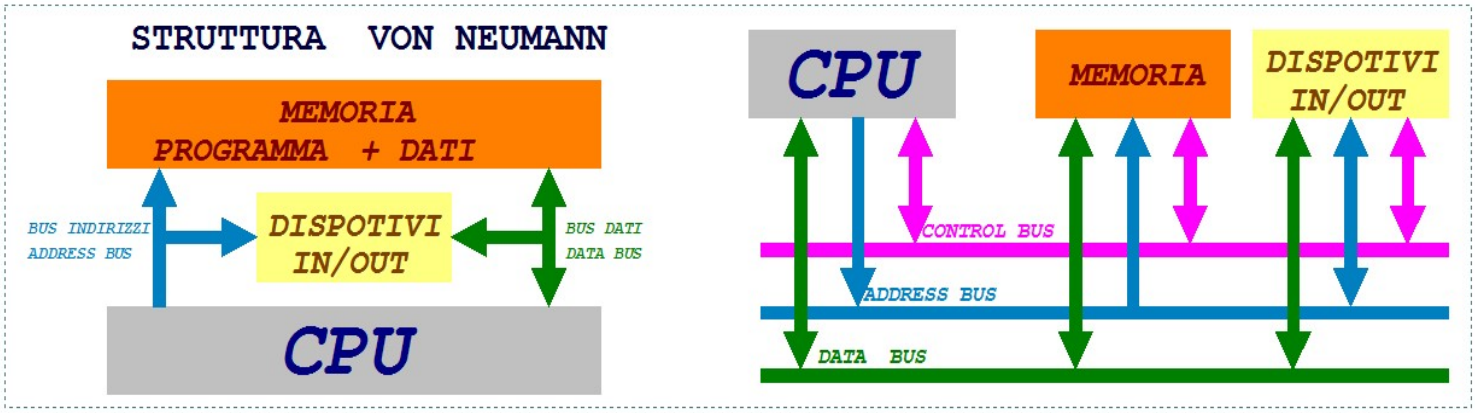
MICROCONTROLLORI PIC – STRUTTURA E REGISTRI

Con i precedenti tutorial, abbiamo visto cos'è un microcontrollore, ed abbiamo visto come scrivere un programma in assembler o in inguaggio C utilizzando gli ambienti di sviluppo della Microchip e della Mikroelektronika.
Con le idee un po' più chiare andiamo ora a guardare al suo interno, per comprendere meglio la sua struttura..

ARCHITETTURA INTERNA

La classica architettura degli elaboratori è quella descritta nel modello di **Von Neumann**, ideata dall'omonimo matematico, descritta sinteticamente nel seguente schema a blocchi:

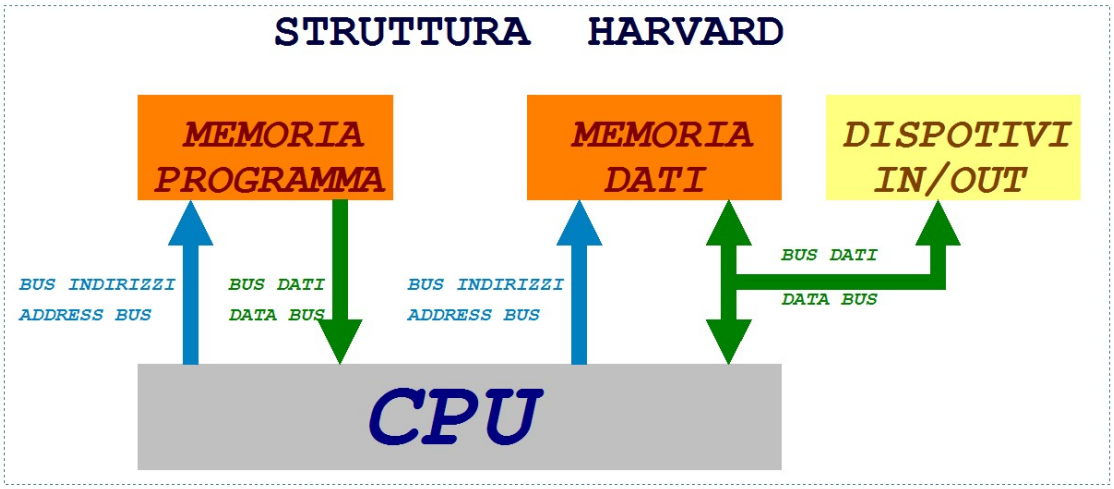


Come si può vedere nell'immagine, la memoria che contiene il programma contiene anche i dati, ed i bus di comunicazione sono comuni a tutto il sistema. I bus di comunicazione rappresentano l'insieme dei collegamenti che consentono alla CPU di comunicare con gli altri dispositivi.

- Il **data-bus**, è l'insieme di collegamenti dove transitano i dati.
- L'**address-bus**, è l'insieme di collegamenti che consente alla CPU di scegliere dove scrivere o leggere il dato.
- Il **control-bus**, è l'insieme di collegamenti che consente alla CPU di selezionare il tipo di operazione da eseguire,scrittura, lettura o operazioni sui dispositivi periferici.

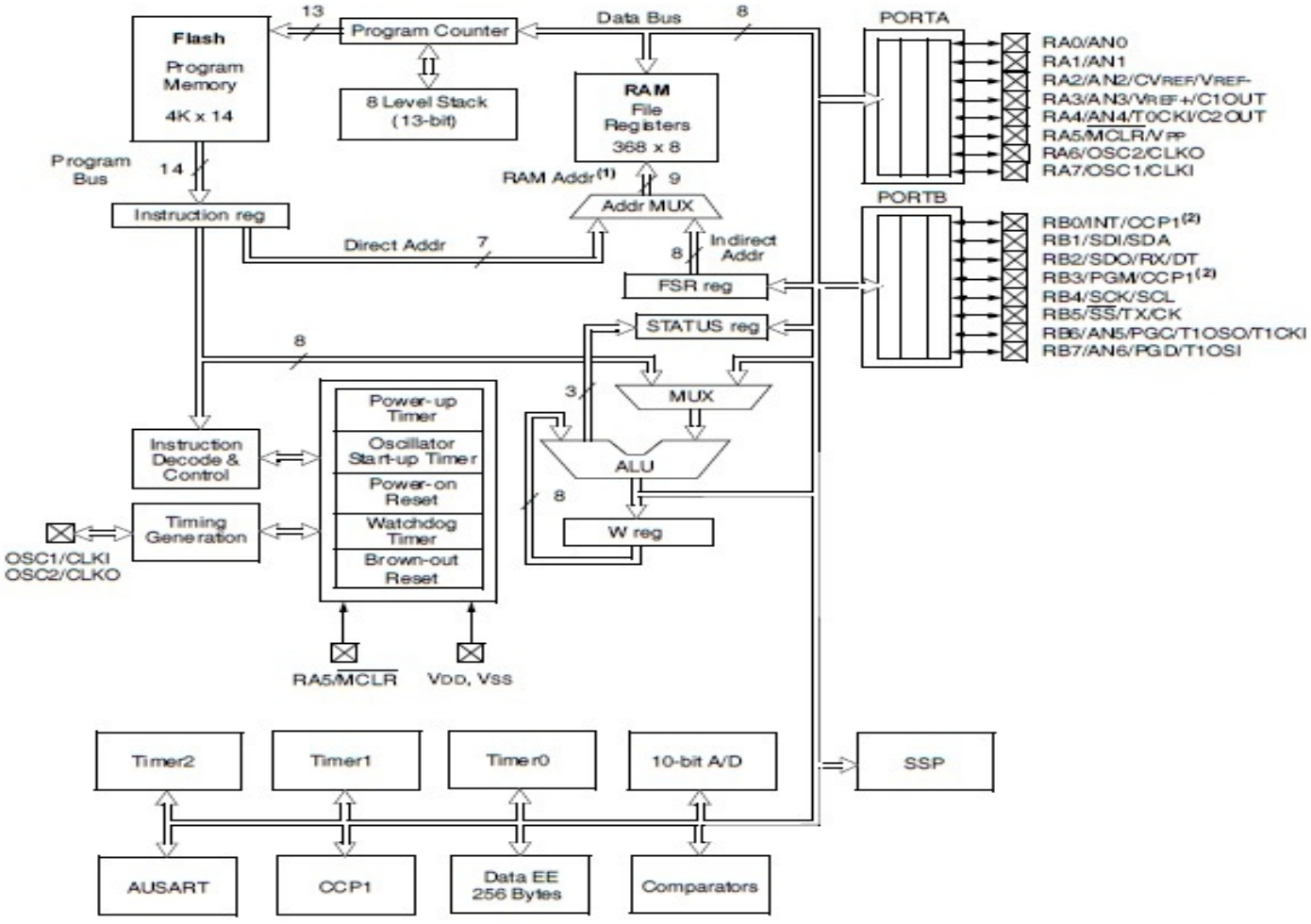
In questa architettura la CPU (**C**entral **P**rocessing **U**nit) ad ogni istruzione esegue due fasi distinte, la fase di prelievo dell'istruzione dalla memoria (fase di **fetch**) e dopo averla decodificata esegue l'istruzione (fase di **execute**).
Nell'architettura Von Neumann le due fasi non possono sovrapporsi, perché le istruzioni ed i dati si trovano sulla stessa memoria, perciò la fase di fetch ed execute vengono eseguite sequenzialmente.
Questo tipo di architettura viene utilizzata nei normali elaboratori, in quanto offre una maggiore flessibilità visto che la memoria è unica e condivisa per il programma e per i dati.

Per applicazioni più specifiche, come nel caso dei microcontrollori, si preferisce utilizzare l'architettura **Harvard**, dove la memoria programma e la memoria dati sono differenti e comunicano con la CPU con bus differenti.



A differenza dell'architettura Von Neumann, la fase di fetch e di execute possono sovrapporsi perché la CPU opera su bus e su memorie differenti. Ogni singola istruzione potrà essere eseguita così con un solo ciclo di clock. Di contro quest'architettura è circuitalmente più complessa e meno flessibile, in quanto le dimensioni delle due memorie sono fisse. Come in molti altri microcontrollori perciò il PIC utilizza l'architettura Harvard.

Di seguito la struttura interna del PIC16F88 estratta dal data-sheet ufficiale della microchip.



Prima di descrivere la struttura interna del pic vediamo di seguito una sintesi delle sue caratteristiche.

Come si può vedere al suo interno ci sono anche altri dispositivi hardware come convertitori A/D, PWM, Comparatori e Timers.

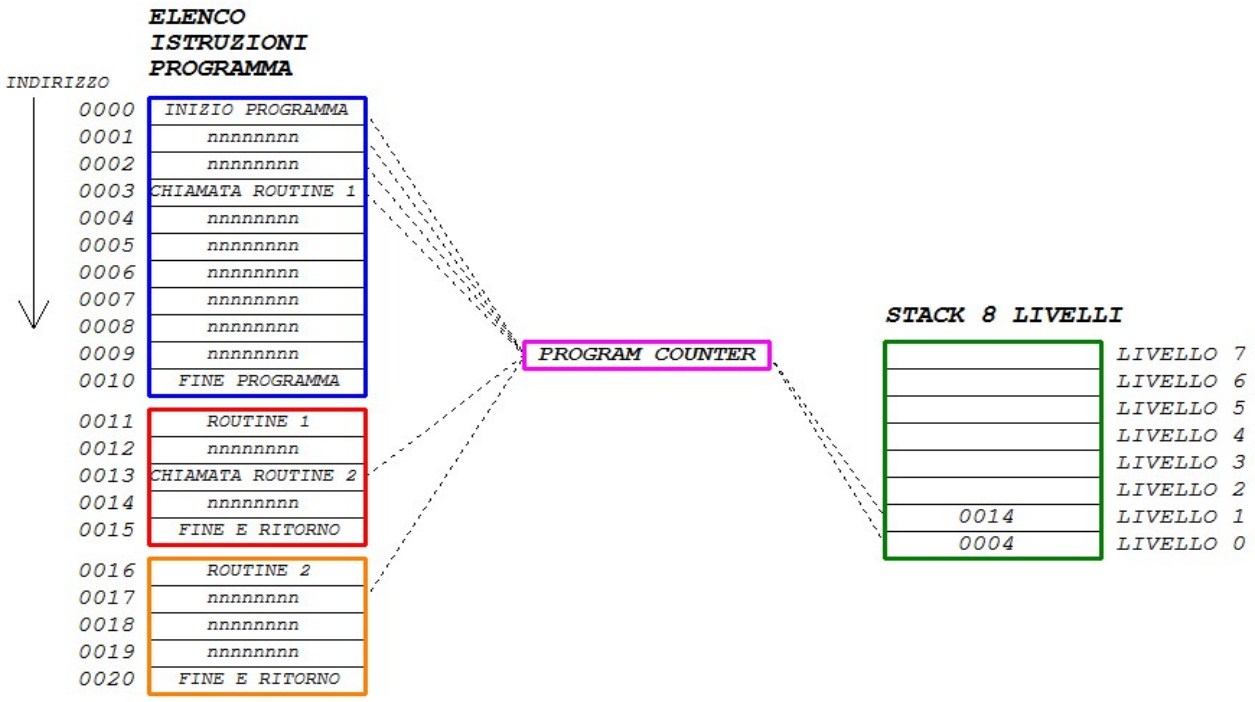
Device	Program Memory		Data Memory		I/O Pins	10-bit A/D (ch)	CCP (PWM)	AUSART	Comparators	SSP	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)							
PIC16F87	7168	4096	368	256	16	N/A	1	Y	2	Y	2/1
PIC16F88	7168	4096	368	256	16	1	1	Y	2	Y	2/1

All'interno possiamo distinguere i seguenti dispositivi:

- In alto a destra la **FLASH MEMORY**, cioè la memoria che contiene il programma, organizzata in questo caso con 4096 istruzioni da 14 bit (nel PIC16F88 un'istruzione sfrutta 14 bit)

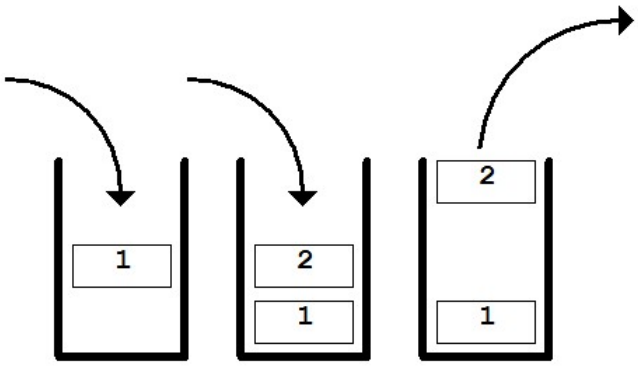
  La flash è una memoria non volatile (mantiene il contenuto in assenza di alimentazione) cancellabile elettricamente. A differenza della EEPROM (che è sempre una memoria non volatile cancellabile elettricamente) la flash non opera sul singolo byte ma su interi blocchi di memoria.
- Vicino alla memoria flash, c'è il **PROGRAM COUNTER**, cioè un registro a 13 bit che contiene l'indirizzo dell'istruzione che la CPU deve eseguire. Tutti i registri sono contenuti nell'area di memoria volatile cioè la RAM, strutturata con diversi registri e con uno spazio libero.

  Il program counter è un registro a 13 bit perché deve indirizzare l'intera memoria flash che occupa 7168 bytes (4096 istruzioni da 14 bit occupano 7168 bytes) in un sistema binario con 13 bit si possono indirizzare fino a $2^{13}=8192$ bytes, più che sufficienti per controllare l'intera memoria di programma.
- Sotto al program counter, c'è lo **STACK** (tadotto "la pila") cioè quello spazio dove può essere salvato il valore del program counter, quando si esegue una chiamata ad una subroutine.



Lo stack può contenere 8 livelli, pertanto si possono eseguire al massimo 8 chiamate ad una subroutine.

Durante l'esecuzione del programma, se viene chiamata una routine, il valore del program counter, viene salvato nello stack, dopo che la routine è stata eseguita, viene ripristinato il precedente valore del program-counter, con un meccanismo chiamato **L.I.F.O. (Last In First Out)** l'ultimo valore del program counter che entra nello stack è il primo ad uscire.



Vediamo il contenuto del PC e dello STACK durante la simulazione in MPLAB del programma di esempio quando viene chiamata la subroutine Delay.

- ✔ *In questo momento si deve elaborare l'istruzione **movlw 0.***
  Il program counter contiene 0x10E cioè l'indirizzo della riga di memoria che contiene l'istruzione,
  e lo stack è vuoto.

PC: 0x10E

Stack Level	Return Address
	0x00
0	0000
1	0000
2	0000
3	0000
4	0000
5	0000
6	0000
7	0000

```
        movlw   0
        movwf   PORTB
        bsf     PORTB,LED

        ;loop del programma
MainLoop
        call    Delay
        btfsc   PORTB,LED
        goto    SetToZero
        bsf     PORTB,LED
        goto    MainLoop
SetToZero
        bcf     PORTB,LED
        goto    MainLoop

;*****************************
        ;SUBROUTINE Delay

Delay
        clrf    Count
        clrf    Count+1
DelayLoop
        decfsz  Count,1
        goto    DelayLoop
        decfsz  Count+1,1

        goto    DelayLoop
        return
```

- ✔ *Stessa cosa per il passo successivo il program counter contiene l'indirizzo di memoria successivo al precedente cioè **0x10F** dove si trova l'istruzione **movwf PORTB** e lo stack è vuoto.*

PC: 0x10F

Stack Level	Return Address
	0x00
0	0000
1	0000
2	0000
3	0000
4	0000
5	0000
6	0000
7	0000

```
        movlw   0
        movwf   PORTB
        bsf     PORTB,LED

        ;loop del programma
MainLoop
        call    Delay
        btfsc   PORTB,LED
        goto    SetToZero
        bsf     PORTB,LED
        goto    MainLoop
SetToZero
        bcf     PORTB,LED
        goto    MainLoop

;*****************************
        ;SUBROUTINE Delay

Delay
        clrf    Count
        clrf    Count+1
DelayLoop
        decfsz  Count,1
        goto    DelayLoop
        decfsz  Count+1,1

        goto    DelayLoop
        return
```

✔ *Ancora una volta il program counter aumenta e lo stack rimane vuoto, perché non c'è alcuna chiamata ad una subroutine.*

PC: 0x110

```
        movlw       0
        movwf       PORTB
        bsf         PORTB,LED

        ;loop del programma
MainLoop
        call        Delay
        btfsc       PORTB,LED
        goto        SetToZero
        bsf         PORTB,LED
        goto        MainLoop
SetToZero
        bcf         PORTB,LED
        goto        MainLoop

;*****************************
        ;SUBROUTINE Delay

Delay
        clrf        Count
        clrf        Count+1
DelayLoop
        decfsz      Count,1
        goto        DelayLoop
        decfsz      Count+1,1

        goto        DelayLoop
        return
```

Stack Level	Return Address
	0x00
0	0000
1	0000
2	0000
3	0000
4	0000
5	0000
6	0000
7	0000

✔ *L'istruzione che il program counter punta in memoria ora, è una chiamata ad una subroutine, lo stack è ancora vuoto ed il program counter contiene l'indirizzo* ***0x111.***

PC: 0x111

```
        movlw       0
        movwf       PORTB
        bsf         PORTB,LED

        ;loop del programma
MainLoop
        call        Delay
        btfsc       PORTB,LED
        goto        SetToZero
        bsf         PORTB,LED
        goto        MainLoop
SetToZero
        bcf         PORTB,LED
        goto        MainLoop

;*****************************
        ;SUBROUTINE Delay

Delay
        clrf        Count
        clrf        Count+1
DelayLoop
        decfsz      Count,1
        goto        DelayLoop
        decfsz      Count+1,1

        goto        DelayLoop
        return
```

Stack Level	Return Address
	0x00
0	0000
1	0000
2	0000
3	0000
4	0000
5	0000
6	0000
7	0000

✔ *Ora il programma salta alla subroutine* Delay*, ed il program counter conterrà l'indirizzo della prima istruzione della subroutine cioè* ***0x118.***

Il primo livello dello stack, conterrà ora l'indirizzo **0x112** *cioè la riga del programma successiva all'istruzione* ***call Delay.*** *Al termine dell'esecuzione della subroutine,il programma continuerà con l'istruzione* ***btfsc PORTB,LED*** *ed il program counter conterrà* ***0x112.***

Stack Level	Return Address
	0x01
0	0112
1	0000
2	0000
3	0000
4	0000
5	0000
6	0000
7	0000

PC: 0x118

```
        movlw       0
        movwf       PORTB
        bsf         PORTB,LED

        ;loop del programma
MainLoop
        call        Delay
        btfsc       PORTB,LED
        goto        SetToZero
        bsf         PORTB,LED
        goto        MainLoop
SetToZero
        bcf         PORTB,LED
        goto        MainLoop

;*****************************
        ;SUBROUTINE Delay

Delay
        clrf        Count
        clrf        Count+1
DelayLoop
        decfsz      Count,1
        goto        DelayLoop
        decfsz      Count+1,1

        goto        DelayLoop
        return
```

- Ancora a sinistra dello Stack in alto, troviamo la **RAM** definita **File register**, composta da 368 byte, contenente i registri, cioè locazioni di RAM con un funzionamento predefinito, e lo spazio libero utilizzabile per le variabili di programma.

- Sotto alla memoria flash troviamo l'**INSTRUCTION REGISTER**, cioè il registro che dopo la fase di fetch (prelievo dell'istruzione) contiene il codice da decodificare ed eseguire.

- Sotto alla RAM troviamo invece il registro **FSR** che si utilzza per l'indirizzamento indiretto, insieme al multiplexer degli indirizzi. Il registro FSR lavora insieme al registro IND che si trova nel File register.
  Il valore scritto in IND va a finire nella locazione indirizzata dal contenuto di FSR.

- Ancora sotto, troviamo il registro di **STATUS** descritto nei precedenti tutorial, contenente i vari flag.

- A destra in alto, troviamo le due porte di ingresso/uscita, **PORTA** e **POTRB**.

- Ripartendo al centro da sinistra, troviamo il blocco per la decodifica delle istruzioni, **Instruction decode and control**.

- Al centro troviamo il gruppo di circuiti ausiliari:
  - **Power Up timer** - *Timer per ritardo dell'avvio del programma dopo l'accensione.*
  - **Oscillator Start-up Timer** - *Timer per controllo della stabilità dell'oscillatore.*
  - **Power on Reset** - *Mantiene il pic in uno stato di hold fino a quando non si raggiunge una condizione stabile sulla tensione e sul clock.*
  - **Watchdog Timer** - *Timer per resettare il microcontrollore in modo da evitare situazioni di stallo, definite in gergo deadlock.*
  - **Brown out Reset** - *Circuito che controlla il valore della tensione di alimentazione, per resettare il microcontrollore quando essa scende sotto a determinati valori.*

- A destra di questo gruppo di circuiti, troviamo la **A.L.U.** (**U**nità **A**ritmetico **L**ogica) ed il registro **accumulatore W**, un registro dove transitano tutti i valori ed i risultati delle operazioni.

- In basso troviamo i **Timer** interni, il convertitore **Analogico-Digitale**, il circuito hardware **AUSART** per il controllo della comunicazione seriale **RS232**, il circuito **CCP** per la gestione del **PWM**, i **comparatori**, il circuito per il controllo della comunicazione seriale sincrona **SSP** (**S**ynchronous **S**erial **P**ort) per gestire la comunicazione I2c e SPI e la EEPROM con i suoi 256byte di dimensione.

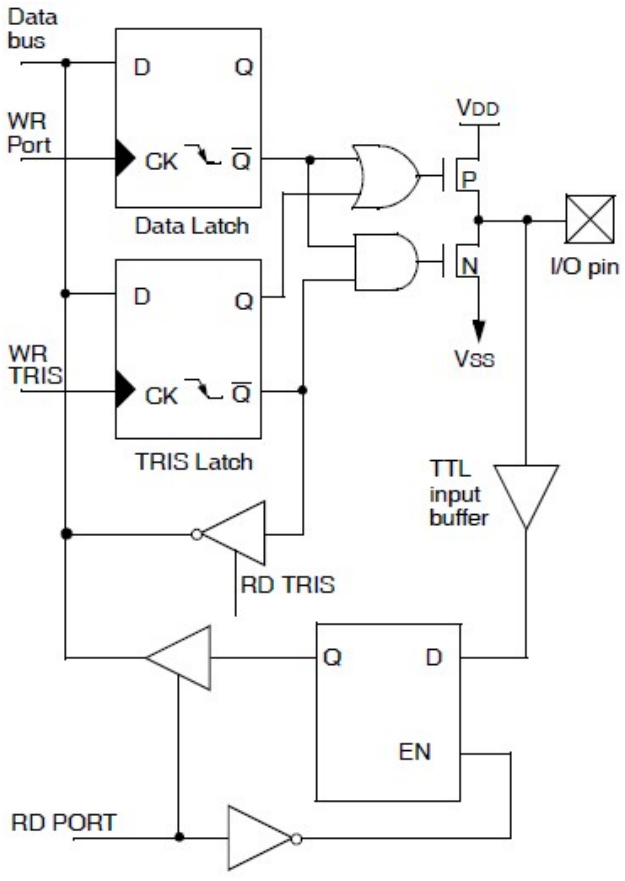
HARDWARE INTERNO

Al'interno di ogni microcontrollore, ci sono diversi circuiti hardware, il primo da analizzare anche se in maniera sintetica, è quello relativo ai pin di ingresso ed uscita. Come abbiamo visto precedentemente, tutti i pin sono multifunzione, e possono essere configurati mediante dei registri. Nel caso del PIC16F88 possono essere oltre che ingressi o uscite digitali, anche collegati a dei comparatori, convertitori, o addirittura avere funzioni più specifiche, come nel caso delle **C.I.P.** (**C**ore **I**ndependent **P**eripherials) dei circuiti che svolgono la loro funzione in maniera indipendente dal programma, consentendo così maggiori prestazioni. Uno dei PIC ad 8 bit più dotato di C.I.P. è il **PIC16F1619** che vedremo successivamente, ma se andiamo nelle famiglie superiori della serie 18F o se andiamo nella categoria dei **DSPIC**, troveremo delle C.I.P. davvero interessanti, di seguito qualche esempio:

- **Convertitori A/D**, convertitore analogico/digitale.
- **Comparatori**, per confrontare il valore in tensione
- **PWM** (**P**ulse **Wi**dth **M**odulator) generatore di impulsi a duty cicle variabile.
- **CLC** (**C**onfigurable **L**ogic **C**ell) cella con porte logiche, configurabile.
- Seriale sincrona ed asincrona, **I2C, SPI, USART RS232, RS485, CANBUS.**
- **AT** (**A**ngular **T**imer) generazione di impulsi sfasati.
- **CRC** (**C**yclic **R**edundancy **C**heck) controllo errori.
- Acceleratore matematico con **PID** (**P**roportional **I**ntegral **D**erivative) controllo con feeback di tipo PID.
- **QEI** (**Q**uadrature **E**ncoder **I**nterface) interfaccia per encoder incrementali.

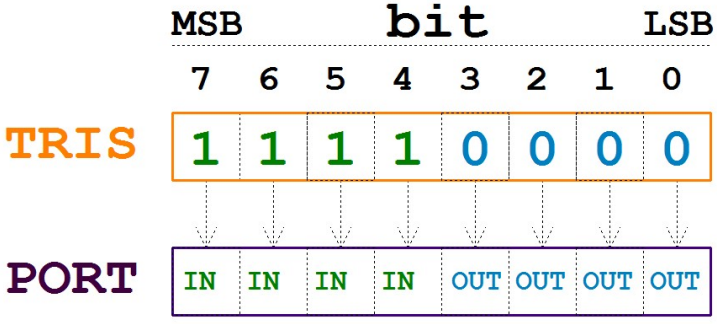
Queste sono solo alcune delle periferiche integrate nei vari microcontrollori, e la loro gestione viene effettuata sempre tramite i registri dedicati presenti nel file register.

Tornando alle porte di input/output, invece di analizzare il PC16F88, facciamo un piccolo passo indietro e vediamo il circuito presente nelle porte del vecchio PIC16F84, dotato di molte meno funzioni. Questo per avere un'idea del meccanismo che regola la configurazione dei GPIO.



Proviamo ad analizzare ad esempio il funzionamento dei registri TRIS e PORT.

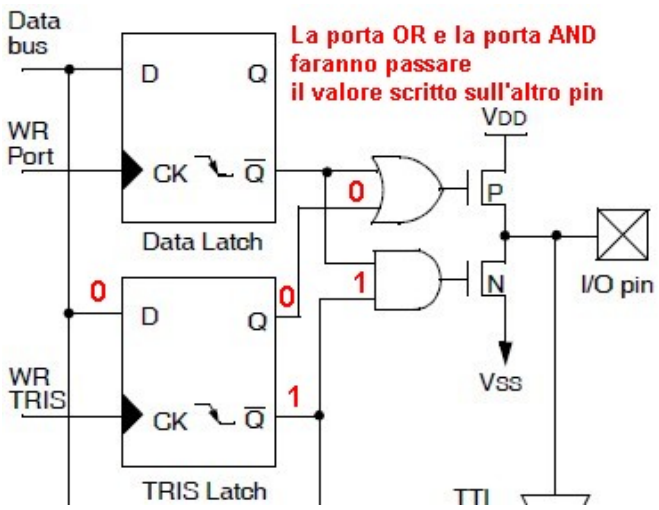
Ricordiamo il funzionamento del registro TRIS, che configura i rispettivi pin della corrispondente PORT come input o output.

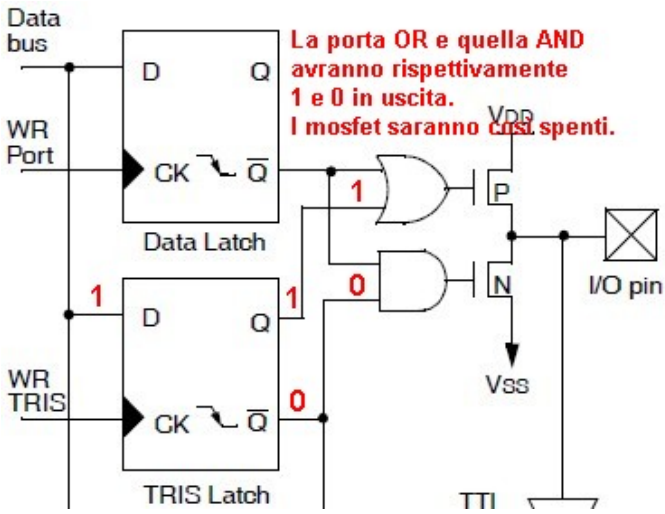


In pratica se scriviamo 0 in uno dei bit di TRIS, avremo un'uscita nel corrispondente bit della relativa PORT.

Se scriviamo 1 in uno dei bit di TRIS, avremo un ingresso nel corrispondente bit della relativa PORT.

Nel circuito a destra vediamo infatti che il valore scritto sul data bus viene portato sulle uscite di uno dei due flip-flop di tipo D, a seconda se scriviamo sul registro TRIS (flip-flop in basso) o se scriviamo sul registro PORT (flip-flop in alto). Se ad esempio scriviamo 0 nel registro TRIS nell'uscita Q del flip-flop in basso troveremo 0 e sull'uscita Q negata troveremo 1. Questi due valori andranno rispettivamente nell'ingresso della porta OR ed AND, in modo che il valore che verrà scritto su PORT presente in maniera invertita sull'uscita del flip-flop in alto attiverà uno dei due mosfet a canale P e vogliamo 1 in uscita a canale N se vogliamo 0. Se invece nel registro TRIS scriviamo 1, le due uscite saranno invertite, e le porte OR and AND non consentiranno il passaggio del segnale. In questo caso il valore scritto sul pin lo troveremo sull'uscita del latch in basso quando verrà effettuata la lettura (RD PORT). Di seguito le due situazioni.





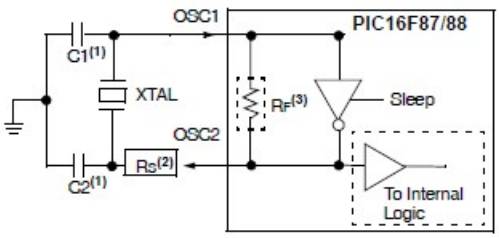
OSCILLATORE E CLOCK

Torniamo ora nel data-sheet del nostro PIC16F88, ed andiamo ad analizzare qualche altro circuito, come ad esempio il circuito che fa capo ai due piedini OSC1 ed OSC2.

Questi due pin, possono essere utilizzati per collegare un circuito oscillatore esterno a quarzo o con una rete RC, qualora non venissero utilizzati in questo modo, possono essere dei pin rispettivamente di input ed ouput (vedi dispensa introduttiva). In quest'ultimo caso, il clock può essere generato da un oscillatore interno e possono essere impostati i seguenti valori; 31kHz, 125kHz, 250kHz, 500kHz, 1MHz, 2MHz, 4MHz ed 8MHz.

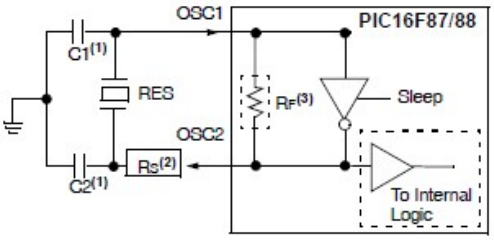
Le possibili impostazioni di clock nel PIC 16F88 sono le seguenti:

- **LP** – (Low Power Crystal) cristalli con frequenza 32kHz
- **XT** – Cristalli con frequenze comprese tra 200kHz e 4MHz
- **HS** – Cristalli con frequenze comprese tra 4MHz e 20MHz



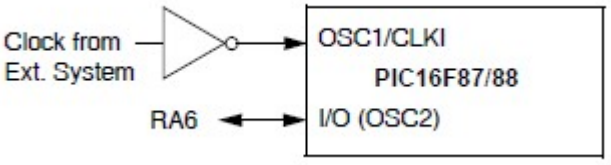
Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
LP	32 kHz	33 pF	33 pF
XT	200 kHz	56 pF	56 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15 pF	15 pF
	20 MHz	15 pF	15 pF

Per l'impostazione XT ed HS, è possibile anche utilizzare un risonatore ceramico anziché un quarzo, in questo caso il circuito di uscita è lo stesso, ma i valori consigliati sono differenti.

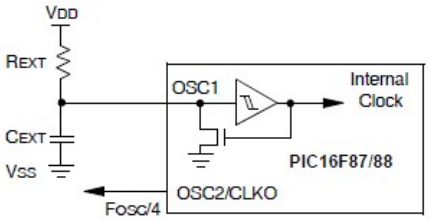


Typical Capacitor Values Used:			
Mode	Freq	OSC1	OSC2
XT	455 kHz 2.0 MHz 4.0 MHz	56 pF 47 pF 33 pF	56 pF 47 pF 33 pF
HS	8.0 MHz 16.0 MHz	27 pF 22 pF	27 pF 22 pF

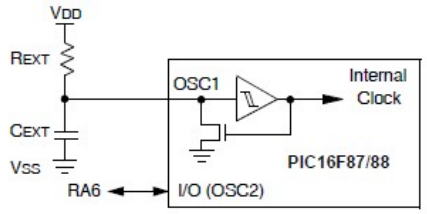
- **EXTCLK RA6 come I/O**, in questo caso viene utilizzato solo il pin OSC1 cioè CLKI, dove è possibile collegare un clock proveniente da un altro sistema, in questo caso, sarà possibile utilizzare il pin OSC2 come un uscita.



- **EXTRC RA6 come CLKO**, il clock viene generato da una rete RC esterna ed il pin OSC2 (RA6) fornirà il clock in uscita.

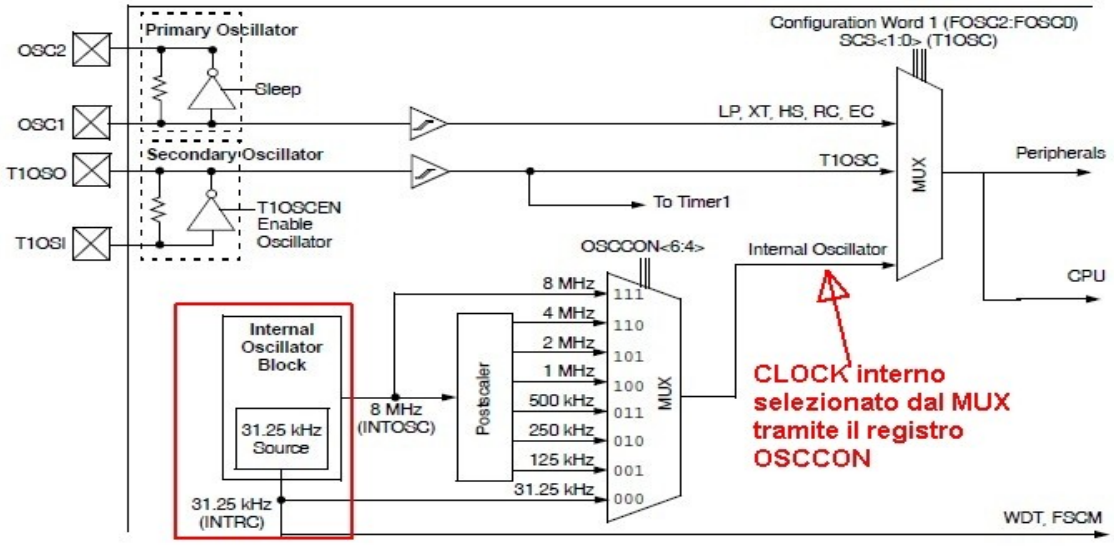


- **EXTRC RA6 come I/O**, come il precedente caso, ma il pin RA6, sarà un'uscita digitale.



- **INTRC RA6 come CLKO**, in questo caso il clock viene generato partendo da un oscillatore interno da 31,25kHz. Questo clock andrà in ingresso ad un multiplexer per generare differenti frequenze, impostabili dal registro OSCCON con i bit 4,5 e 6. Il bit RA6 fornirà in questo caso il clock in uscita, mentre il bit RA7 potrà essere utilizzato come ingresso digitale.

- **INTRC RA6 come I/O**, come nel precedente caso, ma sia il bit RA6 che il bit RA7 saranno rispettivamente un'uscita ed un ingresso digitali. Di seguito lo schema interno dell'oscillatore.

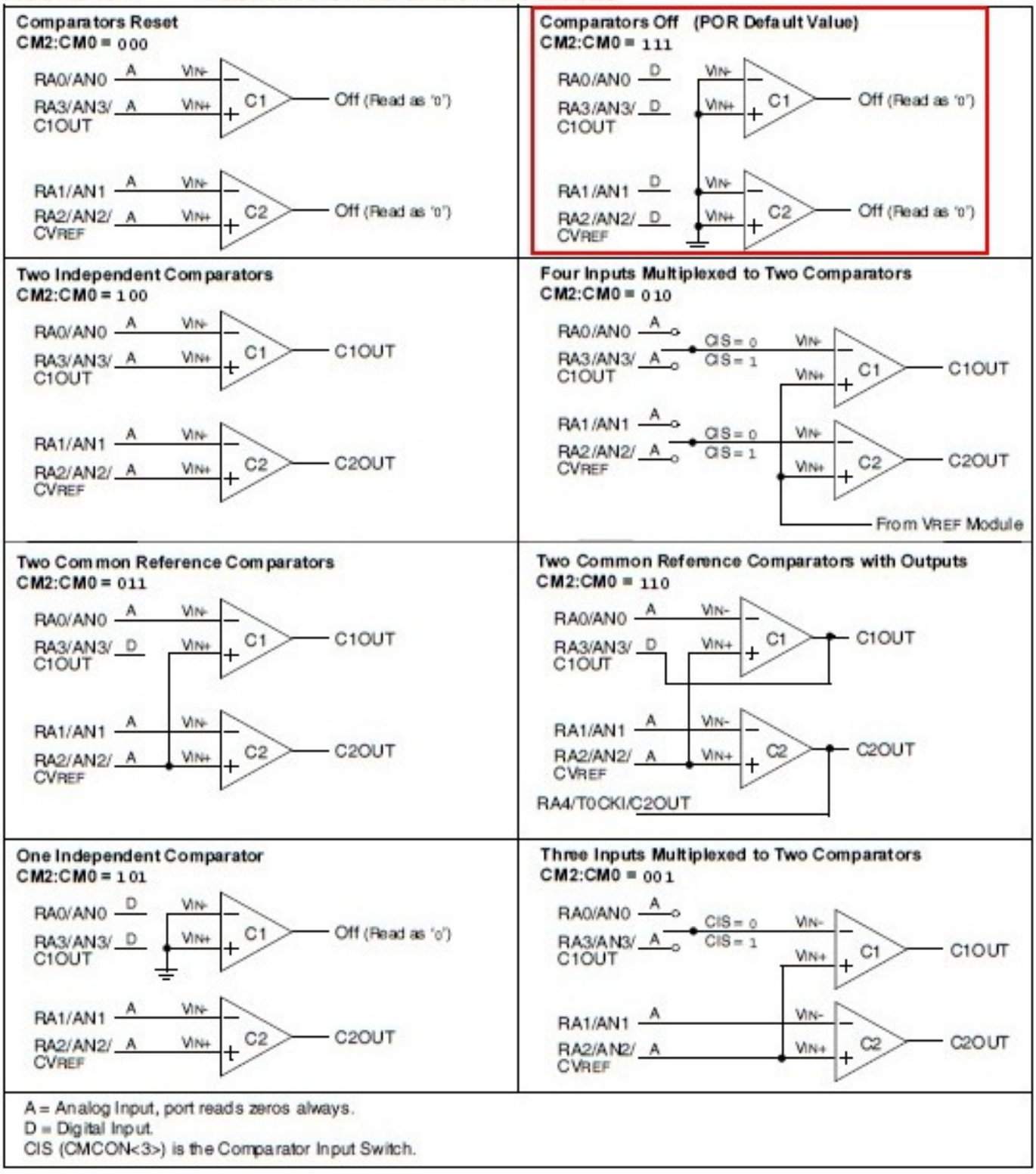


COMPARATORI

Internamente al PIC16F88, ci sono dei comparatori che possono essere collegati in differenti modalità.

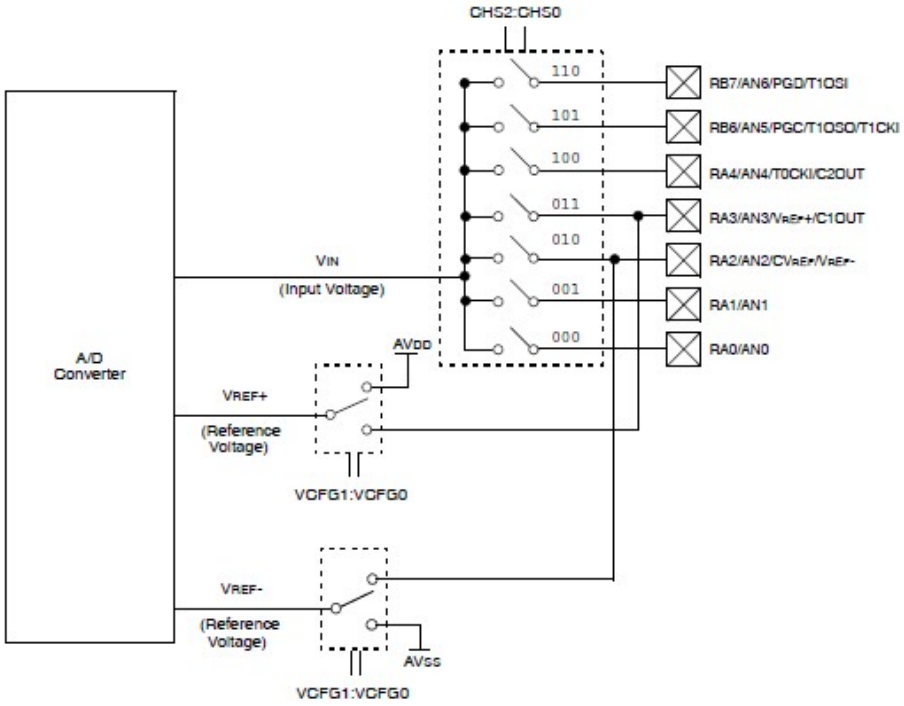
Le possibili impostazioni sono selezionabili tramite il registro CMCON, quella evidenziata è quella che disabilita i comparatori. Perciò se si vogliono utilizzare i bit della PORTA come ingressi/uscite bisogna scrivere nel registro CMCON il valore 0x07 (esadecimale).

FIGURE 13-1: COMPARATOR I/O OPERATING MODES



CONVERTITORE A/D

Internamente al PIC c'è un convertitore A/D a 10 bit (1024 combinazioni). Anche in questo caso, la configurazione dei bit e del convertitore avviene tramite i relativi registri ADCON0, ADCON1 ed ANSEL.



Senza entrare nei dettagli delle impostazioni relative alla conversione, vediamo solamente come impostare i pin come ingresi analogici o digitali. Il registro interessato è ANSEL, dove sono presnti 7 bit dove è possibile scrivere 0 se si vuole che il relativo pin sia un ingresso/uscita digitale, ed 1 se invece il pin si utilizzato come ingresso analogico.

U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0

bit 7 **Unimplemented**: Read as '0'

bit 6-0 **ANS**<6:0>: Analog Input Select bits
Bits select input function on corresponding AN<6:0> pins.
1 = Analog I/O(1,2)
0 = Digital I/O

Ci sono altri dispositivi hardware interni, come ad esempio il TIMER, il crcuito per il PWM, la seriale asincrona RS232, le seriali sincrone SPI ed I2C.

Inoltre come in tutti i microcontrollori, c'è il circuito di watchdog, ed ovviamente il circuito di interrupt.

Ogni microcontrollore anche della stessa famiglia, può presentare delle differenze, e vista la complessità dei circuiti hardware interni, si consiglia in ogni caso di leggere il datasheet.

Approfondiremo comunque questi temi ed i relativi circuiti quando verranno utilizzati nei successivi tutorial.

L'importante è comprendere i vantaggi che tutti questi dispositivi interni possono offrire in termini di prestazione, in quanto offrono un valido supporto alla parte software.

Terminiamo così questa prima introduzione alla struttura hardware del microcontrollore.