *pull-up* sulla PORTA B	
		0	disabilita resistori interni
		1	abilita resistori interni
bit 6	**INTEDG**	seleziona il fronte del segnale di *interrupt* su *RBO/INT*	
		0	fronte di discesa
		1	fronte di salita
bit 5	**TOCS**	seleziona la sorgente per il segnale di *clock* del *timer (TMR0)*.	
		0	*clock* interno
		1	*clock* esterno su *RA4/TOCKI*
bit4	**TOSE**	seleziona il fronte del segnale per il *clock* del *timer* su *RA4/TOCKI*	
		0	fronte di discesa
		1	fronte di salita
bit 3	**PSA**	assegna il *prescaler* al *timer TMR0* o al *WATCH-DOG timer WDT*	
		0	*TMR0*
		1	*WDT*
bit 2	**PS2**	Selezionano il fattore di divisione per il *prescaler* per il TMRO e per il *WDT:* **000** 1:2 - **001** 1:4 - **010** 1:8 – **011** 1:16 – **100** 1:32 – **101** 1:64 – **110** 1:128 – **111** 1:256 **000** 1:1 - **001** 1:2 – **010** 1:4 – **011** 1:8 – **100** 1:16 – **101** 1:32 – **110** 1:64 – **111** 1:128	
bit 1	**PS1**		
bit 0	**PS0**		

INTCON								
Bit 7		Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE		**PEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**
bit 7	GIE	abilitazione generale di tutti gli *interrupt*						
		0	tutti gli *interrupt* sono disabilitati					
		1	tutti gli *interrupt* sono abilitati					
bit 6	PEIE	abilitazione dell interrupt delle periferiche						
		0	disabilitato					
		1	abilitato					
bit 5	TOIE	abilitazione dell'interrupt di superamento di capacità di TMR0						
		0	disabilitato					
		1	abilitato					
bit 4	INTE	abilita l'interrupt per la linea RBO/INT						
		0	disabilitato					
		1	abilitato					
bit 3	RBIE	abilita l'interrupt per il cambio di livello sulle linee RB7-RB4						
		0	disabilitato					
		1	abilitato					
bit 2	TOIF	segnalazione di superamento di capacità di TMR0						
		0	non c'è stato *overflow*					
		1	c'è stato *overflow*					
bit 1	INTF	segnalazione di richiesta d'interrupt sulla linea RB0/INT						
		0	non c'è stato superamento di capacità di TMR0					
		1	c'è stato superamento di capacità di TMR0					
bit 0	RBIF	segnalazione di avvenuto cambio di livello sulle linee RB7-RB4						
		0	non è cambiato nessun livello					
		1	sono cambiati uno o più livelli					

9. I Moduli Periferici del PIC 16F876

9.1 Introduzione

Nella presente unità verranno presi in esame i moduli periferici del microcontrollore PIC 16F876. In particolare verrà esaminato il funzionamento del Convertitore A/D, dei tre TIMER e dei due moduli CCP (Capture/Compare/PWM). Saranno forniti anche semplici esempi di programmazione.

9.2 Il convertitore A/D del PIC 16F876

Il PIC 16F876 ha un modulo convertitore Analogico/digitale a 10 bit con 5 canali d'ingresso, ciascuno i essi selezionabile via software. Il convertitore, ad approssimazioni successive, è dotato internamente di *track-hold*.
I registri specifici associati con il modulo convertitore sono ADCON0 (in pagina 0 all'indirizzo 1Fh) e ADCON1 (in pagina 1 all'indirizzo 9Fh). Il risultato della conversione (10 bit) viene posto nei registri a 8 bit ADRESH e ADREL. Prima di avviare la conversione si debbono impostare tutti i parametri necessari dei registri ADCON0 e ADCON1 nel modo seguente:

1. Con il registro ADCON1 (vedi tabella precedente) si seleziona l'abbinamento dei canali analogici con gli ingressi digitali (RA0 ÷ RA5), la sorgente della tensione di riferimento e la modalità di salvataggio del risultato della conversazione nei registri ADRESH e ADRESL. Nell'esempio illustrato nella tabella 9.1 il canale AN0 è collegato con del registro ADRESL in cui viene posto il risultato, sono posti a zero. Si ricordi che il risultato della conversione è a 10 bit, mentre i due registri contengono complessivamente 16 bit.

Tab.9.1

ADCON1								
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	
0	0	0	0	1	1	1	0	0Eh
i bit meno significativi di ADRESL sono come 0				RA0 ⇐ AN0 $V_{REF} = V_{DD}$				

Il codice per eseguire l'inizializzazione del registro ADCON1 è il seguente:

```
movlw   0Eh      ; in W codice selezione VREF e canale AN0 - allineam. sinistra
movwf   ADCON1   ; sposta il codice nel registro ADCON1 (banco 1)
```

2. Con il registro ADCON0 (vedere tabella precedente) si seleziona il clock di conversione (bit 7 e bit 6), il canale analogico per la conversione (bit 5 ÷ bit 3) e si pone in ON il convertitore A/D (bit 0). Nell'esempio riportato nella tabella 9.2, per il clock si usa *fosc/8*, si seleziona il canale 0 (AN0), e si pone a uno il bit 0, per abilitare il convertitore.

Tab. 9.2

ADCON0								
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE		ADON	
0	1	0	0	0	0	0	1	41h
Fosc/8		Canale AN0					A/D ON	

Il codice per eseguire l'inizializzazione del registro ADCON0 è il seguente:

```
movlw    41h        ;in W codice per frequenza, canale AN0 e A/D ON
movwf    ADCON0     ;sposta il codice nel registro ADCON0 (banco 0).
```

3. si aspetta che trascorra il tempo di acquisizione (tempo impiegato dal condensatore di *holding* per caricarsi al valore del segnale d'ingresso). Come valore risulta circa 20 µs.

4. si avvia la conversione ponendo ad uno il bit 2 (GO/DONE) di ADCON0.

```
bsf   ADCON0, 2     ; pone ad uno il bit 2 di ADCONO e avvia la conve rsione
```

5. si controlla il termine della conversione verificando il bit 2 di ADCON0 (GO/DONE); quando diviene zero la conversione è terminata.

```
loop:  btfsc    ADCON0, 2    ;test del bit 2 di ADCON se O salta l'istruz. su ccessiva
       goto     loop         ; se bit 2 è 1 esegui un nuovo controllo
```

6. si legge il dato convertito nei registri ADRESSH (gli 8 bit più significativi) e ADRESL (i 2 bit meno significativi).

Con l'allineamento a sinistra il risultato della conversione è così rappresentato:

ADRESH										ADRESL					
□	□	□	□	□	□	□	□	□	□	0	0	0	0	0	0
Risultato della conversione (10 bit)															

```
movf    adresh, 1    ; carica in accumulatore la parte alta del risultato conve rsione
movwf   20h          ; salva in memoria la parte alta del risultato
movf    adresl, 1    ; carica in accumulatore la parte bassa del risultato conve rsione
movwf   21h          ; salva in memoria la parte bassa del risultato
```

Il programma completo deve tenere conto anche del cambio di banco per accedere al registro ADCON1, deve poi selezionare nel registro TRISA, come ingresso, la linea RA0 utilizzata per il canale AN0 ed infine deve tenere conto del ritardo per il **Tempo di Acquisizione**.

```
        org 0
        movlw    01h          ; carica in accumulatore la parola per configurare RA0 come i nput
        tris     05h          ; la pone nel registro TRISB
        movlw    8Eh           ; carica in W codice per selezione Vref e canale AN0
        movwf    ADCQN1       ; sposta il codice nel registro ADCON1 (banco 1)
        bcf      03h,05h      ; seleziona banco 0
        movlw    41h          ; carica in W codice per frequenza canale, A/D ON
        movwf    ADCON0       ; sposta il codice nel registro ADCON0 (banco 0)
        call     ritardo      ; chiama la routine di ritardo (circa 20 µs)
        bsf      ADCON0, 2    ; pone ad uno il bit 2 di ADCON0 e avvia la conversione
loop    btfsc    ADCON0, 2    ; test bit 2 di ADCON se 0 (fine conversione) salta l'istruz. su ccessiva
        goto     loop         ; se bit 2 è 1 esegui un nuovo controllo
        movf     adresh, 1    ; carica in accumulatore la parte alta del risultato conversione
        movwf    20h          ; salva in memoria la parte alta del risultato
        movf     adresl, 1    ; carica in accumulatore la parte bassa del risultato co nversione
        movwf    21h          ; salva in memoria la parte bassa del risultato
```

9.2.1 I Registri ADCON1 e ADCON0

Nelle tabelle 9.3 e 9.4 sono riportate le modalità per impostare i vari modi di funzionamento del modulo convertitore.

Tab. 9.3

ADCON1							
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
ADFM				PCFG3	PCFG2	PCFG1	PCFG0

bit 7	
ADFM	
1	I 6 bit più significativi del registro ADRESH sono posti a 0.

	ADRESH								ADRESL							
	0	0	0	0	0	0	□	□	□	□	□	□	□	□	□	□
							Risultato della conversione (10 bit)									

0	I 6 bit più significativi del registro ADRESL sono posti a 0.

	ADRESH								ADRESL							
	□	□	□	□	□	□	□	□	□	□	0	0	0	0	0	0
	Risultato della conversione (10 bit)															

bit 3	bit 2	bit 1	bit 0	RA5	RA3	RA2	RA1	RA0	$V_{REF}+$	$V_{REF}-$
PCFG3	PCFG2	PCFG1	PCFG0	AN4	AN3	AN2	AN1	AN0		
0	0	0	0	AN	AN	AN	AN	AN	V_{DD}	V_{SS}
0	0	0	1	AN	AN	AN	$V_{REF}+$	AN	RA3	V_{SS}
0	1	0	0	DIG	AN	DIG	AN	AN	V_{DD}	V_{SS}
0	1	0	1	DIG	$V_{REF}+$	DIG	AN	AN	RA3	V_{SS}
0	1	1	X	DIG	DIG	DIG	DIG	DIG	V_{DD}	V_{SS}
0	0	0	0	AN	$V_{REF}+$	$V_{REF}-$	AN	AN	RA3	RA2
1	1	0	1	DIG	$V_{REF}+$	$V_{REF}-$	AN	AN	RA3	RA2
1	1	1	0	DIG	DIG	DIG	DIG	AN	V_{DD}	V_{SS}
1	1	1	1	DIG	$V_{REF}+$	$V_{REF}-$	DIG	AN	RA3	RA2

Nota : per i bit PCGF3 ÷ PCGF0 non sono state considerate alcune combinazioni perché danno la stessa configurazione di altri canali poiché il PIC 16F876 ha solamente cinque ingressi analogici.

ADCON0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE		ADON

ADCS1	ADCSO			Massima frequenza possibile per Fosc
0	0	Fosc/2	2 T_{OSC}	1.25 MHz
0	1	Fosc/8	8 T_{OSC}	5 MHz
1	0	Fosc/32	32 T_{OSC}	20 MHz
1	1	F_{RC}	T_{RC}	*clock* derivato da un oscillatore RC interno

CHS2	CHS1	CHS0	CANALE SELEZIONATO
0	0	0	AN0 (su RA0)
0	0	1	AN1 (su RA1)
0	1	0	AN2 (su RA2)
0	1	1	AN3 (su RA3)
1	0	0	AN4 (su RA5)

GO/DONE	
1	Conversione in atto. Si avvia la conversione quando il bit è posto ad 1
0	Conversione non in atto. Il bit è posto automaticamente a 0 al termine della conversione.

ADON	
1	Il modulo A/D è operativo
0	Il modulo A/D non è operativo

9.2.2 I Registri ADRESH e ADRESL

Come evidenziato nella tabella riportata di seguito in base al valore assegnato al bit 7 (ADFM) del registro ADON il dato memorizzato nei registi ADRESH e ADRESL dopo la conversione può essere allineato a destra (6 bit più significativi del registro ADRESH posti a 0) o a sinistra (6 bit meno significativi del registro ADRESL sono a 0).

ADFM - 1	I 6 bit più significativi del registro ADRESH posti a 0 (dato giustificato a destra).																
	ADRESH								ADRESL								
	0	0	0	0	0	0	□	□	□	□	□	□	□	□	□	□	
	Risultato della conversione (10 bit)																
ADFM - 0	I 6 bit meno significativi del registro ADRESL posti a 0 (dato giustificato a sinistra).																
	ADRESH								ADRESL								
	□	□	□	□	□	□	□	□	□	□	0	0	0	0	0	0	
	Risultato della conversione (10 bit)																

Con l'allineamento a destra si ottiene il risultato della conversione a 10 bit (tre cifre esadecimali da 000h a 3FFh). La prima cifra è contenuta nel *nibble* basso di ADRESH, la seconda nel *nibble* alto di ADRESL e la terza in quello basso di ADRESL come mostrato nella tabella 9.5. Il valore di N_{10} è stato calcolato con la relazione riportata di seguito e poi trasformato in esadecimale.

$$N_{10} = V_{IN} * 2^{10} / V_{REF} = V_{IN} * 1024 / 5.0$$

Tab. 9.5

DATO GIUSTIFICATO A DESTRA																	
V_{IN}	N (calcolato)	ADRESH (acquisito)								ADRESL (acquisito)							
1V	0CCh	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
								0		C				C			
2V	199h	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1
								1		9				9			
3V	265h	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	1
								2		6				5			

Con l'allineamento a sinistra la lettura del dato è meno immediata dovendosi suddividere in modo opportuno i bit contenuti nei registri ADRESH e ADRESL, come mostrato nella tabella 9.6.

Tab.9.6

DATO GIUSTIFICATO A SINISTRA																	
V_{IN}	N (calcolato)	ADRESH (acquisito)								ADRESL (acquisito)							
1V	0CCh	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
		0		C				C									
2V	199h	0	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0
		1		9				9									
3V	265h	1	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0
		2		6				5									

Con l'allineamento a sinistra il convertitore fornisce nel registro ADRESH, direttamente il valore del dato corrispondente ad una conversione, con convertitore ad 8 bit come mostrato nella tabella 9.7. In questo caso il valore di N_{10} è calcolato con la relazione :

$$N_{10} = V_{IN} * 2^{10} / V_{REF} = V_{IN} * 256 / 5.0$$

Tab. 9.7

DATO GIUSTIFICATO A SINISTRA																	
V_{IN}	N (calcolato)	ADRESH (acquisito)								ADRESL (acquisito)							
1V	33h	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
		3				3											
2V	66h	0	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0
		6				6											
3V	99h	1	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0
		9				9											

Come si vede in questo caso non è utilizzato il contenuto di ADRESL.

9.3 Tempo di acquisizione e di conversione

Per ottenere la conversione completa di una tensione analogica posta su uno dei canali deve trascorrere un intervallo di tempo detto *tempo di campionamento* (*A/D Sampling Time*) costituito dalla somma di un *Tempo di Acquisizione* T_{ACQ} *(Acquisition Time)* e un *Tempo di Conversione* T_{CNV} *(A/D Conversion Time)*. Il tempo impiegato da convertitore per convertire un solo bit è detto T_{AD}. Questo periodo di tempo non deve essere inferiore a 1,6 µs. Per la conversione completa dei 10 bit sono necessari 12 T_{AD}.

Il *Tempo di Acquisizione* (T_{ACQ}) è il tempo durante il quale il condensatore il tempo di *holding* (120 pF) rimane connesso con la tensione posta in ingresso sul canale analogico prescelto. Il *Tempo di Acquisizione* è legato a fattori quali la temperatura ambiente, la tensione di alimentazione V_{DD}, il valore della capacità di *holding*, e valori resistivi del canale analogico.

Il funzionamento corretto del modulo A/D è legato alla scelta del Tempo di Acquisizione (T_{ACQ} *Acquisition Time*). Per una temperatura ambiente di 50° C una V_{DD} =5V e una esistenza della sorgente analogica inferiore a 10 kΩ si può assumere $T_{ACQ} \approx 20$ µs (valore tipico $T_{ACQ} \approx 40$ µs).

Il *Tempo di Conversione* (T_{CNV}) uguale a 12 T_{AD} è legato invece al tipo di sorgente di clock prescelta per il convertitore e alla frequenza del clock di sistema. Nel caso si scelga a sorgente R, si ha T_{AD} = 4 µs (tipico) che può variare da 2 µs a 6 µs.

Se per esempio, si ha un clock di sistema pari a 4 MHz e si è impostato come sorgente fOSC/8 (con ADCS1 = 0 e ADCS0 = 1), si ha :

$$TAD = 8\ TOSC = 8 / fOSC\ 1 = 8/(4 * 10^{6}) = 2\ \mu s$$

e quindi $TCNV = 12\ TAD = 24\ \mu s$

9.4 Acquisizione con generazione d'interrupt

Per controllare il termine della conversione può essere testato il bit GO/DONE del registro ADCON0 per vedere quando esso va a livello basso o ricorrere alla tecnica dell'*interrupt*.
Infatti, al termine della conversione, il bit 6 del registro IR1 (ADIF) viene posto ad uno generando un *interrupt* se preventivamente è stato abilitato, ponendo a uno, il bit 6 del registro PIE1 (ADIE). Perché l'*interrupt* venga riconosciuto deve essere comunque abilitato anche il bit degli *interrupt* di periferica PEIE (bit 6 del registro INTCO) e quello egli *interrupt* generali GIE (bit 7 di INTCON).
Il programma riportato di seguito, usa la tecnica dell'*interrupt* per eseguire l'acquisizione di un dato analogico.
Notare come la locazione di memoria 004h sia riservata all'inizio della routine di *interrupt* e che quindi, come prima istruzione, deve essere messo un salto all'inizio del programma principale.

```
        goto    inizio

        ;routine d'interrupt
        org 4
        movf    adresh, 1       ; carica in accumulatore la parte alta del risultato conve rsione
        movwf   20h             ; salva in memoria la parte alta del risultato
        movf    adresl, 1       ; carica in accumulatore la parte bassa del risultato conve rsione
        movwf   21h             ; salva in memoria la parte bassa del risultato
        bcf     0Ch, 06h        ; ADIF = 0 del registro PIR1 — azzera bit flag interrupt dell'A/D
        retfie                  ; ritorno dall'interrupt
        ; programma principale
inizio: movlw   01h             ; carica in accumulatore la parola per configurare RA0 c ome input
        tris    05h             ; la pone nel registro TRISB
        bcf     03h,06h         ; bit RP1 = 0 del registro STATUS
        bsf     03h,05h         ; bit RP0 = 1 del registro STATUS – seleziona bank 1
        movlw   8Eh             ; carica in W codice per selezione Vref e canale AN0
        movwf   ADCON1          ; sposta il codice nel registro ADCQN1 (bank 1)
        bsf     8Ch,06h         ; bit ADIE = 1 del registro PIE1 - abilita inter. dell'A/D
        bcf     03h,05h         ; bit RP0 = 0 del registro STATUS – seleziona bank 0
        movlw   41h             ; carica in W codice per frequenza canale, A/D ON
        movwf   ADCON0          ; sposta il codice nel registro ADCON0 (bank 0)
        bcf     0Ch, 06h        ; bit ADIF = 0 del registro PIR1 — azzera bit flag interrupt dell'A/D
        bsf     0Bh,06h         ; abilita interrupt periferiche (PEIE = 1 del registro INTCON)
        bsf     0Bh,07h         ; abilita interrupt generali (GIE = 1 del registro INTCON)
        call    ritardo         ; chiama la routine di ritardo (circa 20 µs) per il tempo di sa mpling
loop:   bsf     ADCON0, 2       ; pone ad uno il bit 2 di ADCON0 e avvia la conversione
        . . . . . .             ; istruzioni varie di el aborazione
        . . . . . .             ; per esempio visualizzazione del dato
        . . . . . .             ; durante l'esecuzione di queste istruzioni finisce la conversione
        . . . . . .             ; e si salta alla routine d'interrupt
        goto    loop            ; si avvia nuova conversione
```

10. I moduli Timer

Il PIC 16F876 possiede al proprio interno tre moduli timer definiti TIMER0 (TMR0), TIMER1 (TMR1) e TIMER2 (TMR2).

10.1 TIMER0

Il funzionamento di questo modulo è simile a quello del PIC 16F84. Il TIMER0 è a 8 bit. E' possibile utilizzare TMR0 con *clock* esterno (ingresso su RA4) o interno (fosc/4) ed abbinare ad esso un *prescaler* ad 8 bit con fattori di divisione 1:1, 1:1, ... 1:128. Tutte le volte che il registro TMR0 passa dal valore FFh a 00h, viene generato un *interrupt*. Nella tabella 10.1 sono riportati i registri associati con TMR0.

Tab. 10.1

TIMER0								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	01h, 101h
Valore a 8 bit								

INTCON								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Bh, 8Bh, 10Bh, 18Bh
GIE		TOIE			TOIF			

GIE = 1	Abilitazione generale degli *interrupt*
TOIE = 1	Abilitazione dell'*interrupt* di TMR0
TOIF	Viene settato quando si genera un *interrupt*

OPTION REG								81h, 181h
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
		TOCS	TOSE	PSA	PS2	PS1	PS0	

TOCS	**0** *clock* interno, **1** *clock* esterno
TOSE	Con *clock* esterno seleziona fronte (**0** fronte di salita, **1** di discesa)
PSA	**0** prescaler assegnato a TMR0, **1** assegnato al WDT
PS2	Selezionano rapporto prescaler:
PS1	Per TMR0: **000** ⇒ 1:2 … **111** ⇒ 1:256
PS0	Per WDT: **000** ⇒ 1:1 … **111** ⇒ 1:128

10.2 TIMER1

Il TIMER1 è un timer a 16 bit in grado di effettuare conteggi da 0 a 65535. può essere usato con *clock* interno in modalità *timer* (con frequenza pari a fosc/4) oppure con *clock* esterno (su RC0), in modalità *counter*. In modalità *counter* si può avere funzionamento *sincrono* (con il clock di sistema) o *asincrono*. E' possibile associare al *timer* un *prescaler* con fattori di divisione **1:1, 1:2, 1:4 e 1:8**. TMR1 genera un interrupt (e abilitato) quando il conteggio passa da FFFFh a 0000h. Il *timer* usa una coppia di registri ad 8 bit per funzioni di temporizzazione.

Nella tabella 10.2 sono rappresentati i registri associati a TMR1.

Tab. 10.2

TMR1L								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Eh
Byte meno significativo								

TMR1H								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Fh
Byte più significativo								

INTCON								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Bh, 8Bh, 10Bh, 18Bh
GIE	PEIE							

GIE = 1	Abilitazione generale degli *interrupt*
PEIE = 1	Abilitazione dell'*interrupt* delle periferiche

T1CON								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	10h
		T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	

T1CKPS1 T1CKPS0	Valore del prescaler: **11** ⇒ 1:8 **10** ⇒ 1:4 **01** ⇒ 1:2 **00** ⇒ 1:1
T1OSCEN	**1** abilitato oscillatore esterno ; **0** oscillatore esterno non abilitato
T1SYNC	**1** segnale esterno non sincronizzato; **0** segnale esterno sincronizzato
TMR1CS	**1** clock esterno (da RC0); **0** clock interno ($f_{OSC}/4$)
TMR1ON	**1** TIMER1 abilitato

PIR1								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Ch
							TMR1IF	

TMR1IF	**1** è avvenuto un *interrupt*; **0** non c'è stato *interrupt*

PIE1								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	8Ch
							TMR1IE	

TMR1IE	**1** abilitato *interrupt* di TMR1; **0** disabilitato *interrupt* di TMR1

10.3 TIMER2

Il TMR2 è un timer con registro ad 8 bit. Ad esso è associato il registro di periodo PR2 a 8 bit. Il clock d'ingresso di TMR2, quello di sistema diviso per quattro (*fosc*/4) che passa, prima di raggiungere il timer, per un prescaler con possibilità di divisione della frequenza **1:1, 1:2, o 1:16**.
Il contenuto di TMR2 viene quindi incrementato, a partire dal valore caricato in esso, con una frequenza stabilita dal clock di sistema e dal prescaler , il valore del registro, viene confrontato in continuazion con il valore contenuto in PR2 per mezzo di un comparatore. Quando il contenuto dei due registri diviene uguale, l'uscita del comparatore attraverso un postscaler, con fattori di divisione da 1:1 a 1:16, pone ad uno il bit di intrrupt per TMR2 (TMR21F). Se abilitato, quindi, viene generato un interrupt. Il segnale d'uscita del TIMER2 può essere utilizzato anche come clock per la porta seriale sincrona. Nella tabella 10.3 è riportata la configurazione del registro INTCON per il controllo degli interrupt (evidenziando i soli bit relativi al TIMER2), del registro di controllo TMR2 (T2CON) e dei registri PIR1, PIE1 (per i soli bit relativi a TMR2).

Tab. 10.3

INTCON								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Bh, 8Bh, 10Bh, 18Bh
GIE	PEIE							

GIE = 1	Abilitazione generale degli interrupt
PEIE = 1	Abilitazione dell'interrupt delle periferiche

T2CON								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	10h
	TOUTPS 3	TOUTPS 2	TOUTPS 1	TOUTPS 0	TMR2 ON	T2CKPS 1	T2CKPS 0	

TOUTPS3 ……. TOUTPS0	Valore del postscaler: **0000** ⇒ 1:1 **0001** ⇒ 1:2 **. . .** **1111** ⇒ 1:16
TMR2ON	**1** TIMER2 abilitato; **0** TIMER2 disabilitato
T2CKPS1 T2CKPS0	Valore del prescaler: **00** ⇒ 1:1 **01** ⇒ 1:4 **1X** ⇒ 1:16

PIR1								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0Ch
						TMR2IF		

TMR2IF	**1** è avvenuto un *interrupt*; **0** non c'è stato *interrupt*

PIE1								INDIRIZZO
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	8Ch
						TMR2IE		

TMR2IE	**1** abilitato *interrupt* di TMR2; **0** disabilitato *interrupt* di TMR2

11. Modulo Capture/Compare/PWM

Il microcontrollore contiene due moduli CCP (*Capture/Compare/PWM*) che lavorano in modalità simile. Ogni modulo contiene due registri ad 8 bit (CCPR1H, CCPR1L per il primo modulo e CCPR2H, CCPR2L per il secondo modulo). Ci si riferisce nel seguito al primo modulo. Ognuno dei moduli può operare in tre modalità diverse: *modo Capture, modo Compare e modo PWM*.
La scelta dei diversi modi di funzionamento viene effettuata predisponendo opportunamente alcuni bit del registro CCP1CON (o CCP2CON) come indicato nella tabella 111

Tab. 11.1

CCP1CON							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCP1X	CCP1Y	Utilizzati solo in PWM. Sono i due bit meno significativi del duty cycle.					
CCP1M3	CCP1M2	CCP1M1	CCP1M0	MODALITÀ			
0	0	0	0	Disabilita tutti i modi.			
0	1	0	0	Capture – ogni fronte di discesa su RC2/CCP1			
0	1	0	1	Capture – ogni fronte di salita su RC2/CCP1			
0	1	1	0	Capture – ogni 4 fronti di salita su RC2/CCP1			
0	1	1	1	Capture – ogni 16 fronti di salita su RC2/CCP1			
1	0	0	0	Compare - porta a livello alto RC2/CCP1			
1	0	0	1	Compare - porta a livello basso RC2/CCP1			
1	0	1	0	Compare - RC2/CCP1 non varia viene generato un interrupt software			
1	0	1	1	Compare - si generano eventi speciali			
1	1	X	X	PWM			

11.1 Modo Capture

In questa modalità, quando si manifesta un evento sull'ingresso RC2/CCP1, il registro a 16 bit CCPR1 (formato dalla coppia di registri ad 8 bit CCPR1H : CCPR1L) cattura il contenuto attuale del registro TMR1 che deve lavorare in modalità timer. L'evento che può attivare la cattura è uno dei seguenti :

- ogni fronte di salita di un segnale posto su RC2/CCP1
- ogni fronte di discesa di un segnale posto su RC2/CCP1
- ogni 4 fronti di salita di un segnale posto su RC2/CCP1
- ogni 16 fronti di salita di un segnale posto su RC2/CCP1

prima di utilizzare la modalità *Capture* il pin RC2 della porta C deve essere impostato come *input*.
Quando si verifica una cattura il bit 2 (CCP1IF) del registro PIR1 viene posto ad uno se è stato precedentemente abilitato (posto a zero) il bit 2 (CCP1PIE) del registro PIE1. Il bit CCP1IF deve essere azzerato via software prima di una nuova operazione di Capture.

11.2 Modo Compare

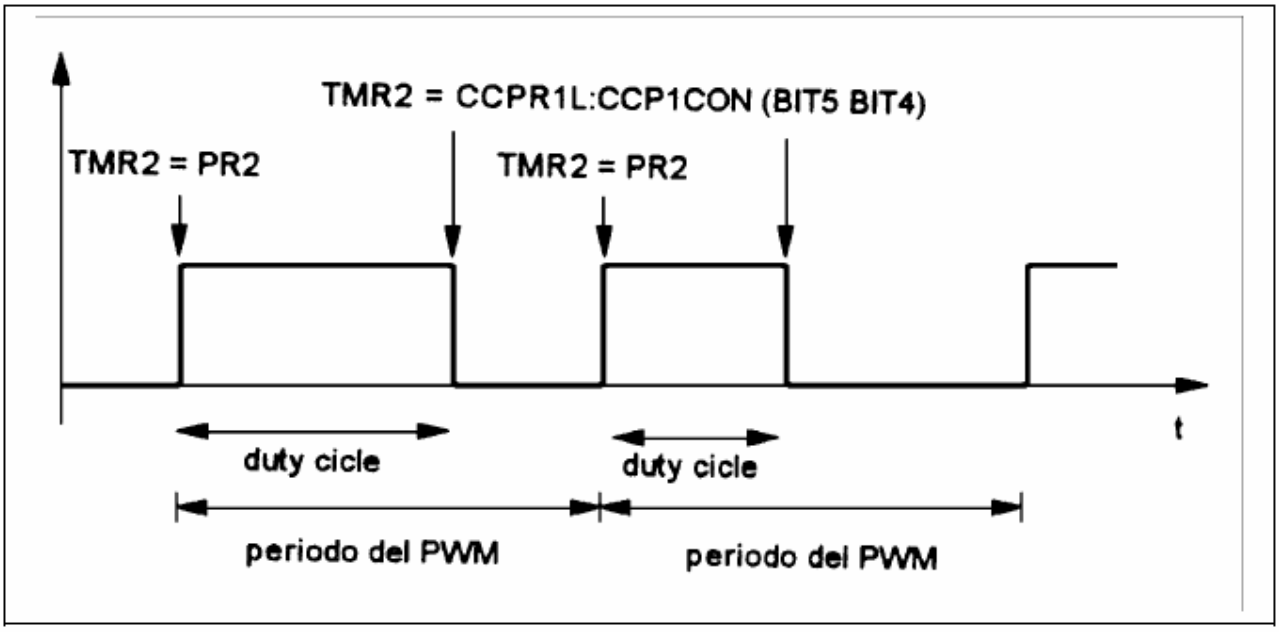
Nella modalità *Compare* il contenuto del registro CPR1 (CCPR1H : CCPR1L) viene confrontato con quello di TMR1. Quando c'è uguaglianza si verifica sul pin RC2/CCP1 (che deve essere preventivamente configurato come *output*), uno dei seguenti eventi :

- RC2/CCP1 viene portato a livello alto.
- RC2/CCP1 viene portato a livello basso
- RC2/CCP1 non subisce variazioni ma viene generato un interrupt software.
- Si genera un speciale evento di *trigger* in modalità diversa in base al modulo usato:
  1. Modulo 1 : viene posto a livello alto il bit CCP1IF – viene *resettato* TMR1
  2. Modulo 2 : viene posto a livello alto il bit CCP2IF – viene *resettato* TMR1 – viene avviata una conversione A/D se il convertitore è abilitato.

Il tipo di evento da generare deve essere programmato nel registro CCP1CON o, per utilizzare il Modulo 2, nel registro CCP2CON

11.3 Modo PWM

Nella modalità PWM (*Pulse Wide Modulation*) viene variato il *duty cycle* di un segnale con un certo periodo prestabilito. Il periodo del segnale deve essere caricato nel registro PR2 (registro di periodo per il TMR2). Il *duty cycle* viene invece caricato nel registro CCPR1L (otto bit di ordine più alto) e sui bit 5 e bit 4 del registro CCP1CON (due bit di ordine più basso). La lunghezza complessiva del valore che può essere attribuito al *duty cycle* è quindi di 10 bit.
Il valore caricato in PR2 viene continuamente confrontato con il valore di TMR2 (che è incrementato con la frequenza $f_{osc}/4$) quando i due valori sono uguali viene posto alto il pin RC2/CCP1 (uscita del PWM) e TMR2 viene azzerato. Poi TMR2 (sempre incrementato con la frequenza $f_{osc}/4$) viene confrontato con CCPR1L e i bit 5 e bit 4 del registro CCP1CON. Quando c'è uguaglianza viene posto a zero il pin RC2/CCP1 (vedere figura). Poi il processo inizia da capo generando un nuovo periodo del segnale.



Le operazioni da compiere per utilizzare il modulo CCP in modalità PWM sono le seguenti :

- Configurare il pin RC2/CCP1 come uscita;
- Caricare in PR2 il valore da attribuire al periodo del PWM;
- Caricare nel registro CCPR1L e nei bit 5 e bit 4 di CCP1CON il valore del *duty cycle*;
- Configurare il *prescaler* di TMR2 con i bit T2CKPS1 ÷ T2CKPS0 del registro T2CON;
- Abilitare TMR2 (porre a uno il bit 2 T2CON);
- Configurare il registro CCP1CON per modalità PWM.

Le formule per il calcolo del periodo di PWM e del *duty cycle* sono :

periodo PWM = Tosc x 4 x (Valore *prescaler* TMR2) x (PR2 + 1)
durata duty cycle = Tosc x 4 x (Valore *prescaler* TMR2) x CCPR1l : CCP1CON (bit5 bit4).