SOMMARIO

- Introduzione
- Architetture: Von Neumann vs. Harvard, RISC vs. CISC
- PIC: Struttura Interna
- PIC: Organizzazione della Memoria
- PIC: Clock e Timing
- PIC: Gestione Interrupt
- PIC: Descrizione delle Periferiche
- PIC: Watchdog Timer e Sleep Mode

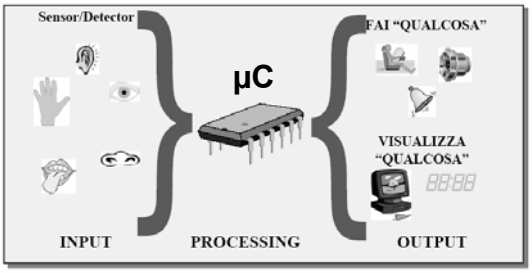
INTRODUZIONE

DEFINIZIONI

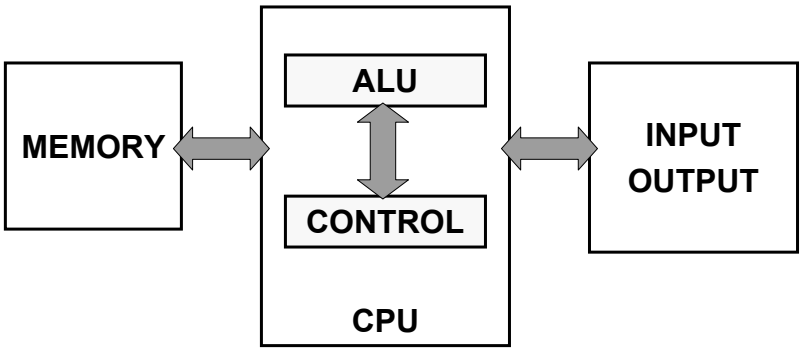
MICROCONTROLLORE (µC): è un microprocessore che opera come un sistema *embedded*

SISTEMA EMBEDDED: è un sistema di elaborazione (computer) specializzato, integrato in un dispositivo fisico in modo da controllarne le funzioni tramite un apposito programma software dedicato.

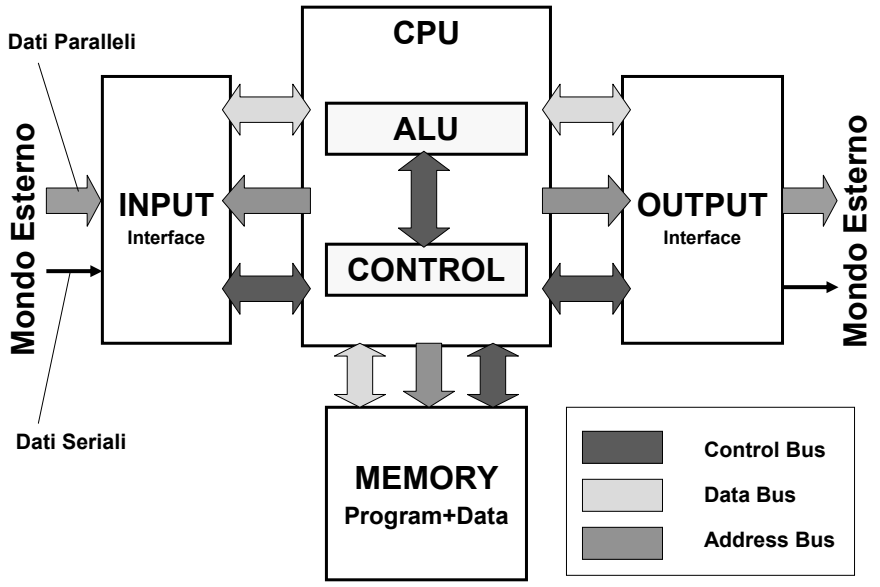
Es: cellulari, ABS, controllo airbag, home automation, dispositivi di automazione di macchine industriali, ecc…



ARCHITETTURA GENERALE DI UN CALCOLATORE



ARCHITETTURA GENERALE DI UN CALCOLATORE



COS'E' UN BUS ?

Si indica con il termine bus l' INSIEME DEI COLLEGAMENTI FISICI tra CPU e ogni altro blocco funzionale presente nel calcolatore (memoria, periferiche, dispositivi di I/O, ecc…).

Su un bus possono circolare:

- **DATI**
- **INDIRIZZI di LOCAZIONI di MEMORIA**
- **SEGNALI di CONTROLLO**

QUALI BUSES SONO PRESENTI IN OGNI CALCOLATORE?

CONTROL BUS: è l'insieme di tutti i segnali di controllo scambiati dalla CPU con i dispositivi di memoria, di elaborazione dati, di input/output che coordinano e sincronizzano le attività dell'intero sistema.

ADDRESS BUS: trasporta gli indirizzi (univoci) corrispondenti ad una locazione di memoria o a un dispositivo di input/output.

DATA BUS: trasporta dati memorizzati in una locazione di memoria (indirizzata preventivamente grazie al bus indirizzi) alla CPU o viceversa.

CPU - FUNZIONI e STRUTTURA

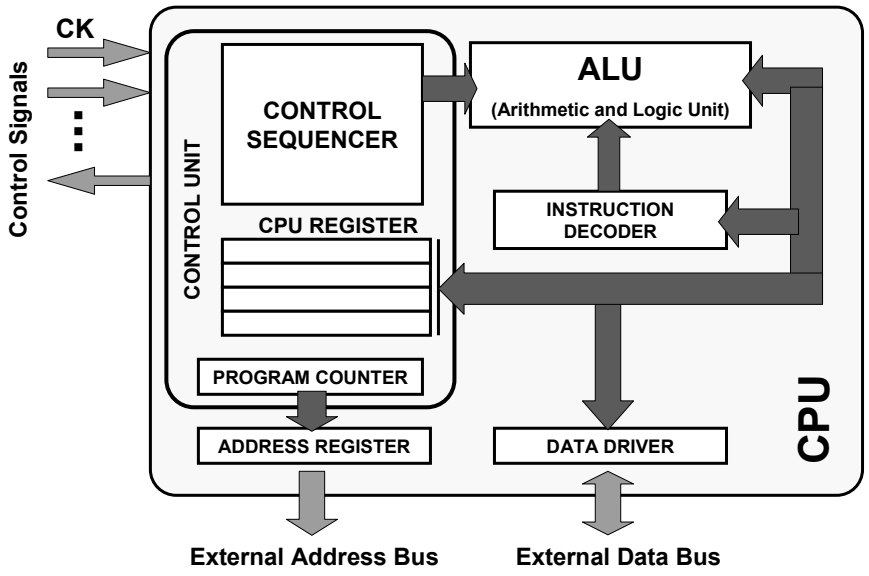
Funzioni principali di una CPU sono:

- **Trasferimento Dati**
- **Controllo di Flusso**
- **Elaborazioni Aritmetiche e Logiche**
  (Addizioni e Sottrazioni, AND, OR, XOR, NOT Incrementi, Decrementi, Shift, Clear, ecc…)

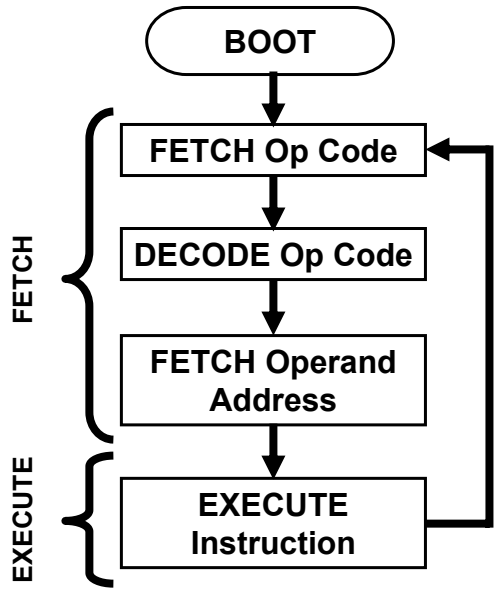
Ogni CPU ha un *array register* con almeno:

- **Un Registro ACCUMULATORE (W)**
- **Il PROGRAM COUNTER (PCL)**
- **L'INSTRUCTION REGISTER (IR)**
- **Lo STACK POINTER (SP)**

CPU BLOCK DIAGRAM



CPU - COME ESEGUE UN'ISTRUZIONE ?



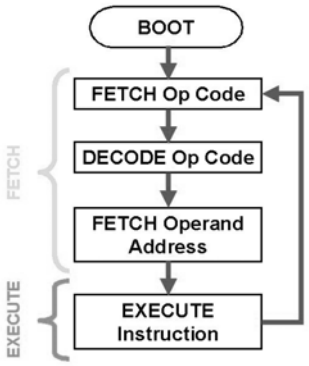
L'esecuzione di un'istruzione da parte della CPU è divisa in 2 fasi:

- **Fase di FETCH**
- **Fase di EXECUTE**

CPU - COME ESEGUE UN'ISTRUZIONE ?

Fase di FETCH

- **La CPU carica sull' address bus l'indirizzo dell'istruzione da eseguire**
- **L'indirizzo caricato è fornito dal Program Counter (PC), registro allocato nella Control Unit della CPU**
- **Sul control bus ci sono le informazioni per leggere la locazione di memoria il cui indirizzo è sull'address bus, mentre sul data bus vengono caricati i dati dalla locazione di memoria contenuta nell'instruction register (IR)**

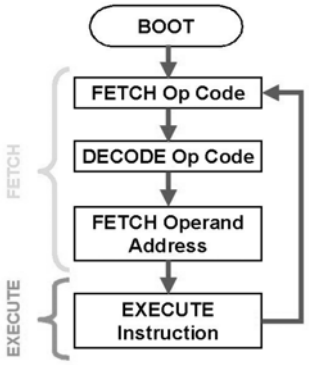


- **Il PC viene aggiornato ed ora punta alla prossima istruzione del programma da eseguire**

CPU - COME ESEGUE UN'ISTRUZIONE ?

Fase di EXECUTE

- **L'istruzione caricata nell' IR viene decodificata**
- **Vengono eseguiti i trasferimenti di dati necessari e le operazioni logiche e/o aritmetiche derivate dalla decodifica dell'op code**
- **Il risultato, a seconda del tipo di operazione eseguita è riscritto in un registro o in una locazione di memoria o su un dispositivo di I/O**



Normalmente, quindi, un istruzione per essere eseguita RICHIEDE ALMENO 2 CICLI MACCHINA (almeno 2 ACCESSI IN MEMORIA, uno in LETTURA e uno in SCRITTURA)

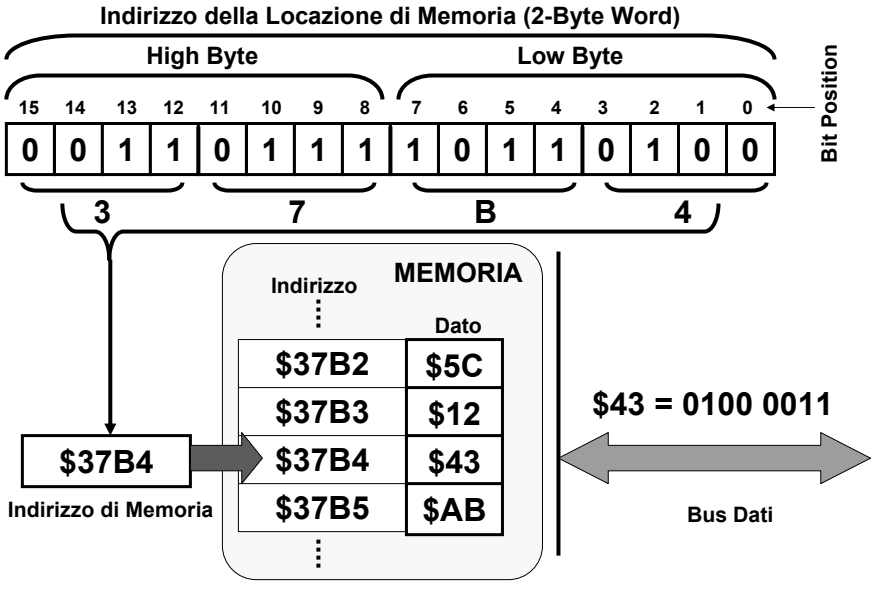
MEMORIA

La memoria in un calcolatore serve per immagazzinare dati e le istruzioni dei programmi da eseguire

I principali tipi di memorie che si possono trovare su un microcontrollore, e su un calcolatore in genere, sono:

- **ROM (Read Only Memory) : programmata permanentemente dal costruttore, non modificabile**
- **RAM (Random Access Memory) : memoria volatile di lettura e scrittura**
- **EEPROM (Electrically EPROM) : memoria non volatile, scrittura e cancellazione di celle entrambe elettriche**
- **FLASH : memoria non volatile, variante delle EEPROM, scrittura e cancellazione entrambe elettriche, non di singole celle, ma di blocchi (updating +veloce)**

ESEMPIO DI INDIRIZZAMENTO



MICROCOMPUTER & MICROCONTROLLER

I termini µP, CPU e MPU (Microprocessor Unit) possono essere considerati sinonimi.

Come visto la CPU è l'insieme di ALU e Control Unit

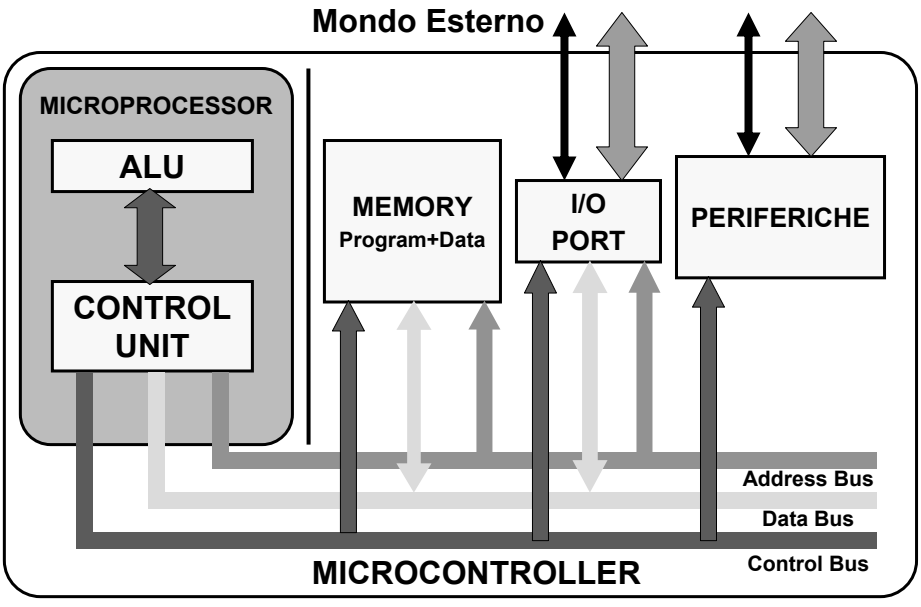
Quando la CPU è un singolo IC è chiamata µP o MPU

L'insieme di MPU, memoria e porte di Input/Output (I/O) è detto MICROCOMPUTER

Quando MPU, memoria e I/O sono integrate su un unico chip si parla di MICROCONTROLLER (µC)

Un Microcontrollore opportunamente programmato è in grado di svolgere attività di controllo acquisendo e inviando dati senza l'ausilio di circuiterie esterne

MICROPROCESSOR vs. MICROCONTROLLER



MICROPROCESSOR vs. MICROCONTROLLER

MICROPROCESSOR (µP)	MICROCONTROLLER (µC)
➢ Alte prestazioni	➢ Alto livello di integrazione
➢ General Purpose	➢ Utilizzati per controlli embedded
➢ "cervello" di PC e Workstations	
➢ Svolge le funzioni di decodifica e controllo istruzioni, operazioni logiche e aritmetiche, controllo del mondo esterno	➢ Svolge le funzioni di un µP con in più "on chip" memoria, porte di I/O, timer, ADC, moduli CAN, USART, ecc…
➢ Costo: da 75 a 500 $	➢ Costo: da 1 a 25 $
➢ Richiesta Annuale: milioni di pz.	➢ Richiesta Annuale: miliardi di pz.

MICROPROCESSOR vs. MICROCONTROLLER

MICROPROCESSOR (µP)

Contiene unità di gestione delle memorie interne ed esterne ed è provvisto di *memoria cache*

La performance (n° di istruzioni exe al sec.) è la caratteristica più importante, il costo è secondario

Viene usato tipicamente nei PC fissi nei laptop o nelle workstations

MICROCONTROLLER (µC)

Ha RAM e ROM integrate, ma è sprovvisto di cache

Ha integrate molte periferiche e viene usato in applicazioni *embedded*

Usato anche in applicazioni di controllo Real Time

Basso costo, basso consumo di potenza

NON C'E' SEMPRE UNA DISTINZIONE CHIARA

µP DI OGGI $\approx$ µC DI DOMANI

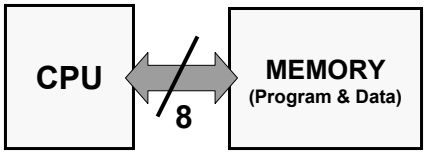
ARCHITETTURE DI UN MICROCONTROLLORE

In tutte le lezioni viene preso come esempio di riferimento il PIC 18F458.

Le considerazioni fatte e le metodologie di programmazione sono comunque di carattere generale e facilmente estendibili a tutte le più comuni famiglie di microcontrollori.

ARCHITETTURA VON NEUMANN

- **Utilizzata di solito per processori *general purpose***
- **Prevede un BUS UNICO tra CPU e memoria**
- **RAM (Data Memory) e Program Memory, quindi devono condividere lo stesso bus, per cui devono avere entrambe parole della stessa lunghezza**

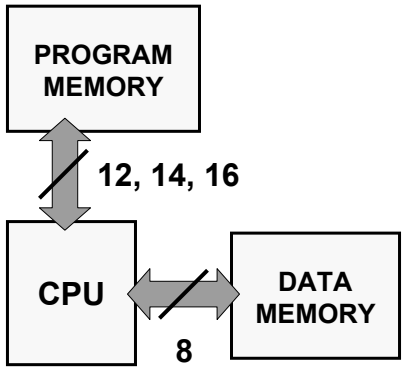


COLLO DI BOTTIGLIA: Il fatto di dover condividere un bus unico fa sì che per completare un'istruzione siano necessari 2 accessi in memoria (uno in RAM e uno in Program Memory) per cui si ha:

UNA ISTRUZIONE ESEGUITA OGNI 2 CICLI MACCHINA

ARCHITETTURA HARVARD

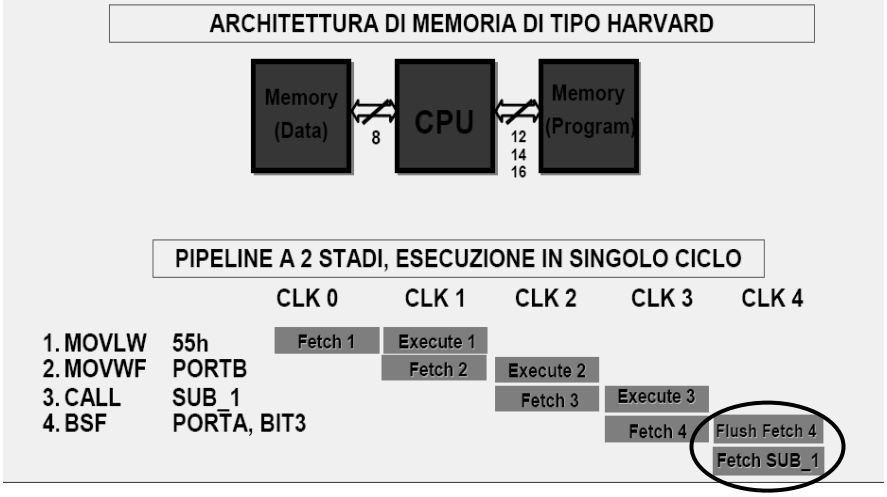
- **Utilizzata di solito per processori RISC, come ad es. i PIC (Peripheral Interface Control)**
- **Prevede 2 BUS SEPARATI tra CPU, program memory e data memory**
- **RAM (Data Memory) e ROM (Program Memory) possono avere parole di lunghezza DIVERSA**



PIC: RAM 8 bit
ROM 12, 14 o 16 bit

La CPU può effettuare un accesso in RAM e uno in ROM contemporaneamente e sfruttando tecniche di *pipeline* si può arrivare ad eseguire 1 ISTRUZIONE OGNI CICLO MACCHINA

ARCHITETTURA HARVARD



CISC vs. RISC

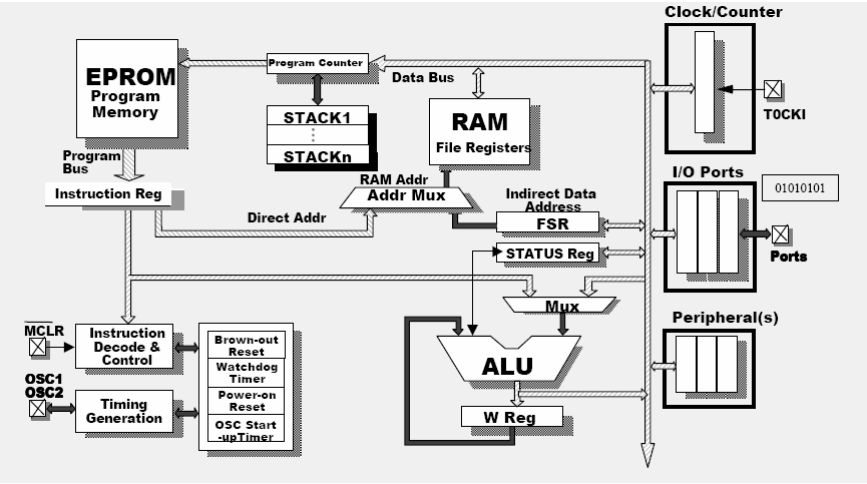
Esistono 2 grandi famiglie di microcontrollori (o più in generale di processori) quelli CISC e quelli RISC.

CISC (Complex Instruction Set Computer)
- **In genere le CPU sono CISC**
- **Normalmente utilizzano architetture Von Neumann classiche**
- **Molte istruzioni (>100)**
- **Molti metodi di indirizzamento**
- **Più di 1 ciclo macchina per eseguire un'istruzione**

RISC (Reduced Instruction Set Computer)
- **Poche istruzioni (<50)**
- **Pochi metodi di indirizzamento (solo diretto e indiretto)**
- **1 ciclo macchina per eseguire un'istruzione (a parte salti e call)**

PIC: STRUTTURA INTERNA

PIC: STRUTTURA INTERNA



FORMATO DELLE ISTRUZIONI

Byte-oriented file register operations

Example Instruction

15 10	9	8	7 0
OPCODE	d	a	f (FILE #)

```
ADDWF MYREG, W, B
```

d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Byte to Byte move operations (2-word)

15 12	11 0
OPCODE	f (Source FILE #)

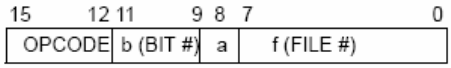
```
MOVFF MYREG1, MYREG2
```

15 12	11 0
1111	f (Destination FILE #)

f = 12-bit file register address

FORMATO DELLE ISTRUZIONI

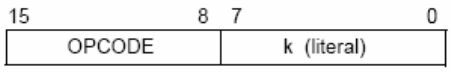
Bit-oriented file register operations




```
BSF MYREG, bit, B
```

b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Literal operations



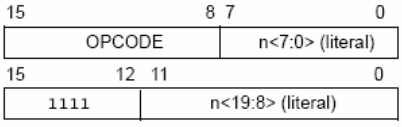

```
MOVLW 0x7F
```

k = 8-bit immediate value

FORMATO DELLE ISTRUZIONI

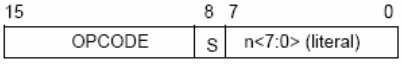
Control operations

CALL, GOTO and Branch operations

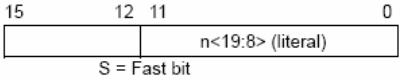



```
GOTO Label
```

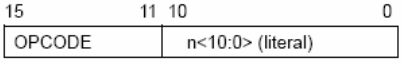

n = 20-bit immediate value



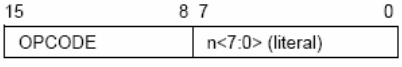

```
CALL MYFUNC
```



S = Fast bit




```
BRA MYFUNC
```




```
BC MYFUNC
```

CPU - Central Processing Unit

- **E' il "cervello" del dispositivo**
- **E' responsabile della corretta sequenza di fetch delle istruzioni da eseguire**
- **Decodifica ed esegue le istruzioni stesse**
- **Lavorare in sincronia con l' ALU quando l'istruzione da eseguire è di tipo logico o artimetico**
- **Controlla l'address bus della program memory**
- **Controlla l'address bus della data memory**
- **Controlla gli accessi allo stack**

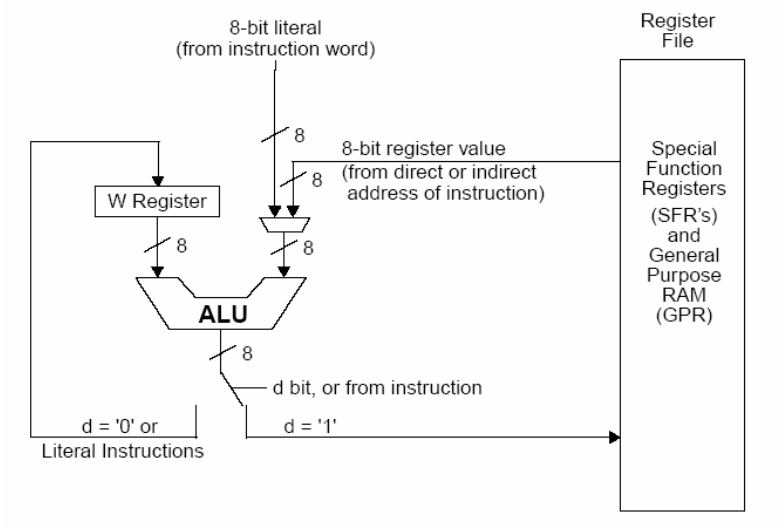
ALU - Arithmetic Logical Unit

I PIC hanno al loro interno una 8-bit ALU e un 8-bit Working Register (W)

L'ALU esegue addizioni, sottrazioni, shift e operazioni logiche tra i dati nel working register e tutti gli altri registri, compresi quelli dedicati alle variabili di lavoro del programma in esecuzione

- **Senza esplicite specificazioni le operazioni sono svolte in base ai criteri del complemento a 2**
- **Nelle istruzioni a 2 operandi tipicamente un operando è il W e l'altro è uno dei file register o una costante**
- **Nelle istruzioni a singolo operando l'operando può essere il W o uno dei file register**

ALU - Arithmetic Logical Unit



PIC: ORGANIZZAZIONE DELLA MEMORIA

PROGRAM COUNTER - PC

Il *Program Counter* (PC) è un registro dedicato in cui la CPU tiene memorizzato l'indirizzo della locazione di program memory in cui è memorizzata la prossima istruzione di programma che si deve eseguire.

Viene incrementato automaticamente ad ogni istruzione eseguita per determinare il passaggio all'istruzione successiva.

Può essere aggiornato via software dal programma in esecuzione (memorizzato sulla program memory del microcontrollore), ad esempio con istruzioni di salto o chiamata a subroutine (funzioni) o tramite il servizio di interrupt

PROGRAM COUNTER - PC

Ha una lunghezza di 21 bit divisi in 3 byte:

- **Low Byte <7:0> è il registro chiamato PCL (R/W)**
- **High Byte <15:8> è il registro chiamato PCH, non è direttamente (R/W), bisogna passare attraverso il registro speciale PCLATH**
- **Upper Byte <20:16> è il registro chiamato, non è direttamente (R/W), bisogna passare attraverso il registro speciale PCLATU**
- **Per prevenire errori di allineamento LSB di PC è fisso a zero = PC viene incrementato di 2 dopo ogni istruzione**

L'esecuzione di un'istruzione di GOTO, comporta l'aggiunta di un offset al PC

STACK E STACK POINTER (SP)

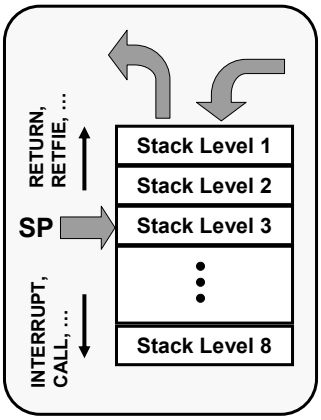
Lo *Stack* è una serie di registri dedicati che la CPU utilizza in caso di chiamata a subroutine o interrupt.

Qui la CPU salva l'indirizzo di quella che sarebbe dovuta essere la successiva istruzione da eseguire se non si fosse verificata una chiamata a subroutine (istruzione CALL) o un interrupt.

Utilizza una politica di gestione LIFO (Last Input First Output).

Si parla di *stack a 32 livelli* (= 32 registri di stack = fino a 32 CALL annidate)

Lo *Stack Pointer* (SP) è il puntatore allo stack, cioè quel registro che mi indica a che livello dello stack mi trovo.



STACK E STACK POINTER (SP)

Lo Stack Pointer non è R/W.

Il PC è caricato (PUSH) nello stack in caso di CALL o salto dovuto ad interrupt.

In caso di PUSH lo SP si muove in basso nello stack

SP si muove in alto in caso di POP, vale a dire quando si esegue un'istruzione di return da subroutine o di fine servizio interrupt

- **Se si fanno più di 32 operazioni di PUSH senza nessuna POP, la 33-esima sovrascrive la prima, la 34-esima la seconda e così via**

PIC: ORGANIZZAZIONE DELLA MEMORIA

Esistono 3 blocchi di memoria all'interno di ogni µC:

Program Memory, Data Memory ed EEPROM Data Memory

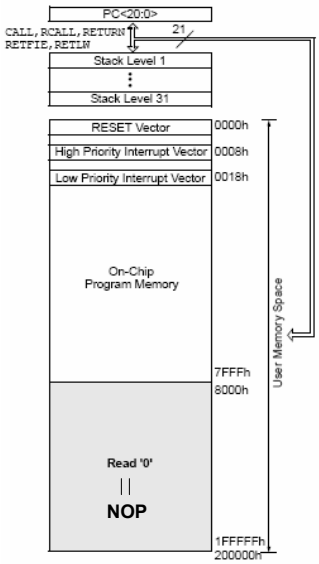
La Program Memory (Flash) è quella parte di memoria in cui risiede il programma da eseguire

La Data Memory (RAM) è quella parte di memoria in cui risiedono le variabili di lavoro generate dal programma

La EEPROM Data Memory offre la possibilità di memorizzare dati in maniera non volatile

La Program Memory e la Data Memory hanno 2 bus divisi (architettura Harvard)

PROGRAM MEMORY



Il PC ha una lunghezza di 21 bit e quindi è in grado di indirizzare 2M di spazio fisico.

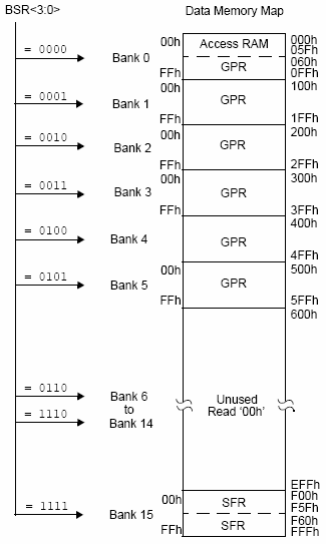
Ogni parola di memoria flash è di 16 bit (che è anche la larghezza del bus di program memory).

32K di progr. mem. = 16K di istruz.

Il *reset vector address* è l'indirizzo 0x00 il quale viene forzato nel PC ad ogni reset

L'*interrupt vector address,* il quale viene forzato nel PC ad ogni interrupt accettato dal µC, è l'indirizzo 0x08 per gli interrupt ad alta priorità e l'indirizzo 0x18 per quelli a bassa priorità

DATA MEMORY



I suoi registri si possono dividere in 2 gruppi principali:

- **GPR (General Purpose RAM), sono quelle locazioni di memoria lasciate libere e disponibili per l'allocazione di variabili di lavoro**
- **SFR (Special Function Register), sono quei registri tramite i quali è possibile controllare l'attività della CPU e tutti i moduli periferici presenti sul µC come timer, ADC, USART, Porte di I/O, CAN (per la serie PIC18), …**

DATA MEMORY

La Data Memory è divisa in *BANCHI*.

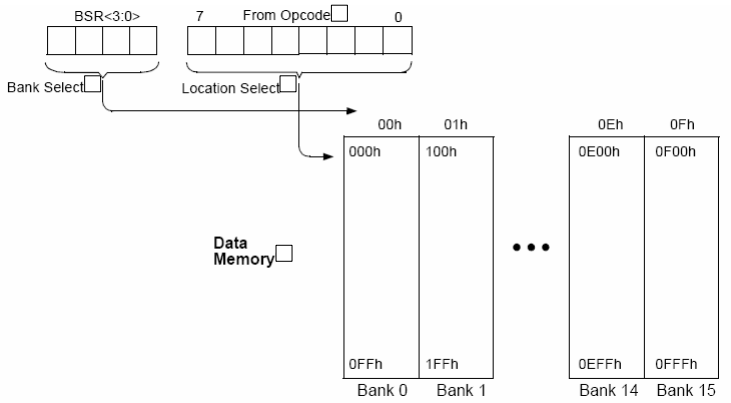
L'intera RAM può essere indirizzata *DIRETTAMENTE* o *INDIRETTAMENTE*.

La selezione di ciascun banco viene fatta attraverso l'uso dei bit di controllo <3:0> del REGISTRO BSR (Bank select Register), uno dei SFR

Sono disponibili 4096 locazioni di Data Memory, per cui l'indirizzamento è a 12 bit

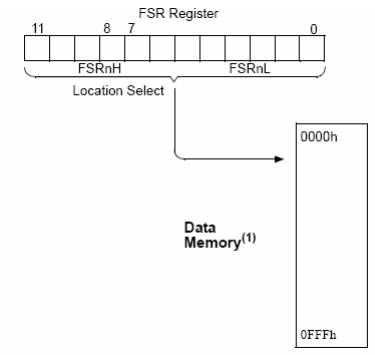
INDIRIZZAMENTO DIRETTO

Per indirizzare una locazione della Data Memory con l'*indirizzamento diretto* vengono utilizzati i bit di selezione del banco di memoria (BSR <3:0>) e i 7 LSB dell'*opcode* dell'istruzione



INDIRIZZAMENTO INDIRETTO

L'*indirizzamento indiretto* può essere usato quando si ha un'istruzione in cui l'indirizzo di data memory non è fisso (ad es. quando si usano nomi simbolici con variabili di lavoro)



Il registro FSR è usato come puntatore alla locazione di Data Memory che deve essere letta o scritta.

PIC: CLOCK E TIMING

CLOCK DI SISTEMA

Il clock è necessario per eseguire le istruzioni di programma e per il corretto funzionamento dei moduli periferici.

4 PERIODI DI CK generano 1 CICLO MACCHINA

Le istruzioni di programma, quindi, vengono eseguite con una frequenza che è ¼ di quella del clock del dispositivo.

E' possibile utilizzare il circuito interno di *timing generation* oppure sfruttare una circuiteria esterna.

DIAGRAMMA TEMPORALE DI CK E CICLO MACCHINA

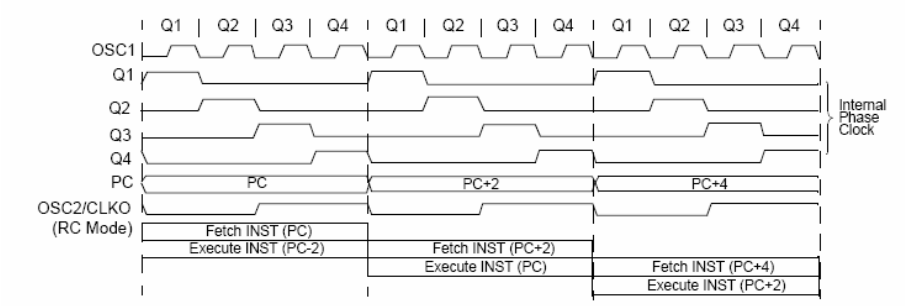


DIAGRAMMA TEMPORALE DI CK E CICLO MACCHINA

Se chiamiamo Q1, Q2, Q3 e Q4 i 4 impulsi di CK che formano 1 ciclo macchina, possiamo dire che:

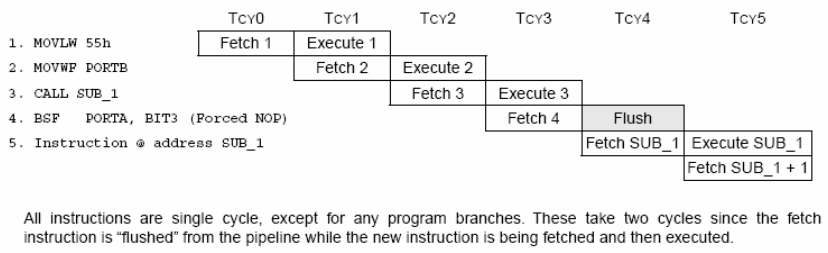
- **La fase di *Instruction Fetch* inizia con l'incremento del Program Counter in Q1**
- **Nella fase di *Execute* l'istruzione fetchata al ciclo macchina precedente viene caricata nel latch Instruction Register (IR) durante Q1**
- **Questa istruzione è decodificata durante Q2, Q3 e Q4**
- **La Data Memory è letta durante Q2 (*operand read*) e scritta durante Q4 (*destination write*)**

FLUSSO DI ISTRUZIONI E PIPELINE

La fase di Fetch impiega un ciclo macchina e la fase di Execute ne impiega un altro.

Ogni istruzione è comunque eseguita in un ciclo grazie al *pipelining*

Se un istruzione provoca un cambiamento "anomalo" del program counter, per completarla è necessario un ciclo macchina aggiuntivo



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

OSCILLATORE

A seconda dei vari microcontrollori (PIC) ci possono essere fino ad 8 modi di generazione del clock di sistema selezionabili settando gli opportuni *configurations bits* (contenuti ad es. all'interno dei vari CONFIG_REG).

I *configurations bits* sono in locazioni di memoria NON volatile e il loro valore è determinato dal valore scritto durante la programmazione del dispositivo.

A seconda di quella che sarà l'applicazione finale che utilizzerà il microcontrollore si sceglierà la configurazione ottimale

ES: Oscillatore RC => Low Cost

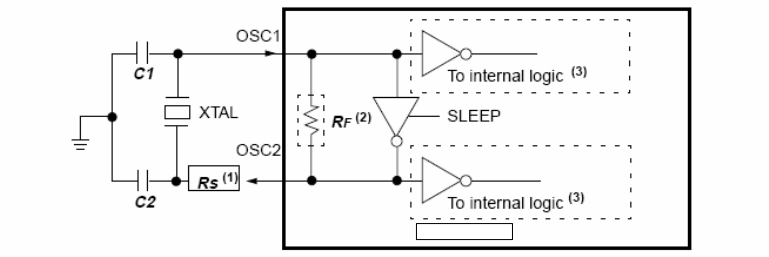
Oscillatore LP => Low Power

OSCILLATORE

FOSC2:FOSC0: Oscillator Selection bits

111 = RC oscillator w/ OSC2 configured as RA6
110 = HS oscillator with PLL enabled/clock frequency = (4 x Fosc)
101 = EC oscillator w/ OSC2 configured as RA6
100 = EC oscillator w/ OSC2 configured as divide-by-4 clock output
011 = RC oscillator
010 = HS oscillator
001 = XT oscillator
000 = LP oscillator

OSC XT, LP o HS: CIRCUITO ESTERNO



Note 1: A series resistor, *Rs*, may be required for AT strip cut crystals.
2: The feedback resistor, *RF*, is typically in the range of 2 to 10 MΩ.
3: Depending on the device, the buffer to the internal logic may be either before or after the oscillator inverter.

PIC: GESTIONE INTERRUPT

COS'E' UN INTERRUPT ?

Un Interrupt è la segnalazione che viene fatta al PIC da una delle sue periferiche o dal mondo esterno che si è verificato un determinato evento.

Il PIC è in grado di riconoscere diverse sorgenti di interrupt.

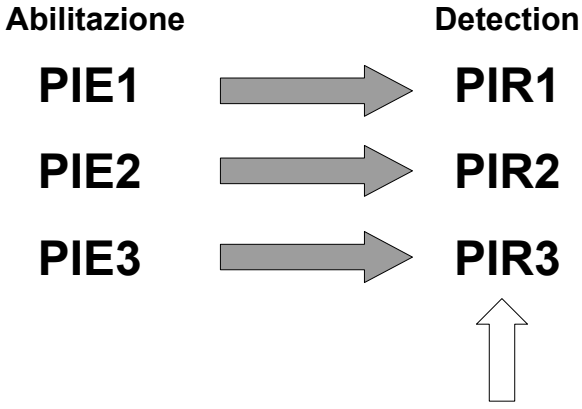
A seconda di qual è l'evento verificatosi, il programma in esecuzione dovrà svolgere una particolare funzione.

Quando si verifica un Interrupt il PIC interrompe immediatamente quella che è l'esecuzione normale del programma e salta alla locazione di memoria contenuta dall' Interrupt Vector Address (0x08 o 0x18) a partire dalle quali è allocata l'ISR (Interrupt Service Routine).

REGISTRI DI GESTIONE INTERRUPT

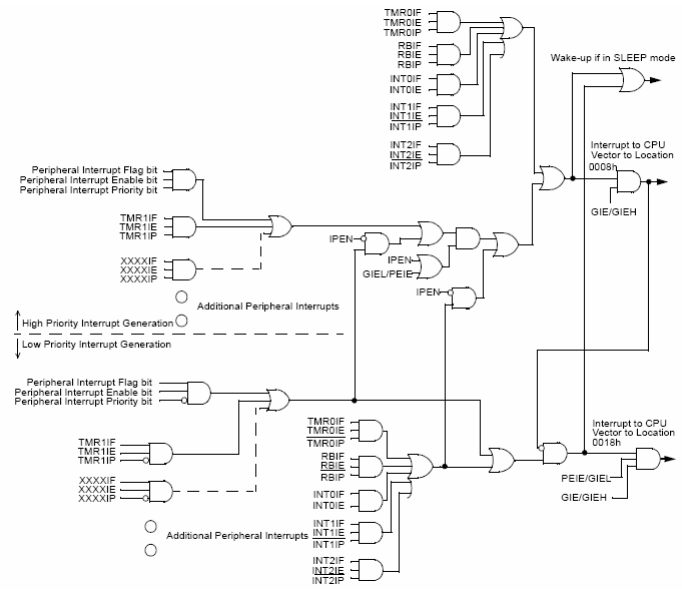
Registro	Bit	USO
INTCON	GIE	Abilitazione Generale all' utilizzo di Interrupt
	PEIE	Abilitazione al Riconoscimento di Interrupt provenienti dalle periferiche integrate sul PIC
	INTE	Abilitazione al Riconoscimento di Interrupt provenienti dall' esterno (linea RB0 e RB1)
RCON	IPEN	Abilitazione Priorità Interrupt
PIE1	xxxE	Contengono i flag (bit) corrispondenti alle periferiche di cui interessa rilevare l'interrupt ABILITAZIONE INTERRUPT
PIE2	xxxE	
PIE3	xxxE	
PIR1	xxxF	Contengono i flag (bit) che rilevano il verificarsi di un interrupt generato dalla corrispondente periferica RICONOSCIMENTO INTERRUPT (DETECTION)
PIR2	xxxF	
PIR3	xxxF	

REGISTRI DI GESTIONE INTERRUPT



SONO QUESTI DA USARE PER IL RICONOSCIMENTO DI QUALE INTERRUPT SI E' VERIFICATO

INTERRUPT: LOGICA DI DETECTION



LATENZA DI INTERRUPT

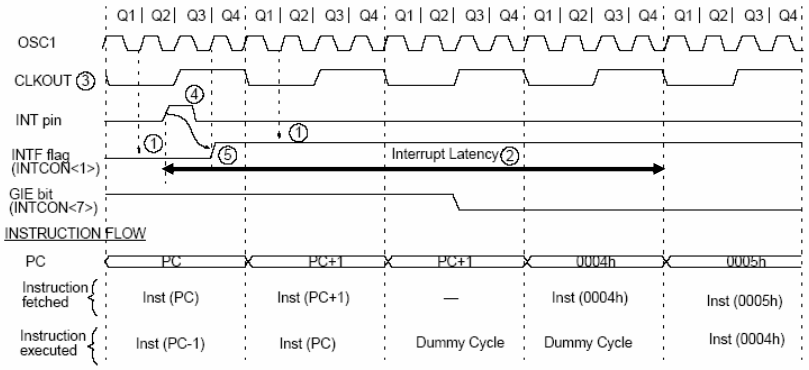
E' definita come l'*intervallo di tempo* che trascorre dal verificarsi dell'evento che genera l'interrupt (*set del corrispondente flag in PIR*) all'istante in cui inizia l'esecuzione dell'istruzione alla locazione 0x08 o 0x18 (*inizio del servizio*)

INTERRUPT INTERNI (Sincroni): 3 cicli macchina

INTERRUPT ESTERNI (Asincroni): 3-3.75 cicli macchina

La latenza esatta dipende dal punto dell'*instruction cycle* in cui si verifica l'interrupt

ESEMPIO DI TIMING PER INTERRUPT ESTERNO



Note 1: INTF flag is sampled here (every Q1).
2: Interrupt latency = 3-4 TCY where TCY = instruction cycle time.
Latency is the same whether Instruction (PC) is a single cycle or a 2-cycle instruction.
3: CLKOUT is available only in RC oscillator mode.
4: For minimum width of INT pulse, refer to AC specs.
5: INTF is enabled to be set anytime during the Q4-Q1 cycles.

Esempio riferito a PIC16F877, ma la logica che sta dietro è la stessa

STRUTTURA DI UN PROGRAMMA

```
        processor xxxxx                 ; tipo di microcontrollore
        include "xxxxx.inc"             ; utilizzo macro simboliche
        __config xxx  (opzionale)
        CBLOCK 0x…                      ; allocazione file register per
        <definizione variabili di lavoro>  ; variabili di lavoro
        ENDC
        ORG 0x00
        nop
        goto xxx
        ORG 0x08                        ;INIZIO HIGH INTERRUPT inizio zona
        bra High_interrupt              ; di memoria riservata al servizio Interrupt
        org 0x0018                      ;INIZIO LOW INTERRUPT
                <ISR Interrupt Service Routines LOW>
High_interrupt  <ISR Interrupt Service Routines HIGH>
xxx
        <Programma Principale>
        <Inizializzazione Porte & Periferiche>
        <Abilitazione Interrupt>
        <Elaborazione Segnali & Dati>
        call …..
….
<Routines di Calcolo o di Servizio Interrupt invocate>
        …return
        END
```

PROCEDURA DI SERVIZIO INTERRUPT E CODE FLOW

```
<Salvataggio Stato del Sistema (Status e W reg.)> (opzionale)
<Ciclo Riconoscimento Interrupt>
            btfsc <registrox>, <bit x (flag)>
            goto <label xxx1>
            btfsc <registroy>, <bit y (flag)>
            goto <label xxx2>
            …
<Chiamata a Funzioni di Interrupt Service>
      xxx1  call <routine1>
            goto FISR
            (è una label non 1 istruz.!!)
      xxx2  call <routine2>
            goto FISR
            …
FISR <Ripristino Stato del Sistema>
            retfie
FINE ROUTINE DI SERVIZIO INTERRUPT
…
PROGRAMMA PRINCIPALE
…
<Routines di programma>
<Routines di Interrupt>
…
```

SALTO A ZONA DI CODICE DIVERSA, CHE HA COME LABEL IL NOME DELLA FUNZIONE DA ESEGUIRE

```
routine1
<Disabilitazione Generale degli Interrupt>
            bcf INTCON, GIE
…
<Elaborazione>
            <eventuali call ad altre funzioni>
…
<Reset Flag dell'Interrupt servito>
            bcf <registrox> , <bit  x (flag)>
            questo serve per poter riconoscere
            altri interrupt di questo tipo!!!
<Riabilitazione Interrupt>
            bsf INTCON, GIE
return
```

TERMINATA LA ROUTINE SI TORNA ALL'ISTRUZIONE IMMEDIATAMENTE SEGUENTE ALLA CALL

PIC: DESCRIZIONE DELLE PERIFERICHE

I/O PORTS

I/O PORTS

Tutti i pin di input/output (I/O) general purpose possono essere considerati come le periferiche più semplici del µC.

Per aggiungere flessibilità e funzionalità al dispositivo alcuni pins sono multiplexati con altre funzioni periferiche.

Normalmente quando un pin è utilizzato da una periferica non può essere usato come general purpose I/O.

La direzione dei pin di I/O è controllata dai *data direction register* (TRIS Register).

TRIS<X> controlla la direzione del PORT<X>.

I/O PORTS

Leggendo il PORT register si legge lo stato del pin, mentre scrivendolo si scriverà sul latch.

Tutte le operazioni di scrittura sugli I/O Ports (es: BCF e BSF) sono operazioni *read-modify-write*:

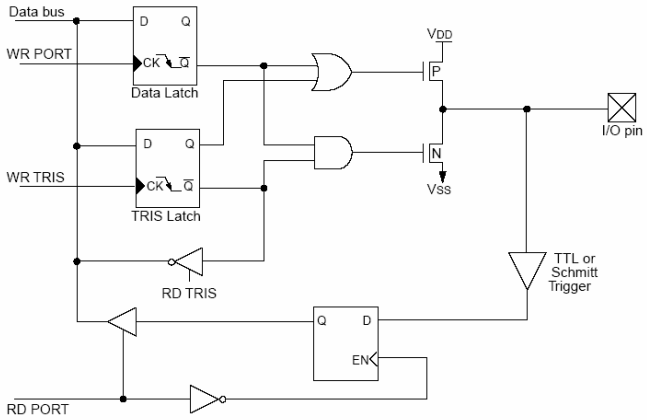
la scrittura su una porta implica la sequenza

- **LETTURA DEL VALORE CORRENTE**
- **MODIFICA**
- **SCRITTURA DEL NUOVO VALORE SUL DATA PORT LATCH**

I/O PORTS

Schema Generale di un Pin di I/O

(non sono state tenute in considerazione eventuali funzionalità multiplexate)



Note: I/O pin has protection diodes to VDD and VSS.

I/O PORTS - Riassunto

Un ‘1’ logico nel corrispondente bit di TRIS significa settare il corrispondente pin di I/O del PORT come INPUT.

Uno ‘0’ logico significa settare il pin come OUTPUT.

ES:

bsf TRISB, 3 => il pin 3 della PORTB è settato come INPUT

bsf PORTB, 3 => viene portato a livello logico alto il pin 3 della PORTB

Quando viene letto un determinato bit del PORT register viene letto il livello presente sul pin corrispondente

Quando si va a cambiare lo stato di un bit di un determinato PORT register viene cambiato lo stato del latch corrispondente.

TIMER 0

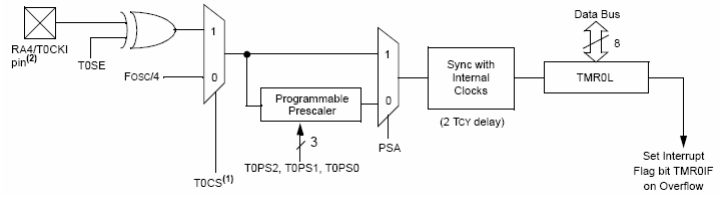
TIMER 0

CARATTERISTICHE:

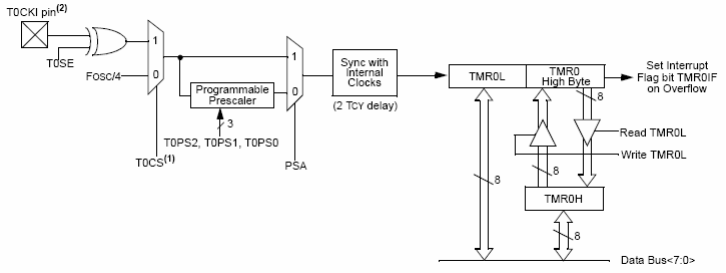
- **8/16 bit timer/counter selezionabile via SW**
- **Leggibile e Scrivibile**
- **Prescaler a 8 bit programmabile via SW**
- **Utilizzo di CK interno o esterno**
- **Possibilità di Interrupt all'overflow**
- **Selezione del fronte utile con CK esterno**

TIMER 0

TIMER 0 BLOCK DIAGRAM IN 8 bit MODE



TIMER 0 BLOCK DIAGRAM IN 16 bit MODE



TIMER 0

Via software è possibile programmare in che modalità farlo operare

Timer Mode **=> T0CON <T0CS > = 0**

Counter Mode **=> T0CON <T0CS > = 1**

- **In Timer Mode, il Timer0 viene incrementato ad ogni ciclo macchina (se prescaler è disabilitato)**
- **In Counter Mode, il Timer0 viene incrementato ad ogni fronte di salita o discesa del pin T0CKI.**

Il fronte utile è programmato via software settando o resettando il bit T0CON <T0SE>

ES: T0CON <T0SE> = 0 => ***fronte di salita***

TIMER 0: PRESCALER

- **Assegnato al TMR0 quando il bit T0CON <T0PSA> = 0.**
- **Non è leggibile o scrivibile e può assumere solo valori predefiniti e selezionabili via SW**
- **Valori selezionabili (T0CON <T0PS2:T0PS0>):**

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

TIMER 0: PRESCALER

PRESCALER NON ASSEGNATO

- **Tutte le scritture sul TMR0 register causano un'inibizione del Timer per 2 cicli macchina.**

!! dopo che il TMR0 è stato scritto con il nuovo valore, non sarà incrementato fino al terzo ciclo macchina successivo.

PRESCALER ASSEGNATO

- **Ogni scrittura sul TMR0 register aggiorna immediatamente il registro e azzera il prescaler.**
- **L'incremento del Timer0 viene inibito per 2 cicli macchina.**

ES. Prescaler Configurato a 1:2

!! dopo una scrittura sul TMR0 register, il Timer non sarà incrementato per 4 CK. Dopodichè TMR0 l'incremeno avverrà ogni numero di CK pari al prescaler impostato.

TIMER 0: ESEMPIO DI INIZIALIZZAZIONE

Sorgente di Clock Interna - Timer Mode

```
        CLRF    TMR0            ; Clear Timer0 register
        CLRF    INTCON          ; Disable interrupts and clear T0IF
                                ;
        MOVLW   0xC3            ; PortB pull-ups are disabled,
        MOVWF   OPTION_REG      ;   Interrupt on rising edge of RB0
                                ;   Timer0 increment from internal clock
                                ;   with a prescaler of 1:16.
;**     BSF     INTCON, T0IE    ; Enable TMR0 interrupt
;**     BSF     INTCON, GIE     ; Enable all interrupts
;
; The TMR0 interrupt is disabled, do polling on the overflow bit
;
T0_OVFL_WAIT
        BTFSS   INTCON, T0IF
        GOTO    T0_OVFL_WAIT
; Timer has overflowed
```

TIMER 0: ESEMPIO DI INIZIALIZZAZIONE

Sorgente di Clock Esterna - Counter Mode

```
    CLRF   TMR0          ; Clear Timer0 register
    CLRF   INTCON        ; Disable interrupts and clear T0IF

    MOVLW  0x37          ; PortB pull-ups are enabled,
    MOVWF  OPTION_REG    ;   Interrupt on falling edge of RB0
                         ;   Timer0 increment from external clock
                         ;   on the high-to-low transition of T0CKI
                         ;   with a prescaler of 1:256.

;** BSF    INTCON, T0IE  ; Enable TMR0 interrupt
;** BSF    INTCON, GIE   ; Enable all interrupts
;
; The TMR0 interrupt is disabled, do polling on the overflow bit
;
T0_OVFL_WAIT
    BTFSS  INTCON, T0IF
    GOTO   T0_OVFL_WAIT
; Timer has overflowed
```

TIMER1 & TIMER3

TIMER1 & TIMER3

E' un timer/contatore a 16 bit (2 registri di 8 bit leggibili e scrivibili)

Ha 3 modi di funzionamento

- ***Timer Sincrono***
- ***Contatore Sincrono***
- ***Contatore Asincrono*** **(funziona anche con µC in *sleep*)**

In *timer mode* l'incremento è ad ogni ciclo macchina

In *counter mode* l'incremento è ad ogni fronte di salita del CK esterno in ingresso sul Pin T1CK

Reset dal modulo CCP (ad es. utile per controllo motori)

TIMER2

TIMER2

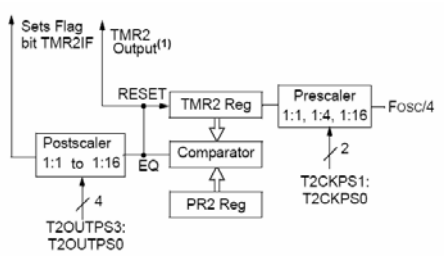
E’ un timer a 8 bit con prescaler e postscaler e la sua sorgente di CK è il CK del ciclo macchina (¼ del CK di sistema)

Di solito è usato come <u>base dei tempi per il modulo PWM</u>

Ha un registro (PR2) su cui viene impostato un certo valore.

Il *timer register* (TMR2REG) si incrementa fino a quando non matcha PR2

Viene disabilitato se il micro entra in *sleep mode*



MODULO CCP

MODULO CCP – Capture Compare PWM

Questo modulo contiene un registro a 16 bit che può operare come:

- ***Capture Register***
- ***Compare Register***
- ***PWM Master/Slave Duty Cycle Register***

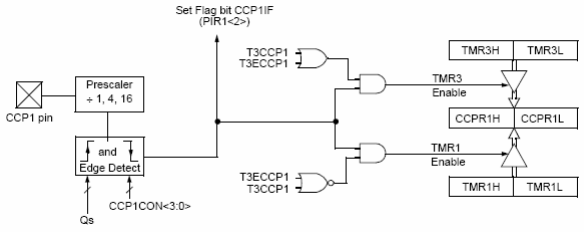
Ogni modo di funzionamento prevede l'utilizzo combinato di un timer specifico

CCP1 Mode	Timer Resource
Capture	Timer1 or Timer3
Compare	Timer1 or Timer3
PWM	Timer2

MODULO CCP - Capture Mode

Quando opera in Capture Mode il modulo "cattura" il valore del Timer1 Register (o del Timer 3) al verificarsi di un determinato evento sul pin CCP1 selezionabile via software tra:

- **Ogni fronte di salita**
- **Ogni fronte di discesa**
- **Ogni 4 fronti di salita**
- **Ogni 16 fronti di salita**

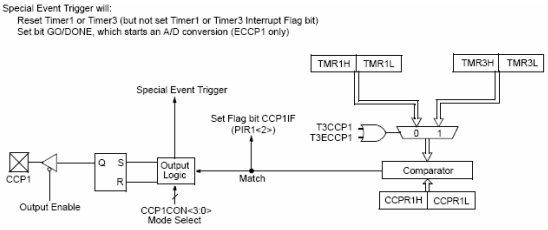


MODULO CCP - Compare Mode

Quando opera in Compare Mode il modulo confronta continuamente il valore caricato nei file register del modulo (CCPR1) con il valore dei Timer1 o Timer3.

Quando si verifica il *match* tra i valori dei due registri il pin CCP1 viene:

- **Portato a livello logico alto**
- **Portato a livello logico basso**
- **Rimane invariato**



- **E' possibile abilitare *special events* (ADC o Reset TMR1)**

Ogni azione (sul pin o special events) è configurabile via SW

MODULO CCP - PWM Mode

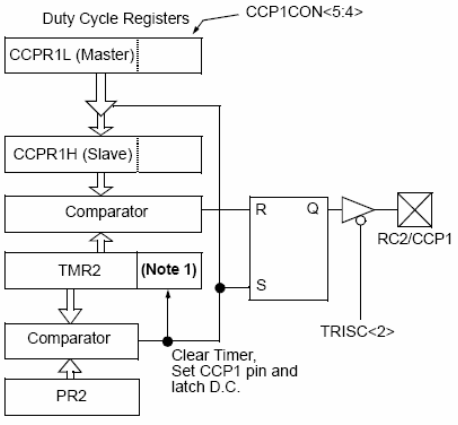
Quando opera in PWM Mode il modulo produce in uscita (CCP1 pin) un segnale PWM con risoluzione fino a 10 bit

Quando TMR2 eguaglia PR2, al ciclo macchina seguente:

TMR2 viene azzerato

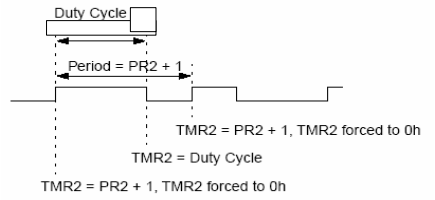
CCP1 pin è settato (eccezione se DC=0%)

Viene caricato il PWM DC



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time-base.

MODULO CCP - PWM Mode



La frequenza dell'uscita PWM è l'inverso di PWM_{Period}

MODULO CCP - ESEMPI DI INIZIALIZZAZIONE

INIZIALIZZAZIONE CAPTURE MODE

```
    CLRF    CCP1CON       ; CCP Module is off
    CLRF    TMR1H         ; Clear Timer1 High byte
    CLRF    TMR1L         ; Clear Timer1 Low byte
    CLRF    INTCON        ; Disable interrupts and clear T0IF
    BSF     TRISC, CCP1   ; Make CCP pin input
    CLRF    PIE1          ; Disable peripheral interrupts
    CLRF    PIR1          ; Clear peripheral interrupts Flags
    MOVLW   0x06          ; Capture mode, every 4th rising edge
    MOVWF   CCP1CON       ;
    BSF     T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Capture_Event
    BTFSS   PIR1, CCP1IF
    GOTO    Capture_Event
;
; Capture has occurred
;
    BCF     PIR1, CCP1IF  ; This needs to be done before next compare
```

MODULO CCP - ESEMPI DI INIZIALIZZAZIONE

INIZIALIZZAZIONE COMPARE MODE

```
    CLRF    CCP1CON        ; CCP Module is off
    CLRF    TMR1H          ; Clear Timer1 High byte
    CLRF    TMR1L          ; Clear Timer1 Low byte
    CLRF    INTCON         ; Disable interrupts and clear T0IF
    BCF     TRISC, CCP1    ; Make CCP pin output if controlling state of pin
    CLRF    PIE1           ; Disable peripheral interrupts
    CLRF    PIR1           ; Clear peripheral interrupts Flags
    MOVLW   0x08           ; Compare mode, set CCP1 pin on match
    MOVWF   CCP1CON        ;
    BSF     T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Compare_Event
    BTFSS   PIR1, CCP1IF
    GOTO    Compare_Event
;
; Compare has occurred
;
    BCF     PIR1, CCP1IF    ; This needs to be done before next compare
```

MODULO CCP - ESEMPI DI INIZIALIZZAZIONE

INIZIALIZZAZIONE PWM MODE

```
    CLRF    CCP1CON        ; CCP Module is off
    CLRF    TMR2           ; Clear Timer2
    MOVLW   0x7F           ;
    MOVWF   PR2            ;
    MOVLW   0x1F           ;
    MOVWF   CCPR1L         ; Duty Cycle is 25% of PWM Period
    CLRF    INTCON         ; Disable interrupts and clear T0IF
    BCF     TRISC, PWM1    ; Make pin output
    CLRF    PIE1           ; Disable peripheral interrupts
    CLRF    PIR1           ; Clear peripheral interrupts Flags
    MOVLW   0x2C           ; PWM mode, 2 LSbs of Duty cycle = 10
    MOVWF   CCP1CON        ;
    BSF     T2CON, TMR2ON ; Timer2 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the TMR2 Interrupt flag bit
;
PWM_Period_Match
    BTFSS   PIR1, TMR2IF
    GOTO    PWM_Period_Match
;
; Update this PWM period and the following PWM Duty cycle
;
    BCF     PIR1, TMR2IF
```

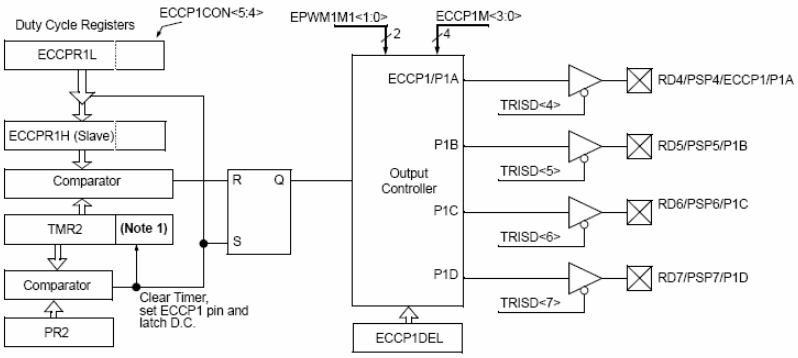
ENHANCED CCP

ENHANCED CCP

Questo modulo lavora in maniera analoga al modulo CCP visto prima e differisce solo per la modalità PWM

- **Oltre alla normale modalità PWM è in grado di funzionare in modalità *Enhanced PWM***
- **In EPWM il modulo è in grado di fornire in uscita fino a 4 segnali di controllo utilizzabili ad esempio per comandare un motore elettrico con la classica configurazione ad H**
- **Il modulo inoltre può essere programmato per andare in *shutdown* al verificarsi di determinati eventi analogici o digitali**

ENHANCED CCP - PWM



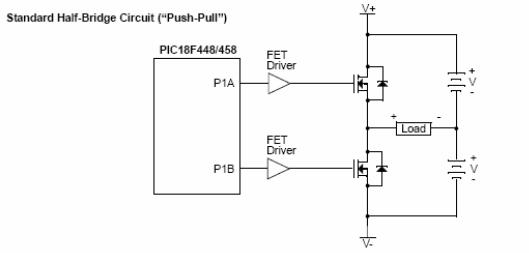
Note: The 8-bit TMR2 register is concatenated with the 2-bit internal Q clock, or 2 bits of the prescaler to create the 10-bit time-base.

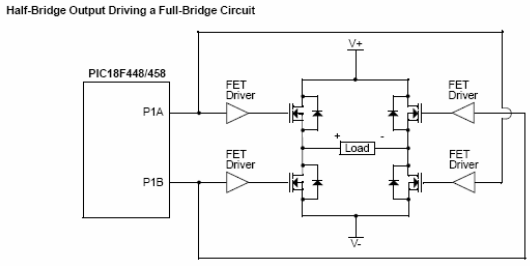
ENHANCED CCP - PWM

Può lavorare in una delle seguenti 4configurazioni a seconda dei valori impostati nei bit <7:6> di ECCP1CON

- **Single Output (uguale al modulo PWM standard)**
- **Half-Bridge Output**
- **Full-Bridge Output Forward Mode**
- **Full-Bridge Output Reverse Mode**

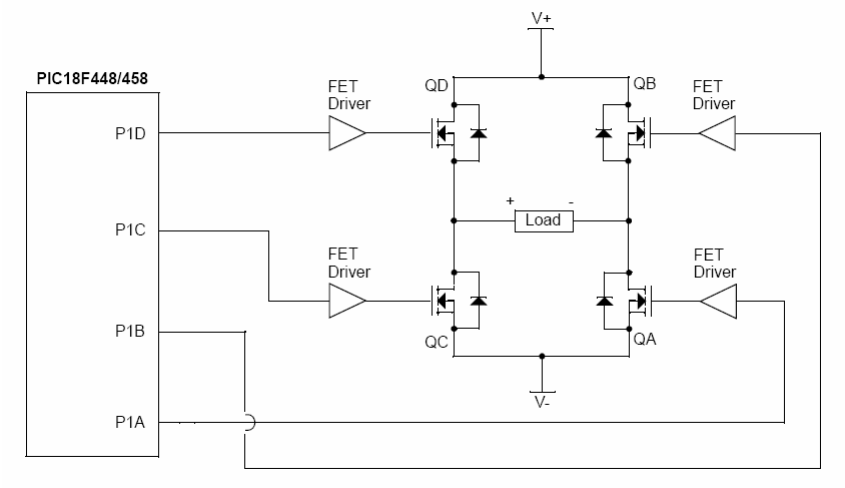
ENHANCED PWM - HALF BRIDGE





- **2 uscite complementari usate per pilotare il carico**
- **La configurazione Half-Bridge può essere utilizzata anche in applicazioni a ponte completo ma pilotati da 2 soli segnali PWM**

ENHANCED PWM - FULL BRIDGE

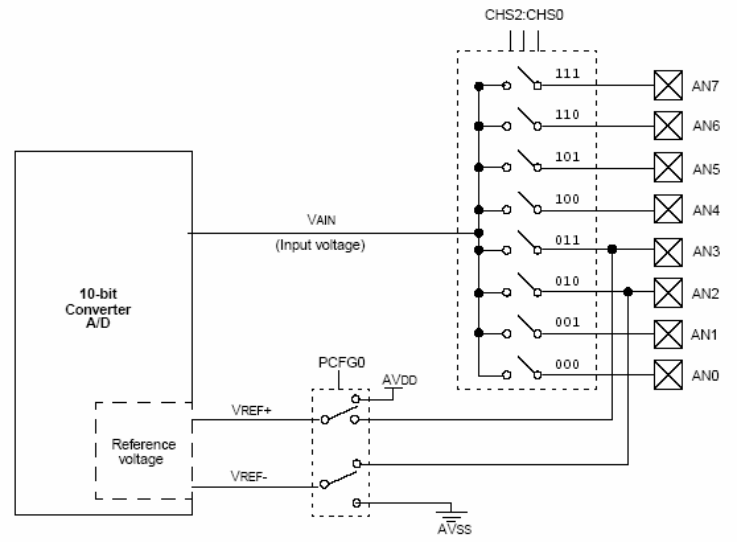


A to D CONVERTER

ADC - ANALOG to DIGITAL CONVERTER

- **Il modulo prevede l'acquisizione del dato analogico (0-5V) grazie all'utilizzo di un semplice circuito di *sample&hold* integrato nel µC e realizzato con un sampling switch e un hold capacitor.**
- **L'uscita dell'hold capacitor è l'input del convertitore.**
- **Il convertitore lavora per *approssimazioni successive* e produce un risultato digitale con risoluzione di 10 bit.**
- **E' possibile avere fino ad un max di 8 canali analogici di ingresso multiplexati.**
- **A seconda dell'applicazione si può decidere via SW se le Vref vengono fornite dall esterno o possono essere utilizzate le tensioni standard fornite dal µC.**

ADC - ANALOG to DIGITAL CONVERTER

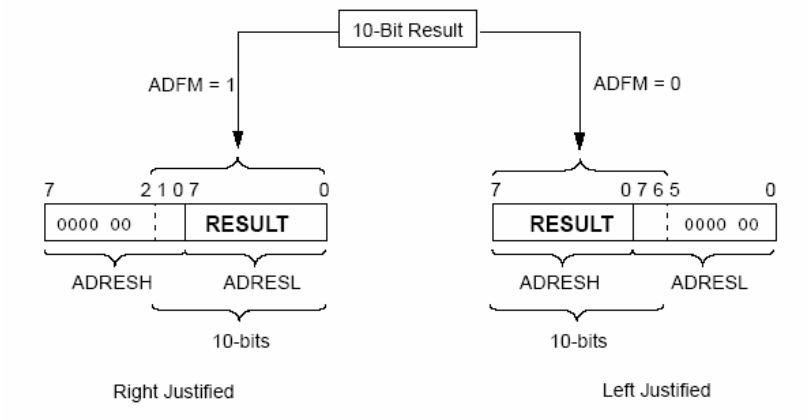


ADC - ANALOG to DIGITAL CONVERTER

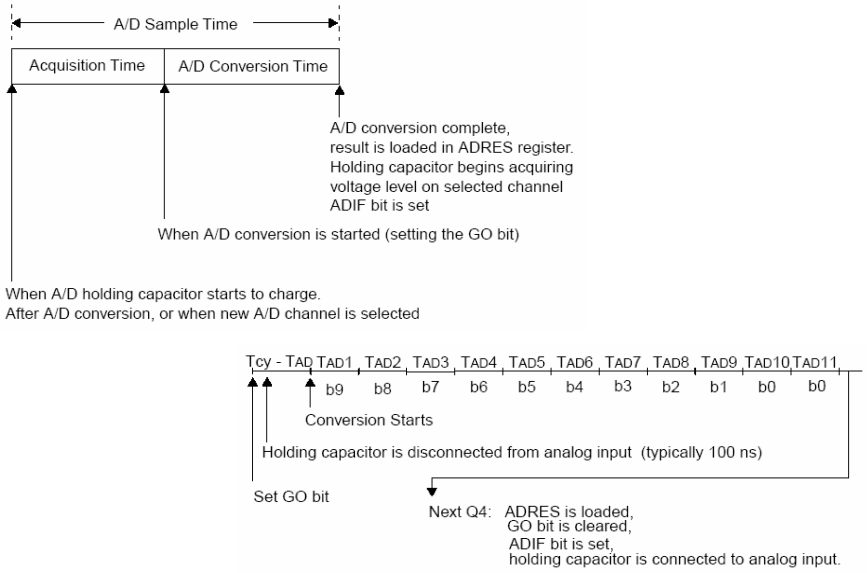
- **Il modulo ha 4 registri base**
  - **A/D Control Register0 (ADCON0)**
  - **A/D Control Register1 (ADCON1)**
  - **A/D Result High Register (ADRESH)**
  - **A/D Result Low Register (ADRESL)**
- **ADCON0 controlla le operazioni di conversione**
- **ADCON1 configura le porte (num di input analogici da multiplexare, sorgente di Vref, ecc…)**
- **ADRESH:ADRESL sono i registri in cui viene caricato il risultato della conversione**

ADC - ANALOG to DIGITAL CONVERTER

Si può avere giustificazione a destra o a sinistra del risultato della conversione



ADC - ANALOG to DIGITAL CONVERTER



ADC - ANALOG to DIGITAL CONVERTER

- **L' *Acquisition Time* è il tempo durante il quale l'hold capacitor è connesso al livello di tensione esterno da convertire**
- **T_{AD} è il tempo di conversione per bit (L'ADC lavora per approssimazioni successive!)**
- **Per una conversione a 10 bit sono necessari 12 T_{AD}**
- **La somma di Acquisition time e tempo di conversione è il *Sampling Time***
- **C'è un tempo di acquisizione minimo per garantire la carica del condensatore di hold**

ADC - ANALOG to DIGITAL CONVERTER

ESEMPIO DI INIZIALIZZAZIONE E CONVERSIONE

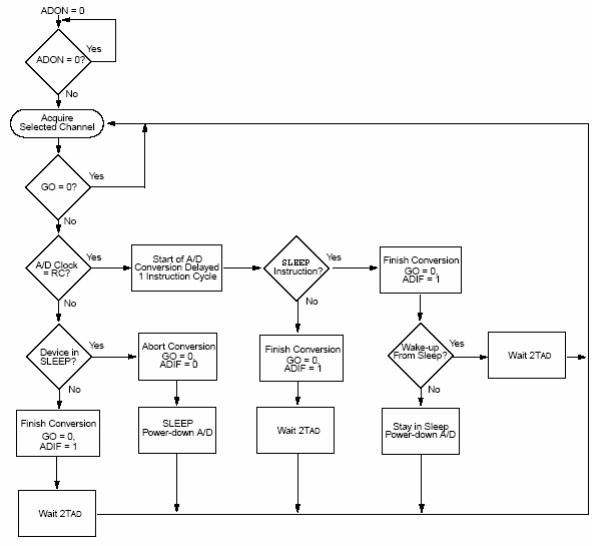
```
    CLRF    ADCON1          ; Configure A/D inputs,
                            ;    result is left justified
    BSF     PIE1, ADIE      ; Enable A/D interrupts

    MOVLW   0xC1            ; RC Clock, A/D is on, Channel 0 is selected
    MOVWF   ADCON0          ;
    BCF     PIR1, ADIF      ; Clear A/D interrupt flag bit
    BSF     INTCON, PEIE    ; Enable peripheral interrupts
    BSF     INTCON, GIE     ; Enable all interrupts
;
;  Ensure that the required sampling time for the selected input
;  channel has elapsed. Then the conversion may be started.
;
    BSF     ADCON0, GO      ; Start A/D Conversion
      :                     ; The ADIF bit will be set and the GO/DONE
      :                     ; bit is cleared upon completion of the
      :                     ;     A/D Conversion.
```

NB: Il bit GO/DONE di Start of Conversion (SOC) NON deve essere settato nella stessa istruzione che accende il modulo a causa dei requisiti minimi di acquisizione

ADC - ANALOG to DIGITAL CONVERTER

- **Il convertitore può lavorare anche quando il µC è in *sleep mode* e, anzi, lo può svegliare generando un interrupt di *end of conversion*.**
- **Il modulo A/D aspetta 1 ciclo macchina prima di iniziare la conversione.**
- **Questo permette di eliminare il rumore derivante dallo switching digitale del canale da convertire**
- **L'istruzione di *sleep* DEVE seguire l'istruzione di SOC**
- **Se il corrispondente interrupt è abilitato alla fine della conversione il modulo "sveglia" il µC.**

ADC - ANALOG to DIGITAL CONVERTER



FLOW CHART di una CONVERSIONE A/D

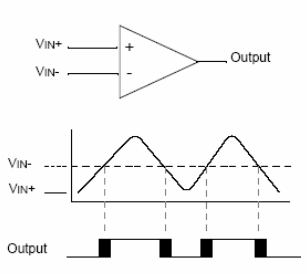
COMPARATORE

COMPARATORE

Il modulo contiene 2 comparatori analogici

Oltre a 4 linee esterne (RD0:RD3) è possibile utilizzare come ingressi anche il generatore di tensioni di riferimento interno al PIC

Il registro CMCON controlla le varie configurazioni in cui può lavorare il modulo (bit CM2:CM0)

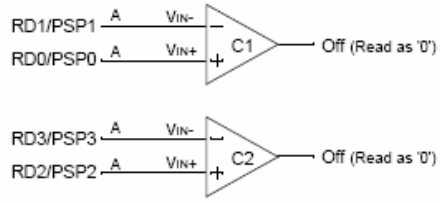


Attraverso il registro CMCON è possibile anche leggere le uscite del modulo (bit 7:6 CxOUT) e cambiarne la polarità (bit 4:5 CxINV)

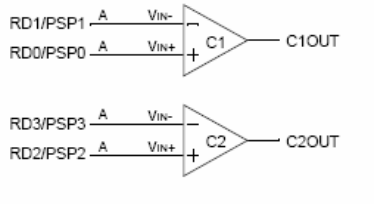
Il modulo può generare un interrupt in grado di svegliare il PIC dallo sleep

COMPARATORE

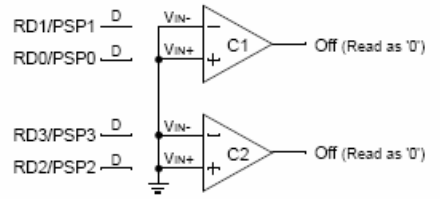
Comparators Reset (POR Default Value)
CM2:CM0 = 000



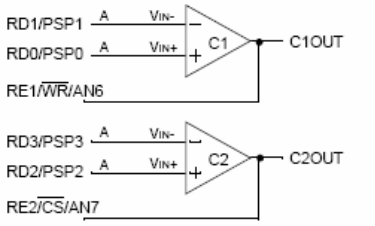
Two Independent Comparators
CM2:CM0 = 010



Comparators Off
CM2:CM0 = 111

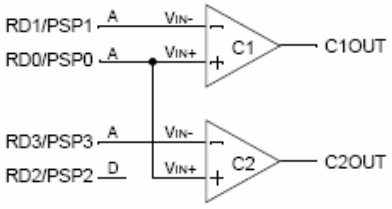


Two Independent Comparators with Outputs
CM2:CM0 = 011

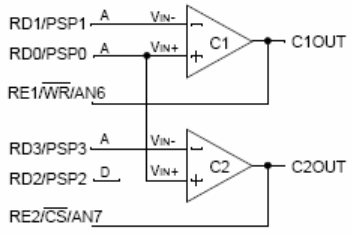


COMPARATORE

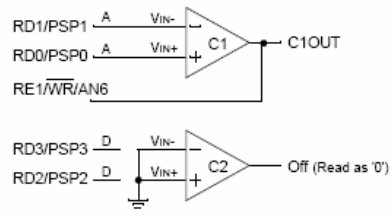
Two Common Reference Comparators
CM2:CM0 = 100



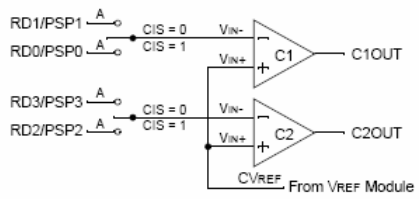
Two Common Reference Comparators with Outputs
CM2:CM0 = 101



One Independent Comparator with Output
CM2:CM0 = 001



Four Inputs Multiplexed to Two Comparators
CM2:CM0 = 110



USART

USART

L’USART (Universal Synchronous Asynchronous Receiver Transmitter è uno dei moduli di comunicazione seriale integrato sul µC.

Può essere configurato come:

- **Asynchronous** **(Full Duplex)**
(può comunicare con dispositivi periferici come ad es. PC)
- **Synchronous - Master** **(Half Duplex)**
- **Synchronous - Slave** **(Half Duplex)**
(può comunicare con dispositivi esterni come ad es. ADC, DAC,EEPROM Dati, ecc…)

USART - BAUD RATE GENERATOR (BRG)

- **BRG serve per entrambi i modi di funzionamento del modulo (sincrono e asincrono) ed è quel blocco che si occupa di generare la frequenza alla quale vengono trasmessi o ricevuti i bit di dati.**
- **Il registro dedicato SPBRG controlla il periodo di un *free-running 8-bit timer*.**
- **Stabilita una determinata baud rate che si vuole ottenere (ad es. 9600 bps) per la comunicazione seriale, e noto il CK del µC (Fosc), è possibile calcolare il valore da inserire in SPBRG.**

USART - ASYNCHRONOUS MODE

In questa configurazione il modulo usa il formato standard NRZ (= Non Return to Zero) con:

1bit di Start, 8 o 9 bit di dati, 1bit di Stop

(Il più usato prevede 8 bit di dati)

- **Utilizza il BRG per ricavare la baud rate desiderata dal CK del sistema**
- **Trasmette e Riceve PER PRIMO l' LSB**
- **Trasmitter e Receiver sono funzionalmente indipendenti, ma utilizzano lo stesso formato dei dati e la stessa baud rate**

USART - ASYNCHRONOUS MODE

- **L'HW del modulo non supporta il controllo di parità che può essere implementato solo via SW**
- **Il suo funzionamento è interrotto durante lo SLEEP**
- **Si parla di modalità di funzionamento 8N1 (= 8 bit di dati, NO parità, 1 bit di stop)**
- **I suoi blocchi principali sono:**

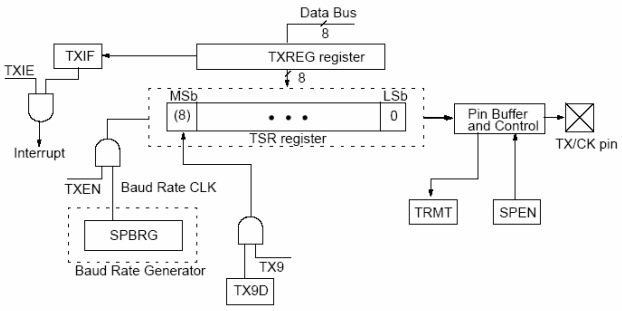
• Baud Rate Generator

• Sampling Circuit

• Asynchronous Transmitter

• Asynchronous Receiver

USART - ASYNCHRONOUS TRASMITTER

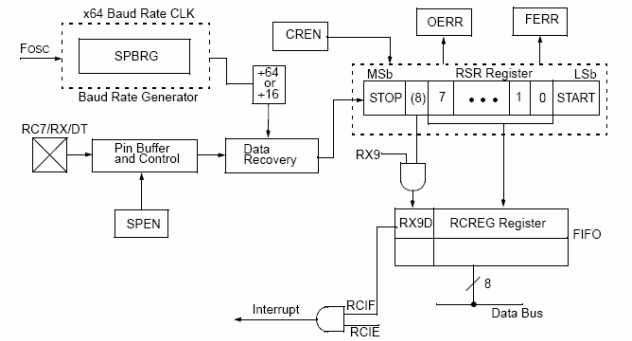


- **TSR non è mappato in memoria per cui non è accessibile dall'utente**
- **TXIF viene settato quando si setta il TXEN e viene resettato dal caricamento di TXREG**
- **La TX non avviene fino a quando TXREG non è caricato e BRG ha prodotto un fronte utile**

USART - ASYNCHRONOUS TRASMITTER

- **TSR prende i dati da TXREG**
- **TXREG viene caricato coi dati da TX via SW**
- **TSR non è caricato fino a quando non è stato trasmesso lo STOP bit del pacchetto precedente**
- **Una volta che TSR è caricato da TXREG, TXREG si svuota e TXIF viene settato e se abilitato viene generato un interrupt**
- **TXIF indica lo stato di TXREG**
- **TRMT indica lo stato di TSR e viene settato quando TSR è vuoto**

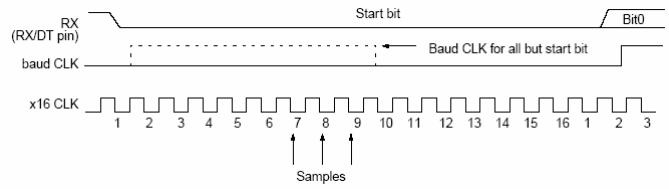
USART - ASYNCHRONOUS RECEIVER



- **Il registro chiave è RSR (Receive Shift Register)**
- **I dati sono RX sul pin RC7 e vengono memorizzati nel Data Recovery Block che altro non è che uno Shift Register che funziona a un CK x16 della baud rate**
- **La RX è abilitata settando CREN**

USART - ASYNCHRONOUS RECEIVER

- **Il dato su RC7 è campionato 3 volte per individuare con maggior precisione il livello logico presente**



- **Dopo che è stato campionato lo STOP bit il dato ricevuto e messo in RSR è trasferito su RCREG (se vuoto)**
- **Se il trasferimento viene completato viene settato RCIF e se abilitato si genera un interrupt**
- **RCIF viene resettato dall'HW quando RCREG è stato letto e svuotato**

USART - SYNCHRONOUS MODE

In questa configurazione la comunicazione è HALF DUPLEX, vale a dire che mentre si trasmettono dati la ricezione è inibita e viceversa.

Le linee RC6 e RC7 assumono il ruolo rispettivamente di CK della comunicazione e linea dati.

Master Mode indica che il µC TX il master CK sulla linea di CK (RC6 quindi è configurato come pin di output)

Slave Mode indica che il master CK è fornito dall'esterno ed arriva su RC6 che quindi è configurato come pin di input.

QUESTO PERMETTE AL MICRO DI TX O RICEVERE DATI DURANTE LO SLEEP MODE

USART - ESEMPI DI INIZIALIZZAZIONE

ASYNCHRONOUS TRASMITTER/RECEIVER

```
MOVLW   <baudrate>      ; Set Baud rate
MOVWF   SPBRG
MOVLW   0x40            ; 8-bit transmit, transmitter enabled,
MOVWF   TXSTA           ; asynchronous mode, low speed mode
BSF     PIE1,TXIE       ; Enable transmit interrupts
BSF     PIE1,RCIE       ; Enable receive interrupts

MOVLW   0x90            ; 8-bit receive, receiver enabled,
MOVWF   RCSTA           ; serial port enabled
```

SYNCHRONOUS TRASMITTER/RECEIVER

```
MOVLW   <baudrate>      ; Set Baud Rate
MOVWF   SPBRG
MOVLW   0xB0            ; Synchronous Master,8-bit transmit,
MOVWF   TXSTA           ; transmitter enabled, low speed mode
BSF     PIE1,TXIE       ; Enable transmit interrupts
BSF     PIE1,RCIE       ; Enable receive interrupts

MOVLW   0x90            ; 8-bit receive, receiver enabled,
MOVWF   RCSTA           ; continuous receive, serial port enabled
```

DATA EEPROM

DATA EEPROM

Questo tipo di periferica permette di memorizzare in maniera permanente (non volatile) dati che possono essere di natura diversa ad es: risultato di elaborazioni, stato del sistema al verificarsi di determinate condizioni, dati acquisiti dall'esterno, ecc…

Risulta molto utile in caso di applicazioni che prevedono acquisizione dati come nel caso di *datalogger*.

Il controllo e l'accesso alla EEPROM Dati avviene *indirettamente* attraverso l'uso di 4 registri SFR:

- **EECON1 controlla la configurazione del modulo**
- **EECON2 permette l'accesso in scrittura**
- **EEDATA contiene i dati letti/da scrivere**
- **EEADR contiene l'indirizzo della locazione da RD/WR**

DATA EEPROM - LETTURA

- **La lettura di una locazione è semplice e prevede solo di caricare in EEADR l'indirizzo della locazione EEPROM che si vuole leggere e di abilitare la lettura (Start)**
- **Il dato è disponibile in EEDATA al ciclo macchina immediatamente successivo e vi viene mantenuto fino a quando non viene effettuata una nuova lettura o fino a quando non viene sovrascritto dall'operatore ad es. durante una operazione di scrittura in EEPROM Dati**

ESEMPIO DI LETTURA

```
MOVLW   DATA_EE_ADDR    ;
MOVWF   EEADR           ;Data Memory Address
                        ;to read
BCF     EECON1, EEPGD   ;Point to DATA memory
BCS     EECON1, CFGS    ;
BSF     EECON1, RD      ;EEPROM Read
MOVF    EEDATA, W       ;W = EEDATA
```

DATA EEPROM - SCRITTURA

- **La scrittura prevede una precisa sequenza d'accesso alla EEPROM Dati, questo per evitare accessi indesiderati che potrebbero alterare il contenuto della memoria stessa.**

ESEMPIO DI SCRITTURA

Required Sequence

```
BCF     INTCON, GIE  ; Disable INTs.
BSF     EECON1, WREN ; Enable Write
MOVLW   55h          ;
MOVWF   EECON2       ; 55h must be written to EECON2
MOVLW   AAh          ;   to start write sequence
MOVWF   EECON2       ; Write AAh
BSF     EECON1,WR    ; Set WR bit begin write
BSF     INTCON, GIE  ; Enable INTs.
```

- **Al completamento del ciclo di scrittura il bit WR è resettato automaticamente dall'HW, viene settato il flag EEIF e, se abilitato, viene generato un interrupt**

MODULO CAN

CAN - Control Area Network

E' un INTERFACCIA SERIALE DI COMUNICAZIONE DIGITALE studiato per applicazioni real-time.

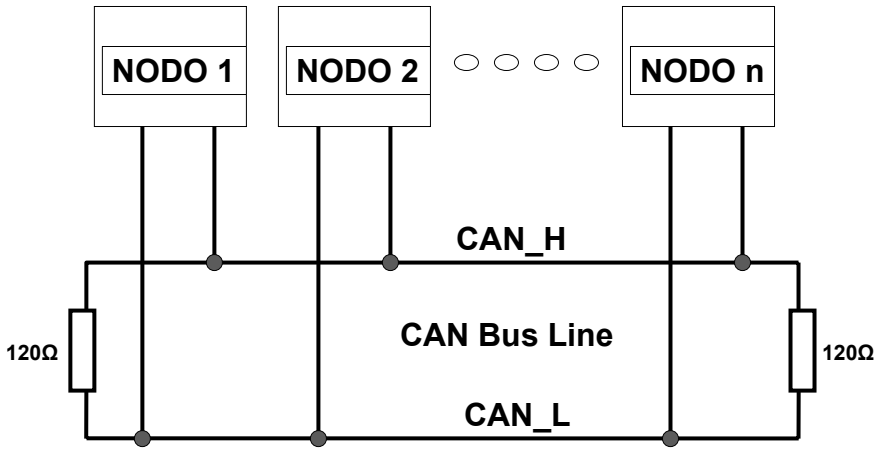
Il protocollo, grazie alle sue caratteristiche di alta flessibilità, robustezza, affidabilità consente a controllori sensori ed attuatori di:

- **Comunicare tra loro a velocità fino a 1 Mbit/sec**
- **Lavorare in condizioni ambientali ostili**
- **Svolgere funzioni di autodiagnostica**

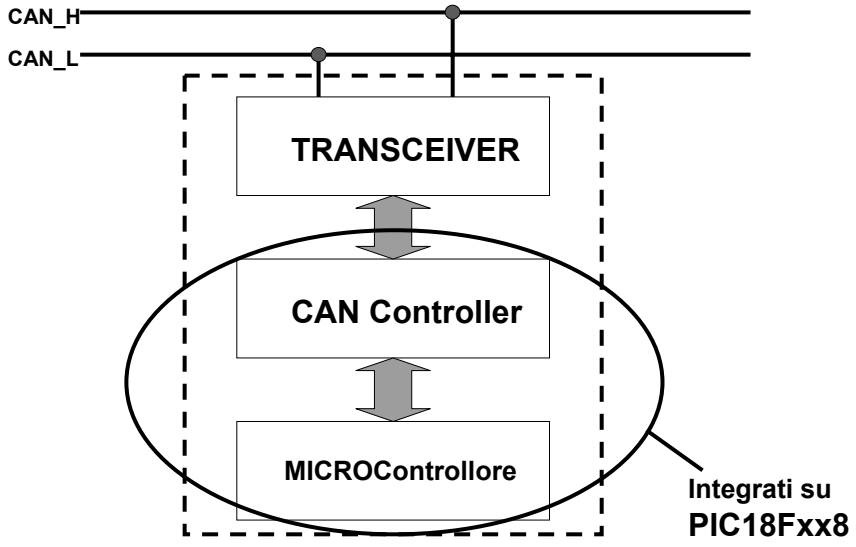
inoltre permette di:

- **Diminuire i costi di progettazione ed implementazione**
- **Inserire in un sistema prodotti di costruttori diversi**
- **Facilità di configurazione e modifica della rete**

CAN - SCHEMA DI RETE



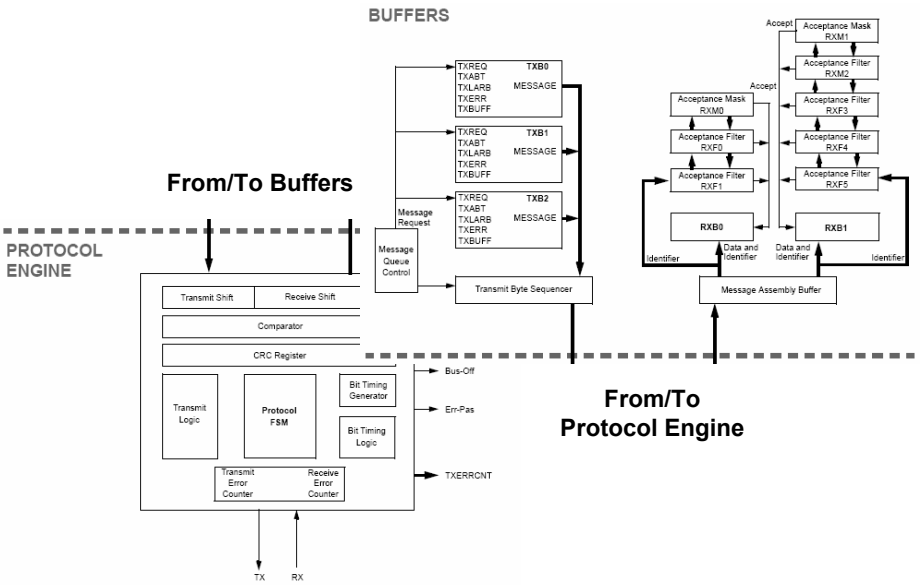
CAN - SCHEMA DI NODO



MODULO CAN - OVERVIEW

- **Il modulo è formato da una parte di *protocol engine* e da una di *message buffering and control*.**
- **Il *protocol engine* gestisce tutte le funzioni di trasmissione e ricezione di messaggi sul bus CAN.**
- **I msg possono essere trasmessi solo dopo essere stati preventivamente caricati negli opportuni data register.**
- **Stato dei msg, ed eventuali errori, possono essere controllati andando a leggere alcuni registri di controllo.**
- **Tutti i msg presenti sul bus vengono controllati e viene fatto il *match* con filtri configurati via SW,filtri che vengono utlizzati per riconoscere solo i msg utili al nodo (CAN è *broadcast* = tutti i msg sul bus sono visti da tutti i nodi)**
- **Se il match dà risultato positivo il msg è ricevuto dal modulo e memorizzato in uno dei 2 registri di ricezione**

MODULO CAN - BLOCK DIAGRAM



MODULO CAN - TIPI DI FRAME

- **Il modulo CAN supporta diversi tipi di frame:**
  - ***Standard Data Frame***
  - ***Extended Data Frame***
  - ***Remote Frame***
  - ***Error Frame***
  - ***Overload Frame Reception***
  - ***Interframe Space***
- **Usa i pin RB3/CANRX and RB2/CANTX/INT2 per interfacciarsi col bus, per cui è necessario configurare**

  bit TRISB<3> = 1

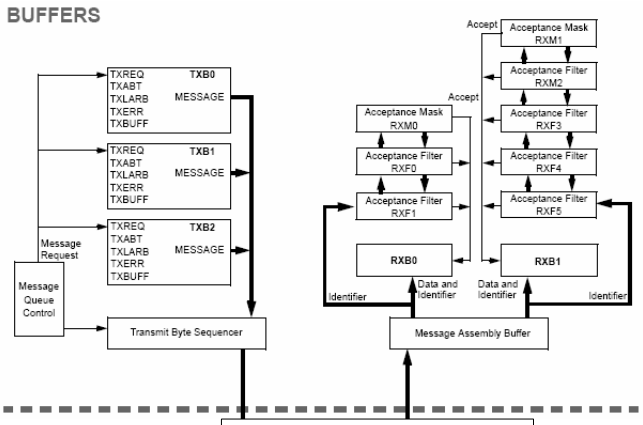
  bit TRISB<2> = 0

MODULO CAN - REGISTRI

Ci sono molti registri associati al modulo CAN che si possono raggruppare in:

- **Control and Status Registers**
- **Transmit Buffer Registers (Data and Control)**
- **Receive Buffer Registers (Data and Control)**
- **Baud Rate Control Registers**
- **I/O Control Register**
- **Interrupt Status and Control Registers**

MODULO CAN - BUFFERS



Il modulo dispone di:

3 buffer di TX, 2 buffer di RX, 2 mask buffer (uno per ogni buffer di RX), 6 filtri

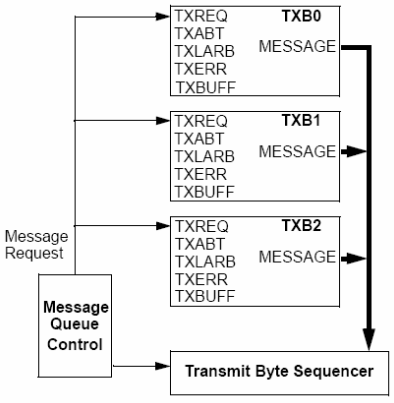
MODULO CAN - MODI DI FUNZIONAMENTO

Prevede 6 modi di funzionamento:

- ***Configuration mode*: utilizzato per la inizializzazione del modulo e non permette TX o RX di msg**
- ***Disable mode***
- ***Normal Operation mode***
- ***Listen Only mode*: il modulo può solo RX e riceve tutti i msg. E' utilizzato come monitor del bus**
- ***Loopback mode*: utilizzato in fase di debug e test del sistema, permette scambio di msg INTERNI tra buffer di TX e RX**
- ***Error Recognition mode*: utilizzato quando si vuole ricevere TUTTI i messaggio ignorando eventuali errori**

MODULO CAN - TX di MESSAGGI

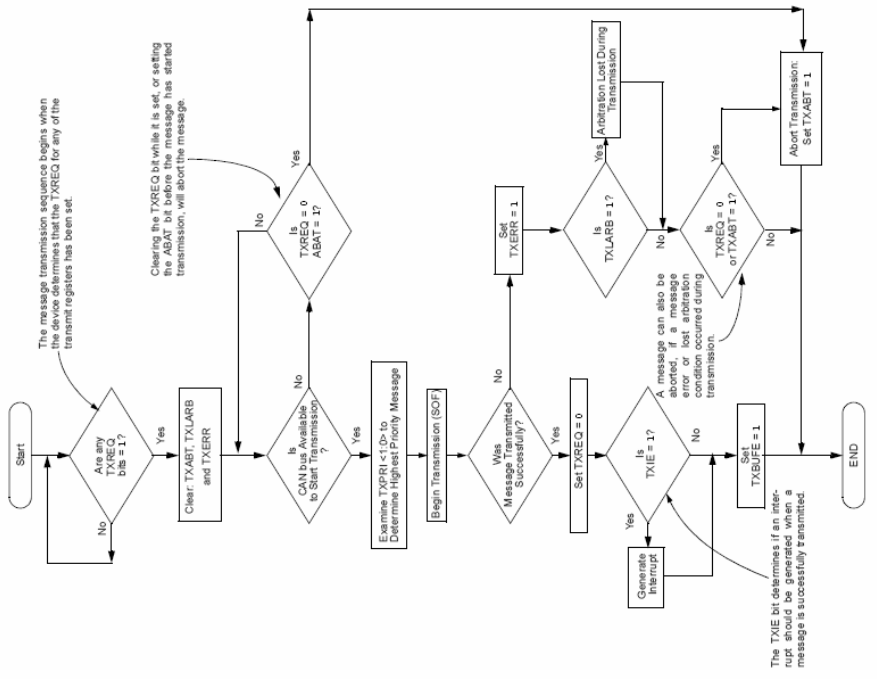
- **3 Buffer di TX (da 14 bytes)**
- **µC può accedere a uno dei buffer solo se TXREQ = 0**
- **I registri dell' identificatore e dell'indicatore del numero di byte di dati del msg da TX sono quelli minimi da caricare nel buffer per poter effettuare la TX**



- **La priorità di TX dei buffer è *indipendente* dalla priorità del protocollo: il buffer a priorità più alta TX per primo**

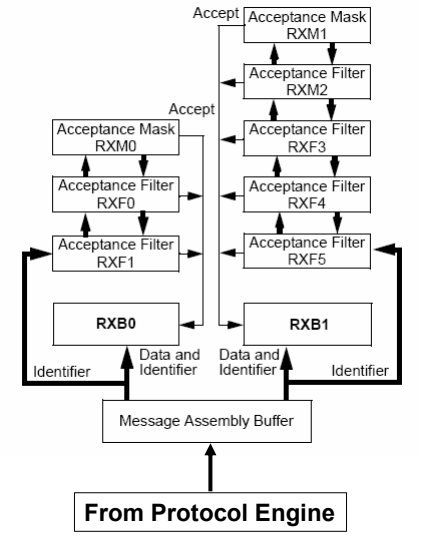
MODULO CAN - TX di MESSAGGI

- **Per iniziare la TX deve essere TXREQ = 1**
- **TXREQ = 1 non è condizione sufficiente per iniziare la TX**
- **La TX può iniziare solo quando il dispositivo rileva che il bus è in stato di *idle***
- **Quando la TX è terminata con successo allora TXREQ viene resettato, TXBnIF = 1 e si genera un interrupt se TXBnIE è stato preventivamente settato a 1**
- **Se la TX fallisce TXREQ rimane settato e si settano i flag corrispondenti ad errori di TX o a perdita dell'arbitraggio**
- **Una TX viene *abortita* se TXREQ è resettato oppure se viene fatta richiesta di abortire tutti i msg pendenti dei vari buffer settando il bit ABAT del registro CANCON**



MODULO CAN - RX di MESSAGGI

- **3 Buffer di RX**
- **MAB riceve ogni msg sul bus dal *protocol engine***
- **µC può accedere a RXB0 o RXB1 mentre l'altro è disponibile per la rx di un nuovo msg oppure contiene un msg vecchio**
- **MAB assembla tutti i msg che vengono caricati in RXB0 o RXB1 solo se è verificato uno dei criteri degli *acceptance filter***



MODULO CAN - RX di MESSAGGI

- **Quando un msg è spostato dal MAB a uno degli altri 2 buffer di RX viene settato il corrispondente interrupt flag (RXBnIF)**
- **L'interrupt verrà generato solo se il corrispondente bit di abilitazione (RXBnIE) è settato**
- **RXBnIF deve poi essere messo a zero dal µC una volta terminato il *processing* del msg per poter consentire il caricamento di msg successivi**
- **RXB0 è il buffer a priorità più alta rispetto a RXB1**

MODULO CAN - FILTRI E MASK

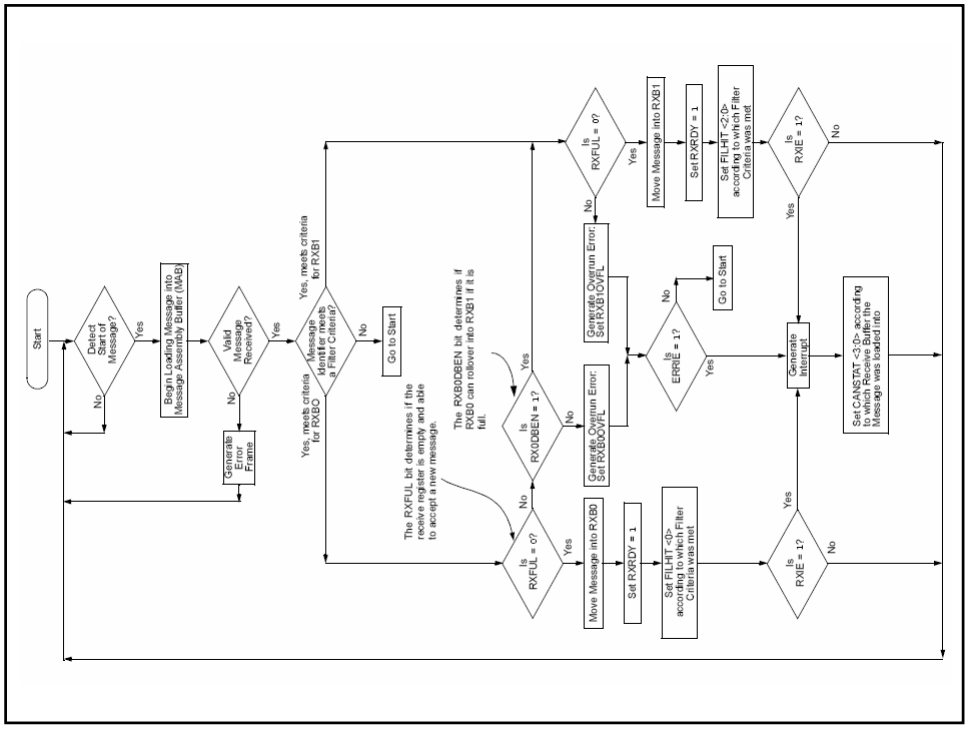
- **I filtri e le mask sono usati per determinare se un messaggio presente sul bus CAN, e quindi nel *message assembly buffer* (MAB), può essere accettato e quindi caricato in uno dei 2 buffer di RX.**
- **Una volta che un messaggio è ritenuto valido dal *protocol engine*, e viene ricevuto e messo nel MAB, il suo campo identificatore è comparato con il valore (configurazione) dei vari filtri del modulo (RXF0 – RXF5).**
- **Se c'è match allora il messaggio è caricato nell'appropriato buffer di RX.**

MODULO CAN - FILTRI E MASK

Mask bit n	Filter bit n	Message Identifier bit n001	Accept or Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Legend: x = don't care

- **Le mask servono per determinare quali bits dell'identificatore del messaggio devono essere presi in considerazione nel matching.**

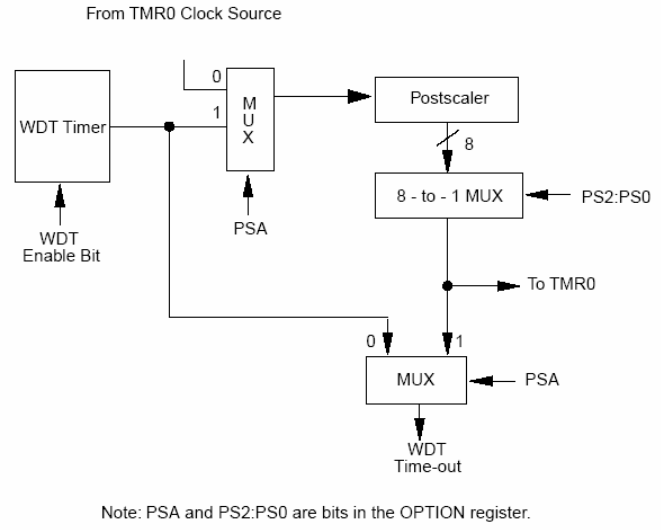


WATCHDOG TIMER e SLEEP MODE

WATCHDOG TIMER

- **Il *Watchdog Timer* (WDT) è un OSC RC *free runnning on-chip* che non richiede nessun componente esterno.**
- **WDT è tenuto separato da ogni altro blocco di temporizzazione interno o esterno al dispositivo e per questo è in grado di funzionare anche quando il normale CK di sistema è fermo, ad es. come conseguenza di un istruzione di SLEEP.**
- **Durante il funzionamento normale del µC un *WDT time-out* genera un RESET del dispositivo.**
- **Se il µC è in SLEEP, invece, un *WDT time-out* causa il risveglio del µC e il ripristino del normale funzionamento (*WDT wake-up*)**

WATCHDOG TIMER



Note: PSA and PS2:PS0 are bits in the OPTION register.

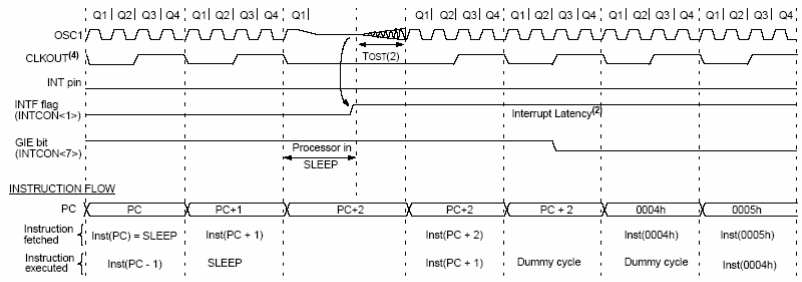
SLEEP MODE (Power Down)

- **Questa caratteristica messa a disposizione da molti microcontrollori permette al dispositivo di mettersi nello stato in cui ha il CONSUMO MINIMO DI CORRENTE (utilissimo ad es. in applicazioni a batteria).**
- **In *sleep mode* l'oscillatore del dispositivo è spento.**
- **Il µC entra in sleep solo dopo aver eseguito una istruzione di SLEEP.**
- **Dopo una istruzione di sleep, se abilitato, il WDT viene azzerato ma continua a funzionare.**
- **Il µC può essere risvegliato dallo *sleep mode* in seguito a: WDT time-out, Reset del dispositivo, Interrupt dovuti a evento esterno, cambio di stato di determinati port pin, fine conversione A/D, Timer1 overflow, ecc…**

SLEEP MODE (Power Down)

- **Una volta risvegliato il µC (a parte il caso di reset) riprende l'esecuzione del programma dall'istruzione immediatamente successiva a quella di sleep.**
- **Questo perché al momento dell'esecuzione della SLEEP nel Program Counter si ha il Pre-Fetch dell'istruzione seguente.**
- **In caso di risveglio dovuto ad un interrupt, il programma salterà alla locazione di memoria riservata al servizio interrupt (es. 0x04 per il PIC16F877)**

SLEEP MODE (Power Down)



Note 1: XT, HS or LP oscillator mode assumed.
2: TOST = 1024TOSC (drawing not to scale) This delay will not be there for RC osc mode.
3: GIE = '1' assumed. In this case after wake- up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.
4: CLKOUT is not available in these osc modes, but shown here for timing reference.