I microcontrollori

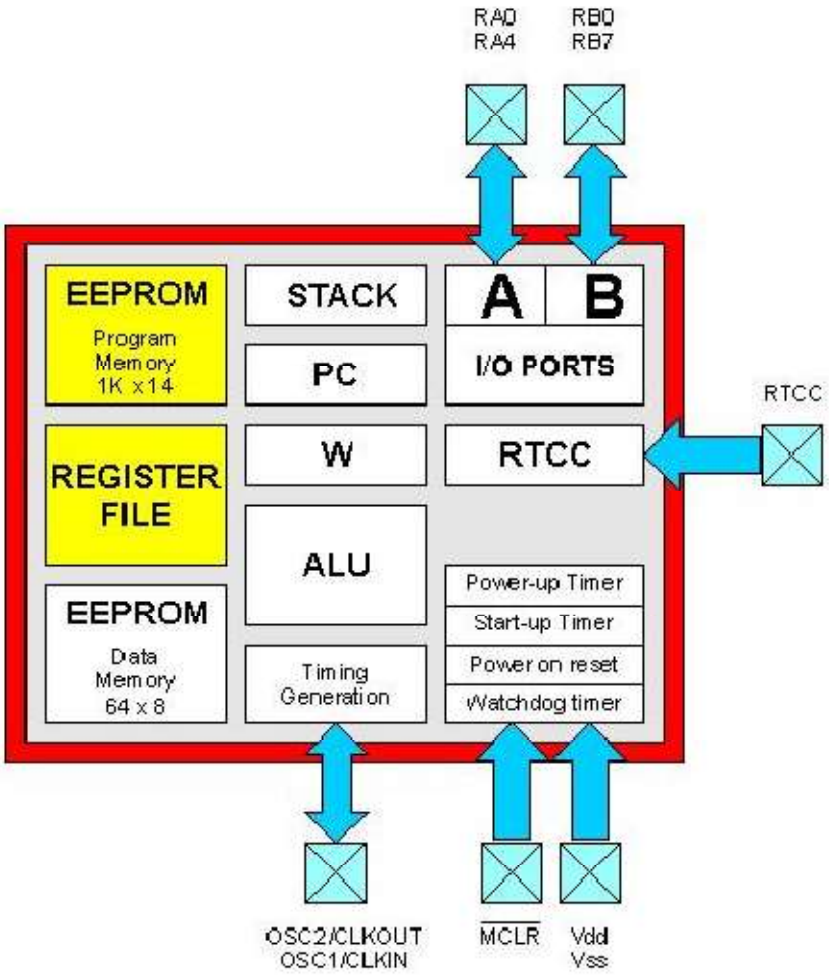
Con il termine di microcontrollori si intende un sistema integrato simile ad un microprocessore ma contente su un unico chip oltre la CPU, anche la memoria RAM, memoria dati, memoria programmi, ADC, PWM, periferiche di input output. Tutte queste applicazioni sono dette embedded cioè incorporate in apparati. Il microcontrollore viene utilizzato in tutto ciò che è automatico come cellulari, telecomandi, chiavi elettroniche, sistemi di sicurezza, misure a distanza, controlli di grandezze fisiche a distanza. I microcontrollori vengono utilizzati in quei sistemi detti EMBEEDDED, cioè integrati e specializzati a svolgere determinate operazioni

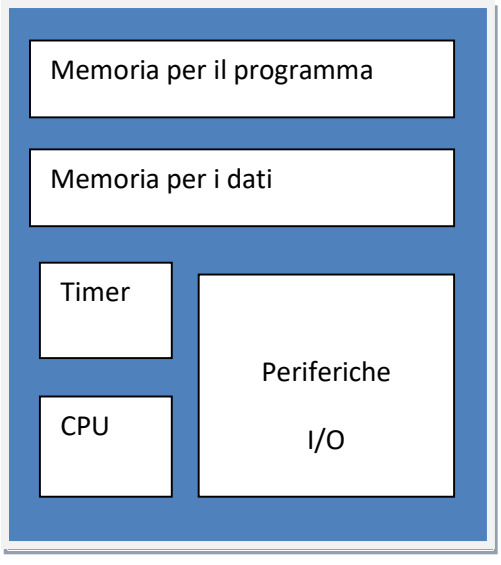
I Pic sono microcontrollori della microchip. Fanno parte della architettura RISC (Reduced Instruction Set Computing) che prevede poche e semplici istruzioni della stessa lunghezza, cioè tutte richiedenti lo stesso tempo macchina.

In sostanza i pic hanno le seguenti caratteristiche:

1. Parte centrale: CPU; è la parte preposta al controllo, all'esecuzione del programma e alla elaborazione dei dati
2. Memoria per il programma- può essere flash se parliamo dei pic di tipo f oppure eprom se parliamo di pic di tipo windowed o OTP. Nel caso in cui la memoria programmi sia di tipo FLASH, il programma può essere cancellato e riscritto semplicemente tramite un dispositivo elettronico detto programmatore con interfaccia seriale o USB ad un pc. Se la memoria programma è EPROM, il programma può essere scritto una sola volta e non cancellato oppure cancellato tramite raggi ultravioletti (pic windowed).
3. Memoria per i dati – di tipo RAM oppure EEPROM
4. Uno o più timer
5. Le periferiche di I/O così suddivise:
   a. Porte I/O
   b. ADC convertitori analogico/ digitale
   c. Comparatori
   d. Moduli CCP
   e. I/O seriale

Struttura interna del microcontrollore pic

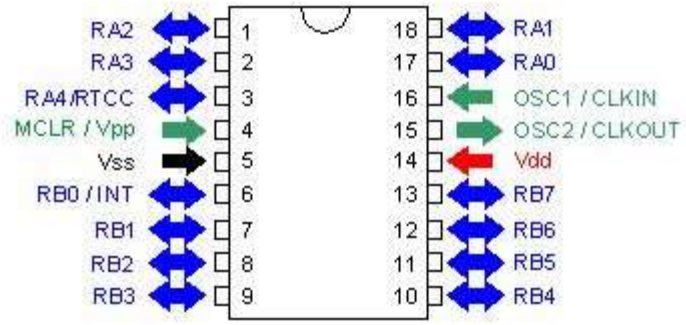




Caratteristiche salienti del pic 16f84

Alimentazione	2-6 V . assorbimento di corrente massimo 2 mA se si lavora a 5 V con frequentza di 4 MHz
Memoria	Programma: di tipo flash di un k/14 bit (1024 righe per 14 bit)
	Dati: di tipo RAM 68 byte
	Dati: di tipo EEPROM 64 byte
Clock	Con frequenza massima di 20 MHz
Timer	Un solo timer TIMER0
Watchdog	Timer che genera un segnale di reset del microcontrollore se si verifica un timer out, cioè se l'esecuzione dura più a lungo del previsto
Periferiche I/O	Sono in tutto 13 periferiche, 5 della porta A e 8 della porta B
Interrupt	Il pic può essere interrotto durante l'esecuzione del programma tramite 4 modalità differenti
Set di istruzioni	I programmi vanno scritti in assembler e ogni codice ha una estensione di 14 bit. Le istruzioni sono in tutto 35

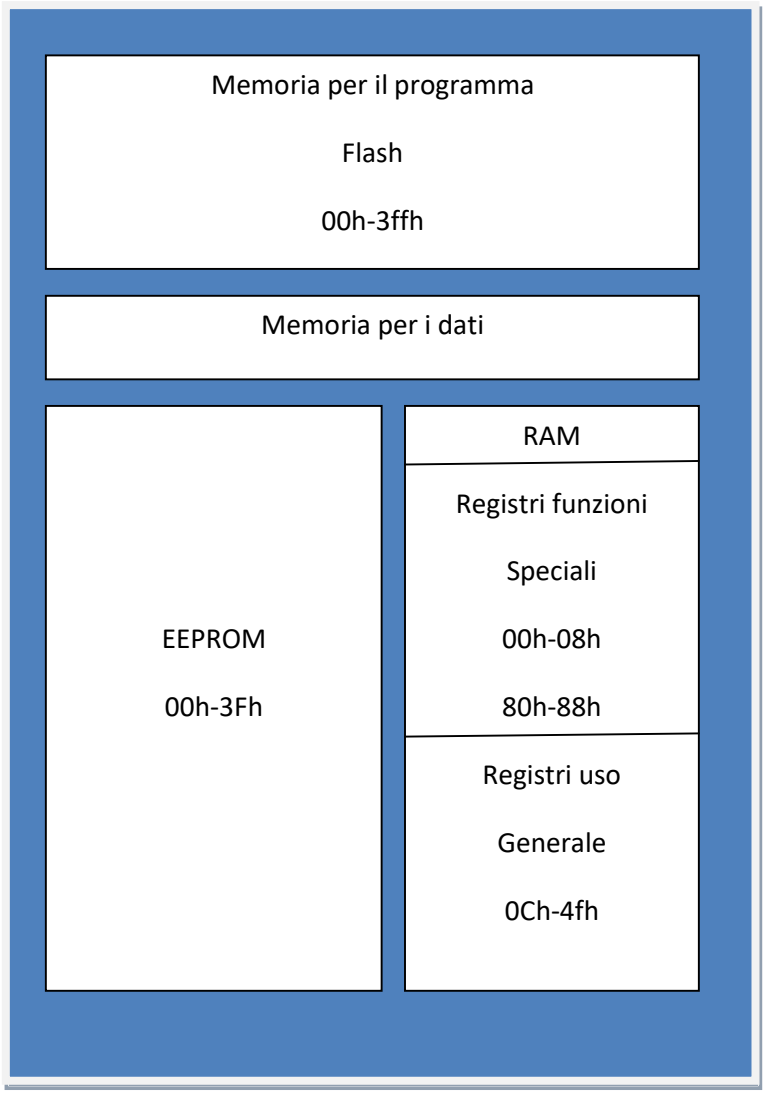
Piedinatura



Il pic 16f84 ha in tutto 18 pin

1	RA2		RA1	18
2	RA3		RA0	17
3	RA4/Tock1:può essere usata come clock esterno		OSC1/CLKIN oscillatore ingresso	16
4	MCLR:reset attivo a livello basso. In condizioni normali deve essere posto a livello alto attraverso una resistenza di pull up. Per il reset va collegato a massa ma tramite una resistenza di 50-100Ω		OSC2/CLKOUT oscillatore uscita. Tra osc1 e osc2 va collegato sempre un quarzo o una rete RC o un risonatore ceramico	15
5	VSS massa		VDD alimentazione 2-6 V	14
6	RB0/INT è utilizzata anche come segnali esterni di interrupt		RB7	13
7	RB1		RB6	12
8	RB2		RB5	11
9	RB3		RB4	10

Organizzazione della memoria



- **Memoria programma**- Nel pic 16f84 la memoria programma è di tipo flash; ogni riga di programma può quindi essere scritta e cancellata con un dispositivo elettrico detto programmatore.
  a. Essa è formata da 1024 linee di 14 bit ciascuna. La cancellazione può riguardare le singole locazioni di memoria o tutto il programma. Ogni locazione è distinta da una numero sequenziale espresso in esadecimale che va da 00h a 3ffh.
  b. Una locazione speciale è 004h da dove viene scritta la routine di interrupt.

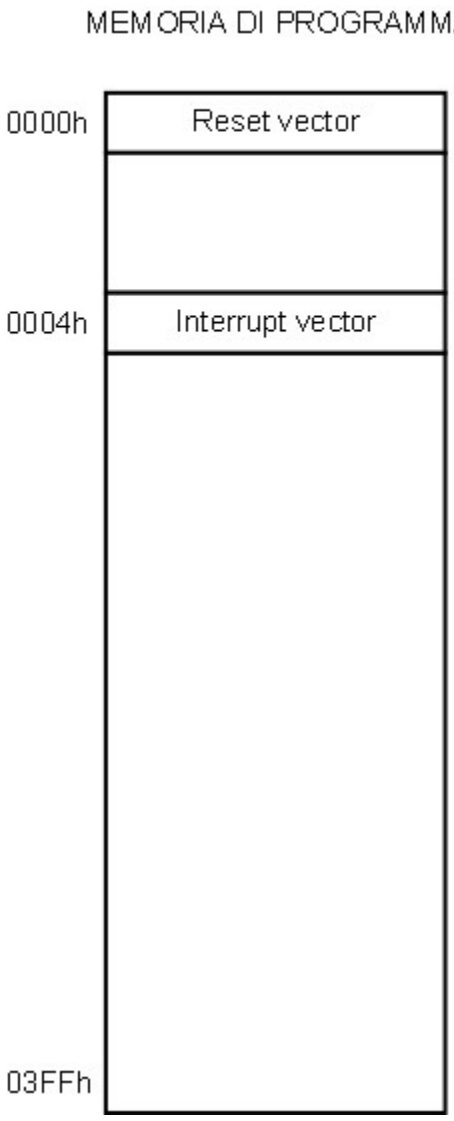
c. Una parte importante della memoria è lo stack pointer; è una memoria RAM di 8 locazioni ciascuna di 13 bit. In ogni locazione viene salvata volta per l'indirizzo di ritorno dalla subroutine. Il fatto che sono otto locazioni sta ad indicare che si possono realizzare programmi con 8 subroutine annidate.

La chiamata a subroutine viene fatta con l'istruzione call. prima di eseguire il salto,. il PIC memorizza, in un altro registro speciale,denominato **STACK**, l'indirizzo di quella che sarebbe dovuta essere la successiva istruzione da eseguire se non si fosse incontrata la CALL.
Vediamo meglio con un esempio:

```
ORG 0x00
Point1 movlw 10
       call Point2
       goto Point1
Point2    movlw 11
```

L'operazione appena effettuata viene denominata CHIAMATA A SUBROUTINE, ovvero una interruzione momentanea del normale flusso di programma per "chiamare" in esecuzione una serie di istruzioni per poi ritornare al normale flusso di esecuzione.
La parola STACK in inglese 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 **L**ast **I**n **F**irst **O**ut, in cui l'ultimo elemento inserito (last in) deve necessariamente essere il primo ad uscire (last out). Grazie a questa caratteristica è possibile effettuare più CALL annidate ovvero l'una nell'altra e mantenere sempre traccia del punto in cui riprendere il flusso almomento che si incontra una istruzione RETURN.

d. Appena si avvia il dispositivo, il program counter punta alla locazione con indirizzo 000h locazione di reset dove è scritta la prima riga di programma

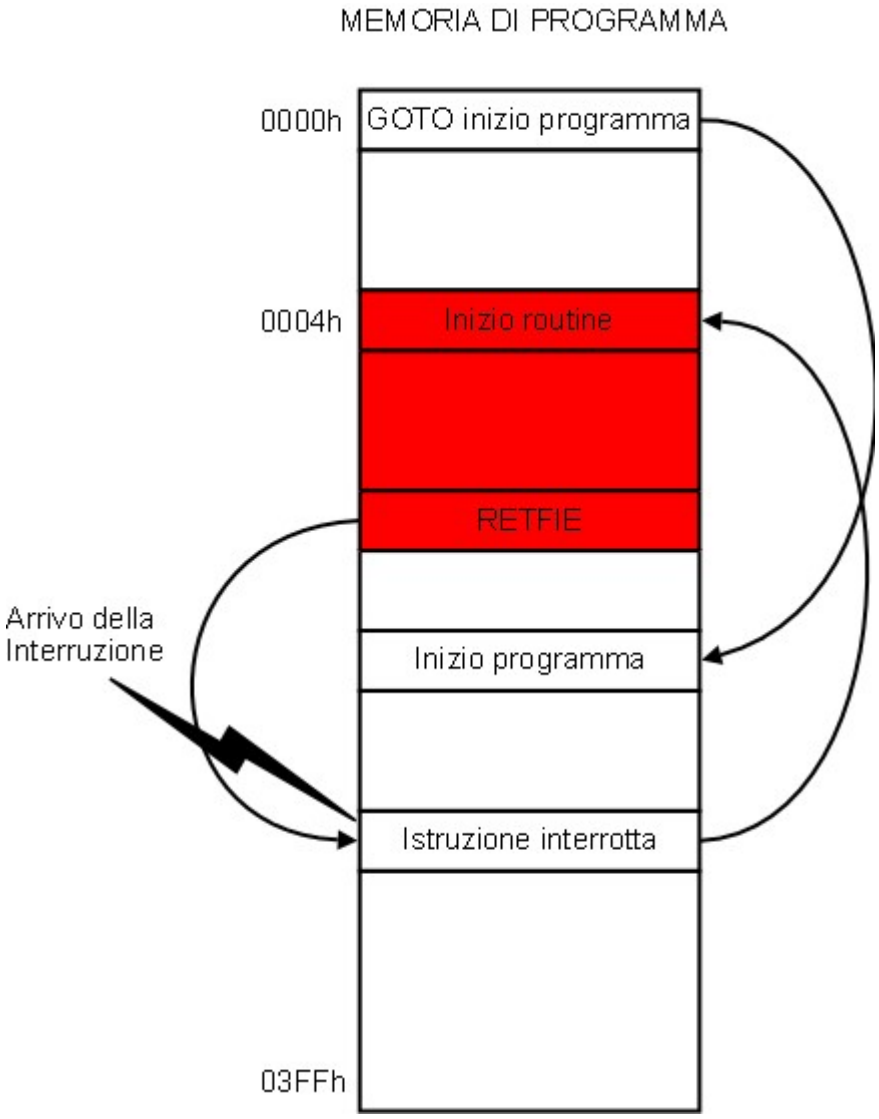


Esiste pero' una locazione speciale l'indirizzo 0004h,detto indirizzo di interrupt.Ogni qualvolta si verifica una interruzione, il programma salta immediatamente a questa locazione.

In questo modo, se si desidera creare una routine di gestione delle interruzioni occorre farla cominciare da questa precisa locazione (0004h) in modo tale che essa verra' eseguita non appena si verifichera' un interrupt. Ovviamente prima della locazione 0004h occorrera' inserire un salto incondizionato per poter "oltrepassare" la routine di gestione delle interruzioni: di solito il salto e' inserito proprio in prima posizione, cioe' in 0000h, cosicche' il microcontrollore, una volta accesso o resettato, passera' immediatamente ad eseguire il salto e quindi si posizionera' alla locazione opportuna scelta dal programmatore.

In figura si mostra graficamente il concetto di salto al vettore di interrupt.

I pic della serie PIC18 hanno 2 vettori d'interrupt (di alta e bassa priorità). Il vettore ad alta priorità è posizionato all'indirizzo 0x0008 ed il vettore a bassa priorità è posizionato all'indirizzo 0x0018.
Quando avviene un'interruzione da parte delle periferiche, prima di saltare al vettore d'interruzione, il PIC salverà tutte le informazioni necessarie nellostack , per poter riprendere dal punto in cui il programma è stato interrotto, una volta eseguite le operazioni necessarie per gestire l'interruzione.
Per abilitare il registro vettore di interrupt ci serviamo del registro speciale INTCON



- **La memoria dati EEPROM** ha una capacità di 64 locazioni di memoria di 1 byte ciascuna con indirizzo 00h-3fh
- **File register** – è formato da due banchi con 80 locazioni di memoria di un byte ciascuna. Il primo banco, banco 0, ha locazioni con indirizzo da 00h a 4fh; il secondo banco, banco 1, ha 12 locazioni che vanno da 80h a 8Bh cioè dodici locazioni. Le prime 12 locazioni del banco 0 e le dodici del banco 1, sono

registri di uso speciali; i rimanenti registri sono per uso specifico del programmatore. I file register di PIC 16F84 si dividono quindi in SFR (Special Function Register) e i restanti 68 byte per i 68 GPR (General Purpose Register). Ogni registro occupa sempre ed esattamente un byte La memoria di stack è di tipo

- **Registro accumulatore**, detto anche W. È un registro speciale attraverso il quale devono passare tutti i dati in ingresso del micro prima di passare alla ALU e viceversa. Il registro accumulatore è di 8 bit e non ha alcun indirizzo.Per qualsiasi istruzione che opera con dati, bisogna ricorrere all'accumulatore W. Infatti, per una istruzione dovremmo usare la memoria così ripartita:
  8 bit per specificare il valore che intendiamo inserire nella locazione di memoria,
  7 bit per specificare in quale locazione di memoria vogliamo inserire il nostro valore,
  6 bit per specificare quale istruzione intendiamo utilizzare.
  Ogni riga di programma è di 14 bit; una istruzione completa sarebbe di 21 bit, troppa per ogni riga programma.

00h INDF	80h INDF	SFR
01h TMR0	81h OPTION	
02h PCL	82h PCL	
03h STATUS	83h STATUS	
04h FSR	84h PSR	
05h PORTA	85h TRISA	
06h PORTB	86h TRISB	
07h	87h	
08h EEDATA	88h EECON1	
09h EEADR	89hEECON2	
0Ah PCLATH	8Ah PCLATH	
0Bh INTCON	8Bh INTCON	
4f		GPR

File register speciali

TMR0 contiene il conteggio attuale del contatore/temporizzatore. Può essere sia scritto che letto nel senso che si può scegliere il valore iniziale dal quale il contatore può iniziare a contare. Il contatore timer può contare sia impulsi interni che esterni posti sul piedino RA4. Essendo a 8 bit conta fino a 255 generando così una interrupt e riparte poi daccapo. Per l'impostazione del timer0 si utilizzano i registri INTCON e OPTION.

TRISA (85h) e TRISB (86h): servono per stabilire quali delle linee delle due porte devono essere considerate come ingresso o come uscite. È importante ricordare che ciascuna linea può essere progettata come ingresso o come uscita indipendentemente dalle altre.

Esempio di impostazione ingresso o uscita delle linee di una porta

			Ingresso	Uscita	Uscita	Ingresso	Ingresso
			RA4	RA3	RA2	RA1	RA0
X	X	X	1	0	0	1	1
1				3			

Esempio di impostazione ingresso o uscita delle linee di una porta

Ingresso	Uscita	Ingresso	Ingresso	Uscita	Uscita	Ingresso	Ingresso
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
1	1	1	1	0	0	1	1
F				3			

PORTA (05h), PORTB (06h) questi registri gestiscono il livello logico delle porte

STATUSSi può accedere al registro sia dal banco 0 che dal banco 1. I singoli bit sono i seguenti

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP=0	RP1=0	RPO	$\overline{TO}$	$\overline{PD}$	Z	DC	C

IRP e RP0	Non vengono utilizzati nel pic16f84 e vengono posti a 0
RP0	Selezione il banco 0 (RP0=0) o il banco 1 (RP0=1)

$\overline{TO}$	È posto a 0 quando il Watchdog Timer va in Time-out
$\overline{PD}$	È posto a 0 quando viene eseguita una istruzione di sleep
Z	È posto a 1 quando il risultato di una operazione logica o aritmetica è 0
DC	È posto a 1 quando c'è riporto del semi byte meno significativo
C	È posto a 1 quando c'è riporto del bit meno significativo

Nel registro status non si può scrivere direttamente ma si possono leggere i singoli bit.

Esempio per configurare il bit necessario per indirizzare i file register posti nel banco 0 o nel banco 1

Es: si scriva nel registro TRISA la configurazione della porta A

EEDATA EEADR sono rispettivamente i registri dati e i registri indirizzi; controllano la lettura e la scrittura nella memoria EEPROM. In EEADR viene immesso l'indirizzo da cui leggere o in cui scrivere il dato; in EEDATA viene immesso il dato da scrivere. Ad essi sono associati altri due registri EECON1 ed EECON2

EECON1 EECON2 sono registri implementati per comandare la EEPROM. Di EECON1 solo i 5 bit più significativi sono implementati. I bit WR e RD sono quelli che avviano il processo rispettivamente di scrittura o di lettura e sono settati a 1; il pic li setta a 0 una volta completato il processo. Il bit WREN (WRite ENable) abilita o disabilita le scritture nella EEPROM; quando viene settato ad 1 permette di effettuare le scritture, quando e' settato a 0 le scritture sono inibite. Al reset del microcontrollore tale bit e' settato a 0.
Il bit WRERR viene settato ad 1 dal pic quando o un reset oppure un time-out del watch dog ha interrotto prematuramente una scrittura nella EEPROM (va ricordato che la scrittura nella EEPROM e' una operazione relativamente lenta se rapportata alla velocita' di esecuzione delle istruzioni di programma).
Il bit EEIF funge da indicatore per il completamento della fase di scrittura. Prima di iniziare una scrittura deve essere posto a 0 e viene posto automaticamente ad 1 dal pic quando la scrittura e' terminata.Il registro EECON2 e' un semplice registro di appoggio utilizzato esclusivamente nella fase di scrittura della EEPROM. Prima di poter scrivere un byte nella EEPROM occorre scrivere in EECON2 una serie prestabilita di valori, cioe' una specie di sequenza di avvio, in grado di abilitare la EEPROM a ricevere il byte in questione. La sequenza deve essere ripetuta ogni volta che si vuole memorizzare un byte.

Struttura del registro EECON1

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
X	X	X	EEIF	WRERTR	WREN	WR	RD

Registro option

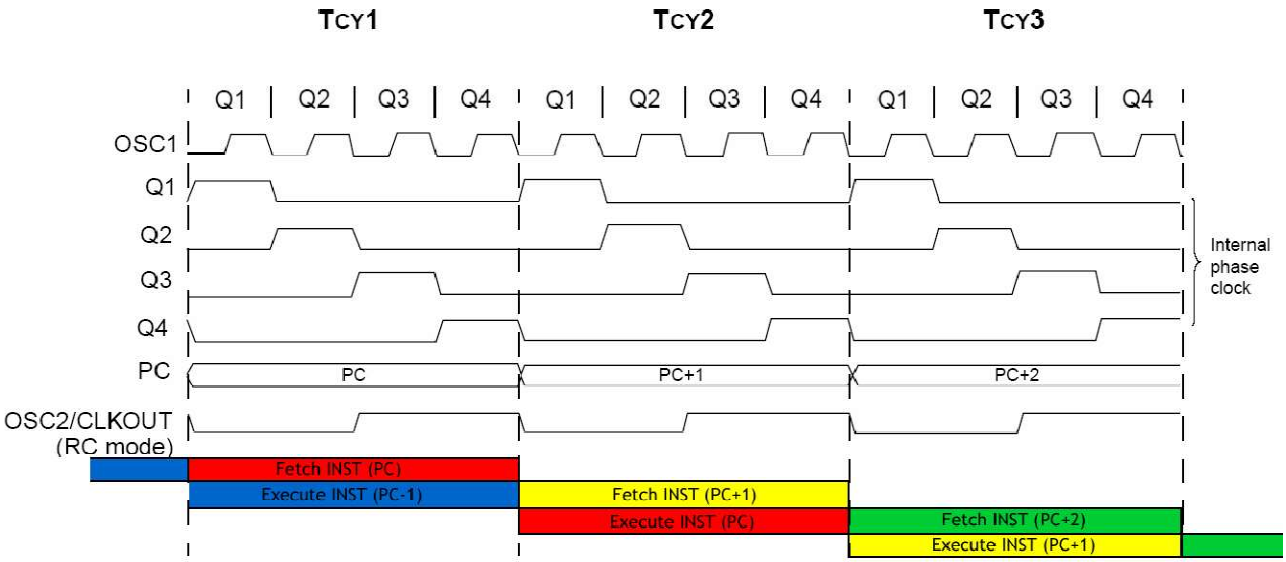
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$\overline{RBPU}$	INTEDG	TOIE	TOSE	PSA	PS2	PS1	PS0

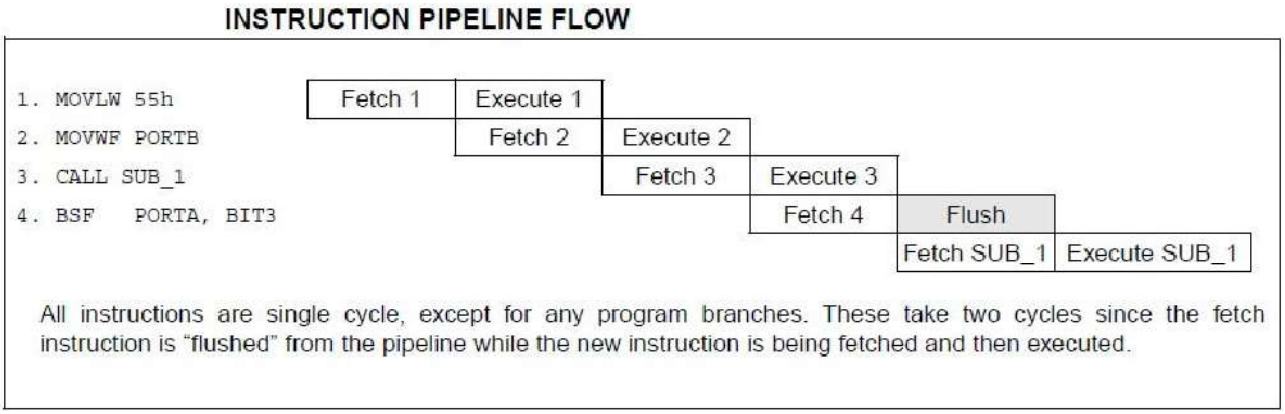
$\overline{RBPU}$	Disabilita o abilita i resistori interni di pull-up sulla porta b	
	0	Disabilita i resistori interni
	1	Abilita i resistori interni
INTDEG	Seleziona il fronte di interrupt su RB0/INT	
	0	Fronte di discesa
	1	Fronte di salita
TOCS	Seleziona la sorgente per il clock del timer (TMR0)	
	0	Clock interno
	1	Clock esterno su RA4/T0CK
TOSE	Seleziona il fronte del segnale per il clock del timer su RA4/T0CKI	
	0	Fronte di discesa
	1	Fronte di salita
PSA	Assegna il prescaler al timer TMR0 o al watch-dog WDT	
	0	TMR0
	1	WDT
PS2	Selezionano il fattore di divisione per il prescaler sia per TMR0 che per il WDT.	
PS1		
PS0		

CLOCK e temporizzazioni

Il microcontrollore è munito di un clock interno con frequenza max di 20 MHh che sincronizza tutte le attività della CPU. Ogni ciclo macchina dura 4 periodi di oscillazione del clock. Il Program counter aggiorna il suo stato ogni ciclo macchina e lo pone nel registro Timer0. Nella figura è riportata la tecnica di pipeline che permette di velocizzare la CPU. Si vede che durante il ciclo macchina *n* viene eseguita la fase di fetch *n* ed eseguite le istruzioni del ciclo *n-1*. Ciò è reso possibile

dal fatto che la fase di fetch è gestita dalla BIU e la fase di istruzione è gestita dalla EU, due strutture indipendenti della CPU.





Il **Watch Dog Timer** e' un oscillatore interno al PICmicro, ma completamente indipendente dal resto della circuiteria, il cui scopo e'quello di rilevare eventuali blocchi della CPU del micro e resettare il PICmicro per riprendere la normale esecuzione del programma.

• Per poter rilevare un eventuale blocco della CPU durante l'esecuzionedel programma principale, viene inserita all'interno di questo, unaistruzione 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 cheil programma si è bloccato per qualche motivo e si effettua il Resetdella CPU.

Interrupt

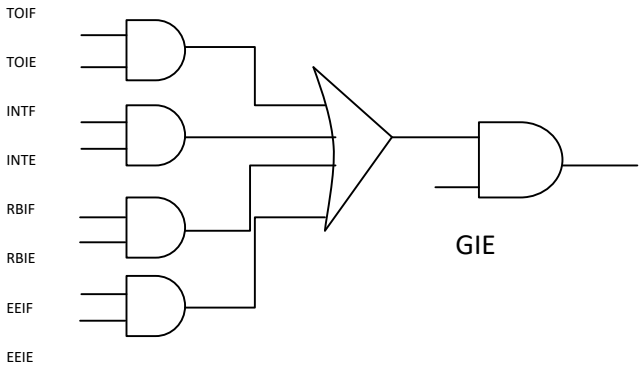
Il pic16f84 ha 4 possibili interrupt:

- Interrupt esterni sulla linea RB0/INT
- Interrupt su overflow del timer/counter TMR0
- Interrupt su fine scrittura EEPROM dati
- Interrupt su cambiamento di livello ai pin RB7-RB4

I casi di interrupt sono 4 ma il vettore interruzione è unico e si trova all'indirizzo 004h. Il flag di abilitazione all'interruzione è GIE, bit 7 del registro INTCON. Per gestire le interruzioni bisogna però specificre quali delle 4 cause li hanno generati abilitando i rispettivi flag.

Flag di interrupt del registro INTCON:

- RBIF (bit 0) individua l'interruzione su cambiamento di linee RB7-RB4
- INTF (bit 1) individua l'interruzione della linea RB0/INT
- TOIF (bit 2) individua l'interrupt su overflow del timer/counter
- EEIF (bit 4 di EECON1) è l'interrupt di fine scrittura
  Ciascuno dei quattro interrupt può essere abilitato o disabilitato indipendentemente da GIE ponendo rispettivamente a 1 o a 0 i corrispondenti bit del registro status. In particolare:
  - RBIE (bit 3) abilita o disabilita il corrispondente RBIF
  - INTE (bit 4)abilita o disabilita il corrispondente INTF
  - TOIE (bit 5) abilita o disabilita il corrispondente TOIF
  - EEIE (bit 6)abilita o disabilita il corrispondente EEIF



GIE è il flag di abilitazione generale degli interrupt. Se è alto abilita tutti quelli non mascherati, se basso li disabilita tutti

esempio: si voglia accendere e spegnere un led collegato alla linea RA0 con un intervallo di 1 secondo utilizzando un quarzo con frequenza di 3,276 MHz

L'uso dell'interrupt da parte del programmatore non è semplicissimo e richiede degli accorgimenti. È importante ricordare che l'evento fisico non determina direttamente l'interrupt ma alza solo il flag corrispondente indipendentemente dal fatto che l'interrupt corrispondente sia abilitato. Il flag qualificatore, il GIE e il flag di abilitazione causano l'interrupt. Es un impulso su RB0/INT determina l'innalzamento di INTF anche se GIE e INTE sono bassi. Se in un tempo successivo vengono posti alti GIE INTE e INTF è ancora alto, si genera un interrupt.

Accorgimenti:

- Al rientro della routine di interruzione (e quindi al ripristino dell'abilitazione generale con GIE che alto), se il flag di richiesta di interrupt è ancora alto la routine parte indefinitamente. Per evitarlo, il flag di richiesta va abbassato all'interno della routine di gestione. È opportuno farlo con l'ultima istruzione della routine stessa prima dell'istruzione di rientro RETFIE, per evitare che un evento del tipo atto a generare l'interrupt lo possa rialzare durante l'esecuzione della routine stessa
- Di default gli interrupt non sono nidifica bili; se si desidera che lo siano, cioè che una routine di interruzione possa a sua volta essere interrotta, si deve alzare GIE all'inizio della routine di gestione.

```
void interrupt(void)

{

 PORTD++;        //Incrementa la porta D di una unità

 delay_ms(500);

 INTCON.INTF = 0;        // cancella il flag di interrupt

}


void main(void)

{

 TRISB = 0x01;

 TRISD = 0x00;
```

```
 INTCON.GIE = 1;              //Cancella il flag globale
 INTCON.INTE = 1;             //abilita RB0/INT
 INTCON.PEIE = 0;             //Disabilita tutti gli interrupt
 OPTION_REG.INTEDG = 1;
 PORTD=0;
do
 {


  } while(1);}
```

N.B: i bit che terminano con E sono di Enable, quelli che terminano con F sono di Flag

Registro INTCON

È possibile accedere sia dal banco 0 che dal banco 1.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE	Abilitazione generale di tutti gli interrupt	
	0	Tutti gli interrupt sono disabilitati
	1	Tutti gli interrupt sono abilitati
EEIE	Segnala il completamento di un ciclo di scrittura su EEPROM dati	
	0	Ciclo di scrittura non completato
	1	Ciclo di scrittura completato
TOIE	Abilitazione dell'interrupt di superamento di capacità di TMR0	
	0	Disabilitato
	1	Abilitato
INTE	Abilita l'interrupt per la linea RB0/INT	
	0	Disabilitato
	1	Abilitato
RBIE	Abilita l'interrupt per il cambio di livello sulle linee RB7-RB4	
	0	Disabilitato
	1	Abilitato

TOIF	Segnalazione di superamento capacità di TMR0	
	0	Non c'è stato overflow
	1	C'è stato overflow
INTF	Segnalazione di richiesta di interrupt sulla linea RB0/INT	
	0	Non c'è stato superamento di capacità di TMR0
	1	C'è stato superamento di capacità
RBIF	Segnalazione di avvenuto cambio di livello sulle linee RB7-RB4	
	0	Non è cambiato nessun livello
	1	Sono cambiati uno o più livelli

Timer0 e prescaler

È possibile far variare la frequenza del segnale di clock per generare un interrupt con frequenza differente. Si possono utilizzare due dei seguenti metodi:

- Utilizzo del timer senza prescaler
    1. Scrivere la routine d'interrupt del timer partendo dalla locazione 004h della memoria di programma. Nella routine devono essere poste le seguenti informazioni:
        a. Ricaricare nel timer il valore di inizio del conteggio
        b. Azzerare il flag di avvenuto interrupt (bit 2 di INTCON)
        c. Porre al termine della routine l'istruzione di ritorno dall'interrupt (RETFIE)
    2. Il programma principale deve essere scritto nel seguente modo:
        a. Impostare in modo timer ponendo a 0 il bit 5 di OPTION
        b. Inizializzare il timer ponendo il valore iniziale del timer
        c. Abilitare l'interrupt del timer ponendo a 1 il bit 5 di INTCON
        d. Abilitare gli interrupt ponendo a 1 il bit 7 di INTCON

Con questo metodo, se il timer è inizializzato a 00h, la frequenza di interrupt sarà: $f_i=f_{macchina}/256$ dove $f_{macchina}=f_{osc}/4$

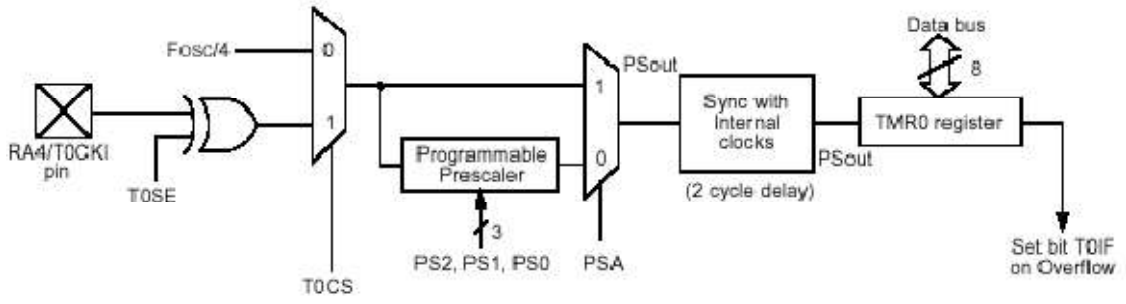
Se il timer è inizializzato ad Nt allora: $f_i=f_{macchina}/(256-Nt)=f_{osc}/4*(256-Nt)$

- Utilizzo del prescaler

1. Scrivere la routine d'interrupt del timer partendo dalla locazione 004h della memoria di programma. Nella routine devono essere poste le seguenti informazioni:
    a. Ricaricare nel timer il valore di inizio del conteggio
    b. Azzerare il flag di avvenuto interrupt (bit 2 di INTCON)
    c. Porre al termine della routine l'istruzione di ritorno dall'interrupt (RETFIE)
2. Il programma principale deve essere scritto nel seguente modo:
    a. Assegnare il prescaler al timer ponendo a 0 il bit 5 di OPTION; seil bit è posto a 1 allora il prescaler è posto al watchdog timer
    b. Selezionare il fattore di divisione come in tabella
    c. Inizializzare il timer ponendo il valore iniziale del timer
    d. Abilitare l'interrupt del timer ponendo a 1 il bit 5 di INTCON
    e. Abilitare gli interrupt ponendo a 1 il bit 7 di INTCON

Con questo metodo, se il timer è inizializzato a 00h, e N_p è la divisione di prescaler, la frequenza di interrupt sarà: $f_i=f_{macchina}/N_p*256$ dove $f_{macchina}=f_{osc}/4$

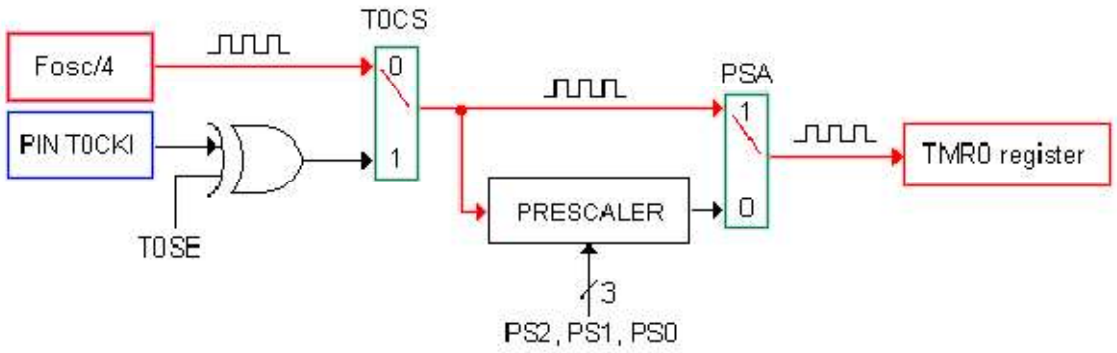
Se il timer è inizializzato ad Nt allora: $f_i=f_{macchina}/N_p(256-Nt)=f_{osc}/4*N_p(256-Nt)$



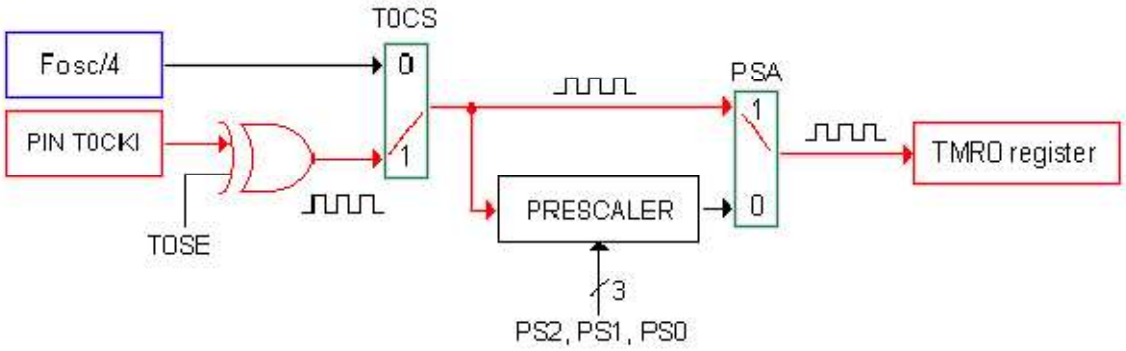
I blocchi Fosc/4 e T0CKI rappresentano le due possibili sorgenti di segnale per il contatore TMR0.

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 T0CKIcorrispondente al pin 3 nel PIC16F84A.

I blocchi T0CS e PSA riportati in verde sono due commutatori di segnale sulla cui uscita vienepresentato uno dei due segnali in ingresso in base al valore dei bit T0CS e PSA del registroOPTION.
Vediamo in pratica come è possibile agire su questi blocchi per ottenere differenti modalità di conteggio per il registro TMR0.
Iniziamo programmando i bit T0CS a 0 e PSA a 1. La configurazione di funzionamento che otterremo è rappresentata nella seguente figura:

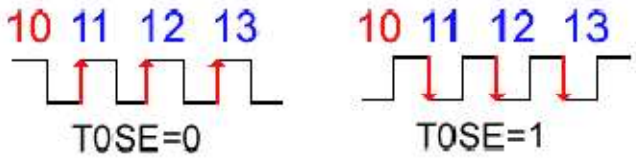


Come abbiamo già detto in precedenza, la frequenza Fosc/4 è pari ad un quarto della frequenza di clock. Utilizzando un quarzo da 4Mhz avremo una Fosc/4 pari ad 1 MHz. Tale frequenza viene inviata direttamente al registro TMR0 senza subire nessun cambiamento. La cadenza di conteggio che se ne ottiene è quindi pari ad 1 milione di incrementi al secondo del valore presente in TMR0.
Ipotizziamo ora di cambiare lo stato del bit T0CS da 0 a 1 la configurazione che otteniamo è la seguente:

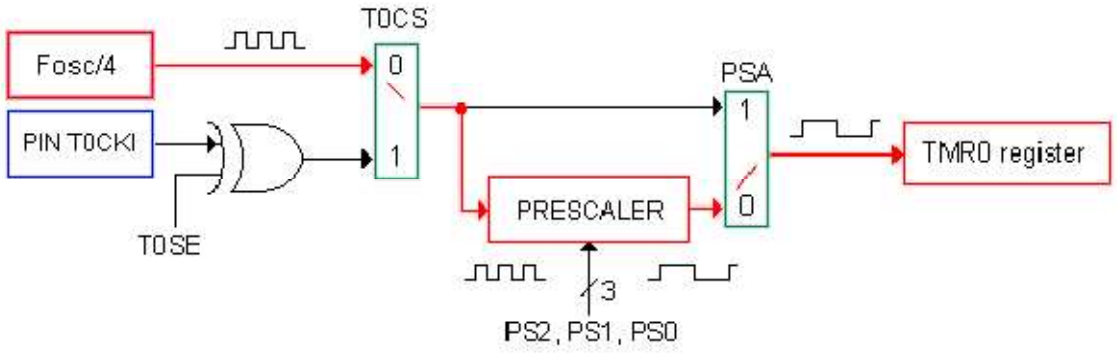


Questa volta sarà il segnale applicato al pin TOCKI del PIC ad essere inviato direttamente al contatore TMR0 determinandone la frequenza di conteggio. Applicando ad esempio a questo pin una frequenza pari ad 100Hz otterremo una frequenza di conteggio pari a cento incrementi al secondo.
La presenza della porta logica XOR (exclusive OR) all'ingresso TOCKI del PIC consente dideterminare tramite il bit T0SE del registro OPTION se il contatore TMR0 deve essereincrementato in corrispondenza del fronte di discesa (T0SE=1) o del fronte di salita (T0SE=0)del segnale applicato dall'esterno.

Nella figura seguente viene rappresentata la corrispondenza tra l'andamento del segnaleesterno ed il valore assunto dal contatore TMR0 in entrambe i casi:



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:



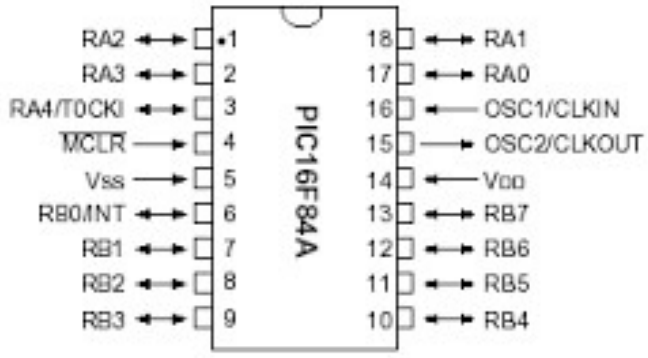
Il PRESCALER consiste in pratica in un divisore programmabile a 8 bit da utilizzare nel caso la frequenza di conteggio inviata al contatore TMR0 sia troppo elevata per i nostri scopi.

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

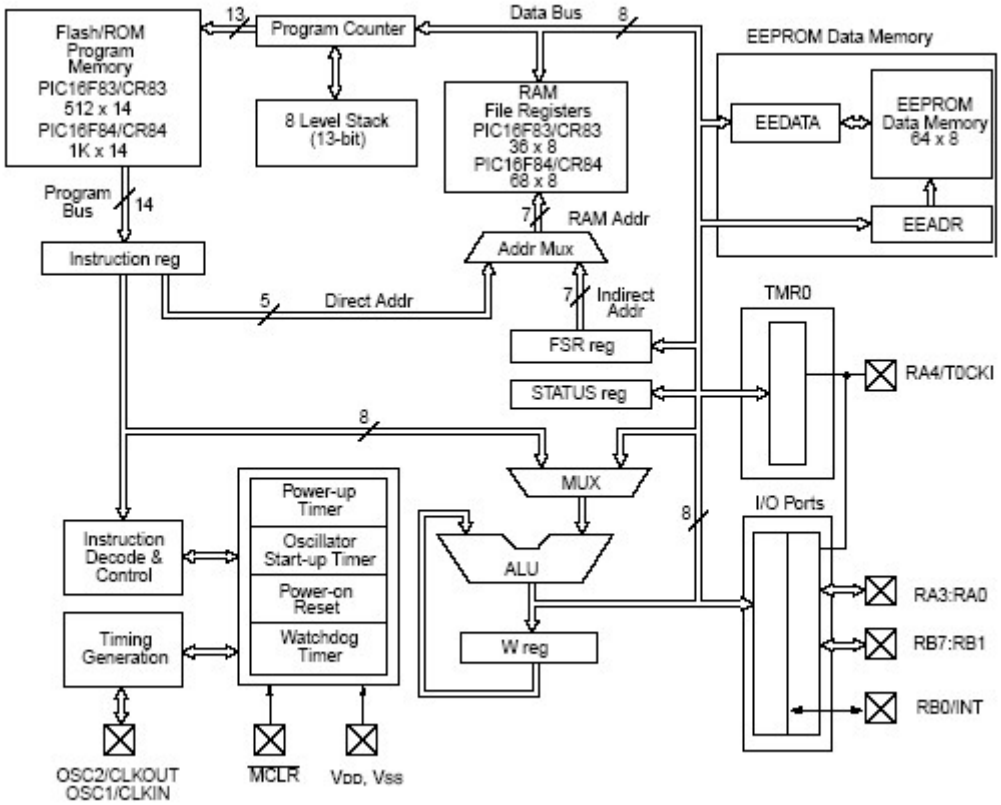
Valore prescaler			Rapporto divisione Counter/timer	Rapporto divisione Watchdog/timer
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Differenza tra pic16f84 e pic16f628

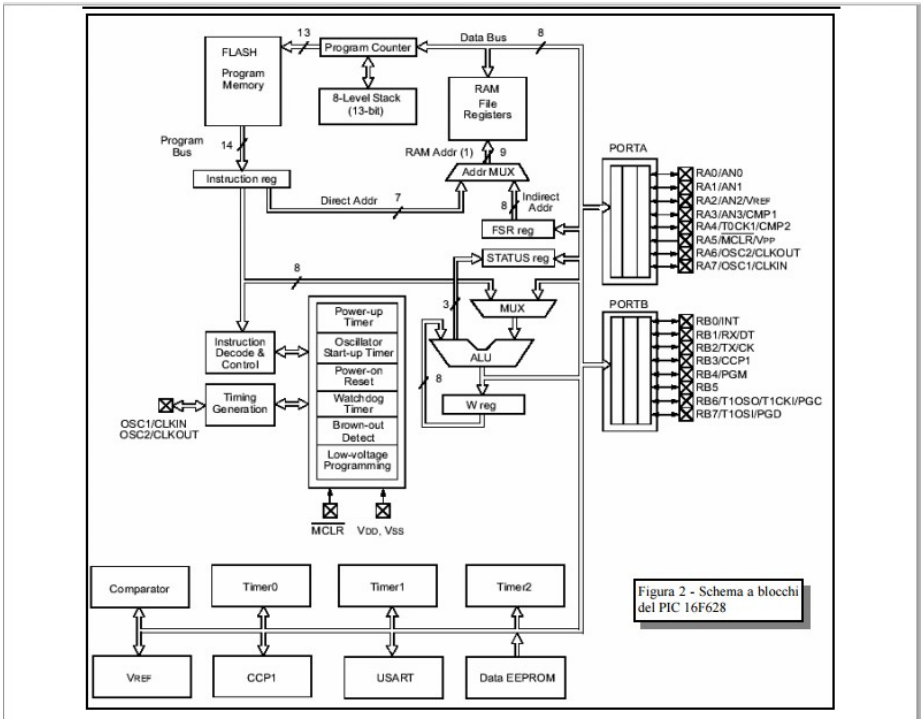
Pic16f84	Pic16f628
1024x14 program memory	2048x14 program memory
64x8 RAM memory	224x8 RAM memory
64x8 memory data	128x8 memory data



Architettura del pic 16f84



Architettura del pic 16f628



Registri del pic 16f84

Address	Bank 0	Bank 1	Address
00h	INDF	←	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	←	82h
03h	STATUS	←	83h
04h	FSR	←	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	Unimplemented	←	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	←	8Ah
0Bh	INTCON	←	8Bh
0Ch - 4Fh	GPR	←	8Ch - CFh

Registri del pic 16f628