APPUNTI

SUI

MICROCONTROLLORI

FAMIGLIA

PIC16F87X

prof. Duilio De Marco, prof. Ercole Mitrotta, prof. Giorgio Tombola

Indice generale

I MICROCONTROLLORI

Con il termine microcontrollore si intende comunemente un sistema a microprocessore integrato su un unico chip che comprende, oltre alla CPU, una memoria di programma, solitamente di sola lettura (PROM, EPROM o EEPROM), memoria RAM, generalmente di dimensioni ridotte, per i risultati intermedi dell'elaborazione e per lo stack e periferici di I/O vari (porte seriali e/o parallele, timer ecc.). Per quanto appena esposto questi IC vengono anche detti **single chip**.
Con queste caratteristiche, dovrebbe essere evidente che i microcontrollori sono stati concepiti soprattutto per applicazioni industriali di controllo, che possono andare dagli elettrodomestici "intelligenti", ai sistemi di comunicazione o sicurezza, alla strumentazione, all'automazione in campo automobilistico, ecc.

LA FAMIGLIA PIC16F87X

Tra i numerosi tipi sul mercato, i microcontrollori CMOS a 8 bit della famiglia PIC prodotti dalla Microchip Technology, in tempi recenti hanno guadagnato ampia diffusione grazie alla loro concezione innovativa che li rende molto versatili e facili da programmare. I microcontrollori PIC (probabile acronimo per Programmable Integrated Controller) si distaccano dalla struttura di un microprocessore classico, essenzialmente perché sono delle CPU RISC *(Reduced lnstruction Set Controller* - elaborazione con insieme di istruzioni ridotto).
La filosofia RISC *consiste* sostanzialmente nel prevedere *poche e semplici* istruzioni *tutte della stessa lunghezza* e (possibilmente) tutte *richiedenti lo stesso numero di cicli macchina sia per il fetch che per l'esecuzione.* Questa caratteristica (che è soddisfatta nei PIC con eccezione delle istruzioni di salto), unita alla separazione fisica dei canali lungo cui fluiscono istruzioni e dati, permette di ottenere una sovrapposizione (Pipelining) delle fasi di fetch di un'istruzione con quella di esecuzione della precedente "senza buchi" e quindi in modo molto efficiente. Nei PIC, tenuto conto del pipelining, la gestione di un'istruzione (fetch + esecuzione) richiede generalmente un solo ciclo macchina, costituito da quattro cicli di clock. Alla frequenza massima di clock di 20 MHz ciò significa un instruction cycle di 200 ns.

In fig. 1 è riportata la piedinatura del PIC16F876, mentre in fig. 2 quella del PIC16F877.

Fig. 1: piedinatura del PIC 16F876

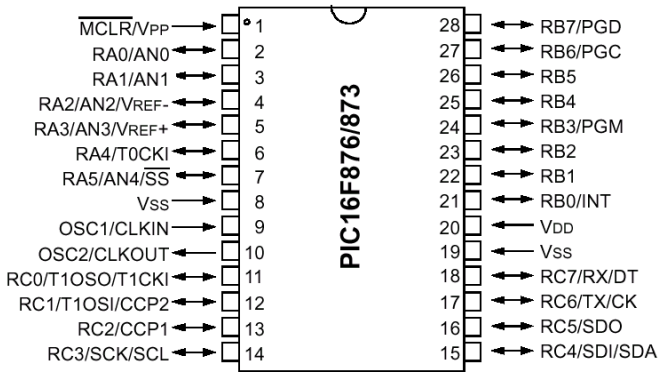
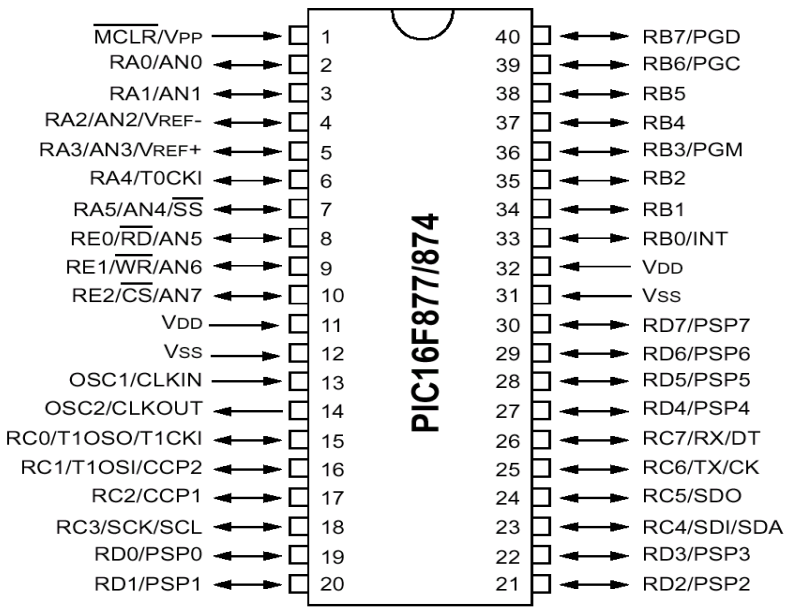


Fig. 2: piedinatura del PIC 16F877



RIASSUNTO CARATTERISTICHE FAMIGLIA PIC16F87X

Fig. 3: Caratteristiche PIC famiglia 16F87X

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

FUNZIONE DEI PIN DEL PIC 16F87X

TABLE 1-1: PIC16F873 AND PIC16F876 PINOUT DESCRIPTION

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP/THV	1	1	I/P	ST	Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input0
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input
RB4	25	25	I/O	TTL	Interrupt on change pin.
RB5	26	26	I/O	TTL	Interrupt on change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I^2C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

PINOUT 16F877

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/ counter. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be soft-ware programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

(continua descrizione pinout 16F877).

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/ output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

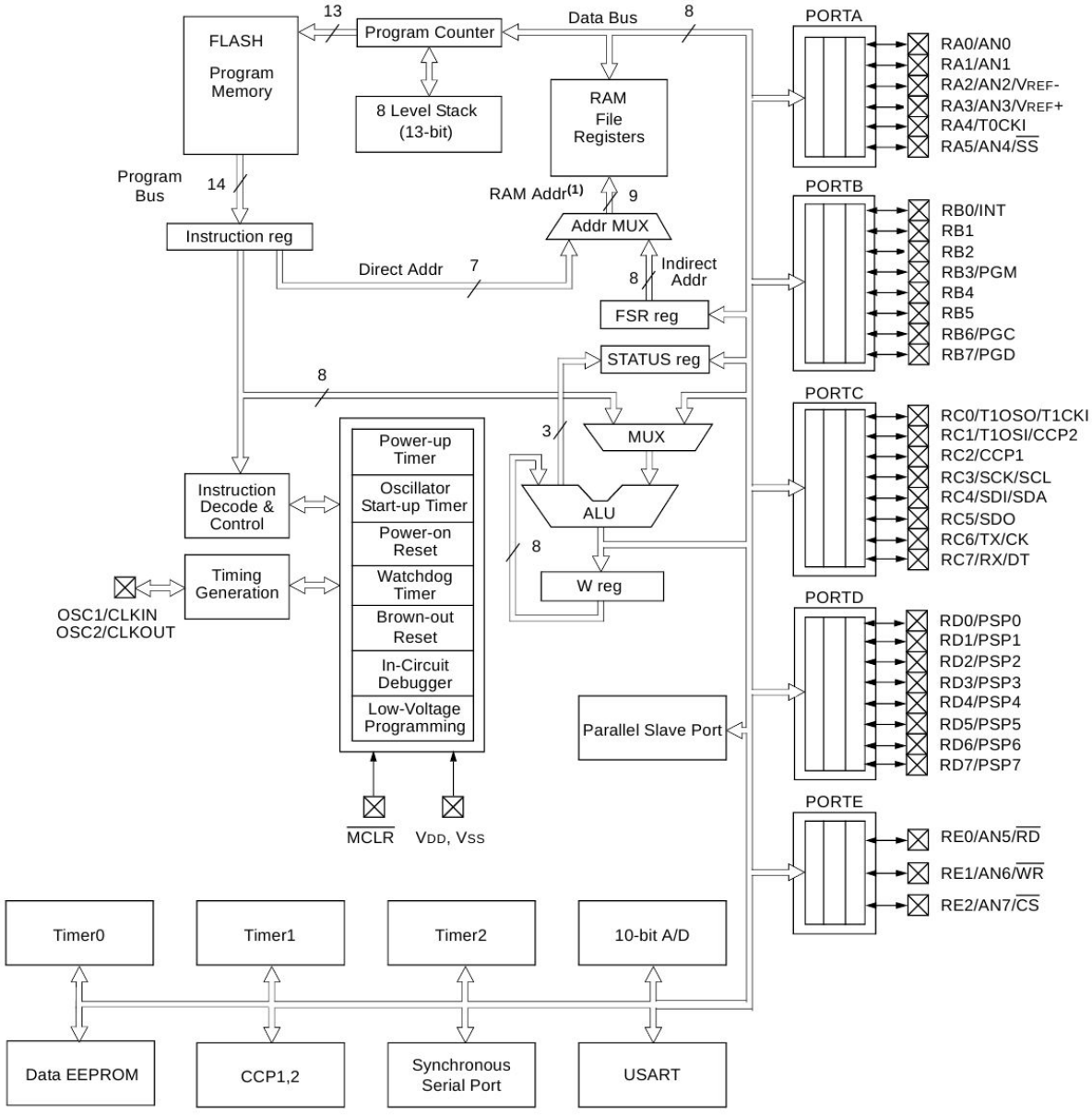
CARATTERISTICHE PRINCIPALI DEL PIC 16F87X

- Set ridotto a 35 istruzioni di una singola parola a 14 bit, tutte di un solo ciclo macchina, tranne le istruzioni di salto, che ne richiedono 2.
- Dati di 8 bit (la struttura Harvard è caratterizzata da separazione fra dati e istruzioni, a differenza dei microprocessori aventi struttura Von Neumann in cui dati e istruzioni condividono la stessa memoria e lo stesso bus).
- Frequenza di clock dalla continua a 20 MHz, generabile con varie opzioni selezionabili da software (quarzo, risuonatore ceramico o gruppo RC).
- Memoria-programma FLASH di 8 kword x 14 bit (quindi fino a 8.000 istruzioni).
- Memoria-dati RAM 368 registri di 8 bit ciascuno.
- Memoria-dati EEPROM 256 byte di memoria dati (scrivibili, con tempi dell'ordine dei ms, durante il normale funzionamento da programma).
- Stack separato dalle aree di memoria dati e programma e non accessibile al programmatore, profondo 8 livelli.
- Possibilità di interrupt (14 sorgenti di interrupt): su richiesta da linea esterna (bit 0 di Port B, RB0/INT), su cambiamento delle linee RB4 ... RB7 di Port B, su overflow del contatore/temporizzatore e infine su completamento scrittura in EEPROM dati.
- linee di I/O con controllo individuale della direzione, TTL compatibili ad alta corrente in uscita (25 mA sink/source), alcune delle quali specializzabili (richiesta interrupt, clock contatore, input con pull-up ecc.).
- Timer0: Contatore/temporizzatore programmabile a 8 bit, con prescaler programmabile di 8 bit.
- Timer1: Contatore/temporizzatore programmabile a 16 bit, con prescaler programmabile di 8 bit, incrementabile in modalità sleep tramite clock esterno.
- Timer2: Contatore/temporizzatore programmabile a 8 bit, con prescaler, postscaler e period register a 8 bit.
- Autoreset all'accensione, senza necessità di componenti esterni (il reset porta il Program Counter alla locazione 000H, da cui parte l'esecuzione del programma).
- Power-up timer: consente di ritardare di alcuni millisecondi l'inizio dell'esecuzione del programma all'atto dell'accensione, per dare modo alla tensione di alimentazione e all'oscillatore di clock di stabilizzarsi.
- Possibilità di mandare (da programma) il microcontroller in *SLEEP,* cioè in uno stato di attesa in cui l'esecuzione del programma si interrompe, le linee di I/O restano nello stato in cui erano precedentemente e il consumo del chip diventa minimo. Dallo stato di *SLEEP* il PIC esce con un livello basso sull'ingresso di reset MCLR, con un reset dal watchdog Timer (vedi oltre) o con un interrupt.
- Watchdog Timer (WDT), attivabile in fase di programmazione, genera un reset ogni intervallo di circa 18 ms (espandibili a 128 x 18 ms = 2,3 s con un prescaler). È utile per sbloccare il microcontrollore da eventuali situazioni critiche o in cui possa finire a causa di errori nel flusso dell'elaborazione.
- Infine, molto importante nelle applicazioni commerciali, è possibile proteggere il codice di programma, cioè renderlo inaccessibile alla lettura, una volta scritto nella ROM.

In Fig. 4: Schema a blocchi 16F877. è riportato lo schema a blocchi interno del 16F877. Si noti la separazione delle memorie di programma e dati e dei relativi percorsi, tipica della struttura Harvard. Al solito, una delle caratteristiche principali di un microprocessore (e anche di un microcontrollore) è la sua dotazione di registri specializzati, fra i quali un accumulatore o registro di lavoro *(working register* W) che intrattiene una relazione speciale con l'ALU, nel senso *che* è *operando e destinazione* di tutte *le operazioni aritmetico/logiche.* Tutti questi registri sono a 8 bit, anche quelli implicati negli indirizzamenti. Tranne W tutti gli altri registri appaiono come locazioni di una RAM dati organizzata a byte, in pagine di 128 byte ciascuna.

Fig. 4: Schema a blocchi 16F877.

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

CONFRONTO TRA LE ARCHITETTURE HARVARD E VON NEUMAN.

L'architettura Harvard ha due memorie separate, una memoria programma e una memoria dati, che sono accessibili con due linee di bus. L'architettura tradizionale von Neumann prevede invece un'unica memoria e un unico bus sia per il programma che per i dati.

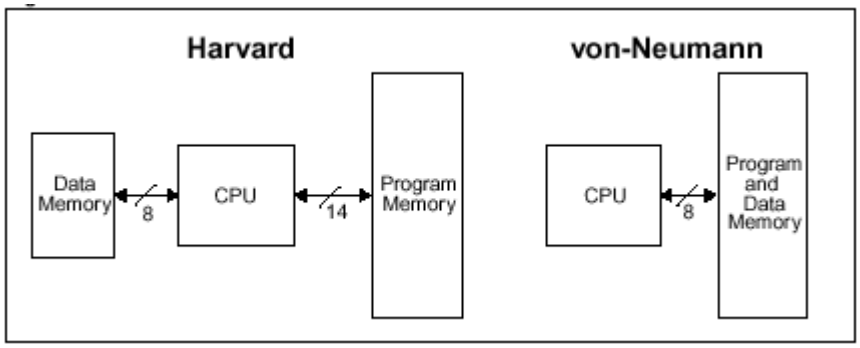


Fig. 5: Confronto Architetture Harvard e Von Neumann

Per eseguire un'istruzione in un dispositivo von Neumann occorre fare uno o più accessi in memoria per prelevare l'istruzione (fase di fetch), attraverso il bus a 8 bit, poi occorre prelevare i dati, elaborarli e molto probabilmente scriverli e per fare tutto ciò occorrono diversi cicli macchina. Invece con i dispositivi Harvard, di nuova concezione, l'istruzione è prelevata (fetch) in un unico ciclo macchina (tutte le istruzioni sono a 14 bit). I due bus separati fanno sì che mentre un'istruzione è eseguita, si può fare contemporaneamente fare il fetch dell'istruzione successiva.

- La struttura Harvard è più veloce di quella von Neumann.
- La struttura von Neumann è però più semplice di quella Harvard.
- Il bus istruzioni della struttura Harvard è più grande (14 bit) di quello dei dati (8 bit). I due bus separati ottimizzano la gestione della memoria.
- Le istruzioni sono a singola parola di 14 bit e sono accessibili con un unico ciclo macchina. In questo modo ogni locazione di memoria della memoria-programma contiene un'istruzione.
- Instruction pipeline: le istruzioni possono essere eseguite contemporaneamente.
- Reduced Instruction Set: set d'istruzioni ridotto (solo 35, micro RISC).
- Register file architecture: il register file/data memory può essere indirizzato in modo diretto e indiretto. Tutti i registri di funzioni speciali e lo stesso PC (registro del program-counter) sono mappati nella memoria dati (data memory).
- Orthogonal (Symmetric) Instructions: le istruzioni ortogonali permettono di riportare il risultato di un'operazione nel registro desiderato usando uno dei due modi di indirizzamento. La natura simmetrica e la scarsità di funzioni speciali rende la programmazione più rapida e più efficiente.

CPU E ALU PIC16F87X.

GENERALITÀ.

La CPU utilizza le istruzioni, situate nella memoria-programma (MPrg) per controllare le operazioni del dispositivo.
Diverse istruzioni operano con i dati situati in memoria-dati (MDt) e per fare ciò si usa l'unità aritmetica-logica ALU.
La ALU, oltre ad eseguire operazioni logico-aritmetiche, controlla lo stato di alcuni bit che si trovano nel registro STATUS.
Il risultato di alcune istruzioni forza lo stato di alcuni bit ad un valore che dipende dal risultato.

FORMATO GENERALE DELLE ISTRUZIONI.

Le istruzioni possono avere una delle quattro forme illustrate in Fig. 6: Formato generale delle istruzioni. Il codice operativo (OPCODE) varia da 3 a 6 bit.

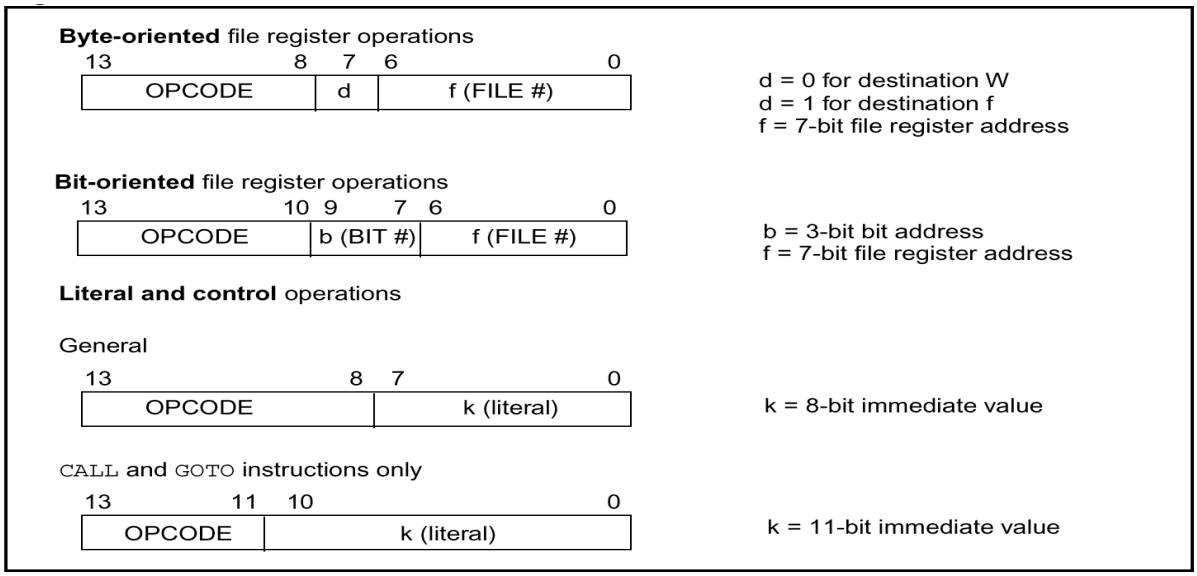


Fig. 6: Formato generale delle istruzioni.

UNITÀ CPU CENTRAL PROCESSING UNIT.

La CPU:

- preleva l'istruzione corrente;
- la decodifica;
- la esegue, lavorando con l'ALU;

inoltre

- controlla il bus-indirizzi della memoria programma;
- controlla il bus-indirizzi della memoria dati;
- accede allo stack.

INSTRUCTION CLOCK.

Ogni istruzione viene completata in un ciclo macchina, detto ciclo istruzione Tcy. Un ciclo Tcy avviene in quattro Q-cicli, Q1, Q2, Q3, Q4. Un ciclo Q è uguale al periodo di clock dell'oscillatore.

Quindi Tcy = 4 Q = 4 Tosc (vedi Fig. 7: Attività cicli Q.)
Durante queste fasi il micro svolge le seguenti attività:

- Q1: decodifica istruzione; o forzamento dell'operazione NOP
- Q2: lettura dato dell'istruzione o NOP
- Q3: elaborazione dato;
- Q4: scrittura del risultato (NOP se il risultato resta in w).

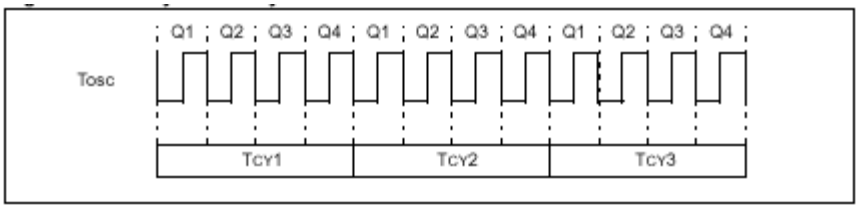


Fig. 7: Attività cicli Q.

ARITHMETIC LOGICAL UNIT (ALU).

Il micro della famiglia PIC16F87x ha un'ALU a 8 bit e un registro accumulatore W (working register) a 8 bit non indirizzabile.
L'ALU esegue funzioni aritmetiche e booleane tra i dati contenuti nel registro W e uno dei registri file, che stanno in memoria RAM. Alcuni sono denominati SFR (Special Function Register) e altri GPR (General Purpose Ram).

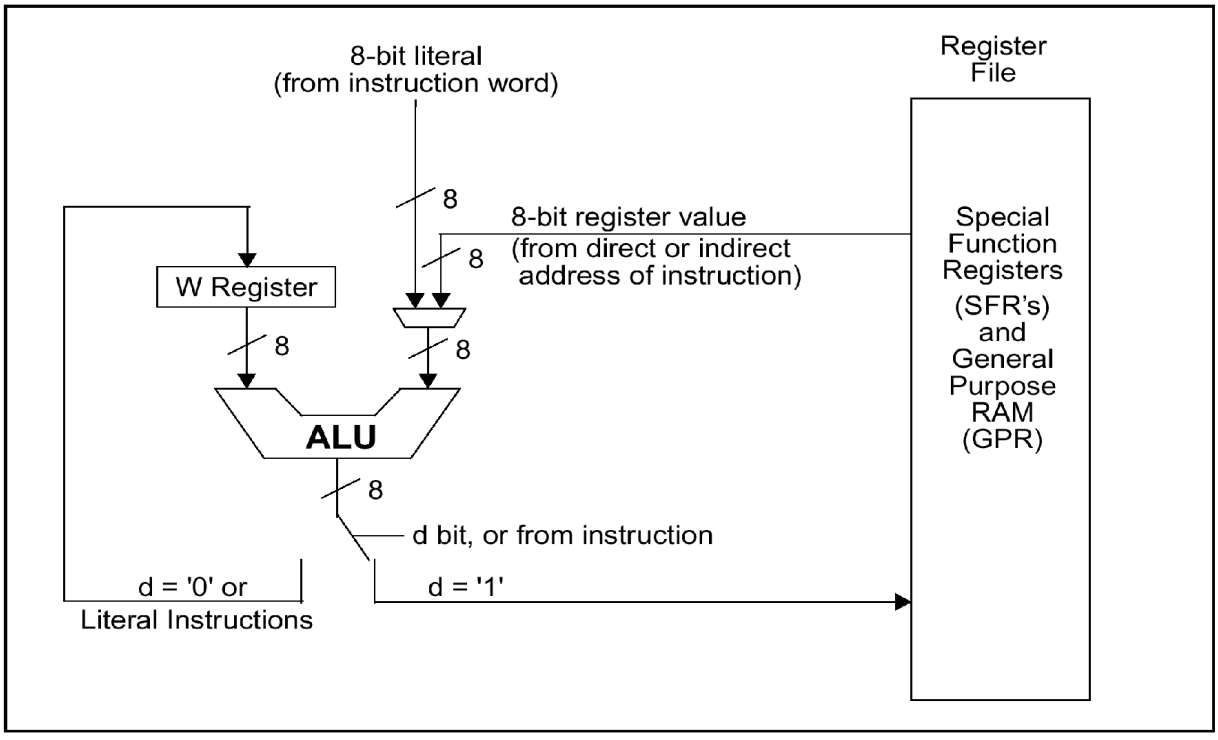


Fig. 8: Operazioni tra l'ALU e il Registro W

La ALU esegue:

- addizione e sottrazioni (in complemento a due)
- operazioni logiche tra bit (bit wise)
- operazioni di shift.

Nelle operazioni a due operandi tipicamente un operando è nel registro W, l'altro è in un registro file oppure è una costante immediata.
Nelle operazioni ad un solo operando questo è nel registro W oppure in un registro file.
A seconda del tipo di istruzione eseguita, l'ALU può influenzare il valore dei bit di Carry (flag C), Digit Carry (DC) e Zero (Z) del registro STATUS:

- DC: carry sul 4° bit; DC=1 se risultato > 15, ovvero se bit 4 è 1;
- C: carry sul 7° bit; C=1 se risultato >255; C=0 se la sottrazione dà risultato negativo
- Z: zero; Z=1 se risultato è uguale a zero.

STATUS REGISTER.

Il registro di stato **STATU**S, contiene:

- lo stato aritmetico dell'ALU;
- lo stato di Reset;
- il banco di memoria selezionato.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 **$\overline{TO}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$**: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Poiché la selezione del banco di data memory è controllata da questo registro, si richiede che esso sia presente in ogni banco (vedi Fig. 9: Organizzazione memoria programma.) ed abbia lo stesso offset in ogni banco.

Il registro di stato può essere la destinazione di ogni istruzione, così come ogni altro registro file. Con alcune eccezioni, in particolare quando il micro sta eseguendo un'operazione che modifica i bit di flag Z, DC, C, prende il controllo del registro e impedisce (disabilita) la scrittura da parte dell'utente.
Ecco come vengono modificati i bit Z, C, DC.

- Z =1 se il risultato dell'operazione è zero;
  =0 se il risultato dell'operazione è diverso da zero;
- DC = 1 se il risultato dell'operazione è >15, ovvero quando il bit 4 del registro diventa 1;
  = 0 se il risultato dell'operazione è <15, ovvero fin quando il bit 4 del registro è 0;
- C =1 se il risultato dell'operazione è >0, ovvero quando il bit 7 del registro è 0;
  =0 se il risultato dell'operazione è < 0, ovvero quando il bit 7 del registro è 1.

REGISTRO OPTION_REG indirizzo 81h, 181h

E' un registro (di lettura /scrittura) che contiene vari bit di controllo per configurare il prescaler del modulo Timer0/WatchDog, l'interrupt esterno INT, l'interrupt TMR0 e i resistori di pull-up del port B.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

ORGANIZZAZIONE MEMORIA PIC16F87X

GENERALITÀ E TIPI DI MEMORIA.

- Il PIC ha due aree di memoria separate: *memoria programma (MPrg) e memoria dati (MDt*).
- Ogni area ha il proprio bus, per cui l'accesso può essere contemporaneo.
- La memoria dati è utilizzata per la gestione dei dati (costanti, variabili, espressioni) e usa la memoria General Purpose RAM e i registri file, detti SFR Special Function Register; ha una capacità di 368 byte di tipo RAM e 256 byte di tipo EEPROM.
- La memoria programma è utilizzata per contenere le istruzioni del programma; è di tipo FLASH EEPROM ed ha una capacità di 8k con word di 14 bit

ORGANIZZAZIONE MEMORIA PROGRAMMA.

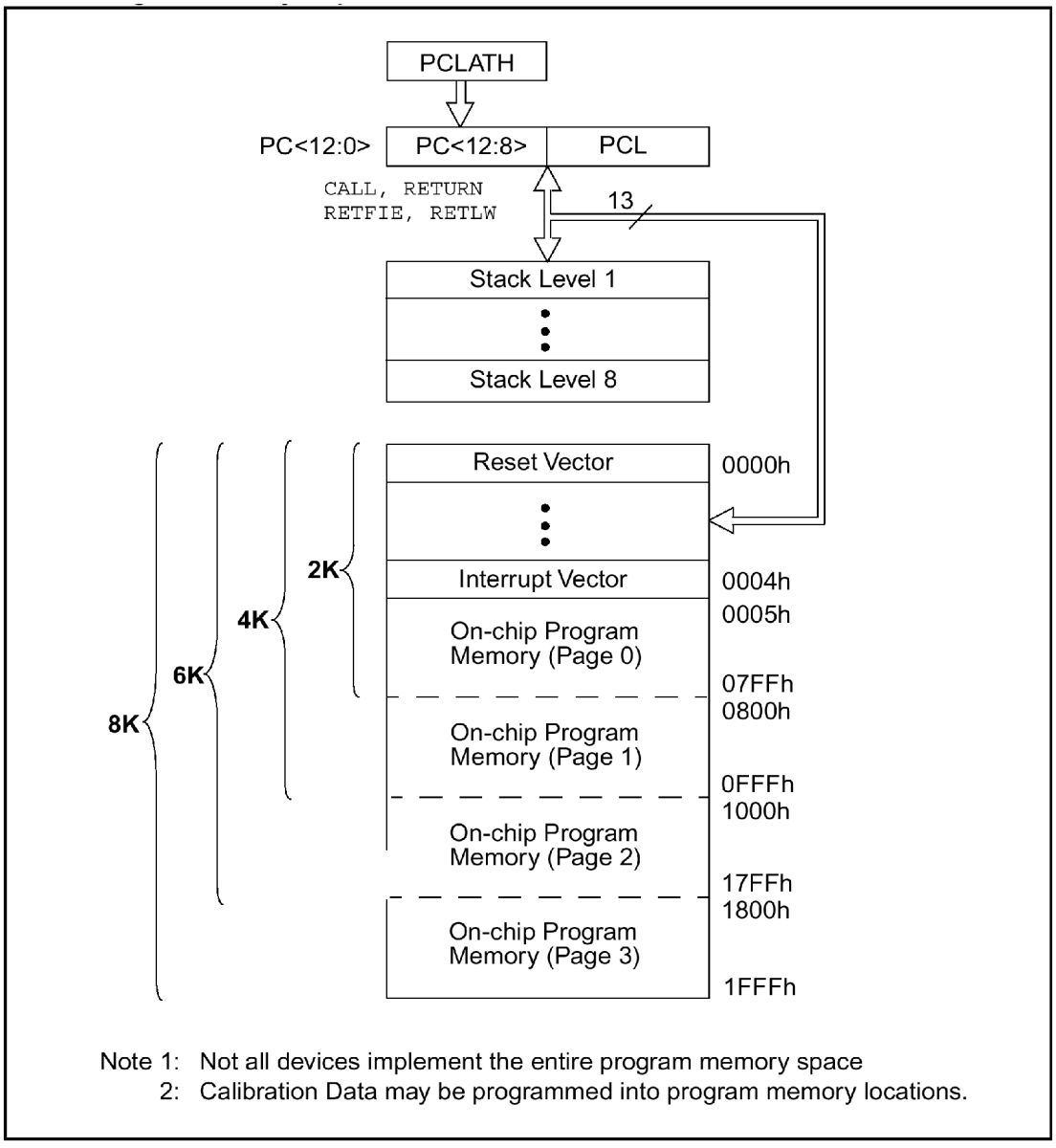


Fig. 9: Organizzazione memoria programma.

- I dispositivi PIC della famiglia 16F87X hanno un Program Counter[1] (PC) a 13 bit, per cui si possono indirizzare 2^{13} = 8.192 (8 kword[2]) locazioni di memoria.
- Il bus della MPrg è a 14 bit, cioè la lunghezza delle parole che codificano le istruzioni è di 14 bit.
- Poiché tutte le istruzioni sono a singola parola, allora il micro può elaborare programmi con circa 8k istruzioni.
- Lo spazio della MPrg di 8 kword è diviso in 4 pagine di 2 kword, i cui indirizzi sono:

1	page 0	0x0000 ... 0x07FF	0 – 2.048
2	page 1	0x0800 ... 0x0FFF	2049 – 4096
3	page 2	0x1000 ... 0x17FF	4.097 – 6.144
4	page 3	0x1800 ... 0x1FFF	6.145 – 8.192

C'è uno stack a 8 livelli (8 locazioni altre di memoria a 14 bit).
Non tutti i micro hanno però 8k di MPrg.
I 2 bit alti del PC servono per impostare in quale pagina di memoria si va a lavorare (indirizzare). Ciò è fatto scrivendo il valore della pagina desiderata in un registro SFR (Special Function Register), chiamato PCLATH (Program Counter LATch High).

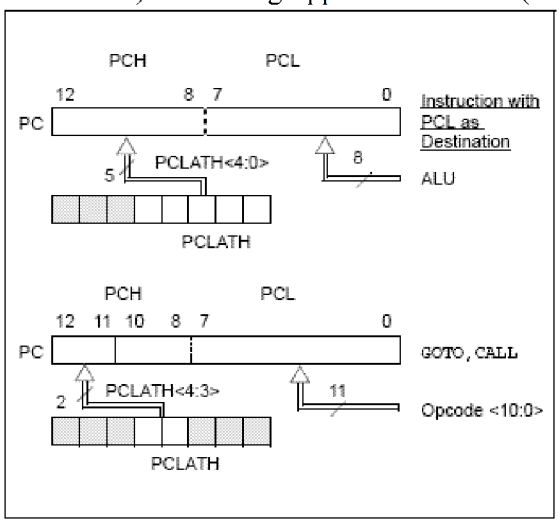
RESET VECTOR indirizzo 0000H
Al reset il micro forza il PC all'indirizzo 0000H della MPrg, cioè si carica in PC l'indirizzo di memoria 0000H. Questo indirizzo è chiamato Reset Vector Address ed è l'indirizzo di memoria a cui il programma deve saltare quando avviene un reset.
Ogni reset pone PCLATH=0, in questo modo viene selezionata la page0 della MPrg.

Osservazione: Le locazioni di memoria 0001H…0003H non sono usate.

INTERRUPT VECTOR indirizzo 0004H
Quando viene riconosciuta l'evento di un interrupt, il PC è forzato all'indirizzo 0004H; a questa locazione ci deve essere la chiamata alla routine di gestione dell'interrupt (Interrupt Service Routine, ISR).

PROGRAM COUNTER E MEMORIA PROGRAMMA
Il program counter (PC) è un registro a 13 bit che contiene l'indirizzo dell'istruzione che sarà eseguita nel ciclo successivo.
Il byte LSB è detto PCL (PC Low) ed è un registro R/W (lettura/scrittura) a 8 bit. Il gruppo di 5 bit MSB (bit 12 ... 8) è detto PCH (PC High). Questo registro non è direttamente R/W e tutti i suoi aggiornamenti avvengono attraverso il registro PCLATH.
Sono necessari 13 bit poiché lo spazio della Memoria Programma è di 8 kword (2^{13} = 8.192); dal momento che le istruzioni CALL e GOTO gestiscono solo gli 11 bit meno significativi, la memoria programma è considerata divisa in 4 pagine di 2kw. Per poter indirizzare una locazione di tale memoria occorre quindi un indirizzo a 13 bit così composto:

- 11 bit per localizzare una locazione in una pagina;
- 2 per selezionare la pagina. Questi bit corrispondono ai bit 3,4 del registro PCLATH



[1] **Program Counter**: registro puntatore alla locazione della memoria di programma che contiene l'indirizzo dell'istruzione da eseguire nel ciclo successivo.
[2] Word: unità elementare (istruzione) che un microcontrollore o un microprocessore può elaborare.

bit4	*bit3*	*page*	*indirizzi locazioni memoria*
0	*0*	*Page0*	*0x0000..0x07FF;*
0	1	Page1	0x0800..0x0FFF
1	0	Page2	0x1000..0x17FF
1	1	Page3	0x1800..0x1FFF

Fintanto che il programma si mantiene al di sotto di 2k istruzioni, i bit PCLATH<4:3> sono uguali a zero e, pertanto, non occorrono accorgimenti particolari per effettuare operazioni di salto. In caso contrario bisogna assicurarsi, prima di ogni istruzione di salto, che i bit di selezione di pagina siano programmati in modo che sia indirizzata la pagina di memoria programma nella quale deve procedere l'esecuzione.
Il registro PCLATH non viene mai aggiornato dal contenuto di PCH, ma i bit 3 e 4 di PCLATH vengono automaticamente caricati al posto dei 2 bit più significativi del program counter ogni volta che viene eseguita un'istruzione di CALL o GOTO.
Quando viene eseguita un'istruzione di RETURN non è richiesta la manipolazione di PCLATH<4:3> perché tutti i 13 bit del Program Counter vengono recuperati dallo stack.

Esempio di chiamata ad una routine in page1 da page0:

```
ORG 0x500
BSF PCLATH,3 ; Seleziono Page1 (800h-FFFh) per la chiamata
CALL SUB1_P1 ; chiama subroutine in Page1 (800h-FFFh)
: ;
: ;
ORG 0x900 ;
SUB1_P1: ; subroutine in  Page1 (800h-FFFh)
: ;
RETURN ; return to Call subroutine in Page0 (000h-7FFh)
```

Quando si effettuano istruzioni con PCL come destinazione (computed GOTO) vengono invece caricati automaticamente tutti i 5 bit di PCLATH nel PCH: si devono prendere le opportune precauzioni ogni volta che si supera un blocco di 256 istruzioni.

STACK (8 LOCAZIONI DI 13 BIT).
Lo stack:

- è un insieme di 8 locazioni di memoria di 13 bit in cui è possibile memorizzare fino a 8 indirizzi di rientro da chiamate a routine e/o interrupt;
- è gestito come un registro a scorrimento LIFO (Last In, First Out); se si effettuano più di 8 chiamate nidificate, la nona determina la perdita della prima e così via senza che sia fornita alcuna segnalazione di errore;
- come spazio di memoria non appartiene né alla memoria programma né alla memoria dati.

Il PC è inserito (pushed) dentro lo stack quando è eseguita un'istruzione di CALL o si genera un evento di interrupt. Dallo stack è prelevato (poped) l'indirizzo top quando viene eseguita una qualsiasi delle istruzioni delle tre istruzioni di ritorno: RETURN, RETFIE o RETLW.

ORGANIZZAZIONE MEMORIA DATI

La memoria riservata alla gestione dei dati è formata da locazioni di memoria di un byte suddivise in due gruppi:

SFR (Special Function Registers): sono usate per operazioni di controllo del micro (periferiche e CPU), ad esempio TRIS e PORT.

GPR (General Purpose Registers): sono locazioni generiche per memorizzare dati e per operazioni veloci

La Memoria Dati è anch'essa suddivisa in 4 banchi di 128 byte per un totale di 512 byte. Ogni banco contiene locazioni sia di tipo SFR sia GPR. Le prime locazioni sono riservate ai registri speciali (SFR), le altre sono di uso generale (GPR).

La selezione del banco avviene tramite i bit STATUS (5 ... 7)

	7	6	5	4	3	2	1	0
STATUS	IRP	RP1	RPO	xxxx	xxxx	xxxx	xxxx	xxxx

Bit 7: IRP Indirect Register Pointer (indirizzamento indiretto)

0: banchi 0 e 1 locazioni di memoria 000H ... 0FFH 0 → 255

1: banchi 2 e 3 locazioni di memoria 100H ... 1FFH 256 → 511

Bit 6,5 RP1 : RP0 Register Pointer (indirizzamento diretto).

RP1	*RP0*	*banco (128B)*	*indirizzi locazioni memoria*	
0	*0*	*Bank0*	*0x000... 0x07F;*	0 → 127
0	1	Bank1	0x080... 0x0FF	128 → 255
1	0	Bank2	0x100... 0x17F	256 → 383
1	1	Bank3	0x180... 0x1FF	384 → 511

Osservazioni.

- Tutta la memoria dati è una RAM statica.
- Alcuni SFR sono presenti in più banchi per facilitare alcune operazioni più frequenti; ad esempio i registri PCL, STATUS, FSR, PCLATH, ecc. Questi registri presenti in più banchi non sono registri doppioni, ma sono solo registri che corrispondono a più di un indirizzo.
- La mappa della memoria RAM (fig 10) ha gli ultimi 16 byte (da 70h ... 7Fh) mappati in tutti e 4 i banchi. Possono essere utilizzati per variabili che lavorano con registri in più banchi, in questo modo si ha una riduzione del codice.
- Per spostare dati da un registro ad un altro, il dato deve passare per il registro w; ciò implica che tutti i movimenti register-to-register avvengono con due istruzioni: una per mettere il dato nel reg. W, l'altra per metterlo nel registro desiderato.

GENERAL PURPOSE REGISTER FILE.
Si può accedere ad ogni registro GPR o in modo diretto o in modo indiretto attraverso il registro FSR (File Select Register; indirizzo: 004h).

mappa MEMORIA DATI (RAM STATICA)

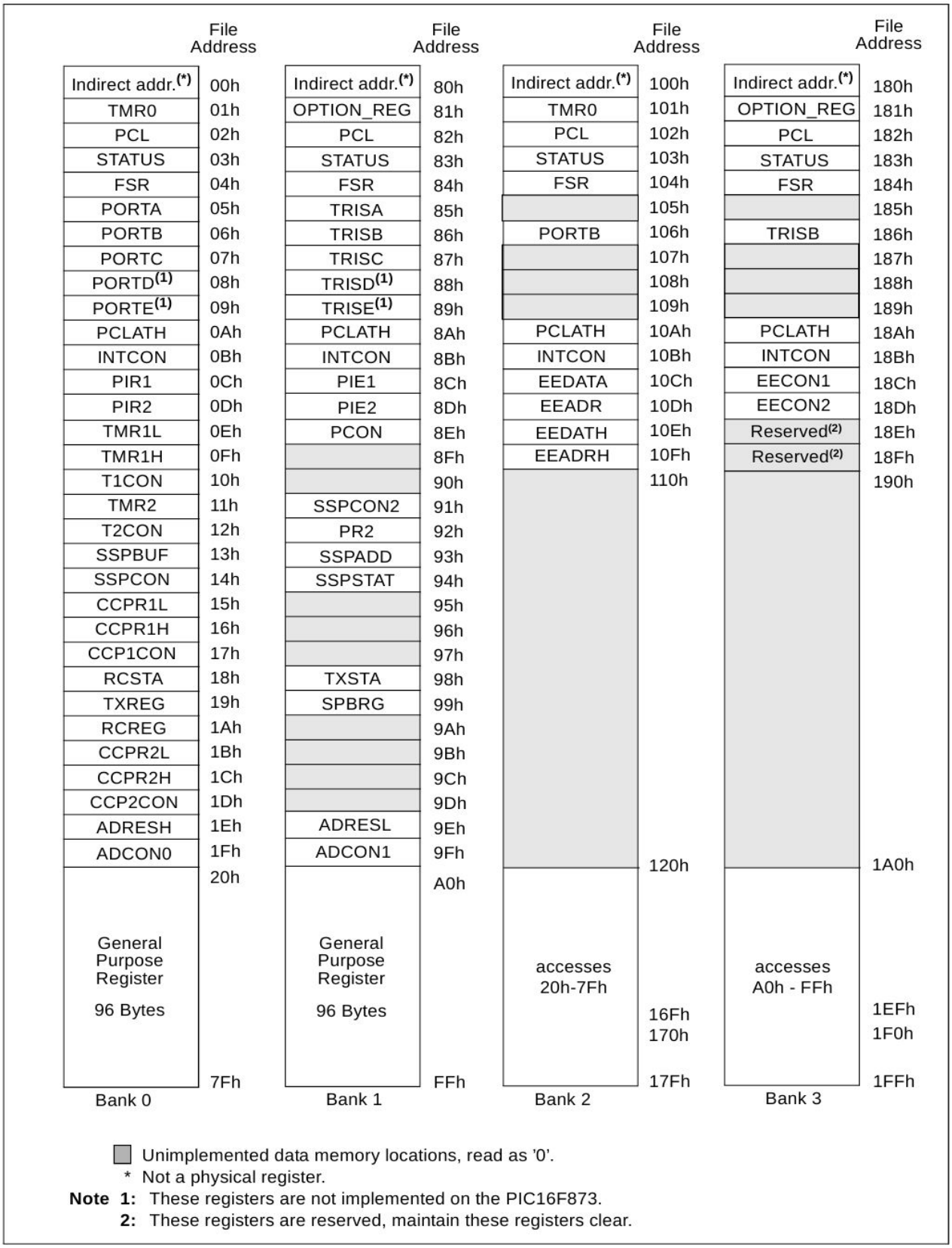


Fig. 10

SPECIAL FUNCTION REGISTER.

Gli SFR sono registri usati dalla CPU e dai moduli periferici per controllare le operazioni del micro. Questi registri sono implementati nella RAM statica. Gli SFR sono suddivisi in due insiemi: quelli per la CPU e quelli per i moduli periferici.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
01h	TMR0	Timer0 Module Register								xxxx xxxx	47
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	29
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	33
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	35
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	36
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
0Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	22
0Dh	PIR2	—	**(5)**	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	24
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	51
11h	TMR2	Timer2 Module Register								0000 0000	55
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	55
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	70, 73
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	67
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	57
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	57
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	58
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	96
19h	TXREG	USART Transmit Data Register								0000 0000	99
1Ah	RCREG	USART Receive Data Register								0000 0000	101
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	57
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	57
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	58
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	116
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	111

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
101h	TMR0	Timer0 Module Register								xxxx xxxx	47
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	26
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
10Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	41
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	41
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						xxxx xxxx	41
10Fh	EEADRH	—	—	—	EEPROM Address Register High Byte					xxxx xxxx	41
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	31
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
18Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	41, 42
18Dh	EECON2	EEPROM Control Register2 (not a physical register)								---- ----	41
18Eh	—	Reserved maintain clear								0000 0000	—
18Fh	—	Reserved maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.
5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

INDIRIZZAMENTO DEI REGISTRI: DIRETTO - INDIRETTO.

L'indirizzamento della RAM dell'archivio registri (memoria RAM dati) può essere *diretto* o *indiretto.*

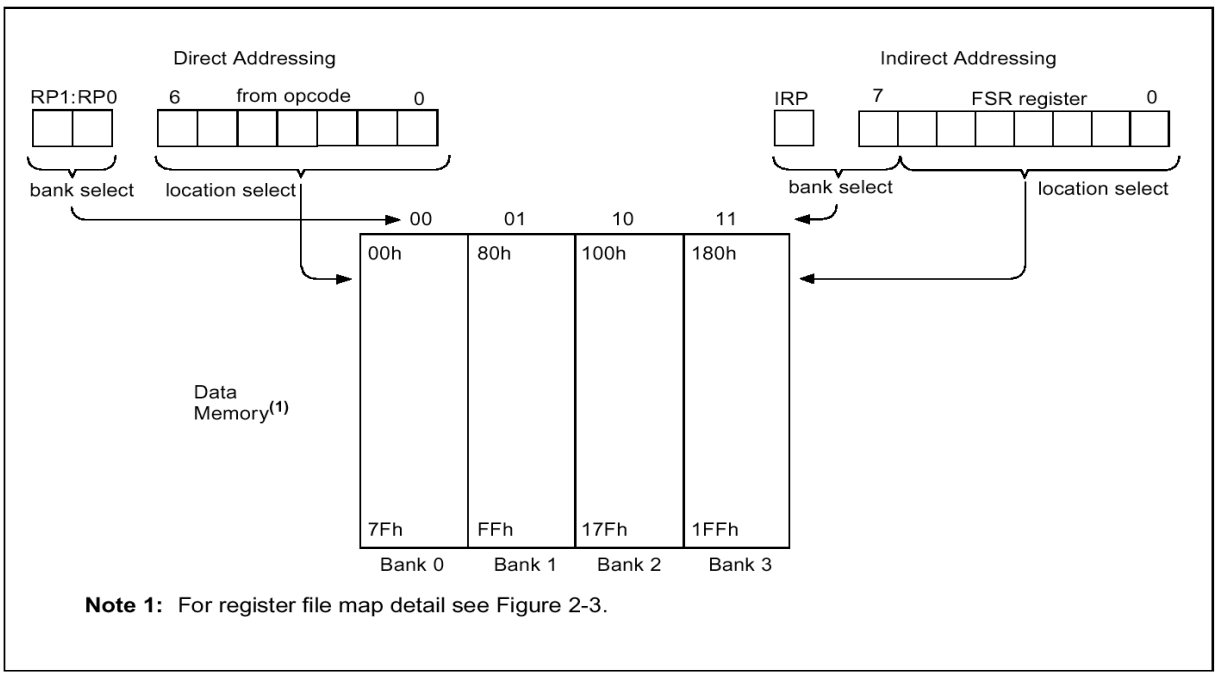
Nell'**indirizzamento diretto** il codice dell'istruzione contiene *direttamente* l'indirizzo del registro (cioè della locazione RAM) su cui opera. Tuttavia nel codice operativo di un'istruzione solo 7 bit sono riservati all'indirizzamento (00H - 7FH); per cui la memoria dati è suddivisa in pagine di 128 byte. Le pagine sono selezionate dai bit *RP0, RP1* del registro STATUS. In altri termini, se per esempio si deve accedere al registro TRISA che è in pagina 1 all'indirizzo 85H, *prima* si deve selezionare la pagina 1 ponendo a 1 il bit *RP0* nel registro STATUS, *poi* riferirsi nell'istruzione al corrispondente indirizzo in pagina 0, cioè 05H. Se si vuole ritornare alla pagina 0, ci si deve ricordare di riportare a 0 *RP0.*
La pagina 0 è quella attiva al reset di accensione, perché in tale occasione *RP0* si azzera automaticamente.

Nell'**indirizzamento indiretto** il registro FSR (indirizzo 04H) fa da indice, cioè contiene l'indirizzo della locazione RAM (registro) su cui si vuole operare. Di per sé *non esistono istruzioni a indirizzamento indiretto.* Se si vuole operare sul registro indirizzato da FSR, è sufficiente riferirsi direttamente al *registro fittizio* INDF (indirizzo 00H). In termini forse più chiari: *una qualunque operazione sul registro INDF agisce in realtà sul registro il cui indirizzo è contenuto in FSR.*

Poiché FSR è a 8 bit, gli indirizzi ricopribili vanno da 00H a FFH, comprendendo così entrambe le pagine 0 e 1. Formalmente, all'indirizzamento indiretto contribuisce anche *IRP* (MSB di STATUS), che seleziona *la coppia* di pagine.

Il procedimento di indirizzamento nei due casi è illustrato graficamente in figura seguente.

Fig. 11 indirizzamento diretto/indiretto.



Esempio: uso indirizzamento indiretto per azzerare la memoria dati RAM dalla locazione 0x20 a 0x2F con un numero minimo di istruzioni.

```
      BCF   STATUS,IRP      ; Indirect addressing Bank0/1
      MOVLW      0x20       ; Initialize pointer to RAM
      MOVWF      FSR        ;
NEXT  CLRF  INDF            ; Clear INDF register
      INCF  FSR,F           ; Inc pointer
      BTFSS FSR,4           ; All done?
      GOTO NEXT             ; NO, clear next
      CONTINUE ;
      ----------            ; YES, continue
```

Esempio di come inizializzare (azzerare) la RAM General Purpose

```
            CLRF  STATUS     ; Clear STATUS register (Bank0)
            MOVLW 0x20       ; 1st address (in bank) of GPR area
            MOVWF FSR        ; Move it to Indirect address register
Bank0_LP
            CLRF INDF0       ; Clear GPR at address pointed to by FSR
            INCF FSR         ; Next GPR (RAM) address
            BTFSS FSR, 7     ; End of current bank ? (FSR = 80h, C = 0)
            GOTO Bank0_LP    ; NO, clear next location
            ;
            ; Next Bank (Bank1)
            ; (** ONLY REQUIRED IF DEVICE HAS A BANK1 **)
            ;
            MOVLW 0xA0       ; 1st address (in bank) of GPR area
            MOVWF FSR        ; Move it to Indirect address register
Bank1_LP
            CLRF INDF0       ; Clear GPR at address pointed to by FSR
            INCF FSR         ; Next GPR (RAM) address
            BTFSS STATUS, C ; End of current bank? (FSR = 00h, C = 1)
            GOTO Bank1_LP    ; NO, clear next location
```

FUNZIONI DI I/O DEI MICROCONTROLLORI.

GENERALITÀ

I pin di I/O, raggruppati in ***Ports***, sono dei dispositivi di ***input/output general purpose*** con lo scopo di interfacciare il microcontrollore con i dispositivi periferici di I/O.
Questi pin permettono al micro di monitorare e di controllare altri dispositivi. Inoltre, per aumentare la flessibilità del micro, alcuni pin sono multiplexati per eseguire alcune funzioni speciali, come rilevare interrupt, conversione A/D, RS232, USART, ecc.

Per la maggior parte dei ports, la direzione dei pin di I/O (input o output) è controllata da un registro speciale, detto **TrisX** (con X = A, B, C, ecc), **data direction register**. Il valore di ogni bit del registro determina la direzione del singolo pin, cioè il bit TRIS<x> controlla il pin PORT<x>, secondo la seguente convenzione:

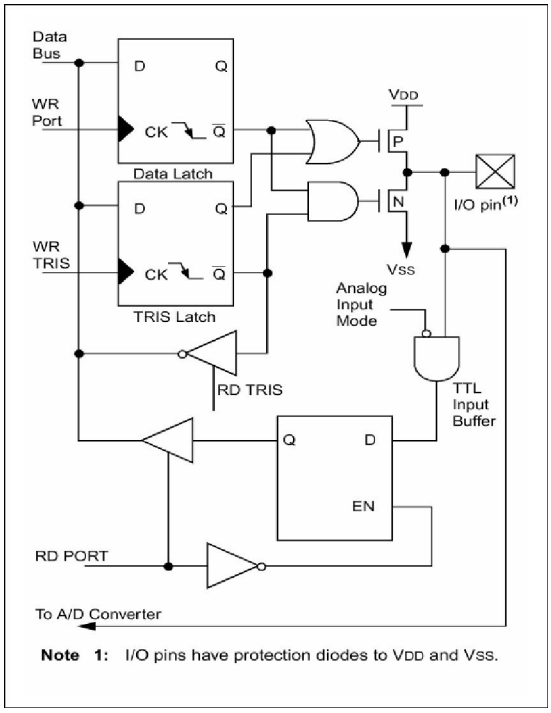
- bit =1 → pin I/O funziona come input;
- bit =0 → pin I/O funziona come output.

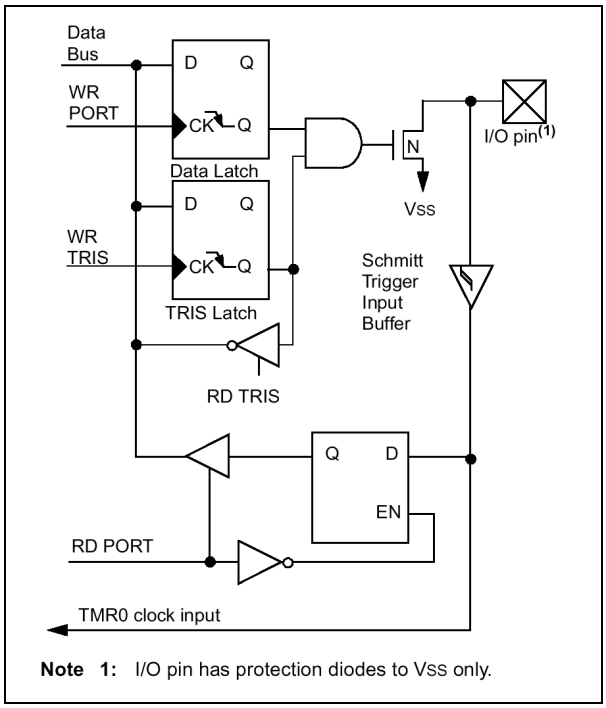
I valori dei pin sono memorizzati nei registri speciali denominati **PortX** (con X = A, B, C, ecc). Nei latch di tali registri viene memorizzato il dato pronto per l'uscita. Quando invece si legge il registro PortX il micro legge il valore dei livelli presenti sui pin di I/O.

REGISTRI PORTA E TRISA.

Il PortA è un port a 6 bit bidirezionale. Il corrispondente data direction register è il registro TrisA.

Fig. 12: Schema a blocchi PortA, pin RA3..RA0 e RA5. **Fig. 13 pin RA4/TM0CKI**





Leggendo PortA si legge lo stato dei pin, mentre scrivendo sul portA si scrive sul latch del port.

OSS: le operazioni di scrittura sono in realtà operazioni read/modify/write, cioè operazioni che comportano prima la lettura del valore del pin dove scrivere, poi la sua modifica ed infine la scrittura nel data-latch.

Il pin RA4 è multiplexato con l'ingresso di clock esterno del Timer0 (RA4/TM0CKI TiMer0 ClocK Input). Perciò il pin RA4/TM0CKI può essere:

- un ingresso a Trigger di Schmitt
- un uscita open drain.

Tutti gli altri terminali della porta A possono essere:

- ingressi: sensibili ai livelli TTL
- uscite: di tipo full CMOS Output driver

I pin RA0.RA3 e RA5 sono multiplexati con input analogici. Le operazioni su questi pin sono selezionabili tramite il registro ADCON1 (A/D Control Register1).

N.B.: Al reset Power On, che è generato quando si applica la tensione di alimentazione al micro, questi pin vengono configurati come ingressi analogici, letti come '0'.

Il registro TrisA controlla la direzione dei pin del PortA, anche se questi vengono configurati come input analogici.

TABLE 3-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input
RA1/AN1	bit1	TTL	Input/output or analog input
RA2/AN2	bit2	TTL	Input/output or analog input
RA3/AN3/VREF	bit3	TTL	Input/output or analog input or VREF
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0 Output is open drain type
RA5/$\overline{SS}$/AN4	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 3-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000

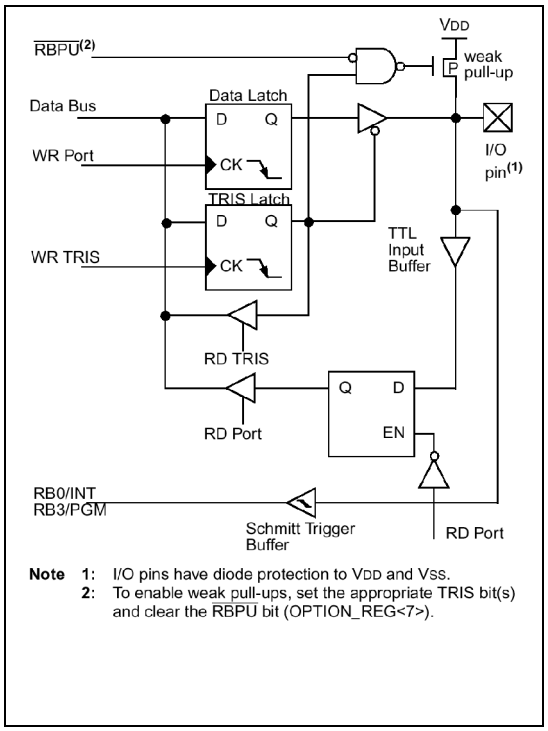
Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by PORTA.

Fig. 14: Funzione svolta dai pin e registri associati al PortA.

REGISTRI PORTB E TRISB.

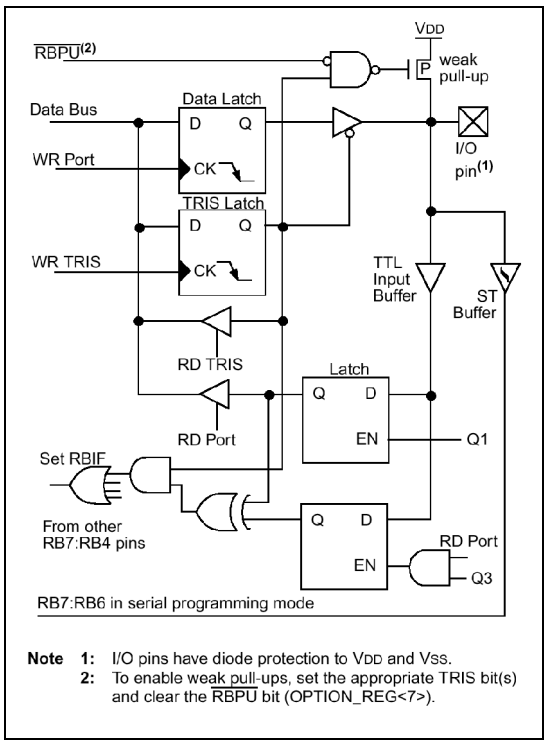
Il PortB è un port a 8 bit bidirezionale. Il corrispondente data direction register è il registro TrisB.

Fig. 15: Schema a blocchi PortB, pin RB3 … RB0



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Fig. 16 pin RB7 … RB4



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Tre pin del PortB sono utilizzati per la programmazione
cioè la scrittura nella memoria programma:
RB3/PGM: pin per la programmazione in LVP (Low Programming Voltage mode);
RB6/PGC pin ICD (In-Circuit Debugger); serve per il clock della programmazione via seriale;
RB7/PGD pin ICD (In-Circuit Debugger); serve per i dati della programmazione via seriale.

Tutti i pin hanno dei circuiti di pull-up *deboli* interni realizzati con MOS-p che possono essere attivati tutti insieme azzerando il bit 7 del registro OPTION_REG (/RBPU). Il pull-up è automaticamente disattivato se il pin viene usato come output. Inoltre viene disabilitato dal Power-on Reset.
Quando è attivo il pull-up l'ingresso eroga una corrente tipica di 250 µA (50 ÷ 400 µA). La corrente è misurata con ingresso collegato a massa.

Se i quattro pin RB7 ... RB4 sono configurati come input, si possono usare come pin di interrupt (sensibili alla transazione di livello H→L o L→H). La relativa segnalazione avviene tramite il bit 0 (RBIF) del registro INTCON.

Il pin RB0/INT, se configurato come input, è un pin sensibile ad un interrupt esterno. La rilevazione avviene quando c'è una transizione di livello. Per la sua configurazione si usa il bit INTEDG (OPTION_REG<6>):
se bit INTEDG = 1 l'interrupt avviene sul fronte di salita L→H;
se bit INTEDG = 0 l'interrupt avviene sul fronte di discesa H→L.

Fig. 17: Funzione svolta dai pin e registri associati al PortA.

TABLE 3-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT	bit0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3/PGM	bit3	TTL	Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB6/PGC	bit6	TTL/ST(2)	Input/output pin (with interrupt on change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST(2)	Input/output pin (with interrupt on change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in serial programming mode.

TABLE 3-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

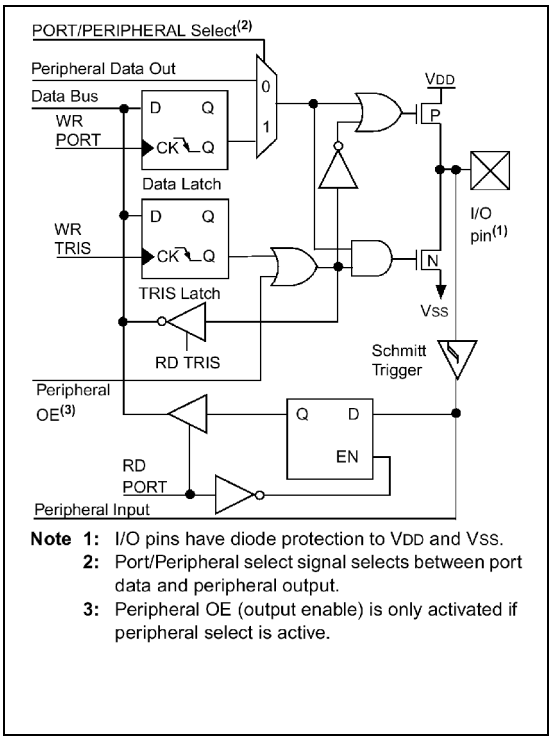
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

REGISTRI PORTC E TRISC.

Il PortC è un port a 8 bit bidirezionale. Il corrispondente data direction register è il registro TrisC.

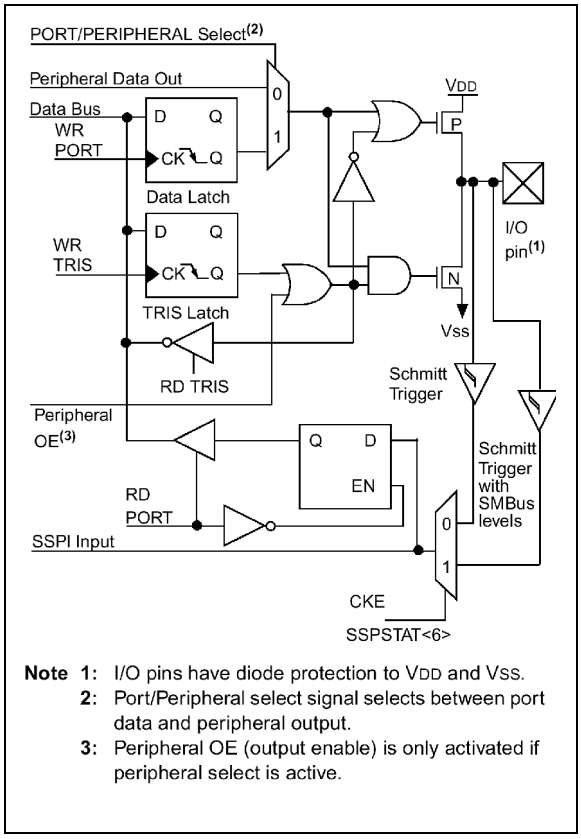
Fig. 18: Schema a blocchi PortC, pin RC0..RC2 RC5 ... RC7. (Peripheral Output Override)



Note 1: I/O pins have diode protection to VDD and VSS.
2: Port/Peripheral select signal selects between port data and peripheral output.
3: Peripheral OE (output enable) is only activated if peripheral select is active.

Il

Fig. 19 pin RC3 ... RC4 (Peripheral Output Override)



Note 1: I/O pins have diode protection to VDD and VSS.
2: Port/Peripheral select signal selects between port data and peripheral output.
3: Peripheral OE (output enable) is only activated if peripheral select is active.

PortC hanno dei buffer a trigger di Schmitt in input. Viene usato per configurare i moduli periferici I^2C, RS232, USART, ecc.

Fig. 20: Funzione svolta dai pin e registri associati al PortC.

TABLE 3-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/PWM1 output
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output
RC6/TX/CK	bit6	ST	Input/output port pin or USART Asynchronous Transmit or Synchro- nous Clock
RC7/RX/DT	bit7	ST	Input/output port pin or USART Asynchronous Receive or Synchro- nous Data

Legend: ST = Schmitt Trigger input

TABLE 3-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged.

REGISTRI PORTD E TRISD.

Fig. 21: Schema a blocchi PortD.

Questa sezione non è presente nel PIC16F876.
Il PortD è un port a 8 bit bidirezionale. Il corrispondente data direction register è il registro TrisD.
Ha gli ingressi con buffer a Trigger di Schmitt. Ciascun pin può essere configurato individualmente come input o come output.
Il PortD può essere configurato modulo Parallel Port Slave.

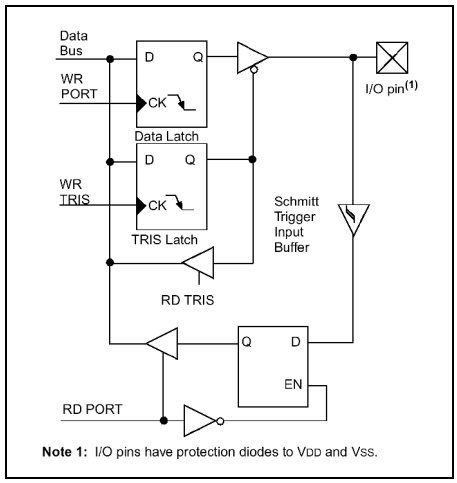


Fig. 22: Funzione svolta dai pin e registri associati al PortA.

TABLE 3-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL(1)	Input/output port pin or parallel slave port bit0
RD1/PSP1	bit1	ST/TTL(1)	Input/output port pin or parallel slave port bit1
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or parallel slave port bit2
RD3/PSP3	bit3	ST/TTL(1)	Input/output port pin or parallel slave port bit3
RD4/PSP4	bit4	ST/TTL(1)	Input/output port pin or parallel slave port bit4
RD5/PSP5	bit5	ST/TTL(1)	Input/output port pin or parallel slave port bit5
RD6/PSP6	bit6	ST/TTL(1)	Input/output port pin or parallel slave port bit6
RD7/PSP7	bit7	ST/TTL(1)	Input/output port pin or parallel slave port bit7

Legend: ST = Schmitt Trigger input TTL = TTL input
Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffer when in Parallel Slave Port Mode.

TABLE 3-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used by PORTD.

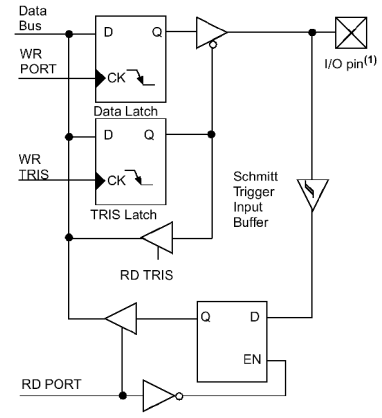
REGISTRI PORTE E TRISE.

Questa sezione non è presente nel PIC16F876.

Il PortE ha tre pin, RE0/RD/AN5, RE1/WR/AN6, RE2/CS/AN7, che sono individualmente configurabili come I/O. Questi pin hanno gli ingressi a buffer Trigger di Schmitt. Possono essere usati come ingressi analogici del convertitore A/D.

Il registro TrisE controlla la direzione dei pin RE (bit 2 ... 0) e le operazioni sul modulo periferico PSP (bit 7 ... 4).

Fig. 23: Schema a blocchi PortE



Note 1: I/O pins have protection diodes to VDD and VSS.

Fig. 24: Funzione svolta dai pin e registri associati alle sporte.

TABLE 3-9: PORTE FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/$\overline{RD}$/AN5	bit0	ST/TTL(1)	Input/output port pin or read control input in parallel slave port mode or analog input: $\overline{RD}$ 1 = Not a read operation 0 = Read operation. Reads PORTD register (if chip selected)
RE1/$\overline{WR}$/AN6	bit1	ST/TTL(1)	Input/output port pin or write control input in parallel slave port mode or analog input: $\overline{WR}$ 1 = Not a write operation 0 = Write operation. Writes PORTD register (if chip selected)
RE2/$\overline{CS}$/AN7	bit2	ST/TTL(1)	Input/output port pin or chip select control input in parallel slave port mode or analog input: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port Mode.

TABLE 3-10: SUMMARY OF REGISTERS ASSOCIATED WITH PORTE

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used by PORTE.

CONVERSIONE A/D

Il convertitore A/D è gestito dai registri **ADCON0** e **ADCON1** e, anche se non deve essere utilizzato, occorre configurarlo per gestire correttamente le linee del PORTA.
Per stabilire la funzione delle linee del PORTA bisogna impostare i 4 bit meno significativi del registro **ADCON1**, denominati **PCFG0, PCFG1, PCFG2** e **PCFG3** secondo lo schema riportato nella tabella seguente.

REGISTRO ADCON1

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.
2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

V_{REF+} e V_{REF-} sono la tensione massima e minima utilizzate nella conversione.

Vediamo adesso la funzione del registro **ADCON0**

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

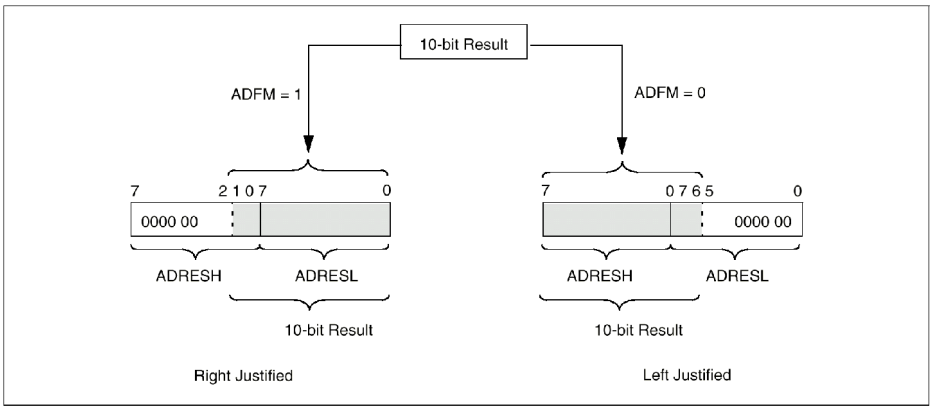
bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

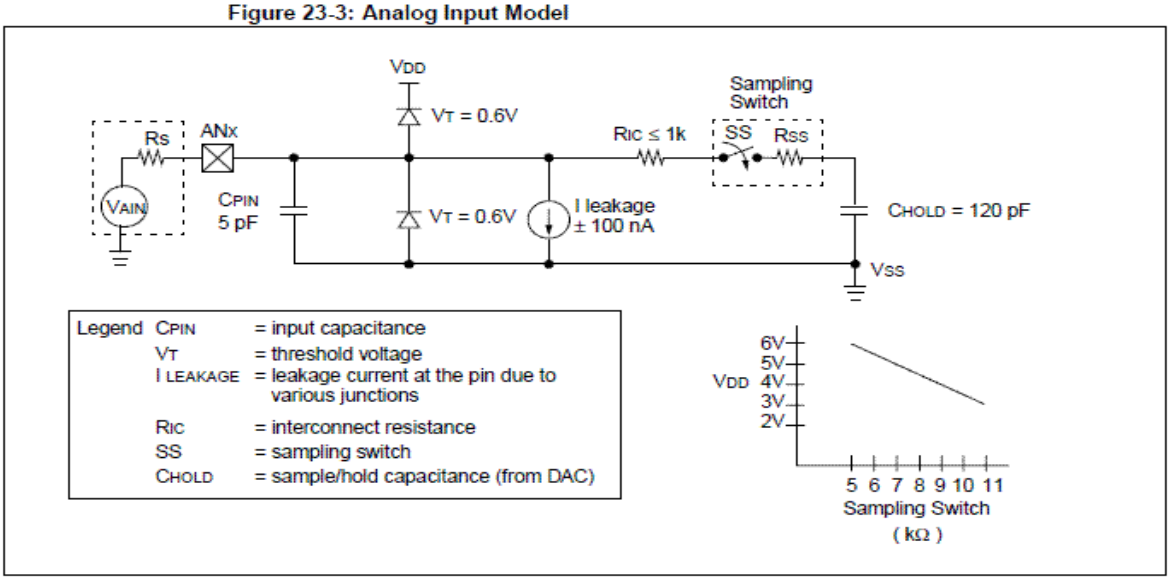
Il bit 0 (**ADON**) accende o spegne il convertitore
Il bit 1 non viene utilizzato
Il bit 2 (**GO/DONE**) deve essere posto a 1 per avviare la conversione e viene automaticamente azzerato alla fine della conversione.
I bit 3, 4 e 5 determinano l'ingresso la cui tensione è convertita.
I bit 6 e 7 (**ADCS0** e **ADCS1**) servono per selezionare la frequenza di conversione.
Il risultato della conversione è a 10 bit ed è memorizzato nei registri **ADRESH** e **ADRESL**. Tramite il bit 7 di ADCON1 (ADFM) si determina il formato di uscita del convertitore: allineamento a destra (ADFM=1) con 8 LSB in ADRSL e i 2 MSB in ADRESH; allineamento a sinistra (ADFM=0) con 8 MSB in ADRSH e i 2 LSB in ADRESH.



PRINCIPIO DI FUNZIONAMENTO DEL CONVERTITORE A/D

Vediamo come è costituito il modulo convertitore A/D all'interno del PIC:

Figure 23-3: Analog Input Model



Sulla sinistra abbiamo la sorgente che fornisce il segnale analogico da campionare (V_{AIN}), collegata al pin ANx. Tale sorgente ha una sua impedenza (Rs) che non va trascurata. Il datasheet dice difatti che tale impedenza non deve essere superiore a 10KΩ per poter avere una corretta acquisizione. Subito dopo il pin ANx abbiamo quindi un condensatore (C_{PIN}) che ha la funzione di filtrare i disturbi, e due diodi di protezione che scaricano eventuali tensioni superiori a V_{DD} o inferiori a V_{SS}. Segue quindi il vero e proprio circuito di campionamento, preceduto da una leggera perdita di corrente (I leakage) dovuta alle giunzioni interne.

In condizioni normali, lo switch di campionamento SS è chiuso permettendo a C_{HOLD}, di potersi caricare allo stesso livello di tensione presente sull'ingresso analogico da convertire. Lo Switch di campionamento ha anch'esso una propria impedenza (Rss) che è inversamente proporzionale alla tensione V_{DD} di alimentazione del picmicro.

Quando viene avviata la conversione (ADGO=1), SS viene aperto in maniera tale da isolare C_{HOLD} dalla linea e avviare quindi il processo di conversione che permette di tramutare il livello di tensione presente in C_{HOLD} in un valore digitale. La conversione del valore analogico in una parola di 8 o 10 bit viene eseguita tramite un metodo chiamato ad approssimazioni successive.

Al termine della conversione il valore digitale è presente nei registri ADRESH e ADRESL, viene settato il **flag di interrupt** di fine conversione analogico/digitale (**ADIF**), ADGO viene rimesso a zero e SS viene richiuso per permettere a CHOLD di ricaricarsi e tenersi quindi pronto per un'altra eventuale conversione.

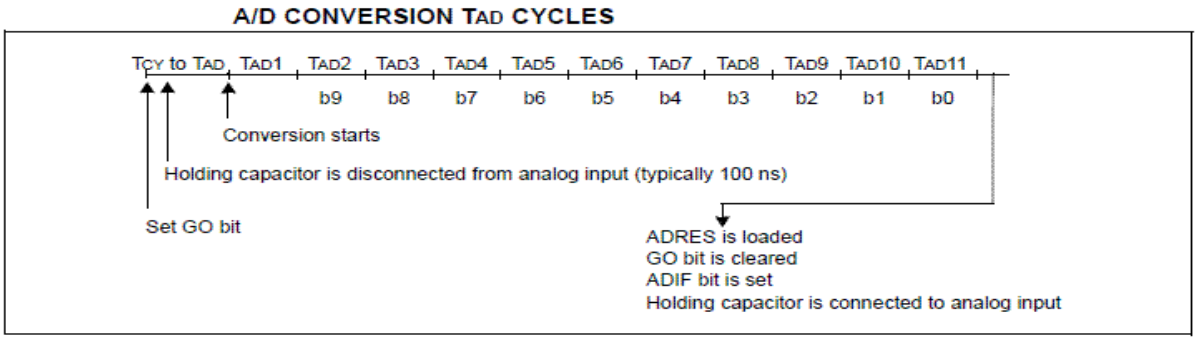
Per effettuare quindi una corretta acquisizione e conversione del segnale bisogna rispettare alcune tempistiche ben precise: abbiamo un **tempo di acquisizione**, che è necessario per poter caricare a piena capacità il condensatore di campionamento, ed un **tempo di conversione**, che inizia quando settiamo ADGO e termina quando viene settato ADIF, e rappresenta il tempo necessario al modulo A/D per poter effettuare l'operazione di conversione. La somma dei due tempi prende il nome di **tempo di campionamento**.

Le impedenze Rs e Rss influiscono sul tempo di acquisizione in maniera non trascurabile in quanto il tempo di carica di un condensatore è uguale a 5*R*C.

Quanto minore è l'impedenza di ingresso del segnale, meno tempo ci vorrà per caricare il condensatore e quindi tanto inferiore sarà il tempo di acquisizione. Con un'impedenza di ingresso di 50Ω si ha un tempo di acquisizione (tempo di carica di C_{HOLD}) tipico di circa 10.6µS, con un'impedenza di 10KΩ il tempo tipico sale a 19.7µS.

Tale tempo, purtroppo, non viene gestito in maniera automatica dal pic e si capisce, quindi, che, appena finita una conversione non possiamo avviarne immediatamente un'altra perchè non diamo al condensatore di campionamento il tempo necessario per ricaricarsi. Tra una conversione e l'altra, pertanto, dovremo rispettare un'attesa che è ragionevole scegliere tra 10 e 20µS.

Il tempo di conversione, invece, è funzione di un parametro chiamato **T_{AD}**, che rappresenta il tempo necessario per convertire un unico bit o, in altre parole, **il tempo di conversione per bit**. La conversione AD richiede un tempo di $12T_{AD}$ per una conversione completa a 10bit:



T_{AD} rappresenta in pratica la frequenza di clock scelta per il convertitore A/D (impostata tramite i bit ADCS2:ADCS0 nei registri ADCON1 e ADCON0). La scelta del T_{AD} va fatta in maniera accurata per alcuni semplici motivi:

- Bisogna che sia rispettato, per T_{AD}, un **valore minimo di 1.6µS**. Quindi un clock troppo rapido non potrebbe rispettare tale requisito.
- T_{AD} non deve nemmeno essere troppo elevato: insieme al segnale da campionare viene difatti inevitabilmente acquisito anche del rumore e con tempi maggiori tale fattore aumenta sempre di più in maniera non trascurabile; inoltre potrebbe accadere che il condensatore di campionamento si scarichi prima che la conversione sia terminata. In entrambi i casi il valore restituito dal convertitore è sicuramente corrotto.

Nella seguente tabella possiamo vedere alcuni valori di TAD in funzione del quarzo utilizzato per il clock del pic e del settaggio di ADCS1:ADCS0 (Nota: tale tabella non riporta tutti i valori di clock per il convertitore AD ed assume che ADCS2=0)

AD Clock Source (TAD)		Device Frequency			
Operation	ADCS1:ADCS0	20 MHz	5 MHz	1.25 MHz	333.33 kHz
2Tosc	00	100 ns(2)	400 ns(2)	1.6 µs	6 µs
8Tosc	01	400 ns(2)	1.6 µs	6.4 µs	24 µs(3)
32Tosc	10	1.6 µs	6.4 µs	25.6 µs(3)	96 µs(3)
RC	11	2 - 6 µs(1,4)	2 - 6 µs(1,4)	2 - 6 µs(1,4)	2 - 6 µs(1)

Notiamo alcuni valori ombreggiati: tali impostazioni non possono essere utilizzate perchè T_{AD} risulta troppo breve o troppo lungo e causa quindi problemi per quanto detto prima.
Oltre al valore di T_{AD} derivato dal clock è possibile selezionare una modalità in cui T_{AD} viene ricavato da un circuito oscillatore RC interno al convertitore. Nei sistemi in cui il dispositivo entrerà in modalità sleep dopo l'avvio della conversione A/D e contemporaneamente operanti ad un clock superiore ad 1MHz, è richiesto l'utilizzo della sorgente di clock RC. In questo modo, il *rumore digitale* proveniente dai moduli in SLEEP viene bloccato: questo metodo fornisce l'accuratezza maggiore. Si capisce quindi che il convertitore AD è l'unico capace di continuare a funzionare anche quando tutto il resto del pic è "addormentato".
La sorgente di clock RC fornisce un T_{AD} tipico di 4µs (oscillante tra 2 e 6µS).
Dalla tabella precedente possiamo quindi vedere che per un pic operante alla frequenza di 20MHz è giusto scegliere la sorgente di clock RC oppure 32Tosc (che fornisce un TAD proprio di 1.6µS). Per altri valori di quarzo, e altre impostazioni fare i calcoli è semplice. Basta calcolare T_{OSC} (che è l'inverso della frequenza del quarzo del picmicro) e quindi moltiplicarlo per 2, per 4, per 8 ecc e vedere quindi qual'è il valore migliore.

TIMER E CONTATORI DI EVENTI

IL TIMER 0

Questo modulo è un contatore di tempo, o di eventi, a 8 bit. Il registro usato per il conteggio è detto TMR0.
In questa periferica si possono vedere 4 sezioni distinte:

1. l'ingresso del clock del contatore
2. il prescaler a 8 bit
3. il modulo del watchdog
4. il registro a 8 bit TMR0

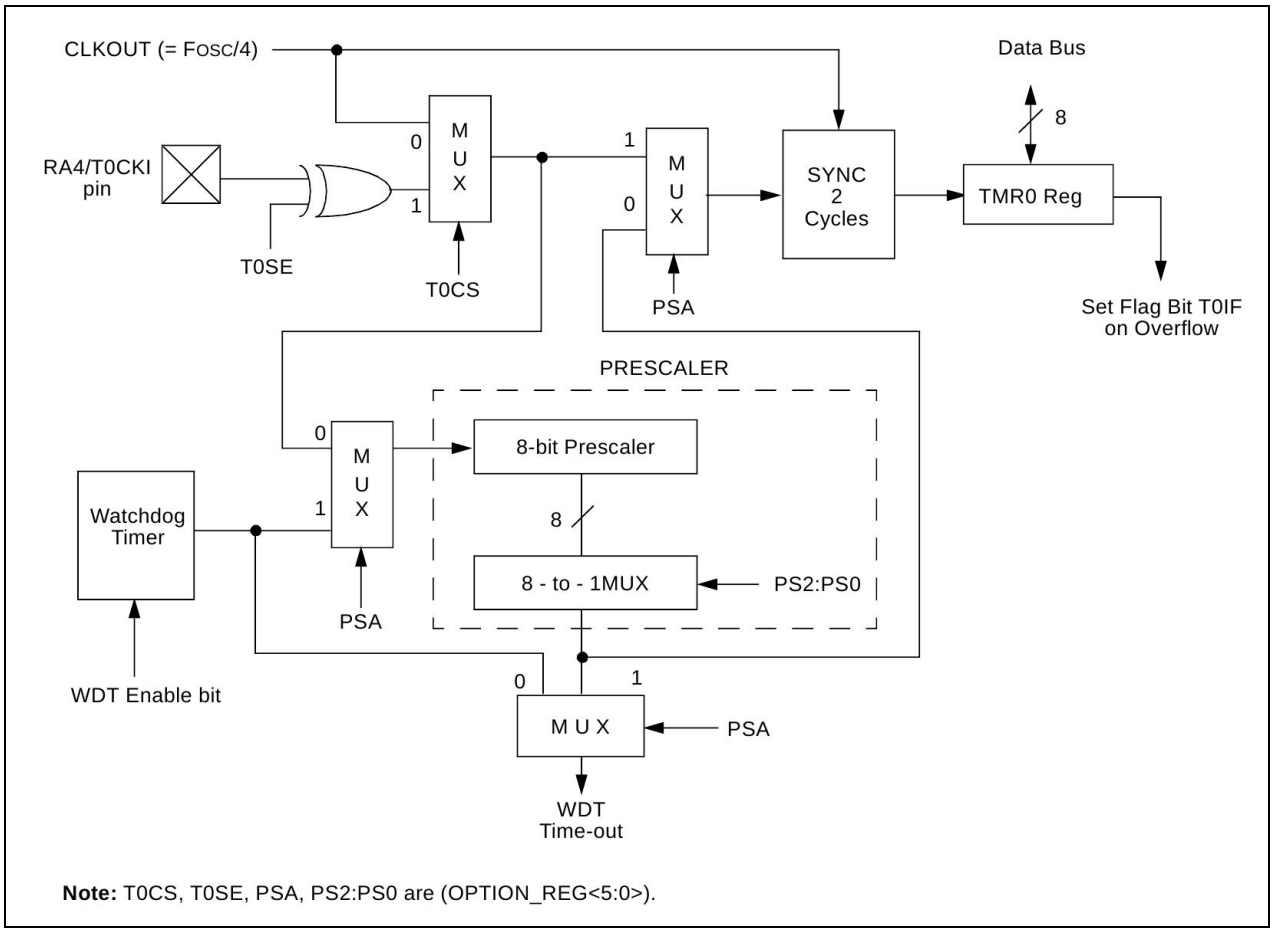


FIG. 5.2-1 - TIMER0

Questo dispositivo è assistito dai bit 0:5 del registro OPTION_REG

OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

FIG. 5.2-2

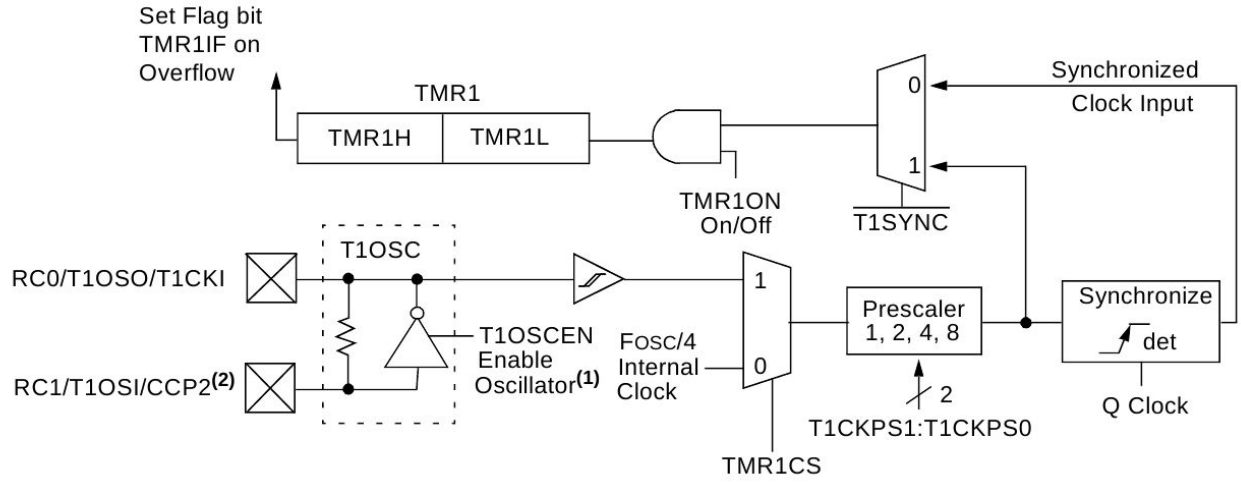
Partendo dalla sezione d'ingresso del clock possiamo vedere che il bit **T0CS** quando è a 0 seleziona come clock del timer il clock di sistema, mentre se è a 1 abilita il clock esterno presente sul piedino 6 (denominato RA4/T0CKI). Il bit **T0SE** permette di selezionare il fronte di salita, del clock esterno, con cui fare procedere il conteggio. Osserviamo che questo dispositivo si presta sia per un conteggio di tempo (sia quando si abilita il clock interno, sia quando si abilita un clock esterno), sia per un conteggio di eventi, quando al piedino 6 fa capo un segnale che commuta per segnalare un evento.
Il prescaler è invece un dispositivo che permette di "dividere" la frequenza di clock (attraverso la scelta opportuna dei bit **PS2:PS0**, vedi figura 4.2-3) e quindi permette di incrementare il contatore con frequenze inferiori. Questo modulo è comune sia al TIMER0 che al Watchdog, quindi la selezione, attraverso il bit PSA, dell'utilizzo per uno ne esclude l'uso da parte dell'altro. (vedi figura 4-1.2)

PS2:PS0: Prescaler Rate Select bits	
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

FIG. 5.2-3

Il **watchdog** è una periferica a se stante rispetto al TIMER0 che ha in comune con esso l'utilizzo del solo Prescaler. Il Watch Dog Timer è un oscillatore interno al PIC, ma completamente indipendente dal resto del circuito, il cui scopo è rilevare eventuali blocchi del flusso normale del programma e attraverso un reset del microcontrollore di ripristinare la normale esecuzione. (Per maggiori chiarimenti sul watchdog fare riferimento al datasheet).
Infine è nel registro TMR0 che è contenuto il dato del conteggio. Questo registro è accessibile sia in scrittura che in lettura, questo comporta sia la possibilità di fare partire il conteggio da un valore desiderato, sia la possibilità di osservare lo stato del conteggio. Essendo a 8 bit, una volta raggiunto il valore 255 all'incremento successivo il contatore si azzera andando in overflow. L'overflow del timer 0 genera un interrupt segnalato dal bit **T0IF** del registro INTCON.

TIMER 1



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

Questo modulo è un contatore di tempo o di eventi a 16 bit. Il registro a 16 bit è detto TMR1 ed è formato da due registri a 8 bit (**TMR1H:TMR1L**) che possono essere letti o scritti. Dopo l'accensione del PIC il valore presente in questi registri è sconosciuto e quindi si deve pensare di aggiornarli se si intende usare il timer1. Questo modulo permette la generazione di un interrupt quando si verifica l'overflow del registro **TMR1** con la segnalazione nel flag **TMR1IF** nel registro **PIR1**.

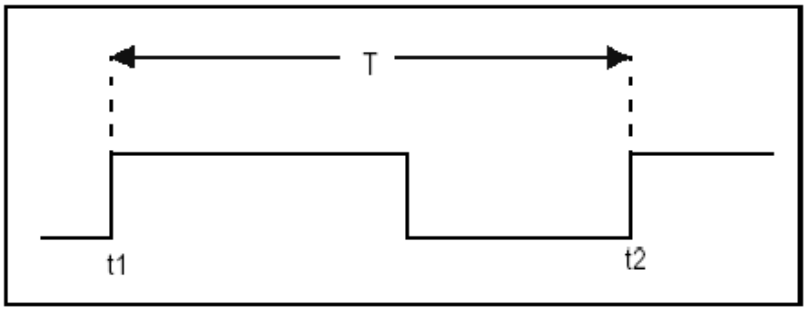
Il timer1 è assistito dal registro **T1CON** che permette la scelta della divisione del prescaler, l'abilitazione o la disabilitazione dell'incremento del registro TMR1 e la scelta del tipo di clock da usare. Caratteristica peculiare di questo timer è la possibilità di collegare un circuito oscillatore dedicato e permettere, quindi, un conteggio sia sincrono che asincrono, alla frequenza desiderata.
E' anche possibile effettuare un conteggio di impulsi provenienti da un altro circuito esterno. In questo caso si utilizza un solo ingresso per il timer1 e cioè quello che ha il contrassegno **T1CKI** (Timer1 Clock Input – che è posizionato su T1OSO). Se si sfrutta questa opzione, T1OSCEN va posto a zero in maniera da disattivare la circuiteria di oscillazione. Per incrementare il Timer1 con l'oscillatore/impulsi esterni (modalità counter) o con l'oscillatore di sistema ($F_{OSC}/4$ – modalità timer) si deve porre rispettivamente a 1 o a 0 il bit **TMR1CS** (Timer1 Clock Source – bit 1 di T1CON). Segue quindi il prescaler (divisore di frequenza) da impostare con i due bit **T1CKPS0-T1CKPS1** (Timer1 ClocK PreScaler bit 0 e 1 – bits 4 e 5 di T1CON): permettono 4 possibili fattori di divisione : 1, 2, 4 e 8.
Segue quindi un selettore che permette di far funzionare il Timer1 in modalità sincrona o asincrona: in modalità sincrona il Timer1 è sincronizzato con l'oscillatore di sistema il che vuol dire che è in grado di rilevare la condizione di SLEEP della CPU. Questo significa che, se la CPU va in sleep, allora va in"letargo" anche il Timer1. In modalità asincrona, invece, il Timer1 continua a funzionare nonostante tutto il resto sia bloccato. Questa modalità di funzionamento è sicuramente vantaggiosa e consente, in alcune applicazioni, di continuare a mantenere il tempo risparmiando corrente. La modalità asincrona si abilita ponendo il bit **T1SYNC** (bit 2 di T1CON) a 1.

Per ottenere un segnale che scandisca i secondi si può utilizzare un interrupt del Timer1 che scatti ogni secondo. Utilizzando il Timer1 in modalità counter, collegandovi un quarzo da 32.768KHz, che genera cioè 32768 oscillazioni complete in un secondo, poiché 32768 vale giusto la metà (più 1) di 0xFFFF si ha un overflow, quindi un interrupt, ogni secondo.

Questa periferica può essere usata da sola oppure insieme alle funzioni **Capture** e **Compare** delle periferiche CCP.

Misura del periodo di un'onda quadra

Un primo esempio dell'utilizzo del modulo CCP e' la misura del periodo **T** di un onda quadra applicata al pin CCP (RB3). Il principio di base e' il conteggio di cicli macchina che avvengono tra due eventi dell'onda quadra di cui si vuole misurare il periodo: per esempio i fronti di salita che si manifestano agli istanti t1 e t2 come mostrato in figura.



Il conteggio si effettua tramite il TIMER1 (a 16 bit). Quando si verifica l'evento "fronte di salita" automaticamente il contenuto del timer 1 e' caricato nei registri CCPR1L e CCPR1H generando anche un interrupt. E' possibile risalire al numero di cicli macchina che intercorrono tra i due istanti t1 e t2 operando una sottrazione tra il valore dei registi caricati nell'istante t2 e il valore salvato nell'istante t1. Se **N** e' il numero dei cicli macchina allora il periodo **T** dell'onda e':

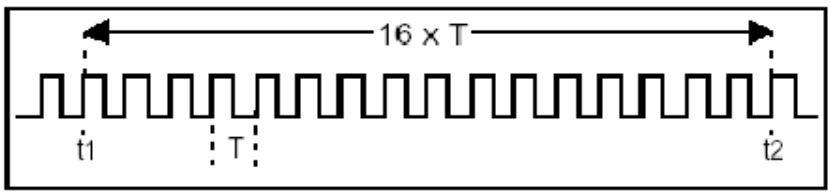
T = N * Tcm con **Tcm=4 / Fosc** (**Fosc** = frequenza del oscillatore del microcontrollore)

I passi da seguire per implementare tale misura sono:

1. Configurare i bit di controllo del registro CCPxM3:CCPxM0 (CCPxCON<3:0>) per catturare ad ogni fronte di salita dell'onda
2. Configurare il timer 1 come contatore asincrono pilotato dal clock di sistema Fosc/4
3. Configurare il prescaler del TIMER1 cosi da evitare l'overflow di Timer1;
4. Abilitare l'interrupt del modulo CCP (bit CCPxIE) del registro PEI1
5. Settare RB3 come pin di ingresso (TRISB<3>=1)

Misura mediata del Periodo di un onda quadra

Il metodo ora proposto differisce dal primo solo per il fatto che l'evento che scatena la cattura non e' il singolo fronte di salita ma il sedicesimo fronte: ogni16 fronti di salita del segnale di ingresso si ha una " cattura". Il vantaggio di tale metodo è l'immunità ai disturbi data dalla media.



Per tale misura bisogna:

1. configurare i bit CCPxM3:CCPxM0 (CCPxCON<3:0>) per catturare ogni 16 fronti di salita del segnale
2. configurare Timer1 prescaler cosi che Timer1 lavorerà per periodi 16 TMAX(TMAX e' il periodo massimo del segnale in ingresso) senza overflow.
3. abilitare l'interrupt CCP (CCPxIE bit di PIE1).
4. quando si verifica l'interrupt:
    a) Sottrarre il valore salvato della cattura in t1 dal valore catturato in t2 e salvare il valore ottenuto in N
    b) Salvare il valore catturato in t2.
    c) Azzerare il flag del Timer1 dovuto all'interrupt
    d) Dividere per 16 il numero N (basta operare uno shift a destra per quattro volte)

Il periodo del segnale in ingresso e' dato da: **T= Tcm * N / 16**

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value

10 = 1:4 Prescale value

01 = 1:2 Prescale value

00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit

1 = Oscillator is enabled

0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **T1SYNC**: Timer1 External Clock Input Synchronization Control bit

When TMR1CS = 1:

1 = Do not synchronize external clock input

0 = Synchronize external clock input

When TMR1CS = 0:

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit

1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)

0 = Internal clock (FOSC/4)

bit 0 **TMR1ON**: Timer1 On bit

1 = Enables Timer1

0 = Stops Timer1

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Esempio:

```
movlw   00110100B  ; inizializza il timer1 (prescaler 1:8)
movwf   t1con      ; con fosc=4MHz la freq. di conteggio è 125kHz
…….
…….
bsf     t1con,0    ; Start Di Timer1
…….
…….
bcf     t1con,0    ; stop del conteggio con timer1
```

TIMER2

Questo modulo è un contatore a 8 bit esclusivamente alimentato dal clock di sistema: può essere quindi solo un contatore di tempo. Il registro dove viene memorizzato il conteggio è il TMR2.

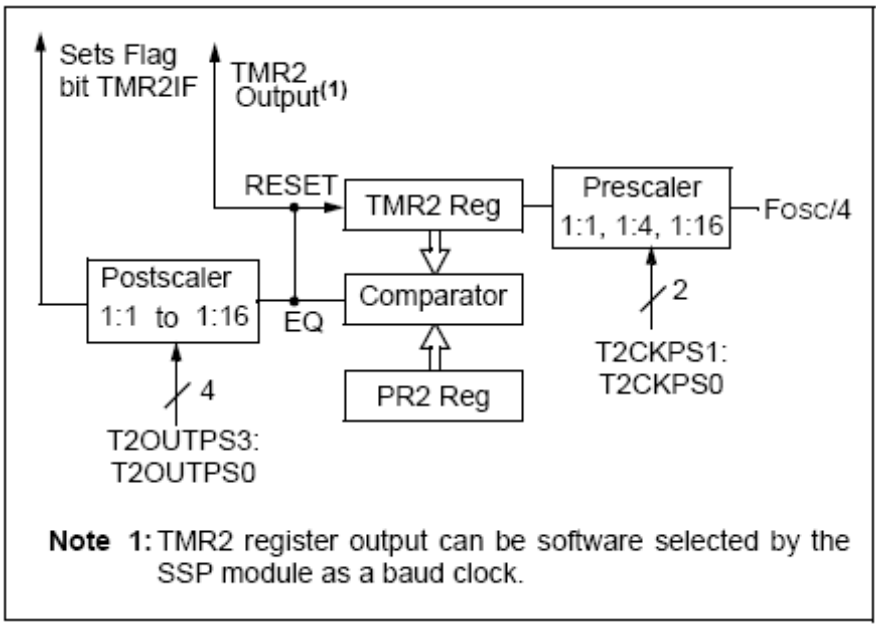


FIG. 5.4.3-5.4-1 – TIMER2

Il Timer2 ha diverse particolarità:

- Possiede un prescaler per la divisione del clock che permette le sole configurazioni 1:1, 1:4, 1:16.
- Possiede un registro di supporto a 8bit, detto PR2, accessibile sia in lettura che in scrittura, che viene costantemente confrontato con il registro TMR2. Quando i due registri sono uguali viene generata una segnalazione che origina tre eventi: azzeramento del registro TMR2, generazione di un segnale "TMR2 Output" (usato da modulo SSP – Syncronous Serial Port) e generazione di un segnale di interrupt.
- Possiede, infine, un postscaler che gestisce il segnale di interrupt generato dal comparatore. Le divisioni possibili sono tutte quelle nell'interballo da 1:1 a 1:16, permettendo quindi di segnalare l'avvento dell'interrupt al bit TMR 2IF di PIR1 solo dopo che sono avvenute un numero prefissato di uguaglianze tra TMR2 e PR2.

Il registro che assiste il Timer 2 è il T2CON.
Anche questa periferica può essere usata da sola o per mezzo della funzione PWM del modulo CCP.

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

PWM

Il microcontrollore PIC 16F876 è dotato di due moduli CCP (Capture Compare PWM) per la generazione hardware di un segnale a modulazione di impulso. Settando in modo appropriato i due moduli, il PIC 16F876 genera in uscita sui pin RC1 e RC2 due segnali con PWM di stessa frequenza e con duty-cycle anche differenti. Il PWM hardware ha una risoluzione a 10 bit, il che vuol dire che si possono ottenere 2^{10}=1024 differenti passi. Questo numero di passi potrebbe risultare eccessivo per alcune applicazioni, quindi usando un singolo byte si possono ottenere 256 passi, suddividendoli se si decide di pilotare un motore elettrico, in 128 (0-127) per un senso di rotazione e 128 (128-255) per il senso di rotazione opposto. I due moduli CCP, utilizzano entrambi il Timer2 del PIC, che li vincola a generare il segnale sui due canali con la stessa frequenza. Per l'utilizzo dei moduli CCP bisogna settare in modo appropriato i seguenti registri:
T2CON e **PR2** registri abbinati con il Timer2
CCP1CON, CCP2CON, CCPRxL registri abbinati con i due moduli CCP
TRISC registro abbinato con il PORTC.
Il registro dal quale dipende la frequenza del PWM e il registro di configurazione del Timer2 T2CON.

T2CON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Bit 7 non utilizzato (settare a 0)
Bit 6-3 utilizzati per il valore del postacaler (non utilizzati per determinare la frequenza del PWM)
Bit 2 se settato ad 1 abilita il TIMER2 se settato a 0 lo disattiva
Bit 1-0 utilizzati per il valore di prescaler (divisore di frequenza) che servono per variare la frequenza del PWM, se impostati a 0 il valore è unitario.
Il periodo del PWM viene impostato attraverso il registro **PR2** scrivendo un valore da 0-255 ricavandolo dalla seguente espressione:

PWM periodo=[(PR2)+1]*4*Tosc*(TMR2 valore di prescaler)

La frequenza del PWM è definita dalla seguente formula

F_{PWM}=1/[periodo PWM]

Per configurare i due moduli CCP (CCP1 e CCP2) usiamo i relativi registri di configurazione CCP1CON e CCP2CON, i quali contengono la configurazione del relativo modulo e parte del valore binario con cui è espresso il valore del duty-cycle.

CCPxCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0

Sostituire alla x il valore 1 oppure 2 se trattasi del registro CCP1CON oppure CCP2CON
Bit 7-6 non utilizzati
Bit 5-4 contengono i bit meno significati del valore di duty-cycle con risoluzione a 10 bit
Bit 3-0 settaggi per il funzionamento del modulo PWM (bit 3-2 vanno settati ad 1), i bit da 1-0 sono in ininfluenti (settarli a 0)
Il registro **TRISC** relativo al PORTC deve essere settato in modo che i pin 1 e 2, quelli sui quali avremo in uscita il segnale con PWM, devono essere posti ad 0.
Nei registri **CCPR1L e CCPR2L** deve essere inserito il valore relativo agli 8 bit più significativi del valore del duty-cycle.
Il duty-cycle è calcolato utilizzando la seguente espressione:

PWM duty-cycle=(CCPR1L:CCP1CON<5:4>)*Tosc*(TMR2 valore di prescaler)

(i due punti non indicano l'operatore matematico di divisione,ma l'unione dei valori dei due registri)

Riepilogo del SET-UP per il PWM

1. settare il periodo di PWM attraverso PR2;
2. settare il duty-cycle del PWM attraverso CCPR1L e CCP1CON bits<5:4>
3. settare i pin del PORTC relativi al modulo da usare come output (0)
4. impostare il valore di prescaler del TMR2 ed abilitarlo attraverso T2CON
5. configurare il relativo modulo CCPx per operazioni di PWM

N.B

Se si utilizza il modulo CCP1 per il PWM, il pin di uscita è il TRISC2, mentre se si utilizza il modulo CCP2 il pin di uscita risulta essere TRISC1.

INTERRUPT PIC16F87X

1. GENERALITÀ

L'interrupt è una particolare caratteristica dei PIC-micro (e dei microprocessori in generale) che consente di intercettare un evento, interrompere momentaneamente il programma in corso, eseguire una porzione di programma specializzata per la gestione dell'evento verificatosi e riprendere l'esecuzione del programma principale. Un evento può essere una variazione di livello, la fine di una trasmissione, il cambiamento di stato di una linea, ecc.
Ad esempio, per verificare il cambiamento di stato di un ingresso (cioè il passaggio da 1 a 0 o viceversa) si dovrebbe continuamente controllare quell'ingresso, aspettando la commutazione; con le interruzioni, invece, si possono eseguire altre operazioni e l'evento sarà intercettato automaticamente.
Volendo fare un paragone con il mondo reale si può dire che l'interrupt rappresenta per il microcontrollore quello che per noi rappresenta la suoneria del telefono.
Per poter ricevere telefonate non dobbiamo preoccuparci di alzare continuamente la cornetta per vedere se c'é qualcuno che vuol parlare con noi, ma, grazie alla suoneria, possiamo continuare tranquillamente a svolgere le nostre faccende perché saremo avvisati ogni volta che qualcuno ci sta chiamando.
E' evidente quanto sia più efficiente gestire un evento con un interrupt anziché controllare ciclicamente il verificarsi dell'evento con il programma principale. Gran parte degli aspetti legati alla gestione dell'interrupt vengono inoltre trattati direttamente dall'hardware interno del PIC per cui il tempo di risposta all'evento è praticamente immediato.

2. EVENTI INTERRUPT.

I μC PIC hanno diverse sorgenti di interrupt. In pratica ad ogni modulo periferico è associata una o più sorgenti di interrupt:

1. ***Interrupt esterno***: cambio di livello sul pin INT/RB0;
2. ***interrupt overflow TRM0***: raggiunta fine conteggio da parte del registro TIMER0
3. interrupt di cambiamento di livello su uno dei pin RB4 ... RB7;
4. interrupt fine conversione del convertitore A/D;
5. interrupt di ***receive*** (RS232)
6. interrupt di ***transmit*** (RS232)
7. termine scrittura dati in EEPROM;
8. interrupt USART :
   - interrupt di ***receive*** (RS232)
   - interrupt di ***transmit*** (RS232)
9. interrupt Parallel Slave Port (PSP);
10. ***interrupt overflow TRM1***: raggiunta fine conteggio da parte del registro TIMER1
11. ***interrupt overflow TRM2***: raggiunta fine conteggio da parte del registro TIMER2
12. interrupt CCP (Capture/Compare);
13. interrupt SSP (Porta seriale sincrona).

3. ABILITAZIONE/CONTROLLO INTERRUPT.

Per gestire efficientemente un interrupt occorrono due controlli:

- ***bit di enable***, per abilitare/disabilitare l'interrupt;
- ***bit di flag***, per rilevare se si è verificato un evento interrupt.

Questi bit di enable e di flag sono contenuti nei seguenti registri:

- INTCON (indirizzo 0x0B) INTerrupt CONtrol register;
- PIE1-PIE2, Peripheral Interrup Enable register
- PIR1-PIR2, Peripheral Interrup flag Register.

registro INTCON (indirizzi 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

bit 3 **RBIE (1):** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

bit 0 **RBIF (1):** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend		
R = Readable bit	W = Writable bit	
U = Unimplemented bit, read as '0'		- n = Value at POR reset

L’Option Register è un registro modificabile, che contiene il bit (INTEDG) di controllo del fronte attivo del pin RB0/INT.

registro OPTION

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

I registri PIE1 e PIR1 contengono rispettivamente i bit di enable e di flag di ogni interrupt collegato alle periferiche interne al PIC.

registro PIR1 (indirizzo 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF(1):** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag
1 = The SSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI
  - A transmission/reception has taken place.
- I^2C Slave
  - A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated START condition was completed by the SSP module.
  - The initiated STOP condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A START condition occurred while the SSP module was idle (Multi-Master system).
  - A STOP condition occurred while the SSP module was idle (Multi-Master system).

0 = No SSP interrupt condition has occurred.

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: PSPIF is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro PIE1 (indirizzo 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Altri registri di controllo dell'interrupt sono PIE2 e PIR2

registro PIE2 (indirizzo 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

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

bit 6 **Reserved**: Always maintain this bit clear

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIE**: EEPROM Write Operation Interrupt Enable
1 = Enable EE Write Interrupt
0 = Disable EE Write Interrupt

bit 3 **BCLIE**: Bus Collision Interrupt Enable
1 = Enable Bus Collision Interrupt
0 = Disable Bus Collision Interrupt

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro PIR2 (indirizzo 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

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

bit 6 **Reserved**: Always maintain this bit clear

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF**: Bus Collision Interrupt Flag bit
1 = A bus collision has occurred in the SSP, when configured for I2C Master mode
0 = No bus collision has occurred

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IF**: CCP2 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Qualunque sia l'evento abilitato, al suo manifestarsi il PIC interrompe l'esecuzione del programma in corso, memorizza automaticamente nello STACK il valore corrente del PROGRAM COUNTER e salta all'istruzione presente nella locazione di memoria 0004H denominata Interrupt vector (vettore di interrupt).

E' da questo punto che dobbiamo inserire la nostra subroutine di gestione dell'interrupt denominata Interrupt Handler (gestore di interrupt).

Potendo abilitare più interrupt, tra i primi compiti dell'interrupt handler è la verifica di quale, tra gli eventi abilitati, ha generato l'interrupt per attivare l'esecuzione della parte di programma relativo.

Questo controllo può essere effettuato utilizzando gli Interrupt flag.

Quando viene generato un interrupt il PIC disabilita automaticamente il bit GIE (Global Interrupt Enable) del registro INTCON in modo da disabilitare tutti gli interrupt mentre è già in esecuzione un interrupt handler. Terminata la gestione dell'interrupt si torna al programma principale tramite l'istruzione RETFIE che riporta automaticamente a 1 il bit GIE.

Il vettore di interruzione è quindi unico; le diverse situazioni di interrupt comunque si "qualificano" impostando il relativo flag di richiesta contenuto nel registro INTCON o PIR. Il software può così (anzi, di solito *deve)* individuare quale condizione ha determinato interrupt controllando tali flag e diramandosi di conseguenza.

Per l'uso degli interrupt è importante soprattutto tenere presente che l'evento che chiama l'interruzione, non determina direttamente l'interrupt, ma si limita a impostare il corrispondente bit di flag qualificatore, indipendentemente dal fatto che l'interrupt in questione sia abilitato o meno. *L'innalzamento del flag qualificatore,* se GIE e il flag di abilitazione specifica sono entrambi alti, *causa l'interrupt.*

Ad esempio, un impulso sulla linea RBO/INT (dichiarata come input) determina l'innalzamento di INTF anche se GIE e/o INTE sono bassi; se, in un tempo successivo, GIE e INTE vengono entrambi alzati e INTF è ancora alto, il microcontrollore va in interruzione, accettando la richiesta così memorizzata.

All'interno della routine di interruzione deve essere resettato il flag di richiesta dell'interrupt altrimenti la routine riparte immediatamente dopo il rientro al programma principale. Quindi è necessario che il flag di richiesta sia abbassato via SW all'interno della routine di gestione;

La routine che gestisce gli interrupt (Interrupt Handler) deve trovarsi all'indirizzo 0x04 (Interrupt Vector), perciò generalmente si usa questo codice:

```
        ORG     0x00

        goto    Start

        ORG     0x04
        ; istruzioni di gestione interrupt
        ....
        ....
        retfie

Start
        ; inizio programma
        ....
        ....
```

Per utilizzare uno o più interrupt bisogna impostare il bit GIE e impostare i bit di abilitazione degli interrupt che ci interessa utilizzare.

4. ESEMPI DI GESTIONE DELL'INTERRUPT

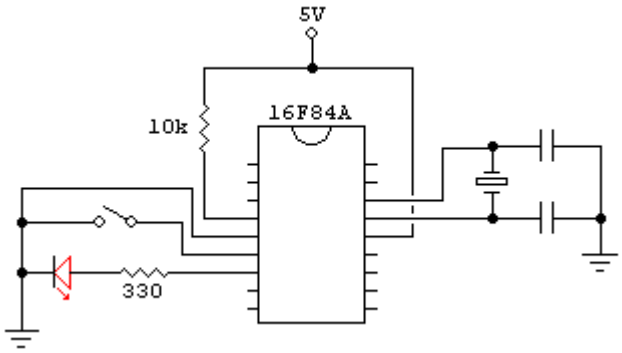
Interrupt su RB0

Questo interrupt permette di "catturare" un fronte di salita o discesa sul piedino RB0; per scegliere quale fronte è sufficiente agire sul bit INTEDG del registro OPTION (OPTION_REG [6]):

- se 1 fronte di salita
- se 0 fronte di discesa

Vediamo ora un esempio completo di come usare un'interruzione per accendere un led alla pressione di un tasto NA, e poi spegnerlo con un'altra pressione.

Approfittando dell'esempio si segnala la possibilità di abilitare delle resistenze di pull-up interne; basta resettare il bit NOT_RBPU del registro OPTION (OPTION_REG [7]) per avere un pull-up per ogni pin settato come ingresso della Porta B; questo ci consente di collegare facilmente dei pulsanti al PIC senza componenti aggiuntivi.




```
        include   "P16F84A.INC"

        ORG       0x00

        goto      Start

        ORG       0x04

        comf      PORTB,F          ; inverte lo stato del LED
        bcf       INTCON,INTF      ; resetta il flag dell'interrupt
        retfie

Start

        bsf       STATUS,RP0       ; banco 1
        movlw     B'11111101'      ; RB0 ingresso, RB1 uscita
        movwf     TRISB
        bcf       OPTION_REG,NOT_RBPU ; attiva pull-up
        bcf       OPTION_REG,INTEDG ; fronte di discesa
```

```
        bcf     STATUS,RP0      ; banco 0

        bsf     INTCON,INTE     ; attiva interrupt su RB0
        bsf     INTCON,GIE      ; abilita gli interrupt

Main
        goto    Main

        END
```

Se il pulsante non è anti-rimbalzo, il LED potrebbe accendersi o spegnersi in modo anomalo. Per ovviare a tale problema si potrebbe introdurre un ritardo di qualche millisecondo prima di uscire dall'Interrupt Handler.

Interrupt su RB4-RB7

In questo caso, l'interruzione viene generata ad ogni cambiamento di stato (sia fronte di salita che discesa) su uno dei pin RB4-RB7; il flag settato è lo stesso per tutti i 4 pin, perciò è compito del programmatore verificare quale sia il cambiamento avvenuto.

Preservare Status e W

Un interrupt può interrompere il programma principale in qualunque momento, per questo motivo la routine di gestione delle interruzioni non deve in alcun modo influire sul comportamento del programma. Qualunque operazione si effettui all'interno dell'Interrupt Handler va inevitabilmente a modificare i registri W e STATUS perciò, se questi vengono in qualche modo utilizzati nel software principale, è necessario preservarne il contenuto.

OSSERVAZIONE: a volte può accadere che quando avviene una richiesta di interrupt ci si trovi in un banco diverso dal banco 0. Per salvare i valori temporanei (come W_TEMP, STATUS_TEMP, ecc.) è opportuno che vengano utilizzate locazioni di memoria situate in posizione comune a tutti i banchi: per questo il costruttore ha previsto delle locazioni comuni a tutti i banchi cha vanno da 070H a o7FH (si veda la mappa della memoria SFR).

```
        ORG     0x04
        movwf   W_TEMP
        swapf   STATUS, W
        movwf   STATUS_TEMP
        ....
        ....
        swapf   STATUS_TEMP, W
        movwf   STATUS
        swapf   W_TEMP, F
        swapf   W_TEMP, W
        retfie
```

In questo snippet viene impiegata l'istruzione *swapf* perché non influenza lo STATUS.

SLEEP

Spesso il PIC si trova in situazioni nelle quali non deve fare nulla, se non attendere un evento esterno; per questo motivo è stato progettato un sistema di risparmio d'energia che limita il consumo di corrente fino a qualche µA.

Per attivarlo è sufficiente usare l'istruzione *sleep* che manda il micro in stand-by. Il PIC può essere "risvegliato" da uno di questi eventi:

- Reset tramite il pin MCRL'

- Timeout del WDT

- Interrupt su RB0, RB4-RB7 o fine scrittura della EEPROM

Durante lo sleep le periferiche interne vengono disattivate.

Se il risveglio (wake-up) è causato da un interrupt, viene subito eseguita l'istruzione che segue *sleep*, e solo poi si passa all'Interrupt Handler; se questa condizione è indesiderata, è sufficiente inserire dopo *sleep* un *nop*.

SET ISTRUZIONI PIC16F8XX

	Sintassi	Descrizione Microchip	Operazione equivalente.		
1	ADDLW k	Add literal and W	w = w+k	arit	
2	ADDWF f,d	add w and f	d = w +f	arit	
3	ANDLW k	and literal with w	w = w & k	log	
4	ANDWF f,d	and w with f	d = w & f	log	
5	BCF f,b	bit clear f	f(b) = 0	azz	
6	BSF f,b	bit set f	f(b) = 1	azz	
7	BTFSC f,b	bit test f, skip if clear	f(b) = 0? se si salta un'istruzione	salto	
8	BTFSS f,b	bit test f, skip if set	f(b) = 1? se si salta un'istruzione	salto	
9	CALL k	subroutine call	chiama sub all'indirizzo k	sub	
10	CLRF f	clear f	f = 0	azz	
11	CLRW -	clear w	w = 0	var	
12	CLRWDT -	clear watchdog timer	watchdog timer = 0	arit	
13	COMF f,d	complement f	d = !f	azz	
14	DECF f,d	decrement f	d=f-1	arit	
15	DECFSZ f,d	decrement f, skip if 0	d = f-1; se d = 0 salta un'istruzione	salto	
16	GOTO k	go to address	salta all'indirizzo k	salto	
17	INCF f,d	increment f	d=f+1	arit	
18	INCFSZ f,d	increment f, skip if 0	d=f+1; se d=0 salta una istruzione	salto	
19	IORLW k	inclusive OR k with w	w= w OR k	log	
20	IORWF f,d	inclusive OR w with f	d= w OR f	log	
21	MOVF f,d	move f	d = f	mem	
22	MOVLW k	move literal to w	w = k	mem	
23	MOVWF f	move w to f	f= w	mem	
24	NOP -	no operation	nessuna operazione	var	
25	RETFIE -	return from interrupt	ritorna da una sub di interrupt	sub	
26	RETLW k	return literal to w	ritorna da una sub con w = k	sub	
27	RETURN -	return from sub	ritorna da una subroutine	sub	
28	RLF f,d	rotate left f through carry	d = f << 1	arit	
29	RRF f,d	rotate right f through carry	d = f >> 1	arit	
30	SLEEP -	go to in standby mode	mette in standby il PIC	var	
31	SUBLW k	substract w from literal	w = k-w	arit	
32	SUBWF f,d	substract w from f	d = f-w	arit	
33	SWAPF f,d	swap f	f = swap bit 0123 con bit 4567 di f	var	
34	XORLW k	exclusive OR k with w	w= w XOR k	log	
35	XORWF f,d	exclusive OR w with f	d= w XOR f	log	

Tabella 1: Set istruzione in ordine alfabetico.

Il PIC ha un set di 35 istruzioni di 14 bit, di cui:

- 3 ... 6 bit rappresentano il codice operativo (OPCOD);
- i rimanenti rappresentano gli operandi (uno o due)

Gli **operandi** possono essere:

- un registro a 8bit (1byte): - **w** (working register) registro accumulatore interno;
  -**f** (file register) registro utente.
- un bit di uno dei registri sopra citati.

Le **istruzioni** sono suddivise in 3 gruppi (vedi Tab.1):

- byte oriented: operandi 2 registri;
- bit oriented: operandi: 1 bit e un registro;
- operazioni con costanti o di controllo (literal e control operation).

Tipo operazioni	**1° GRUPPO**	**2° GRUPPO**	**3° GRUPPO**
ARITMETICHE/ LOGICHE	ADDWF f,d	BCF f,b	ADDLW k
	ANDWF f,d	BSF f,b	ANDLW k
	CLRF f		CLRWDT -
	CLRW -		
	COMF f,d		
	DECF f,d		
	INCF f,d		
	IORWF f,d		IORLW k
	RLF f,d		
	RRF f,d		
	SUBWF f,d		SUBLW k
	XORWF f,d		XORLW k
MOVIMENTO TRA REGISTRI	MOVF f,d		MOVLW k
	MOVWF f		
SALTO	DECFSZ f,d	BTFSC f,b	
	INCFSZ f,d	BTFSS f,b	
VARIE	NOP -		CALL k
	SWAPF f,d		GOTO k
			RETFIE -
			RETLW k
			RETURN -
			SLEEP -

Tabella 2: Set istruzione raggruppate per la funzione/operazione svolta

FORMATO ISTRUZIONI.

Il formato generale di un'istruzione può assumere 3 forme. (vedi Fig. 25)
I bit riservati all'OPCOD possono variare da 3 a 6. Ciò permette di avere un set di istruzioni di sole 35 istruzioni.

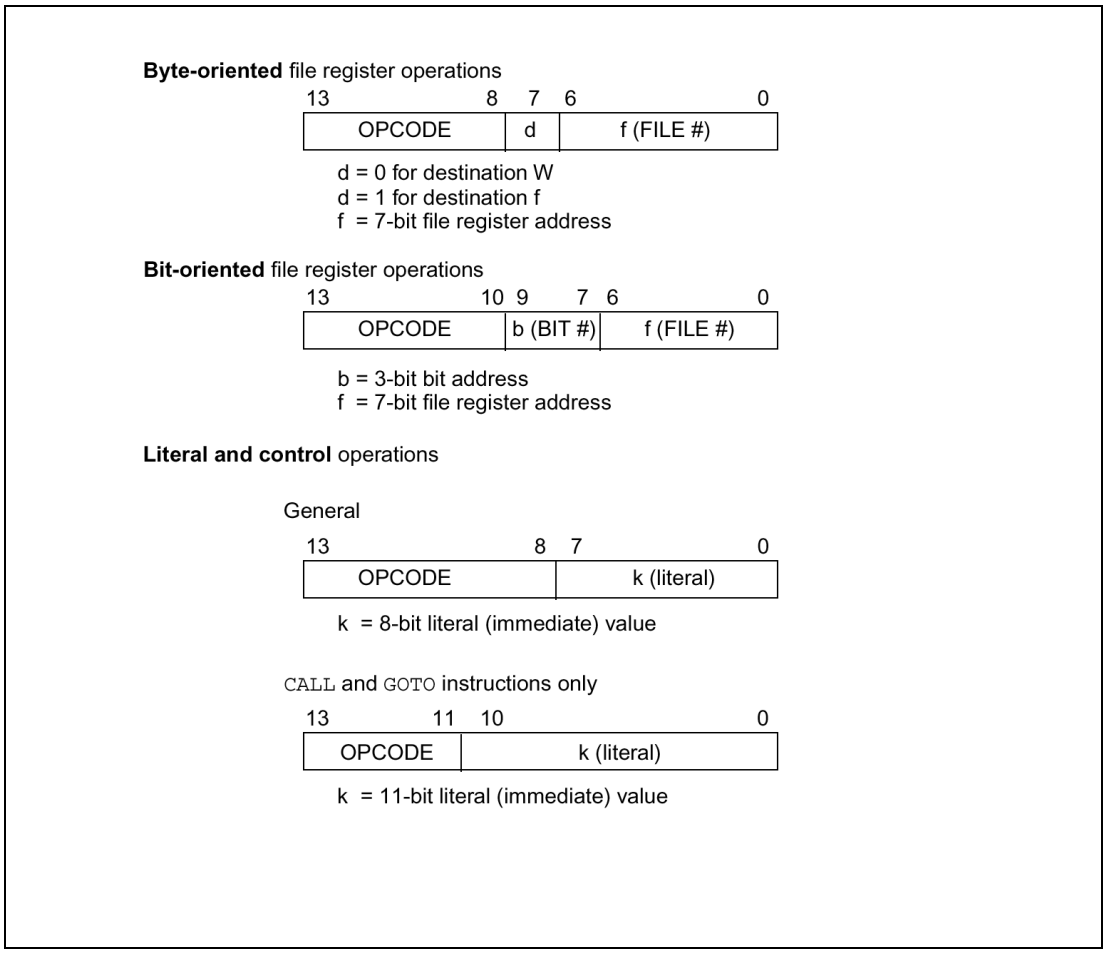


Fig. 25:Formato generale delle istruzioni.

CONVENZIONI.

f = indirizzo di un *file register* a 8 bit (indirizzi possibili: 0x00 ... 0x7F). Corrisponde, in C, al nome di una variabile;
w = indirizzo working register, registro accumulatore interno;
(f) , (w) = contenuto di un registro. Corrisponde, in C, al valore di una variabile;
f <n> = bit di posizione n del registro f:
b = posizione di un bit in un byte (valori 0 ... 7);
k = campo letterale, dato costante, etichetta a 8 o 11 bit (call, goto);
d = registro destinazione di un'operazione: 0→risultato memorizzato in W
1→risultato memorizzato in f

REGISTRI USATI COME SORGENTE E/O DESTINAZIONE DI UN'OPERAZIONE.

SFR Special Function Register, area di memoria dati di 8 bit, registri speciali;
GPR General Purpose Register, area di memoria dati di 8 bit usata dall'utente per la memorizzazione delle variabili.
Tutte le istruzioni permettono sia la lettura che la scrittura dei file register sia gli SFR che i GPR.

Ci sono alcune eccezioni.

MANIPOLAZIONE DI BIT

Osservazione le istruzioni possono essere di:

- sola lettura (read R);
- sola scrittura (write W);
- lettura-modifica-scrittura (read-modify-write R-M-W) nel senso che questo tipo di istruzioni prima leggono il registro, operano sui bit e quindi scrivono il risultato nello stesso registro.

Si deve sempre tenere in mente ciò, soprattutto quando si opera con i registri speciali quali i Ports.

N.B.:
I bit di stato che sono manipolati dal micro (inclusi i bit di flag degli interrupt) sono posti a 1 o a 0 durante il tempo di ciclo Q1 di Tcyl, quindi con le istruzioni R-M-W, che operano con questi registri, non si è sicuri del valore dei bit che essi contengono.

ATTIVITÀ SVOLTE DAI CICLI Q.

Ogni istruzione è elaborata in un periodo di tempo, detto periodo di ciclo Tcy, corrispondente a 4 periodi dell'oscillatore (Tcy = 4*Tosc).

Esempio:
fosc = 4MHz (Tosc=1/fosc=250ns) → fcy = 1/4 fosc = 1MHz quindi → Tcy = 1/fcy = 1μs

Il Tcy è suddiviso in quattro cicli elementari, detti Q cycle, durante i quali l'attività svolta dal micro è:

periodo	attività svolta
Q1	decodifica istruzione
Q2	lettura istruzione
Q3	elaborazione operandi
Q4	scrittura risultato

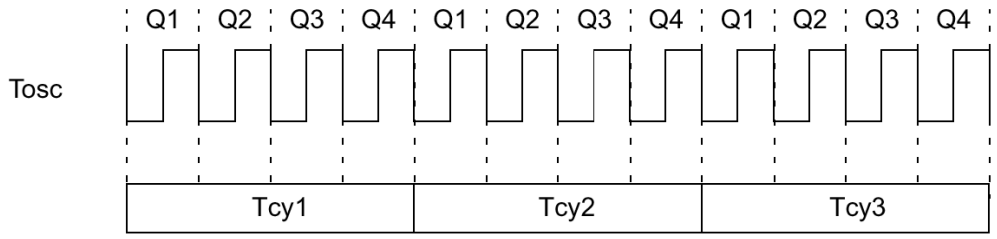


Fig. 26: Attività cicli Q.

Nel manuale, per ogni istruzione, sono mostrate le attività svolte in ogni ciclo.

SPIEGAZIONE DELLE ISTRUZIONI CON ESEMPI

1 **ADDLW k** somma la costante k al valore memorizzato in w; il risultato è messo in w. w=k+w
Esempio:

```
        org     0x00;
start
        movlw   10      ;carico valore 10 in w
        addlw   15      ;w=10+15  ;Z=0; C=0; DC=1
```

2 ADDWF f,d esegue la somma dei contenuti di due reg. d = w + f; se d = 0 → d = w; se d = 1 → d = f
Esempio: fare la somma di due variabili.

```
add1    equ     0x20    ;indirizzo per var. add1
add2    equ     0x21    ;indirizzo per var. add2
        org     0x00    ;inizio main
start
        movlw   10      ;carico valore 10 in w
        movwf   add1    ;sposto 10 in 0x20
        movlw   15      ;carico valore 15 in w
        movwf   add2    ;sposto 15 in 0x21
        movf    add1,w  ;carico add1 in w, w=add1
        addwf   add2,w  ; w=add2+w=add2+add1
```

3 ANDLW k esegue l'AND bit per bit di w con una costante; w = k AND w
Esempio:

```
        org     0x00;
start
        movlw   10101010B   ;carico valore 10101010 binario in w
        andlw   11110000B   ;w=w AND 11110000=10100000
```

OSS:
In genere l'operazione di and viene utilizzata come maschera per estrarre uno o più bit da un byte. La maschera conterrà un 1 nei bit da rilevare e uno zero in quelli che non interessano.

4 ANDWF f,d esegue l'AND bit per bit dei contenuti tra due registri.
Esempio:

```
        org     0x00;
start
        movlw   10101010B   ;carico valore 10101010 binario in w
        movwf   0x20
        movlw   00001111B   ;carico valore maschera in w
        andwf   0x20,w      ;risultato in w, w=00001010
```

5 BCF f,b ;azzera bit di posizione b del registro f; f(b)=0
Esempio:

```
parm1   equ     0x20
        org     0x00;
start
        movlw   11111111B   ;carico valore binario in w
        movwf   parm1       ;parm1=w
        bcf     parm1,2     ;parm1=11111011
```

6 BSF f,b ;come BCF solo che pone il bit b di f ad 1; f(b)=1
Esempio: vedi 5

7 BTFSC f,b ; testa bit b di f, se zero salta l'istruzione successiva; f(b)=0? Si → salta istruzione successiva; no → esegui istruzione successiva;

Esempio:

```
parm1   equ     0x20
        org     0x00;
start
        movlw   11111110B   ;carico valore binario iniziale in w
        movwf   parm1       ;parm1=w
ciclo   btfsc   parm1,0     ;essendo parm1(0)=0 esce dal loop
        goto    ciclo       ;
;no     continu
        a
```

8 BTFSS f,b ; testa bit b di f, se uno salta l'istruzione successiva; f(b)=1? si → salta istruzione successiva; no → esegui istruzione successiva;

Esempio: vedi 7

9 CALL k ;richiama la routine memorizzata alla locazione di memoria k; k può essere una label o un indirizzo.

Esempio:

```
led1    equ     0x20
        org     0x00;
main
        call    LedOn       ;chiamo routine LedOn
        ……..                ;parm1=w
;inizio routine
LedOn   bsf     portB,led1  ;pone a 1 bit led1 di portB
        return              ;
;end    LedOn
```

Quando la CPU del PIC incontra un'istruzione CALL:

- memorizza nello STACK il valore del PC+1 in modo da poter riprendere l'esecuzione dall'istruzione successiva;
- carica nel PC l'indirizzo dalla routine (nell'esempio LedOn) e salta a tale indirizzo;
- esegue le istruzioni della routine fino all'istruzione RETURN o RETLW;
- ritorna caricando in PC l'indirizzo memorizzato in cima allo STACK; sono previsti 8 livelli di stack, per cui si potrebbero nidificare fino a 8 routine, però non è una buona strategia di programmazione.
- riprende l'esecuzione del main.

10 CLRF f ;azzera il contenuto del registro indirizzato da f

Esempio:

```
        ……
        clrf    TMR0        ;azzera reg. Timer0
        ……
```

11 CLRW - ; azzera il contenuto del registro indirizzato da w; nessun argomento

Esempio:

```
        ……
        Clrw                ;azzera reg. w; pone Z=1
        ……
```

12 CLRWDT - ;esegue il reset del watchdog timer

Se attivato il watchdog timer, viene abilitato un timer che dopo un tempo imposta (e programmabile) effettua il reset del micro.

13 COMF f,d ; d = f; effettua il complemento (a 1 o negazione) del valore contenuto nel reg. f; questa istruzione influenza z.

Esempio:

```
parm1  equ      0x20
       org      0x00        ;
main
       movlw    11111110B   ;carico valore binario iniziale in w
       movwf    parm1       ;parm1=w
       comf     parm1,f     ;parm1=00000001
```

14 DECF f,d ;d=f-1; il contenuto di f viene decrementato di 1; se d=w → w=f-1; se d=f → f=f-1; questa istruzione influenza z.

Esempio:

```
parm1  equ      0x20
       org      0x00        ;
main
       movlw    82          ;carico valore binario iniziale in w
       movwf    parm1       ;parm1=w
       decf     parm1,f     ; parm1=81
```

15 DECFSZ f,d ; decrementa il valore del reg. f e se il risultato è 0, salta l'istruzione successiva;
d=f-1; d=0? se si salta;
questa istruzione è utile per effettuare cicli (tipo for, while, repeat)

Esempio:

```
count  equ      0x20
er
       org      0x00;
start
       movlw    10          ;carico valore iniziale in w;
       movwf    counter     ;counter=w
ciclo  decfsz   counter,f   ; questo ciclo viene eseguito 10 volte
       goto     ciclo       ;
;no    continu
       a
```

16 GOTO k ;determina un salto del programma in esecuzione alla locazione k, che può essere una label o un indirizzo;

17 INCF f,d ; d=f+1; il contenuto di f viene incrementato di 1; se d=w → w=f+1; se d=f → f=f+1; questa istruzione influenza z.

Esempio: vedi 14

18 INCFSZ f,d ; incrementa il valore del reg. f e se il risultato è 0, salta l'istruzione successiva;
d=f+1; d=0? se si salta;
questa istruzione è utile per effettuare cicli (tipo for, while, repeat)

Esempio:

```
count  equ      0x20
er
       org      0x00;
```

```
start
      movlw    246          ;carico valore iniziale in w;
      movwf    counter      ;counter = w
ciclo decfsz   counter,f    ; questo ciclo viene eseguito 10 volte (256-
                            246) perché quando f=255 dopo assume il valore
                            0;
      goto     ciclo        ;
;no   continu
      a
```

19 IORLW k ;w = w OR k viene eseguito l'OR inclusivo, bit per bit, con una costante
Esempio: vedi istruzione ANDLW

20 IORWF f,d ; d = w OR f viene eseguito l'OR inclusivo, bit per bit, con registro
Esempio: vedi istruzione ANDWF

21 MOVF f,d ; d=f; copia il valore di f in w o in se stesso; l'utilità di questa istruzione sta nel fatto che
essa modifica il bit zero di STATUS

22 MOVLW k ; w=k; assegno all'accumulatore w il valore di k;

23 MOVWF f ; f=w; copia il valore di w in f.

24 NOP - ;nessuna operazione svolta; comunque, poiché viene eseguita, si ritarda il programma di un tempo t=Tcyl

25 RETFIE - ;ritorno da una routine di servizio di un'interrupt; questa istruzione è posta alla fine della subroutine di gestione dell'interrupt per ridare il controllo dell'esecuzione al main;
Esempio:

```
      org      0x00;
start
      goto     Start        ;
-----  -------  ------
      org      0x04         ;vettore d'interrupt; punto del programma dove
                            viene inviato il controllo del main, quando si
                            verifica qualsiasi interrupt
intSu                       ;inizio sub gestione int.
b
      ------   --------     ;corpo sub
      retfie                ;ritorno al main
;end  intSub
```

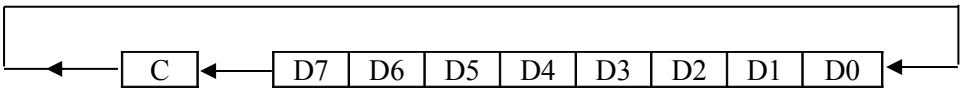
26 RETLW k ;
Esempio:

```
value equ      0x20
      org      0x00;
      call
      sub1
      movwf    value        ;memorizzo in value il valore di w in cui è
                            stato messo dalla sub1
-----  -------  ------
```

```
sub1                        ;inizio sub.
      nop                   ;corpo sub
      retlw    10           ;ritorno al main
;end  sub1
```

27 RETURN - ;ritorna da una sub senza ritornare al main alcun valore

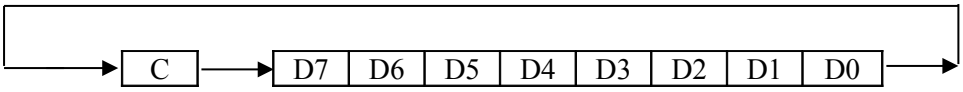
28 RLF f,d ;d = f<<1; ruota (shift) i bit del reg. f verso sinistra, ovvero dai bit LSb a quelli MSb, passando per il bit C (carry) di STATUS. Il contenuto del carry va nel bit 0, mentre il bit 7 pone un nuovo valore nel carry. Questa è una rotazione a 9 bit.
Oss. : questa operazione corrisponde alla moltiplicazione per 2 se C=0.



sequenza RLF: APP=D7; D7=D6… D1=D0; D0=C; C=APP;

Esempio:
```
parm1  equ      0x20
       org      0x00         ;
main   bcf      STATUS,C     ;
       movlw    01010101B    ;carico valore binario iniziale in w
       movwf    parm1        ;parm1=w
       rlf      parm1,f      ;parm1=10101010 ; C=0
```

29 RRF f,d ; d = f>>1; ruota (shift) i bit del reg. f verso destra, ovvero dai bit MSb a quelli LSb, passando per il bit C (carry) di STATUS. Il contenuto del carry va nel bit 7, mentre il bit 0 pone un nuovo valore nel carry. Questa è una rotazione a 9 bit.
Oss. : questa operazione corrisponde alla divisione per 2.



sequenza RRF: APP=D0; D0=D1… D6=D7; D7=C; C=APP;

Esempio:
```
parm1  equ      0x20
       org      0x00         ;
main   bcf      STATUS,C     ;
       movlw    11110000B    ;carico valore binario iniziale in w
       movwf    parm1        ;parm1=w
       rrf      parm1,f      ;parm1=11100000 ; C=1
```

30 SLEEP - ;Questa istruzione blocca l'esecuzione del programma in corso e mette il micro in modo standby. Non viene influenzato nessun bit.

31 SUBLW k ;w = k-w ; si sottrae il valore dell'accumulatore w dalla costante k; il risultato è messo in w. Vengono influenzati i bit C,Z, del reg. STATUS.

Esempio:
```
       org      0x00         ;
main                         ;
       movlw    10           ;carico valore 10 iniziale in w
```

```
        sublw   13          w=13-10=3
```

32 SUBWF f,d ; d = f-w ; si sottrae il valore dell'accumulatore w dal reg. **f**; il risultato è messo in d che può valere w o f. Vengono influenzati i bit C,Z, del reg. STATUS.

Esempio: simile al 31

33 SWAPF f,d ;scambia il nibbi MSB con quello LSB, cioè i bit D7 ... D4 con i bit D3 ... D0 del reg. **f** e il risultato è messo in d, che può essere f o w.

Esempio:

```
parm1   equ     0x20
        org     0x00        ;
main                        ;
        movlw   11100001B   ;carico valore binario iniziale in w
        movwf   parm1       ;parm1=w
        swapf   parm1,f     ;parm1=00011110
```

34 XORLW k ; w = w EXOR k viene eseguito l'OR esclusivo, bit per bit, con una costante k. Viene influenzato il bit Z di STATUS.

Esempio: vedi ist. 35

35 XORWF f,d ; d = f EXOR w : viene eseguito l'OR esclusivo, bit per bit, dell'accumulatore w con il reg.f. Viene influenzato il bit Z di STATUS.

Queste istruzioni, l'XORLW e l'XORWF eseguono l'or esclusivo che dà 0 se i bit sono uguali e dà 1 se diversi; si possono quindi usare per fare confronti

Esempio:

```
valore  equ     75
parm1   equ     0x20
        org     0x00        ;
main                        ;
        movlw   valore      ;carico valore in w
        xorwf   parm1,f     ;
        btfss   status,Z
        goto    diverso
        goto    uguale
        ….      ……
uguale  ----
        -----
        --
diverso -----               ;
        --
```

INTRODUZIONE ALLA PROGRAMMAZIONE IN ASSEMBLY.

TECNICHE DI PROGRAMMAZIONE IN ASSEMBLY.

Il linguaggio assembly, essendo legato alla struttura e al set di istruzioni della particolare CPU utilizzata, lascia molte libertà al programmatore: questo spesso si trasforma in un programma non progettato nel suo complesso, ma assemblato in base alle particolari specifiche di ogni singola routine che non sempre si armonizzano in un progetto complessivo organico. Per evitare questa prassi, che spesso conduce ad errori di programmazione difficili da individuare in fase di verifica e spesso rimediate con modifiche estemporanee che rendono il programma incomprensibile, conviene quindi farsi guidare da alcune regole di ***programmazione strutturata***.

STRUTTURA GENERALE DI UN PROGRAMMA.

In genere un programma di controllo industriale scritto per microcontroller (μC) single chip non si appoggia su un sistema operativo, che nei calcolatori ha il compito di gestire i dispositivi di I/O e di fornire le funzioni di utilità generale. Un programma di controllo industriale non ha quindi la possibilità , terminati i suoi compiti, di ripassare il controllo al sistema operativo, ma resta sempre in funzione fino a quando non si spegne la macchina. La sua struttura generale , a parte una routine di inizializzazione degli I/O e delle variabili, è ciclica, cioè il μC, giunto alla fine del codice programma, prosegue l'esecuzione tornando all'inizio del programma principale.

Per comprendere questo concetto occorre distinguere tra:

- *programma:* sequenza finita di codici (istruzioni); ha un inizio e una fine; evento statico;
- *processo*: programma durante l'esecuzione; codice eseguito dalla CPU; ha un inizio (all'accensione della macchina devono essere eseguiti dei compiti particolari), ma non avrà una fine, tutt'al più, dopo aver eseguito alcuni compiti, resterà in attesa di un nuovo comando; evento dinamico.

Quindi, a livello generale, il programma di un controllo industriale, ha la seguente struttura:

```
INIZIAL     <esegui routine di inizializzazione>
MAIN  <esegui programma principale>
      <ritorna a MAIN>
```

Ovviamente questo è solo un primo passo nella realizzazione di un programma; ognuna delle frasi scritte a fianco delle etichette, INIZIAL e MAIN, comporterà poche o numerose istruzioni, in ogni caso una buona tecnica di programmazione consiste proprio nello sviluppare i concetti generali del programma fino al dettaglio della singola istruzione (programmazione *top-down*).

OSS: l'esecuzione di un programma di controllo (meglio, di un processo) può essere:

- **sequenziale**: una istruzione dopo l'altra, non necessariamente poste in aree consecutive di memoria
- **asincrona**: il processo interrompe la sua normale sequenza di operazioni per svolgere dei compiti particolari e una volta finito, ritorna al funzionamento sequenziale.

La routine di inizializzazione.

In genere la routine di INIZIAL inizializza un gruppo di variabili in RAM.

IL PROGRAMMA PRINCIPALE.

- Il programma principale segue, nel processo di esecuzione, la routine di inizializzazione anche se non è detto che fisicamente sia collocato in memoria immediatamente di seguito a tale routine, l'esecuzione può essere trasferita all'inizio del programma con un'istruzione di salto.

Ritengo inoltre opportuna una verifica del programma principale che preceda la stesura completa delle singole sub, magari utilizzando spezzoni di programma scritti appositamente per effettuare dei test. (Ricordare che il funzionamento esatto in genere è uno solo, mentre i funzionamenti anomali o errati sono infiniti!!!).

La struttura del programma principale solitamente è a *menu*, il programma infatti analizza le varie situazioni diverse che si possono verificare sugli ingressi ed esegue le relative routine di gestione. La natura degli eventi che si possono verificare dipende naturalmente dal progetto, il programma principale potrebbe occuparsi di analizzare i codici che provengono da una tastiera ed eseguire i relativi comandi, oppure leggere i dati che provengono dai sensori collegati al sistema e prendere le opportune decisioni; inoltre potrebbe occuparsi di aggiornare periodicamente i dispositivi di output (display per l'utente, relè che attivano dispositivi esterni, led che indicano in quale fase del programma è in esecuzione). Al termine di questa analisi il programma principale ricomincia da capo. Il tempo che impiega determina la risoluzione temporale dell'analisi dei dati di ingresso, in genere è dell'ordine delle frazioni di secondo, talvolta può essere importante determinarlo con precisione o contenerlo entro i limiti accettabili per non avere ritardi troppo lunghi in certe situazioni. Per fare un esempio concreto, può essere ritenuta accettabile una durata dell'ordine di mezzo secondo se si deve analizzare lo stato di una tastiera che viene azionata manualmente, mentre può essere troppo lunga una durata di 100μs se deve essere letta una linea di ingresso seriale a 9600 baud. In ogni caso la gestione di eventi che si verificano raramente, ma che hanno bisogno di immediata attenzione dalla CPU quando si verificano, può essere demandata alle routine di gestione dell'interrupt.
Prima di scrivere il programma principale in assembly conviene scrivere le cose che il programma deve fare in italiano utilizzando però parole che ricordino il linguaggio ad alto livello (Pascal, C) in modo da evidenziare la struttura del programma. Queste frasi potranno restare utilmente nel programma sorgente come commenti, conviene quindi seguire le regole per i commenti ponendo un asterisco nella prima colonna. Si useranno quindi parole come SE...ALLORA...ALTRIMENTI per rappresentare una struttura di decisione analoga alla IF...THEN...ELSE, INIZIO...FINE o le parentesi graffe per delimitare un blocco logico. Queste parole ovviamente servono solo al programmatore per riconoscere la struttura del programma anche dopo che è passato del tempo, le regole non sono pertanto rigide e ognuno può utilizzare un modello tratto dal linguaggio ad alto livello che preferisce, anche se, per chi vuole dedicarsi alla programmazione in ambiente industriale, è preferibile il linguaggio C, questo essenzialmente per due motivi:

- il linguaggio C, pur essendo un linguaggio ad alto livello e, nello stesso tempo, vicino al linguaggio assembly è stato progettato per la realizzazione di sistemi operativi;
- esistono compilatori C anche per i microcomputer single chip impiegati nei controlli industriali, quindi l'abitudine a usare questo linguaggio può sempre tornare utile.

Subroutine di impiego generale

In genere si possono individuare quattro situazioni hardware che si verificano con frequenza in uno schema di un piccolo controllo industriale: la lettura di un sensore che fornisce un segnale analogico, la lettura di una tastiera, il comando di un relè in uscita e il comando di un display. In questo paragrafo analizzeremo le subroutine che controllano queste quattro situazioni tipiche.

Nella realizzazione di una subroutine il problema preliminare che ci si deve porre è la comunicazione della stessa con il programma principale ovvero come devono essere passati i parametri. Nello schema a fianco (figura 1) la subroutine è rappresentata come un blocco in cui entrano alcune variabili (che provengono dal programma principale) e che restituisce al programma principale altre variabili (tra cui eventualmente quelle di ingresso modificate opportunamente dalla subroutine). Senza volerci dilungare sui vari metodi di passaggio dei parametri che possono essere utilizzati con microprocessori più complessi del 6805, possiamo distinguere due semplici casi:

- le variabili di ingresso e di uscita sono al massimo due numeri di otto bit ciascuno: in questo caso la scelta più logica è quella di impiegare i due registri A e X del microprocessore per i parametri, prima della chiamata della subroutine essi conterranno i valori di ingresso, alla fine quelli di uscita;

- le variabili sono più lunghe, in questo caso é preferibile utilizzare i registri (soprattutto X) per trasmettere alla e dalla subroutine gli indirizzi di inizio delle stesse.

Per esempio, se dovessimo moltiplicare tra di loro due numeri potremmo mettere nei registri i due valori da moltiplicare e la subroutine potrebbe restituire sempre nei registri i due byte del risultato, oppure assegnare ai registri gli indirizzi (in pagina 0) dei due numeri e restituire in X l'indirizzo del risultato.

SINTASSI DI UN PROGRAMMA IN ASSEMBLY.

DIFFERENZA TRA SUBROUTINE E INTERRUPT.

- Una **subroutine** è un sottoprogramma da utilizzare in più punti e/o più volte nel programma principale. Perché ciò sia possibile bisogna che quando la subroutine termina l'esecuzione del programma prosegua esattamente dall'istruzione successiva alla chiamata. Il meccanismo con cui tale prosecuzione viene effettuata prevede il salvataggio automatico del contatore di programma PC in un'area di memoria detta *stack*. L'ultima istruzione eseguita dalla subroutine, RETURN o RETLW, ripristina l'ultimo contatore di programma salvato. In questo modo, finché c'è spazio nello stack (8 byte nel PIC16F87x) una subroutine può chiamarne un'altra (subroutine nidificate, che comunque quando possibile sono da evitare).

- L'**interrupt** è un sottoprogramma che normalmente non viene chiamato da un'istruzione ma da un evento hardware (una transizione su una linea di ingresso). Può verificarsi pertanto in qualsiasi parte del programma, per questo motivo, oltre che il contatore di programma viene salvato anche il registro di stato, essenziale per un corretto funzionamento, i registri vengono ripristinati dall'istruzione RTFIE.

STRUTTURA GENERALE DI UN PROGRAMMA ASSEMBLY PER MICRO 16F87X.

```
;*****************************************************************
; Il file contiene lo schema generale di un programma con il micro; PIC16F877          *
. Contiene il codice di base sia di un programma main                          *
; sia di un programma di gestione dell'Interrupt                        *
;*****************************************************************
;       Filename:       exf877.asm                                              *
;       Date:                                                                   *
;       File Version:   1.0                                                     *
;       Author:         ing. Duilio De Marco                                    *
;       Company:                                                                *
;*****************************************************************
;                                                                               *
;   Files required:     p16f877.inc     o p16f876 o altri                                   *
;                                                                               *
;*****************************************************************
;                                                                               *
;Note:                                                                          *
;                                                                               *
;*****************************************************************

        list    p=16f877        ; definisce il tipo di processore usato
```

```
        #include <p16f877.inc>       ; definizione delle variabili specifiche del processore

        __CONFIG _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _RC_OSC &
        _WRT_ENABLE_ON & _LVP_ON & _DEBUG_OFF & _CPD_OFF

; la direttiva '__CONFIG' setta o resetta i bit di configurazione specifici del processore
; da notare che i bit per default sono tutti a 1, con l'operazione di AND resetto
; in particolare per il PIC16F877 sono state scelte le seguenti opzioni di configurazione:
;       _CP_OFF         H'3FFF'         disabilita la protezione del codice
;       _WDT_ON         H'3FFF'          abilita il watchdog
;       _BODEN_ON  H'3FFF'          abilita il Brown-out Reset (reset che avviene quando la tensione;
                                ; scende sotto 4 V per 100 MSN)
;       _PWRTE_ON  H'3FF7'          abilita il Power Up Timer (Ritarda 75 ms all'applicazione
                                    dell'alimentazione)
;       _RC_OSC         H'3FFF'         sceglie l'oscillatore RC
;       _WRT_ENABLE_ON  H'3FFF'         abilita la scrittura sulla memoria FLASH
;       _LVP_ON         H'3FFF'         abilita la scrittura "in circuit" tramite il pin RB3
;       _DEBUG_OFF          H'3FFF'     disabilita la possibilità di debug "in circuit" tramite i pin ;
                                        RB6 e RB7
;       _CPD_OFF        H'3FFF'         memoria EEPROM non protetta

;per ogni tipo di processore bisogna riferirsi al rispettivo manuale, i valori sono definiti nel file *.inc

;***** DEFINIZIONE DELLE VARIABILI E DELLE COSTANTI
;*** esempi
w_temp          EQU     0x70            ; variabili usata per salvare W durante l'interrupt
status_temp     EQU     0x71            ; variabile usata per salvare lo stato durante l'interrupt

;***************RESET DEL PROCESSORE****************************************
                ORG 0x000               ; inizio del programma dopo il reset
                clrf    PCLATH          ; il programma principale inizia in pagina 0
                goto    main            ; va all'inizio del programma

;***************INTERRUPT DEL PROCESSORE************************************
                ORG     0x004           ; inizio del programma dopo il reset
                ;salvo stato
                movwf   w_temp          ; salva il registro W
                movf    STATUS,w
                movwf status_temp       ; salva il registro di STATO

;********QUI SI INSERISCE LA ROUTINE di INTERRUPT ******************
;********(se avviene con una call a subroutine si consuma una posizione di stack) ****
                ;recupero lo stato
                movf    status_temp,w   ; recupera lo STATO prima dell'interrupt
                movwf STATUS            ; ripristina il registro di stato
                swapf   w_temp,f
                swapf   w_temp,w        ; recupera il registro W, senza toccare lo STATO
                                        ;(per questo non si usa l'istruzione mov)
                retfie                  ; ritorna dall'interrupt
```

```
;********** INIZIO DEL PROGRAMMA PRINCIPALE: ************
;********** INIZIALIZZAZIONE DEL SISTEMA          ************
main

;************* ISTRUZIONI DA RIPETERE CICLICAMENTE ****************
ciclomain       call    subroutine
                goto    ciclomain       ;viene ripetuto sempre

;*************SUBROUTINES CHIAMATE DAL PROGRAMMA PRINCIPLAE

subroutine      nop
                return

                END                     ; fine del listato sorgente, non sempre indispensabile
```

FILE INCLUDE PIC16F877.

```
        LIST
; P16F877.INC  Standard Header File, Version 1.00    Microchip Technology, Inc
        NOLIST
; This header file defines configurations, registers, and other useful bits of information for the PIC16F877
;microcontroller.  These names are taken to match the data sheets as closely as possible.

; Note that the processor must be selected before this file is included.  The processor may be selected the following
ways:

;       1. Command line switch:
;            C:\ MPASM MYFILE.ASM /PIC16F877
;       2. LIST directive in the source file: es: LIST   P=PIC16F877
;       3. Processor Type entry in the MPASM full-screen interface
;==========================================================================
;       Revision History
;==========================================================================
;Rev:   Date:    Reason:
;1.12   01/12/00 Changed some bit names, a register name, configuration bits
;               to match datasheet (DS30292B)
;1.00   08/07/98 Initial Release
;==========================================================================
;       Verify Processor

        IFNDEF __16F877
        MESSG "Processor-header file mismatch.  Verify selected processor."
        ENDIF
;==========================================================================
;       Register Definitions
;==========================================================================

W               EQU     H'0000'
F               EQU     H'0001'

;----- Register Files------------------------------------------------------

INDF            EQU     H'0000'
TMR0            EQU     H'0001'
PCL             EQU     H'0002'
STATUS          EQU     H'0003'
FSR             EQU     H'0004'
PORTA           EQU     H'0005'
```

```
PORTB           EQU     H'0006'
PORTC           EQU     H'0007'
PORTD           EQU     H'0008'
PORTE           EQU     H'0009'
PCLATH          EQU     H'000A'
INTCON          EQU     H'000B'
PIR1            EQU     H'000C'
PIR2            EQU     H'000D'
TMR1L           EQU     H'000E'
TMR1H           EQU     H'000F'
T1CON           EQU     H'0010'
TMR2            EQU     H'0011'
T2CON           EQU     H'0012'
SSPBUF          EQU     H'0013'
SSPCON          EQU     H'0014'
CCPR1L          EQU     H'0015'
CCPR1H          EQU     H'0016'
CCP1CON         EQU     H'0017'
RCSTA           EQU     H'0018'
TXREG           EQU     H'0019'
RCREG           EQU     H'001A'
CCPR2L          EQU     H'001B'
CCPR2H          EQU     H'001C'
CCP2CON         EQU     H'001D'
ADRESH          EQU     H'001E'
ADCON0          EQU     H'001F'

OPTION_REG      EQU     H'0081'
TRISA               EQU     H'0085'
TRISB               EQU     H'0086'
TRISC               EQU     H'0087'
TRISD               EQU     H'0088'
TRISE               EQU     H'0089'
PIE1                EQU     H'008C'
PIE2                EQU     H'008D'
PCON                EQU     H'008E'
SSPCON2             EQU     H'0091'
PR2                 EQU     H'0092'
SSPADD              EQU     H'0093'
SSPSTAT             EQU     H'0094'
TXSTA               EQU     H'0098'
SPBRG               EQU     H'0099'
ADRESL              EQU     H'009E'
ADCON1              EQU     H'009F'

EEDATA              EQU     H'010C'
EEADR               EQU     H'010D'
EEDATH              EQU     H'010E'
EEADRH              EQU     H'010F'

EECON1              EQU     H'018C'
EECON2              EQU     H'018D'

;----- STATUS Bits --------------------------------------------------------

IRP                 EQU     H'0007'
RP1                 EQU     H'0006'
RP0                 EQU     H'0005'
NOT_TO              EQU     H'0004'
```

```
NOT_PD                  EQU     H'0003'
Z                       EQU     H'0002'
DC                      EQU     H'0001'
C                       EQU     H'0000'

;----- INTCON Bits --------------------------------------------------------

GIE                     EQU     H'0007'
PEIE                    EQU     H'0006'
T0IE                    EQU     H'0005'
INTE                    EQU     H'0004'
RBIE                    EQU     H'0003'
T0IF                    EQU     H'0002'
INTF                    EQU     H'0001'
RBIF                    EQU     H'0000'

;----- PIR1 Bits ----------------------------------------------------------

PSPIF                   EQU     H'0007'
ADIF                    EQU     H'0006'
RCIF                    EQU     H'0005'
TXIF                    EQU     H'0004'
SSPIF                   EQU     H'0003'
CCP1IF                  EQU     H'0002'
TMR2IF                  EQU     H'0001'
TMR1IF                  EQU     H'0000'

;----- PIR2 Bits ----------------------------------------------------------

EEIF                    EQU     H'0004'
BCLIF                   EQU     H'0003'
CCP2IF                  EQU     H'0000'

;----- T1CON Bits ---------------------------------------------------------

T1CKPS1                 EQU     H'0005'
T1CKPS0                 EQU     H'0004'
T1OSCEN                 EQU     H'0003'
NOT_T1SYNC              EQU     H'0002'
T1INSYNC                EQU     H'0002'    ; Backward compatibility only
T1SYNC                  EQU     H'0002'
TMR1CS                  EQU     H'0001'
TMR1ON                  EQU     H'0000'

;----- T2CON Bits ---------------------------------------------------------

TOUTPS3                 EQU     H'0006'
TOUTPS2                 EQU     H'0005'
TOUTPS1                 EQU     H'0004'
TOUTPS0                 EQU     H'0003'
TMR2ON                  EQU     H'0002'
T2CKPS1                 EQU     H'0001'
T2CKPS0                 EQU     H'0000'

;----- SSPCON Bits --------------------------------------------------------

WCOL                    EQU     H'0007'
SSPOV                   EQU     H'0006'
SSPEN                   EQU     H'0005'
```

```
CKP                 EQU     H'0004'
SSPM3               EQU     H'0003'
SSPM2               EQU     H'0002'
SSPM1               EQU     H'0001'
SSPM0               EQU     H'0000'

;----- CCP1CON Bits --------------------------------------------------------

CCP1X               EQU     H'0005'
CCP1Y               EQU     H'0004'
CCP1M3              EQU     H'0003'
CCP1M2              EQU     H'0002'
CCP1M1              EQU     H'0001'
CCP1M0              EQU     H'0000'

;----- RCSTA Bits ----------------------------------------------------------
SPEN                EQU     H'0007'
RX9                 EQU     H'0006'
RC9                 EQU     H'0006'    ; Backward compatibility only
NOT_RC8             EQU     H'0006'    ; Backward compatibility only
RC8_9               EQU     H'0006'    ; Backward compatibility only
SREN                EQU     H'0005'
CREN                EQU     H'0004'
ADDEN               EQU     H'0003'
FERR                EQU     H'0002'
OERR                EQU     H'0001'
RX9D                EQU     H'0000'
RCD8                EQU     H'0000'    ; Backward compatibility only

;----- CCP2CON Bits --------------------------------------------------------

CCP2X               EQU     H'0005'
CCP2Y               EQU     H'0004'
CCP2M3              EQU     H'0003'
CCP2M2              EQU     H'0002'
CCP2M1              EQU     H'0001'
CCP2M0              EQU     H'0000'

;----- ADCON0 Bits ---------------------------------------------------------

ADCS1               EQU     H'0007'
ADCS0               EQU     H'0006'
CHS2                EQU     H'0005'
CHS1                EQU     H'0004'
CHS0                EQU     H'0003'
GO                  EQU     H'0002'
NOT_DONE            EQU     H'0002'
GO_DONE             EQU     H'0002'
ADON                EQU     H'0000'

;----- OPTION_REG Bits -----------------------------------------------------

NOT_RBPU            EQU     H'0007'
INTEDG              EQU     H'0006'
T0CS                EQU     H'0005'
T0SE                EQU     H'0004'
PSA                 EQU     H'0003'
PS2                 EQU     H'0002'
PS1                 EQU     H'0001'
```

```
PS0                     EQU     H'0000'

;----- TRISE Bits --------------------------------------------------------

IBF                     EQU     H'0007'
OBF                     EQU     H'0006'
IBOV                    EQU     H'0005'
PSPMODE                 EQU     H'0004'
TRISE2                  EQU     H'0002'
TRISE1                  EQU     H'0001'
TRISE0                  EQU     H'0000'

;----- PIE1 Bits ---------------------------------------------------------

PSPIE                   EQU     H'0007'
ADIE                    EQU     H'0006'
RCIE                    EQU     H'0005'
TXIE                    EQU     H'0004'
SSPIE                   EQU     H'0003'
CCP1IE                  EQU     H'0002'
TMR2IE                  EQU     H'0001'
TMR1IE                  EQU     H'0000'

;----- PIE2 Bits ---------------------------------------------------------

EEIE                    EQU     H'0004'
BCLIE                   EQU     H'0003'
CCP2IE                  EQU     H'0000'

;----- PCON Bits ---------------------------------------------------------

NOT_POR                 EQU     H'0001'
NOT_BO                  EQU     H'0000'
NOT_BOR                 EQU     H'0000'

;----- SSPCON2 Bits ------------------------------------------------------

GCEN                    EQU     H'0007'
ACKSTAT                 EQU     H'0006'
ACKDT                   EQU     H'0005'
ACKEN                   EQU     H'0004'
RCEN                    EQU     H'0003'
PEN                     EQU     H'0002'
RSEN                    EQU     H'0001'
SEN                     EQU     H'0000'

;----- SSPSTAT Bits ------------------------------------------------------

SMP                     EQU     H'0007'
CKE                     EQU     H'0006'
D                       EQU     H'0005'
I2C_DATA                EQU     H'0005'
NOT_A                   EQU     H'0005'
NOT_ADDRESS             EQU     H'0005'
D_A                     EQU     H'0005'
DATA_ADDRESS            EQU     H'0005'
P                       EQU     H'0004'
I2C_STOP                EQU     H'0004'
S                       EQU     H'0003'
```

```
I2C_START               EQU     H'0003'
R                       EQU     H'0002'
I2C_READ                EQU     H'0002'
NOT_W                   EQU     H'0002'
NOT_WRITE               EQU     H'0002'
R_W                     EQU     H'0002'
READ_WRITE              EQU     H'0002'
UA                      EQU     H'0001'
BF                      EQU     H'0000'

;----- TXSTA Bits ----------------------------------------------------------

CSRC                    EQU     H'0007'
TX9                     EQU     H'0006'
NOT_TX8                 EQU     H'0006'    ; Backward compatibility only
TX8_9                   EQU     H'0006'    ; Backward compatibility only
TXEN                    EQU     H'0005'
SYNC                    EQU     H'0004'
BRGH                    EQU     H'0002'
TRMT                    EQU     H'0001'
TX9D                    EQU     H'0000'
TXD8                    EQU     H'0000'    ; Backward compatibility only

;----- ADCON1 Bits ---------------------------------------------------------

ADFM                    EQU     H'0007'
PCFG3                   EQU     H'0003'
PCFG2                   EQU     H'0002'
PCFG1                   EQU     H'0001'
PCFG0                   EQU     H'0000'

;----- EECON1 Bits ---------------------------------------------------------

EEPGD                   EQU     H'0007'
WRERR                   EQU     H'0003'
WREN                    EQU     H'0002'
WR                      EQU     H'0001'
RD                      EQU     H'0000'

;=========================================================================
;;      RAM Definition
;=========================================================================

      __MAXRAM H'1FF'
      __BADRAM H'8F'-H'90', H'95'-H'97', H'9A'-H'9D'
      __BADRAM H'105', H'107'-H'109'
      __BADRAM H'185', H'187'-H'189', H'18E'-H'18F'

;=========================================================================
;;      Configuration Bits
;=========================================================================

_CP_ALL                 EQU     H'0FCF'
_CP_HALF                EQU     H'1FDF'
_CP_UPPER_256           EQU     H'2FEF'
_CP_OFF                 EQU     H'3FFF'
_DEBUG_ON               EQU     H'37FF'
_DEBUG_OFF              EQU     H'3FFF'
_WRT_ENABLE_ON          EQU     H'3FFF'
```

```
_WRT_ENABLE_OFF         EQU     H'3DFF'
_CPD_ON                 EQU     H'3EFF'
_CPD_OFF                EQU     H'3FFF'
_LVP_ON                 EQU     H'3FFF'
_LVP_OFF                EQU     H'3F7F'
_BODEN_ON               EQU     H'3FFF'
_BODEN_OFF              EQU     H'3FBF'
_PWRTE_OFF              EQU     H'3FFF'
_PWRTE_ON               EQU     H'3FF7'
_WDT_ON                 EQU     H'3FFF'
_WDT_OFF                EQU     H'3FFB'
_LP_OSC                 EQU     H'3FFC'
_XT_OSC                 EQU     H'3FFD'
_HS_OSC                 EQU     H'3FFE'
_RC_OSC                 EQU     H'3FFF'
      LIST
```

MPLAB

CHE COS'È MPLAB?

- E' un ambiente di sviluppo (IDE Integrated Development Environment), organizzato a finestre e basato sul Windows, per i microcontrollori della famiglia PIC della Microchip Tecnology Incorpored.
- MPLAB permette di scrivere, effettuare il debug e ottimizzare un progetto firmware per i micro PIC.
- MPLAB include un text editor, un simulatore e un manager di progetti.
- MPLAB gestisce anche gli emulatori MPLAB-ICE e PICMASTER, i programmatori PICSTART Plus e PRO MATE II, il debugger-programmatore ICD (In Circuit Debugger), altri prodotti di sviluppo di applicazioni.

COMPITI DELL' MPLAB.

L'organizzazione degli strumenti MPLAB è il solito a menu e a finestre tipo Windows.
L'MPLAB permette di:

- assemblare, compilare e linkare programmi sorgente in assembly con il compilatore MPASM), esistono anche compilatori per C Basic);
- fare il debug dell'eseguibile tramite finestre di osservazioni insieme al simulatore o all'emulatore;
- fare misure di temporizzazioni;
- vedere l'evolversi di variabili in Watch windows;
- programmare il firmware con il programmatore PICSTART Plus;
- avere aiuti in linea.

MPLAB IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

L'MPLAB è uno strumento di sviluppo integrato, facile da imparare ed usare, che fornisce gli strumenti per sviluppare e debuggare il firmware per i micro PIC e gira sotto Windows.

L'MPLAB fornisce funzioni che permettono di :

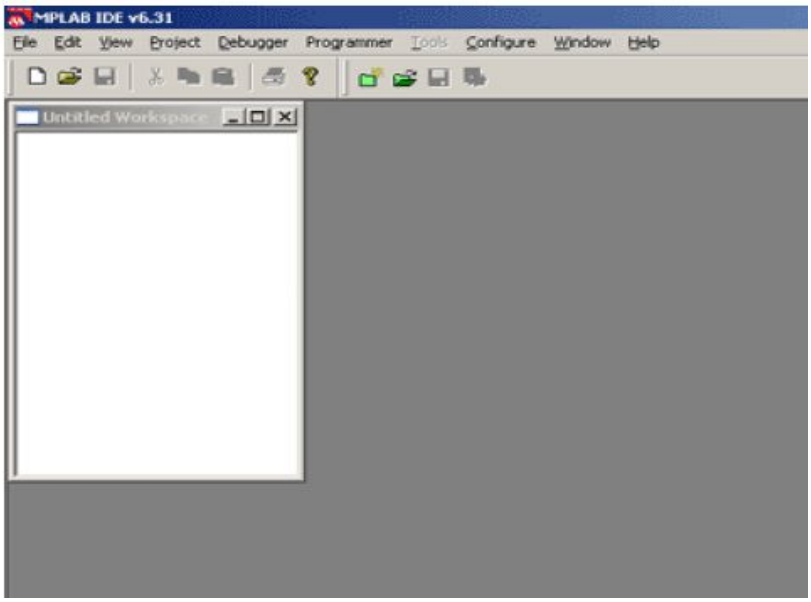
- creare ed editare file sorgenti tramite un completo text-editor;
- raggruppare più file sorgenti , file oggetto precompilati, librerie e file di link in un unico file progetto tramite un Project Manager;
- debuggare codici sorgenti con l'aiuto di una finestra di errori di compilazione;
- debuggare codici eseguibili con l'uso di simulatori o emulatori.
- vedere l'evolversi dell'esecuzione col metodo passo-passo e/o mettere dei punti di break.

L'MPLAB fornisce inoltre un certo numero di finestre che permettono di vedere
il contenuto di tutte le locazioni di memoria sia dell'area programma che di quella dati.

STRUMENTI DI SVILUPPO MPLAB IDE.

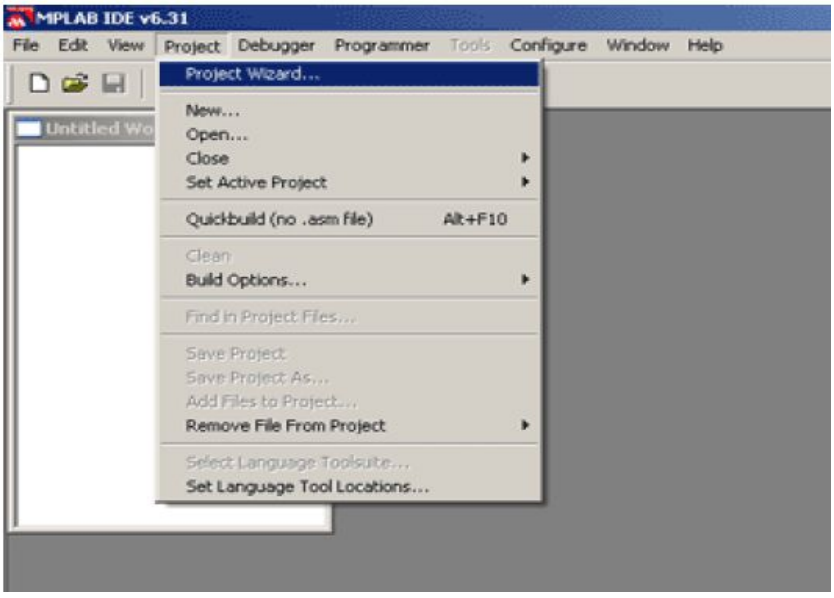
- MPLAB Project Manager;
- MPLAB Editor;
- MPLAB-ICD In-Circuit Debugger;
- MPLAB-SIM Simulator;
- MPLAB-ICE Emulator;
- MPASM Assembler/MPLINK Linker/MPLIB Librarian;
- PICSTART Programmer:
- altri.

INTRODUZIONE AL PROGRAMMA MPLAB.

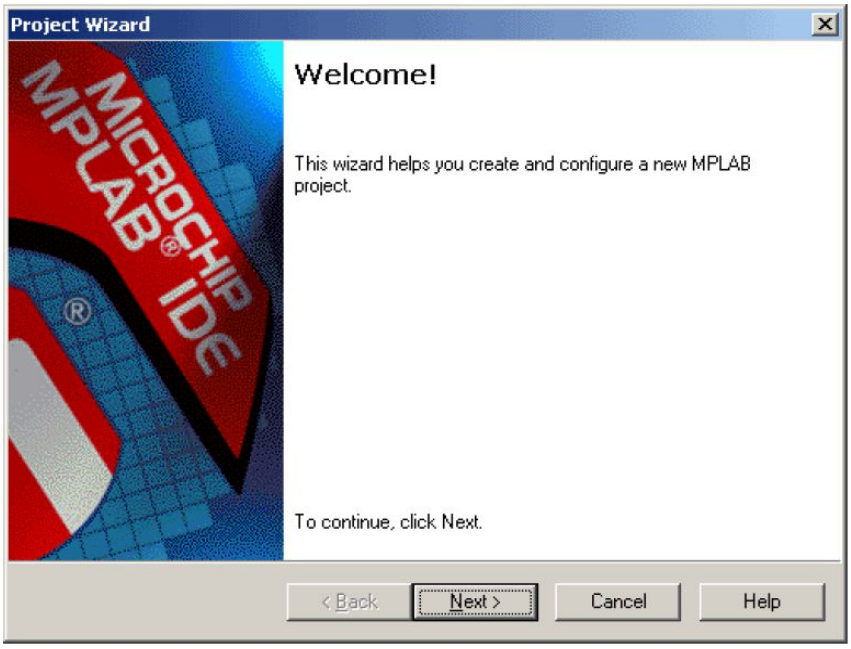


La preparazione del programma da caricare nel microcontrollore richiede i seguenti passi:

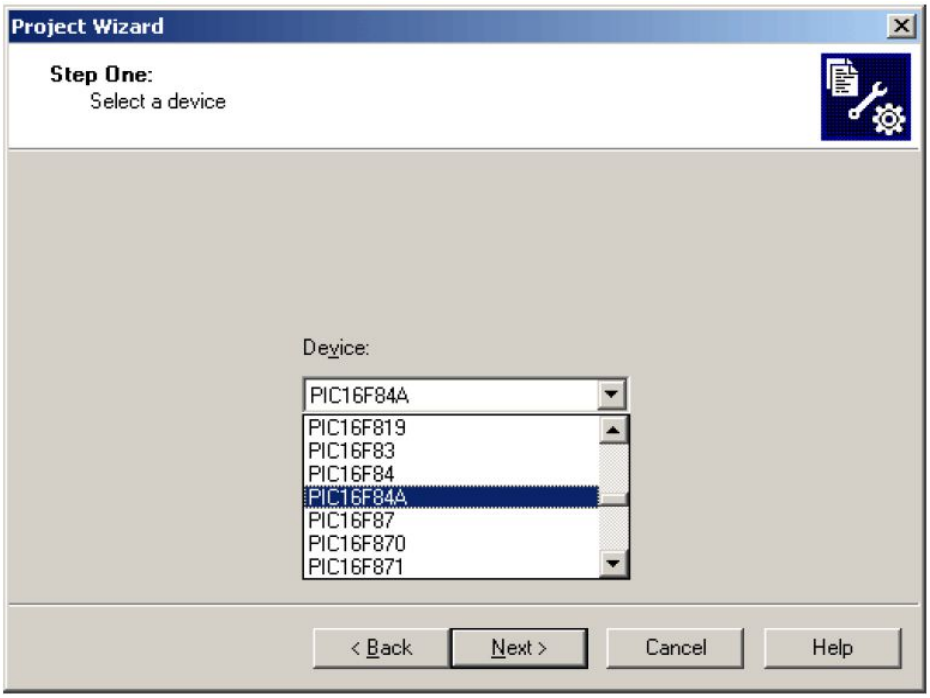
1. Creare un progetto
2. Scrivere il programma
3. Convertirlo in binario (compilare).



Cliccare su PROJECT e successivamente su PROJECT WIZARD, sarà aperta la seguente finestra.



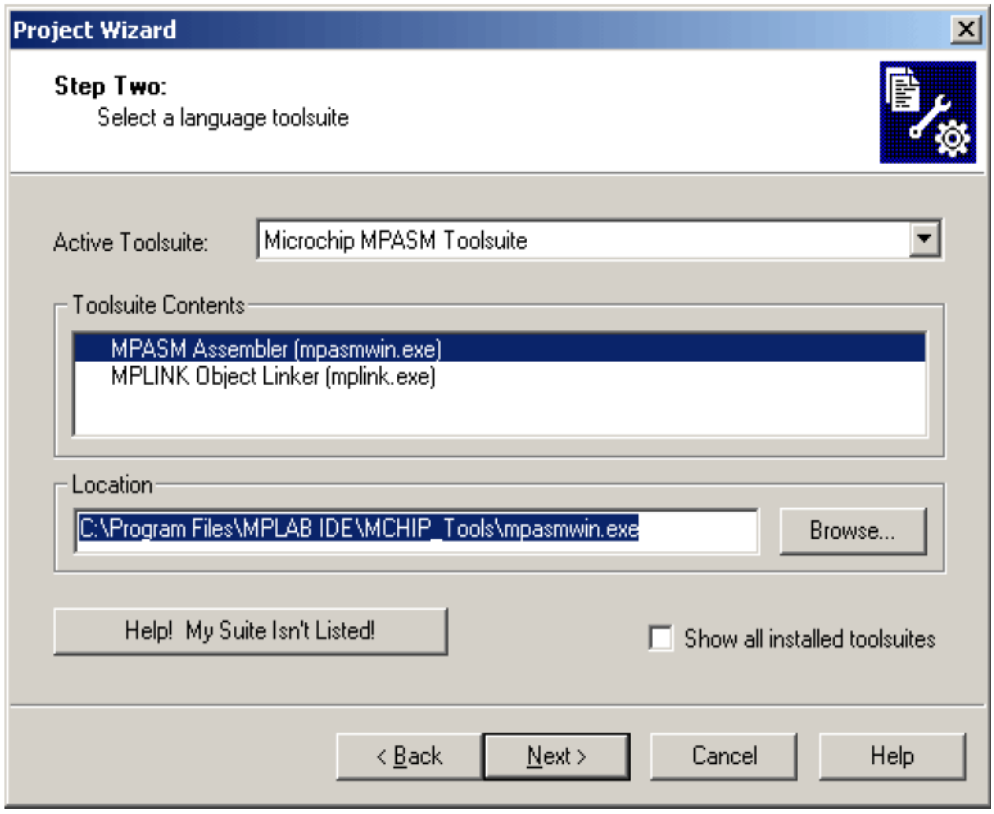
Cliccare su NEXT per continuare. Occorre adesso scegliere l'appropriato microcontrollore della famiglia PIC.



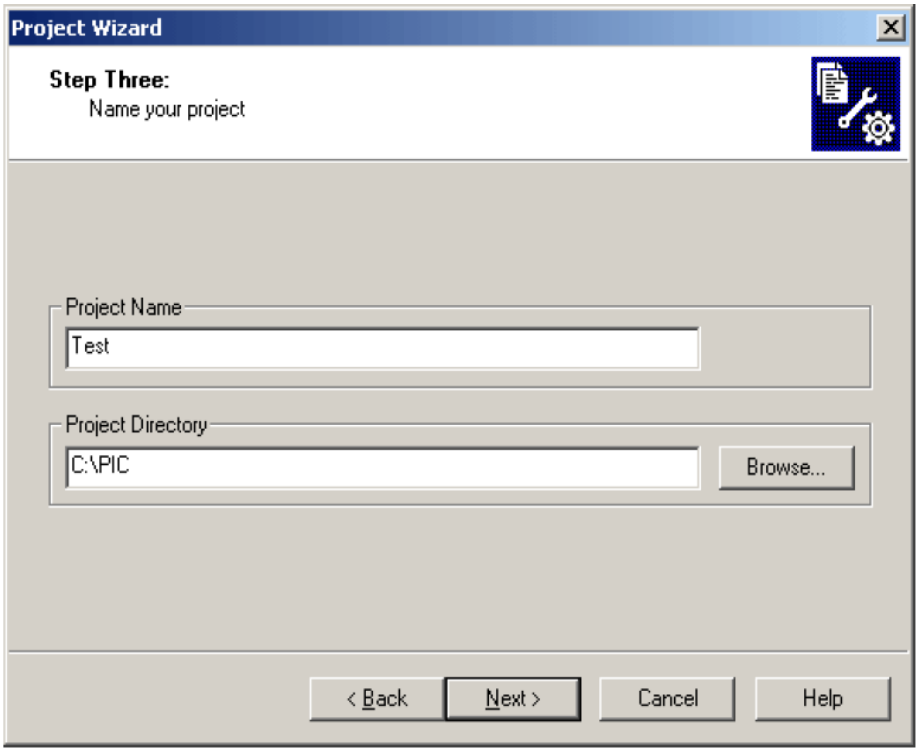
Il passo successivo consiste nel definire il linguaggio che si intende utilizzare per la programmazione. Useremo direttamente il linguaggio assembler, pertanto occorre selezionare l'opzione MPASM Assembler come indicato in figura seguente.

In questo punto ci viene mostrato l' ambiente di programmazione da utilizzare (toolsuite), e quello che esso contiene. Quello scelto da noi è una versione gratuita e contiene diversi assemblatori, come l'MPASM, l'MPLINK e l'MPLIB. Nella locazione sottostante verrà specificato il percorso in cui si trova l'assemblatore scelto da noi (in questo caso quello dell' MPASM). Se invece noi abbiamo bisogno di un' altro assemblatore non presente nella lista, basterà cliccare nella casella sottostante

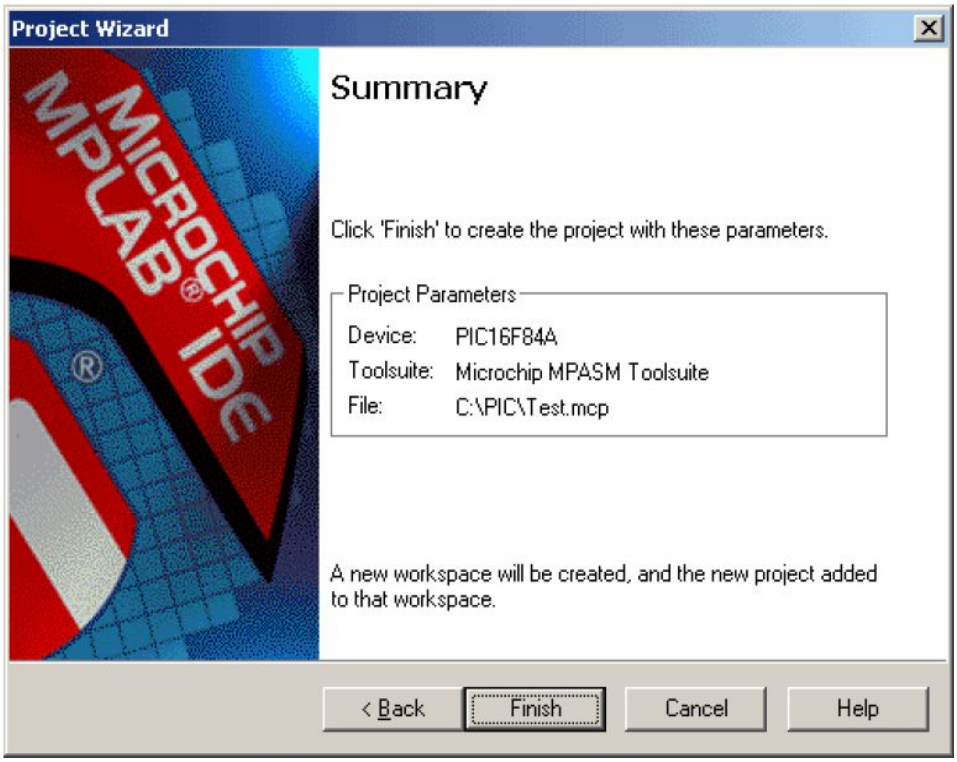
alla “Location”.



Indichiamo il nome del progetto e il percorso. Il nome dovrebbe riflettere lo scopo e il contenuto del programma. Raggruppiamo il progetto in una cartella in grado di ricordarci il contenuto; nel nostro caso si utilizzera il nome PIC come qui di seguito riportato in figura.

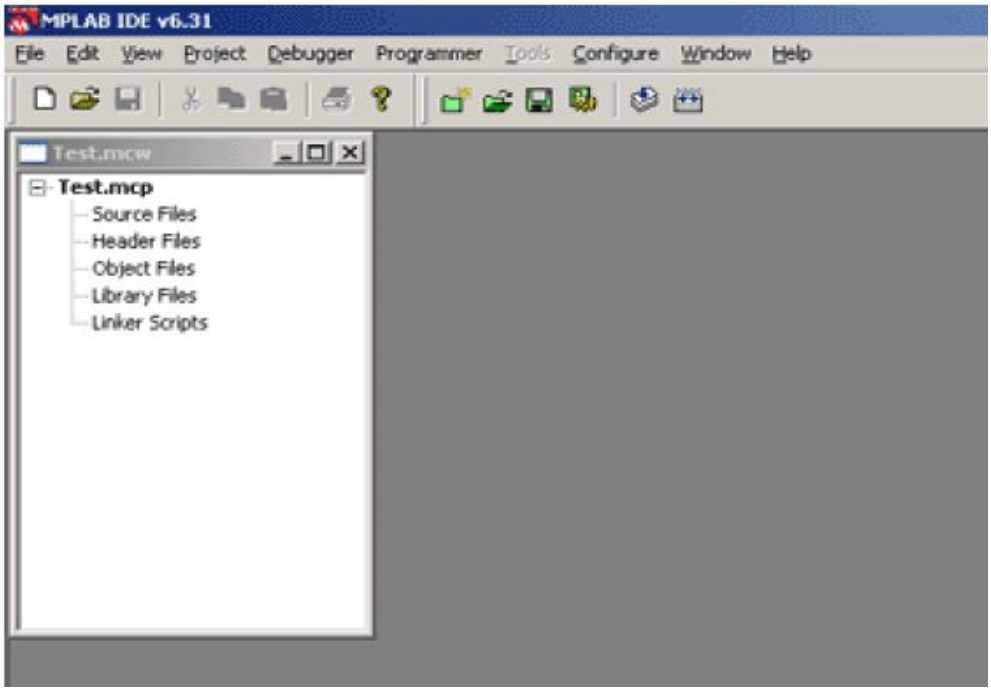


Cliccare NEXT --> si aprirà la finestra “summary”.



Cliccare FINISH per creare il progetto.

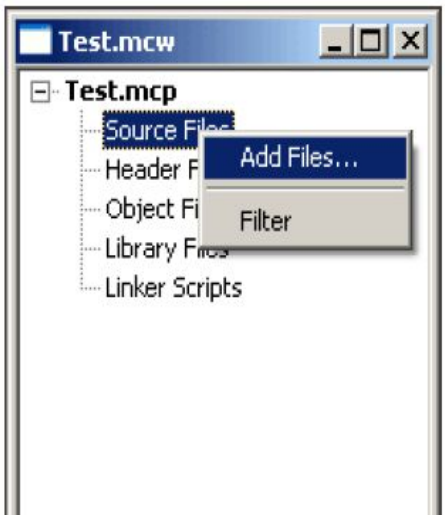
Dopo aver creato il progetto mediante il wizard dovrebbe apparire la seguente schermata.



Adesso dobbiamo scrivere il programma, questa operazione richiede ulteriori file da dover aprire. Cliccare su FILE > NEW, si aprirà una nuova finestra all'interno della working area di MPLAB (La nuova finestra conterra il programma che si dovrà scrivere). Dopo aver aperto il

nuovo file dobbiamo salvarlo (è buona regola), useremo il solito percorso C:\PIC ed il nome del file sarà "Blink.asm", questo è un nome che ci permetterà di ricordare la natura del programma (ad esempio questo farà labpeggiare (blink) uno o più diodi led sulla porta B di un microcontrollore).

Il nuovo file, "Blink.asm" dovrebbe così essere incluso al progetto. Premere il tasto destro del mouse sul "source file" nella finestra "blink.mcw". Questo aprirà una piccola finestra con due opzioni – scegliamo la prima, "Add Files".



MPLAB ICD

CHE COS'È MPLAB ICD

- MPLAB ICD utilizza le capacità di In-Circuit Debugging del PIC16F87X e il protocollo In-Circuit Serial Programming (ICSP) della Microchip sia per programmare che per funzionare come un debugger in-circuit per i micro della famiglia 16F87x.
- Esso opera sotto MPLAB IDE, connesso al circuito applicativo .
- MPLAB ICD viene usato come aiuto nello sviluppo di applicazioni.

L'MPLAB ICD ha le seguenti caratteristiche:

- esecuzione del codice in real-time e single-step;
- possibilità di breakpoint;
- debug direttamente sull'applicativo (in-circuit);
- programmazione integrata (built-in);
- range operativo 3.0V … 5,5V;
- la tensione di alimentazione V_{DD} è prelevata dal circuito applicativo;
- range di frequenza: 32kHz … 20MHz;
- debug a livello sia di sorgente che di simbolico;
- usa l'interfaccia grafica MPLAB IDE;
- compatibile con Windows;
- uso dell'interfaccia RS-232.

RISORSE USATE DA MPLAB ICD (ICD 1.4)

L'MPLAB ICD, per eseguire il debug in-circuit e/o la programmazione integrata (buit-in), usa le seguenti risorse del chip:

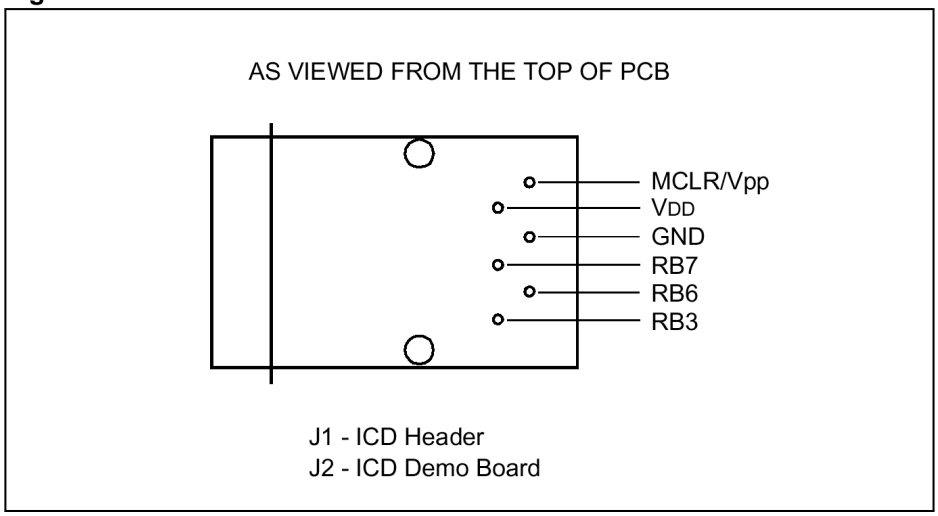
- il pin MCRL/Vpp di programmazione è condiviso;
- la programmazione Low-voltage ICSP è disabilitata; la programmazione low-voltage non è supportata dal MPLAB ICD e quindi durante il funzionamento in modo debug, questa opzione deve essere disabilitata;
- i pin RB6 e RB7 sono riservati per la programmazione e il debug;
- sei o sette locazioni di memoria dati (general purpose file) sono riservate per il controllo di debug (vedi tabella);
- la prima locazione di memoria programma (indirizzo 0x000) deve essere una operazione di NOP;
- le ultime 256 o 288 locazioni della memoria programma sono riservate per il codice di debug (vedi tabella);
- un livello dello stack non è utilizzabile.

Processor	File Register usati	Program Memory usata
PIC16F870/871/872	0x70, 0x0BB-0x0BF	0x06E0-0x07FF
PIC16F873/874	0x6D, 0x16D-0x0EB; 0x0F0, 0x1EB-0x1F0	0x0EE0-0x0FFF
PIC16F876/877	0x70, 0x1EB-0x1EF	0x1F00-0x1FFF

ICD Module Connector

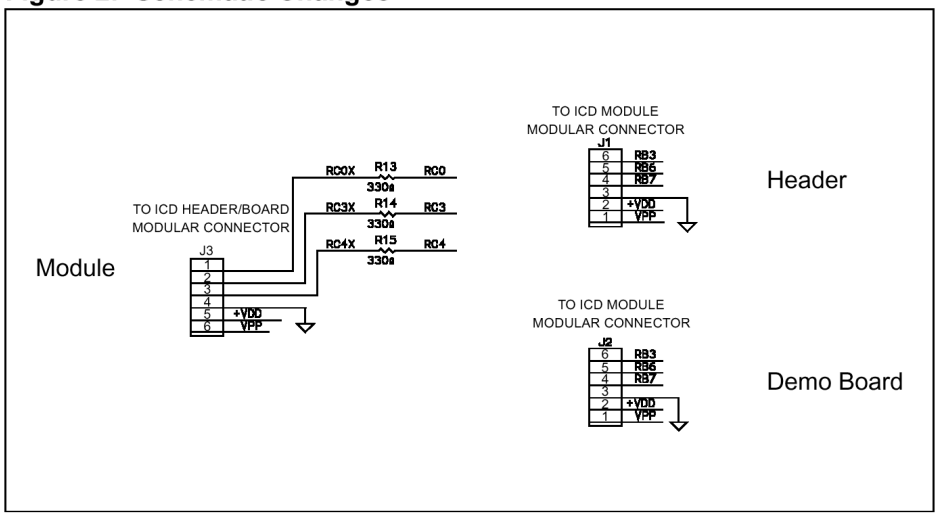
You can put a connector on your target board to connect the MPLAB-ICD debugger to the PIC16F87X processor on your application. The cable connecting the MPLAB-ICD Demo Board or ICD Header to the ICD Module has the following end plug: AMP 5-554710-3, Plug, 6-6 Strand Round Modular Plug or equivalent. This part is available from Digikey, A9117-ND. The pinout for the cable connector is shown in the Figure 1.

Figure 1: ICD Module Connector Detail



For consistency, the following changes have been made to the schematics as shown in Figure 2.

Figure 2: Schematic Changes



SCHEDA PER ESERCITAZIONI DEMO BOARD

La scheda per esercitazioni DEMO BOARD è stata progettata e realizzata nel laboratorio di TDP dagli insegnanti e dal personale non docente per fornire agli studenti un circuito che permette di eseguire agevolmente una serie di esercitazioni. Ciò permette di familiarizzare con la programmazione del microcontrollore, con il debug dei programmi e dà un'idea delle applicazioni di base del micro.

Nello schema della pagina seguente si individuano facilmente alcune sezioni:

ALIMENTAZIONE
In basso a sinistra.
Il circuito può essere alimentato con una tensione alternata applicata al connettore M1 compresa tra 10 e 18 V e raddrizzata dal ponte a diodi PD1 e filtrata dall'induttore L1 e dai condensatori C7 e C8 che la rendono quasi continua.
Il valore della tensione varia però con la tensione alternata applicata a M1 mentre il circuito ha bisogno di una tensione fissa. L'integrato U2 riduce appunto la tensione a circa 5V, più precisamente 5 V ± 5 % (4,75 ÷ 5,25 V) e assicura l'alimentazione del resto del circuito. La tensione d'uscita è ulteriormente stabilizzata dai condensatori C9 e C10.
Infine il LED verde dà un'indicazione luminosa della presenza dell'alimentazione.

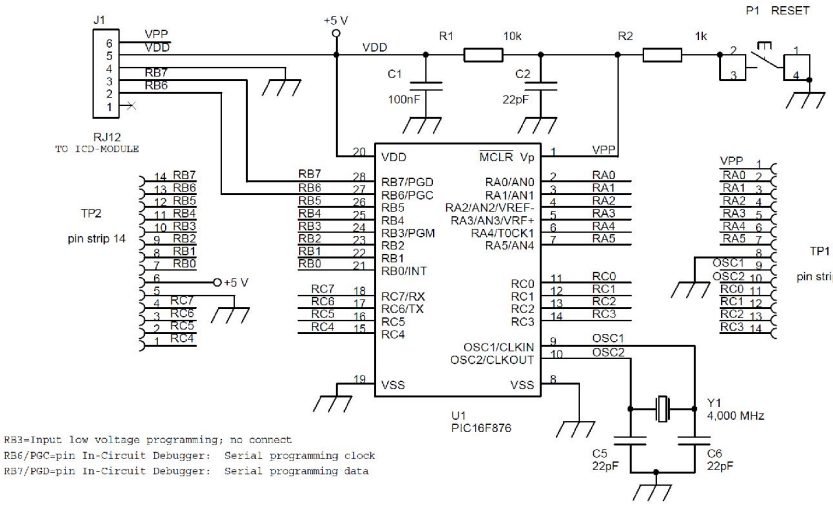
LED
Sono collegati alla porta C se il connettore J2 è nella posizione di sinistra cioè collega la fila centrale e quella di sinistra dei contatti.
Nello stampato è indicato il "peso" binario dei LED, il 128 indica che il LED è collegato bit più significativo della porta.
Sommando i valori dei LED accesi si può convertire in decimale il valore memorizzato nella porta C.

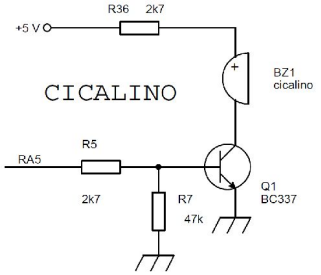
VISUALIZZATORI a LED a 7 SEGMENTI
Se il connettore J2 è nella posizione di destra alla porta C sono collegati i due visualizzatori a LED a 7 SEGMENTI. Per accenderli è necessario anche inserire il ponticello PT1 che permette il corretto funzionamento dei transistor Q2 e Q3.

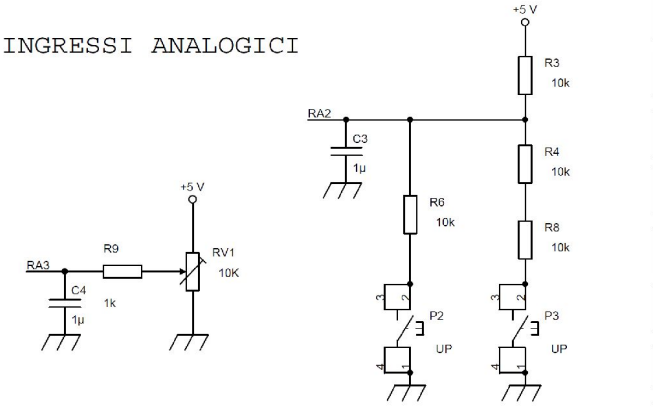
CICALINO
Quando è presente un valore logico alto sul terminale RA5 il transistor Q1 è in saturazione e il cicalino BZ1 emette il suono caratteristico.
R36 limita la corrente che attraversa il cicalino per evitare un rumore eccessivo.

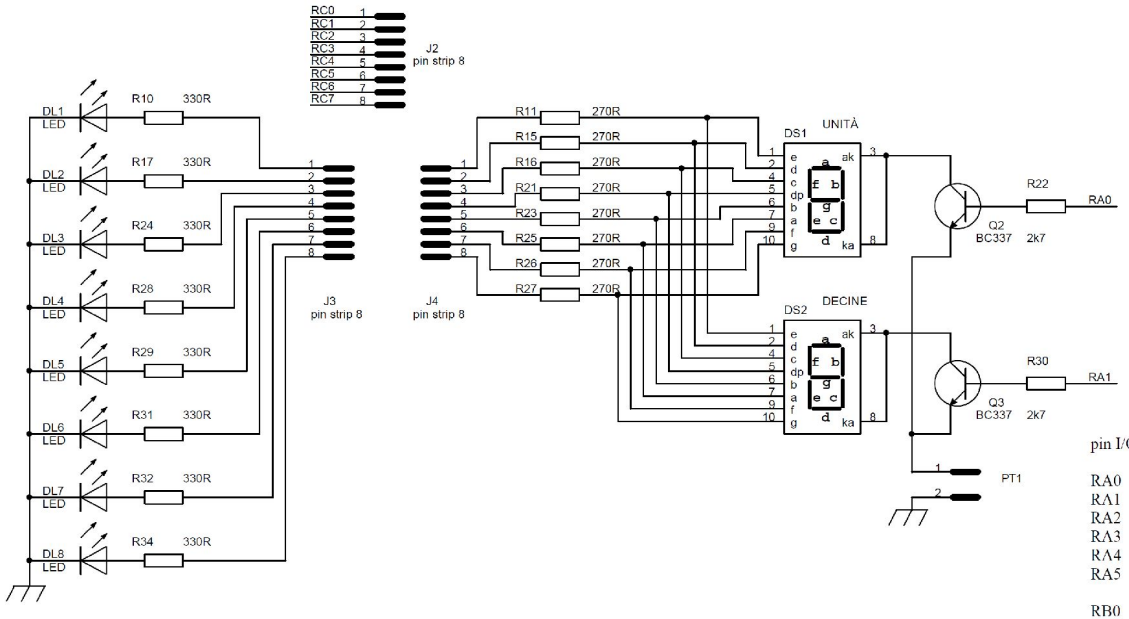
LCD
Nel visualizzatore LCD si possono mostrare 16 caratteri per riga su due righe oppure 16 caratteri su una sola riga. Nel caso di una sola riga i caratteri sono un po' più grandi di quelli visualizzati quando si usano due righe.

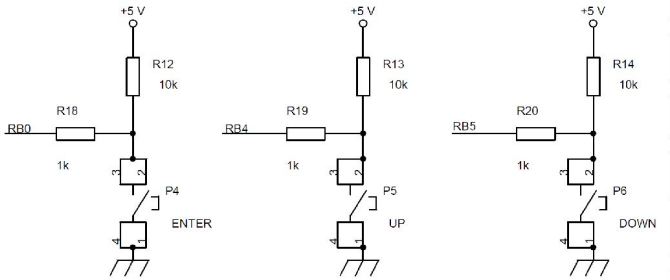
INGRESSI ANALOGICI
Questa sezione genera tensioni diverse da quelle associate ai livelli logici e permette di usare il convertitore analogico-digitale. In particolare premendo solo P2 la tensione dell'ingresso RA2 vale circa metà della tensione di alimentazione, premendo invece solo P3 RA2 scende a circa 1/3 Vcc.
RV1 invece è un trimmer che manda a RA3 una tensione variabile tra 0 e Vcc. Se si usa la stessa Vcc anche per la tensione di riferimento del convertitore A/D ruotando il trimmer il risultato della conversione varierà tra tutti 0 e tutti 1.









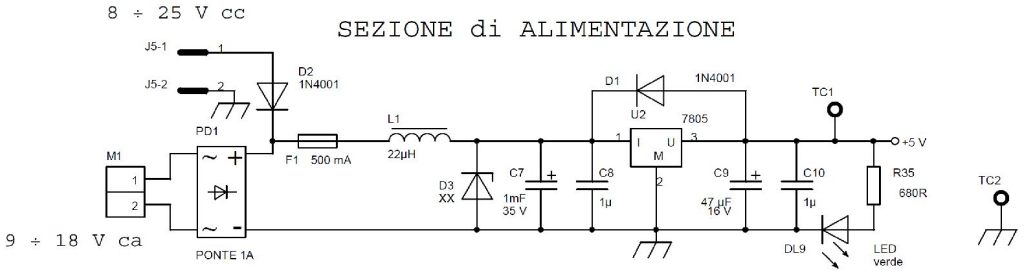


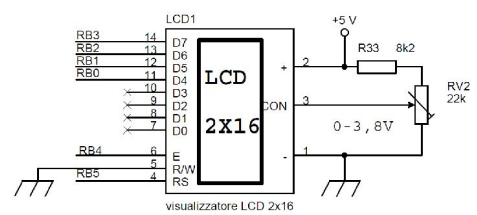
pin I/O funzione

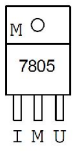
RA0 unità
RA1 decine
RA2 pulsanti analogici
RA3 trimmer
RA4 n. c.
RA5 cicalino

RB0 D4 del LCD o P4-ENTER
RB1 D5 del LCD
RB2 D6 del LCD
RB3 D7 del LCD
RB4 E del LCD o P5-UP
RB5 RS del LCD o P6-DOWN
RB6
RB7

RC0 DL1 o <e> del 7 segmenti
RC1 DL2 o <d> del 7 segmenti
RC2 DL3 o <c> del 7 segmenti
RC3 DL4 o punto decimale
RC4 DL5 o <b> del 7 segmenti
RC5 DL6 o <a> del 7 segmenti
RC6 DL7 o <f> del 7 segmenti
RC7 DL8 o <g> del 7 segmenti







x = 5; y = 2 su Laserjet 4000N

Ing. Duilio De Marco

Title: Scheda applicativa PIC 16F876

Size	Document Number	Rev
A3	1.0	4.0

Date: Saturday, December 10, 2005 Sheet 1 of 2

RS-232

Lo standard RS-232 è un protocollo seriale di tipo asincrono

Seriale significa che i bit che costituiscono l'informazione sono trasmessi uno alla volta su di un solo conduttore. Questo termine è in genere contrapposto a "parallelo": in questo caso i dati sono trasmessi contemporaneamente su più fili.
Parlando astrattamente si potrebbe pensare che la trasmissione seriale sia intrinsecamente più lenta di quella parallela (su di un filo possono passare meno informazioni che su 8 o 16). In realtà questo non è vero in assoluto, soprattutto a causa della difficoltà di controllare lo skew (disallineamento temporale tra i vari segnali) dei molti trasmettitori in un bus parallelo, e dipende dalle tecnologie adottate: p.e. in una fibra ottica, in un cavo ethernet, USB o FireWire (tutti standard seriali) le informazioni transitano ad una velocità paragonabile a quella di un bus PCI a 32 fili. In questa nota applicativa si parlerà solo di un'interfaccia seriale "lenta" gestibile da PC e microcontrollori: RS-232.

Asincrono significa**, i**n questo contesto, che i dati sono trasmessi senza l'aggiunta di un segnale di clock, cioè di un segnale comune che permette di sincronizzare la trasmissione con la ricezione; ovviamente sia il trasmettitore che il ricevitore devono comunque essere dotati di un clock locale per poter interpretare i dati. La sincronizzazione dei due clock è necessaria ed è fatta in corrispondenza della prima transizione sulla linea dei dati.

LE UNITÀ DI MISURA

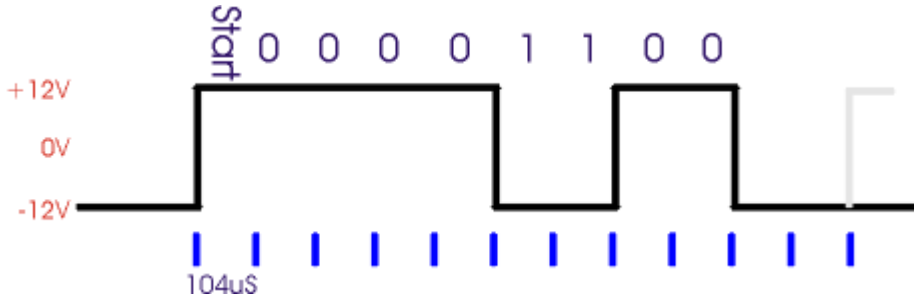
Le unità di misura della velocità di trasmissione sono essenzialmente due: il *baud* ed il *bit per secondo* (bps o, meno spesso, b/s), spesso trattate erroneamente come sinonimi.
Il *baud rate* indica il numero di simboli al secondo che vengono trasmessi sulla linea; il *bps* indica, come dice il nome, quanti bit al secondo sono trasmessi lungo la linea.
Nel caso di trasmissione binaria (cioè è presente un livello alto ed uno basso) le due cose ovviamente coincidono, da cui la *parziale* equivalenza dei due termini. Nel caso di trasmissioni a più livelli, invece, è possibile associare a un solo livello più bit: se per esempio posso trasmettere otto diversi valori di tensione tra 0 e 7 volt, con un solo valore di tensione invio tre bit (0 V = 000, 1 V = 001, 2 V = 010…) ed in questo caso una trasmissione a 1000 baud equivale ad una a 3000 bps.
Nello standard RS232 sono trasmessi segnali binari: quindi il baud rate, pari all'inverso della durata di un bit, coincidente numericamente con il bps.

HALF-DUPLEX E FULL-DUPLEX

Il termine Half-duplex è usato per indicare trasmissioni bidirezionali non simultanee nelle due direzioni: un dispositivo (ricevitore, listener o Rx) ascolta e l'altro (trasmettitore, talkner o Tx) emette segnali. Quando è necessario si scambiano i ruoli.
La trasmissione full-duplex indica che la trasmissione è bidirezionale e contemporanea. In questo caso è necessario ovviamente un altro collegamento. Lo standard RS232 è full-duplex in quanto è utilizzato un conduttore separato per ciascuna direzione delle informazioni. Il vincolo è in genere limitato alla necessità che trasmissione e ricezione abbiano lo stesso formato.
Se la trasmissione è sempre in un solo verso viene detta simplex.

COME È FATTO UN SEGNALE RS-232

La cosa più semplice per descrivere un segnale RS232 è partire con un esempio.
Nell'immagine che segue è visualizzato, in modo idealizzato, cosa appare collegando un oscilloscopio ad un filo su cui transita un segnale RS-232 8n2 a 9600 bps rappresentante il valore binario 00110000.



L'ampiezza del segnale è caratterizzata da un valore "alto" pari a circa +12V ed un valore "basso" pari a –12V. Da notare che, nello standard RS-232 un segnale alto rappresenta lo zero logico ed uno basso un uno, come indicato nel disegno e rovesciato rispetto al comune pensare.
A volte un segnale alto (+12V, cioè uno zero logico) è indicato come **space** ed uno basso (-12V, uno logico) come **mark**.
Tutte le transizioni appaiono in corrispondenza di multipli di 104us (pari a 1/9600 cioè ciascun bit dura esattamente l'inverso del baud rate).
La linea si trova inizialmente nello stato di riposo, bassa (nessun dato in transito); la prima transizione da basso in alto indica l'inizio della trasmissione (inizia il "bit di start", lungo esattamente 104us). Segue il bit meno significativo (LSB), dopo altri 104 MSN il secondo bit, e così via, per otto volte, fino al bit più significativo (MSB). Da notare che il byte è trasmesso "al contrario", cioè va letto da destra verso sinistra. Segue infine un periodo di riposo della linea di almeno 208us, cioè due bit di stop e quindi (eventualmente) inizia un nuovo pacchetto di bit.
Le varianti possibili sono le seguenti:

- Se la trasmissione è più veloce o più lenta, la distanza tra i fronti varia di conseguenza (p.e. a 1200 bps le transizioni avvengono a multipli di 0,833 ms, pari a 1/1200)
- Invece di trasmettere 8 bit, ne posso trasmettere 6, 7 o anche 9 (ma quest'ultima possibilità non è prevista dalle porte seriali dei normali PC)
- Alla fine è possibile aggiungere un bit di parità
- Alla fine la linea rimane nello stato di riposo per almeno 1 o 1.5 o 2 bit (bit di stop); notare che, se non ho più nulla da trasmettere, il riposo è molto più lungo, ovviamente. Molti sistemi non possono utilizzano 1.5 bit di stop

In genere il formato del pacchetto trasmesso è indicato da una sigla composta da numeri e cifre, per esempio 8n1 e 7e2:

- La prima cifra indica quanti bit di dati sono trasmessi (nei due esempi 8 e 7)
- La prima lettera il tipo di parità (rispettivamente nessuna ed even-parity)
- La seconda cifra il numero di bit di stop (rispettivamente 1 e 2)

Tenendo conto che esiste sempre un solo bit di start, un singolo blocco di bit è quindi, per i due esempi riportati, costituito rispettivamente da 10 (1+8+0+1) e 11 (1+7+1+2) bit. Da notare che di questi bit solo 8 e, rispettivamente, 7 sono effettivamente utili.
Lo standard originale prevede una velocità fino a 20Kbps. Uno standard successivo (RS-562) ha portato il limite a 64Kbps lasciando gli altri parametri elettrici praticamente invariati e rendendo quindi i due standard compatibili a bassa velocità. Nei normali PC le cosiddette interfacce seriali RS-232 arrivano in genere almeno a 115Kbps o anche più: pur essendo formalmente al di fuori di ogni standard ufficiale non si hanno particolari problemi di interconnessione.
Ovviamente sia trasmettitore che ricevitore devono accordarsi sul modo di trasmettere i dati prima di iniziare la trasmissione.
E' importante garantire il rigoroso rispetto della durata dei singoli bit: infatti non è presente alcun segnale di clock comune a trasmettitore e ricevitore e l'unico elemento di sincronizzazione è dato dal fronte si salita del

bit di start. Come linea guida occorre considerare che il campionamento in ricezione è effettuato di norma al centro di ciascun bit: l'errore massimo ammesso è quindi pari alla durata di mezzo bit (circa il 5% della frequenza di clock, considerando che anche il decimo bit deve essere correttamente sincronizzato). Naturalmente questo limite non tiene conto della possibile difficoltà di riconoscere con precisione il fronte del bit di start (soprattutto su grandi distanze ed in ambiente rumoroso) e della presenza di interferenze intersimboliche tra bit adiacenti: per questo spesso si consiglia caldamente di usare un clock con una precisione migliore del 1 % imponendo di fatto l'uso di oscillatori a quarzo.

IL BIT DI PARITÀ

Oltre ai bit dei dati (in numero variabile tra 5 e 8) viene inserito un bit di parità (opzionale) per verificare la correttezza del dato ricevuto. Esistono cinque tipi di parità:

- None: nessun tipo di parità, cioè nessun bit aggiunto
- Pari (even): il numero di mark (incluso il bit di parità) è sempre pari
- Dispari (odd): il numero di mark (incluso il bit di parità) è sempre dispari
- Mark: il bit di parità vale sempre mark
- Space: il bit di parità vale sempre space

I PARAMETRI ELETTRICI RS-232

La tensione di uscita a un trasmettitore RS232 deve essere compresa in valore assoluto tra 5 V e 25 V (quest'ultimo valore ridotto a 13 V in alcune revisioni dello standard). A volte le tensioni in uscita sono intenzionalmente diminuite a +/- 6 V anziché 12 V per limitare le emissioni EMC, peraltro sempre critiche, e favorire maggiori velocità di trasmissione.
Il ricevitore deve funzionare correttamente con tensioni di ingresso comprese, sempre in modulo, tra 3 V e 25 V. Molti ricevitori commerciali considerano semplicemente una tensione di soglia al valore di +2V (sopra viene riconosciuto un segnale alto, sotto uno basso) anche se ciò non è pienamente aderente alle norme.
L'impedenza di uscita deve in ogni situazione essere maggiore di 300 ohm (anche a dispositivo spento). L'impedenza di ingresso deve essere compresa tra 3 e 7 kΩ. La corrente prelevabile in uscita mantenendo i corretti valori logici deve essere di almeno di 1.6 mA (in genere però è maggiore di quasi un ordine di grandezza) e nel caso di corto circuito deve essere minore di 100mA.
Infine lo slew-rate (cioè la velocità di variazione della tensione del segnale durante la commutazione) deve essere minore di 30 V/μs per evitare eccessive emissioni elettromagnetiche.

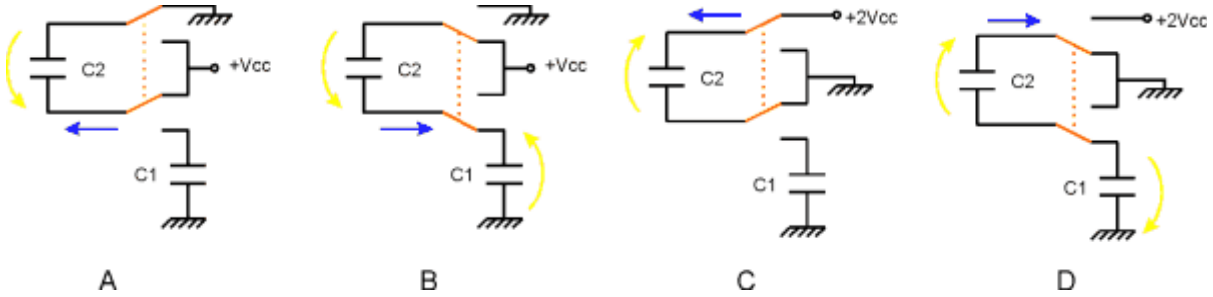
COME COLLEGARE PORTE TTL O CMOS ALLA RS232

In genere i segnali utilizzati dai sistemi digitali variano tra 0 e 5 V e non sono quindi direttamente compatibili con la standard RS232. In commercio esistono appositi *traslatori di livello* che hanno il compito di fornire sia in trasmissione che in ricezione gli opportuni livelli pur non modificando la forma del segnale trasmesso.
Il MAX232 (ed integrati simili, fatti da un po' tutti i produttori di semiconduttori) è un circuito integrato che permette il collegamento tra logica TTL o CMOS a 5 V e le tensioni RS-232, partendo solo da un'alimentazione a 5 V.
Per ottenere la tensione positiva e negative necessarie per il funzionamento dell'integrato è usata una configurazione a pompa di carica, costituito da circuiti interni all'integrato e quattro condensatori esterni generalmente da 1 μF. La capacità effettiva dipende dal tipo di integrato e dalla relativa frequenza di commutazione; a volte i condensatori sono presenti all'interno dell'integrato stesso.
Sono disponibili anche integrati che richiedono un'alimentazione di solo 3.3 V (p.e. il MAX3232).
La sezione ricevente del MAX232 è costituita da due porte invertenti che accettano in ingresso una tensione di +/- 12 V (o altra tensione compatibile allo standard RS232) ed in uscita forniscono un segnale TTL compatibile.

La sezione trasmittente ha due driver con ingresso TTL compatibile e in uscita un driver capace di erogare una tensione di poco meno di +/- 10 V, compatibile con lo standard RS232 (dipende dal carico).

UN CIRCUITO A POMPA DI CARICA

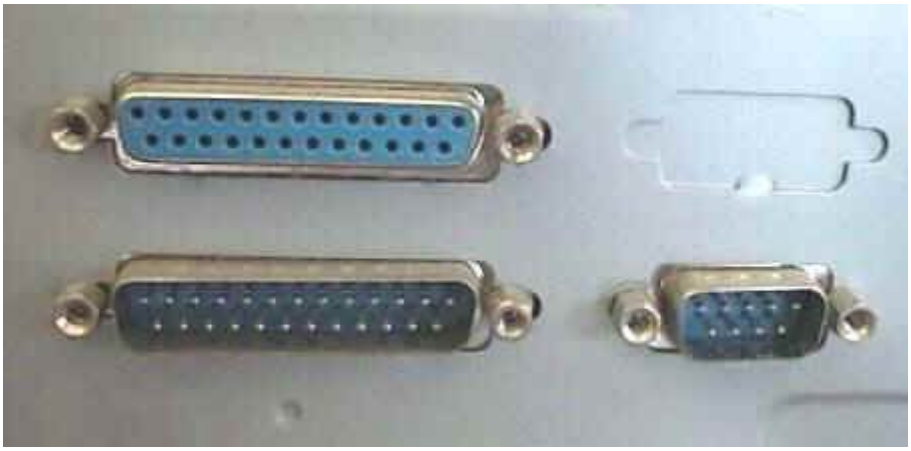
Per ricavare le tensioni positive e negative necessarie per garantire i livelli richiesti dalla RS232 è pratica comune utilizzare un duplicatore ed un invertitore di tensione a pompa di carica.



Le figure A e B mostrano come viene ottenuto il raddoppio della tensione.
Inizialmente il condensatore C2 viene connesso tra massa e Vcc; quindi la corrente carica C2 alla tensione di alimentazione.
C2 viene successivamente connesso tra Vcc ed un secondo condensatore C1; La tensione ai capi di C1 deve essere uguale alla somma di Vcc e della tensione ai capi di C2 che, scaricandosi verso C1, ne aumenta la tensione rispetto a massa.
Il processo è ripetuto fino a quando la tensione ai capi di C1 è uguale a 2Vcc: in questo caso infatti C2 non si scarica più.
Analogamente le figure C e D mostrano l'inversione di tensione:
Inizialmente C2 è caricato alla tensione 2Vcc, ricavata con il precedente circuito)
Quindi C2 è connesso tra massa e C1 avendo cura di invertire le polarità affinché C1 si carichi a –Vcc.
Il limite dei circuiti a pompa di carica è la limitata quantità di corrente disponibile (infatti se prelevo corrente da C1 questo tende a scaricarsi, facendo scendere la tensione).

LA PIEDINATURA DEL CONNETTORE RS-232 DEL PC

Sono disponibili sul PC due tipi di connettori RS-232: DB9 (nove pin) e DB25 (25 pin, il connettore originale e presente solo sui PC più vecchi); ambedue i connettori sono maschi e praticamente identici dal punto di vista funzionale anche se non coincidente con quello proposto dallo standard ufficiale.

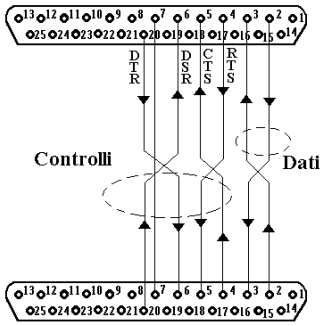


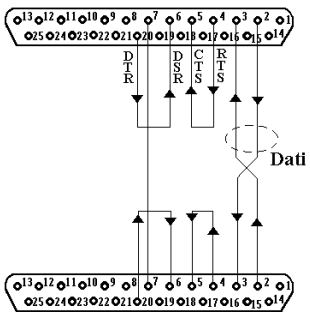
Nell'immagine sono visualizzati in basso i due tipi di connettori utilizzati sui PC per l'interfaccia RS232. Quello in alto è relativo alla porta parallela.
Di seguito è riportata una tabella con indicati i nomi dei segnali, il numero dei pin e la direzione dei segnali stessi(O = uscita dal PC).

Sigla	25pin	9pin	In/Out	Nome
TxD	2	3	O	Dati trasmessi
RxD	3	2	I	Dati ricevuti
RTS	4	7	O	Rest To Send
CTS	5	8	I	Clear To Send
DTR	20	4	O	Data Terminal Ready
DSR	6	6	I	Data Set Ready
RI	22	9	I	Ring Indicator
DCD	8	1	I	Data Carrier Detect
GND	7	5	-	Massa
-	-	1	-	Terra

PERCHÉ TANTI FILI ?

In teoria per ricevere e trasmettere un segnale RS-232 bastano tre fili: ricezione, trasmissione e massa. Spesso lo è anche in pratica.
Gli altri fili (spesso opzionali, ma dipende dall'applicazione) servono per il cosiddetto handshake tra PC e periferica (o tra PC e PC) cioè per sincronizzare in hardware la comunicazione.
Sono presenti due coppie di fili:
RTS/CTS: quando il PC inizia la trasmissione pone RTS alto, la periferica risponde quando pronta ponendo CTS alto. Per interrompere la trasmissione la periferica pone CTS basso.
DTR/DSR: Quando il PC è collegato per la prima volta, pone alto DTR. La periferica risponde ponendo alto DSR.
Purtroppo questo modo di procedere ha un'infinità di variazioni.
Un uso alternativo dei pin RTS e DTR è l'utilizzo come fonte di alimentazione del dispositivo collegato alla porta seriale stessa. L'esempio classico è il mouse seriale ma nulla impedisce di collegare un microcontrollore generico o qualche altro circuito. Unico ed importante limite è la corrente erogata, visto che questi pin non sono pensati per questo uso: è opportuno limitarsi ad un paio di mA anche se molti PC (ma non i portatili) permettono di arrivare tranquillamente a 10mA o anche più. Attenzione che è una sorgente non regolata.



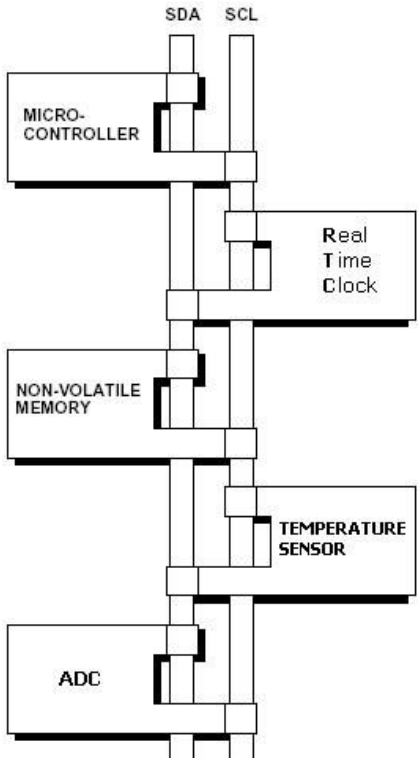


LO STANDARD SERIALE I²C BUS

L'**I²C** è un bus seriale che permette il collegamento a stella di periferiche con solo 2 fili più massa. È consigliato per applicazioni di controllo digitale a 8bit a microcontrollore dove sono presenti memorie ed espansioni I/O.

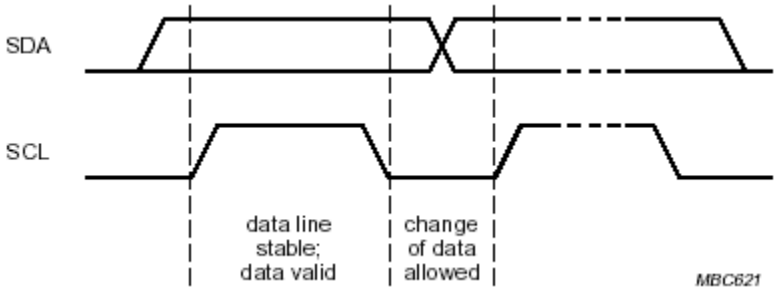
CARATTERISTICHE SALIENTI

- Sono richieste solo due linee: SDA ed SCL
- Ogni dispositivo connesso al bus è indirizzabile via software attraverso un indirizzo univoco
- Il master può operare sia come Tx sia come Rx.
- E' un bus che permette il vero multimaster consentendo la rilevazione di collisione dei dati e l'arbitrato per evitare la perdita di dati se 2 o più masters iniziano contemporaneamente il trasferimento dei dati.
- Trasferimento bi-direzionale seriale a 8bit con velocità di trasmissione fino a 100kbit/s in standard mode, fino a 400kbit/s in Fast mode e fino a 3,4Mbit/s in High speed mode.
- Il numero di periferiche che può essere connesso allo stesso bus è limitato soltanto dal carico capacitivo massimo del bus stesso (400pF).



Glossario	
Termini	**Descrizione**
Trasmettitore	Periferica che spedisce i dati sul bus
Ricevitore	Periferica che riceve i dati dal bus
Master	Periferica che inizia il trasferimento dati, genera il clock e termina il trasferimento.
Slave	Periferica indirizzata dal master
Multi-Master	Più di un master può detenere il controllo del bus
Arbitraggio	Procedura che permette la suddetta situazione, se più di un master tenta di controllare il bus simultaneamente, solo uno ne risulterà abilitato
Sincronizzazione	Procedura per sincronizzare i segnali di clock di due o più periferiche

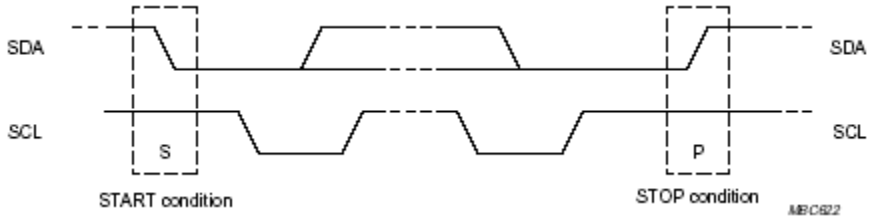
LE LINEE SDA E SCL



Trasferimento di un bit

Sia SDA sia SCL sono linee bi-direzionali mantenute alte tramite una resistenza di pull-up.
Quando il bus è libero entrambe le linee sono Hi.
Gli stadi di uscita dei dispositivi connessi al bus devono avere una configurazione open-drain o open-collector per realizzare la funzione wired-AND.
Un impulso di clock è generato ad ogni bit trasmesso nel bus.
Il dato sulla linea SDA deve essere stabile durante il periodo HI del clock.
Lo stato Hi o Low della linea dati può cambiare soltanto quando il segnale di clock SCL è Low.

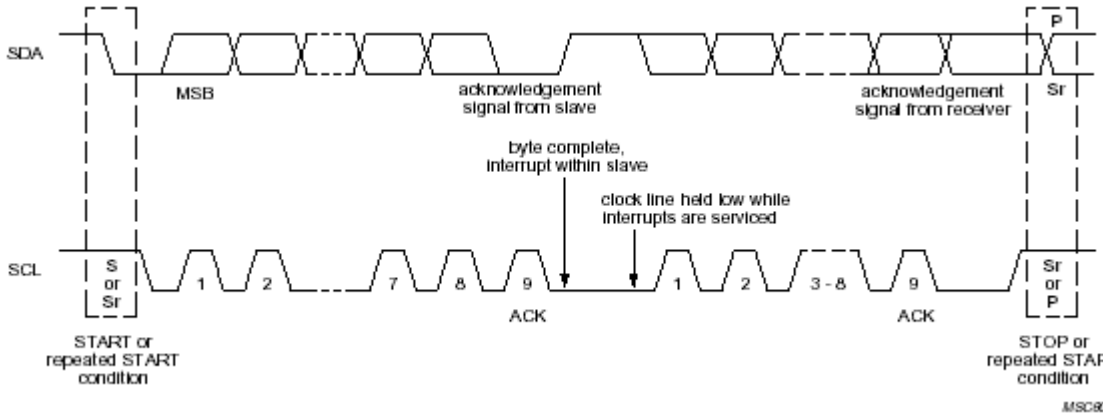
START E STOP



Nelle procedure dell'**I²C** bus esistono situazioni specifiche definite come condizioni di Start (**S**) e di Stop (**P**). Una transizione da Hi a Low su SDA mentre SCL è Hi definisce lo Start; una transizione da Low ad Hi su SDA mentre SCL è Hi definisce la condizione di Stop. Le condizioni di Start e Stop sono sempre generate dal master. Il bus viene considerato occupato (**busy**) dopo la condizione di Start. Il bus è considerato nuovamente libero (**free**) dopo un determinato tempo dalla condizione di stop. Il bus si mantiene busy se un ripetuto start (**SR**) è generato al posto di una condizione di stop.

TRASFERIMENTO DATI

FORMATO DEL BYTE

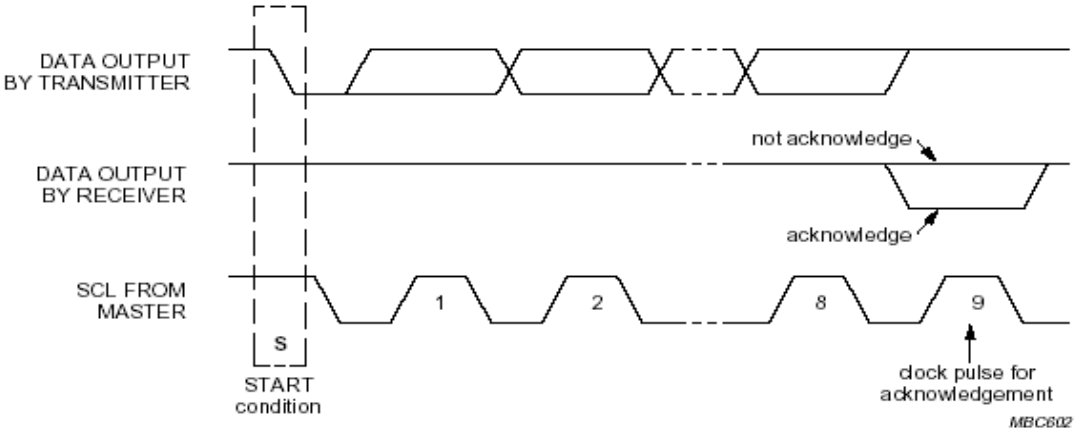


Ogni byte immesso sull'SDA deve essere di 8bit; il numero di byte che può essere trasmesso in ogni trasferimento è illimitato. Ogni byte deve essere seguito da un bit di conferma (ACK). Il dato è trasferito con MSB in testa. Se lo slave non può ricevere o trasmettere un altro byte di dati completo perché non ha terminato qualche altra attività (ad es. la gestione di un interrupt) esso può mantenere la linea di clock SCL

Low per forzare il master allo stato di attesa (**wait state**); il trasferimento dati quindi continuerà quando lo slave, pronto per un altro byte, rilascerà la linea SCL.

ACKNOWLEDGE (ACK)

Nel trasferimento dati è obbligatorio l'ACK. Il relativo impulso di clock è generato dal master. Il trasmettitore rilascia la linea SDA (Hi) durante l'impulso di clock relativo all'ACK e, durante questo impulso di clock il ricevente deve portare basso SDA. Un ricevente è obbligato a generare un ACK dopo ogni byte ricevuto. Se uno slave non riconosce l'indirizzo trasmesso SDA rimane Hi (Not ACK) e il master può allora generare o una condizione di stop per bloccare il trasferimento o uno start ripetuto per iniziare un nuovo trasferimento. Il Not ACK può intervenire anche se il ricevitore slave deve sospendere la ricezione dei dati perché impegnato in altra attività. Lo slave lascia la SDA Hi e il master genera uno stop o una nuova condizione di start. Se il ricevitore master è coinvolto in un trasferimento deve segnalare la fine dati al trasmettitore slave con un Not ACK sull'ultimo byte che era stato spedito dallo slave; il trasmettitore slave



deve abbandonare SDA così da permettere al master di generare stop o nuovo start.

SINCRONIZZAZIONE

Ciascun master genera il proprio clock per trasferire dati sul Bus. Il dato è valido soltanto nel periodo Hi del clock; è per tanto necessario un clock ben definito per dar luogo alla procedura di arbitrato *bit by bit*.

ARBITRATO

Un Master può iniziare un trasferimento solo se il bus è libero. Se due o più master generano una condizione di start contemporaneamente, sulla linea SDA ha luogo l'arbitrato mentre la linea SCL è Hi: il master che trasmette un livello Hi mentre un altro master sta trasmettendo un livello Low spegnerà il suo stadio d'uscita dati perché il livello sul bus non corrisponde a quello da lui previsto. L'arbitrato può continuare per molti bit. Il primo stadio è la comparazione dei bit di indirizzo: se i master stanno ciascuno cercando di indirizzare lo stesso dispositivo l'arbitrato continua con la comparazione dei bit dati se essi sono dei master trasmittenti o dei bit ACK se sono dei master riceventi. Poiché gli indirizzi e i dati sul bus sono determinati dal master vincitore nessuna informazione viene perduta durante il processo di arbitrato. Un master che perde l'arbitrato può generare impulsi di clock fino alla fine del byte nel quale ha perso l'arbitrato. Se il master incorpora anche una funzione slave e perde l'arbitrato mentre veniva indirizzato dal master vincitore esso terminerà sempre l'arbitrato durante il primo byte cioè durante la fare di indirizzamento in quanto anche un master ha un unico codice a **8bit**. Il master perdente deve perciò commutare immediatamente al modo slave. Ciò non modificherà il data tranfer iniziato dal master vincitore. Dal momento che il controllo del bus **I²C** è assunto dal vincitore dei master in competizione, non c'è alcun master centrale e non sussiste un ordine di priorità sul bus.
Gli slave non sono coinvolti nell'arbitrato.

SSPSTAT: registro SYNC SERIAL PORT STATUS (indirizzo94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in slave mode
In I^2C Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high speed mode (400 kHz)

bit 6 **CKE**: SPI Clock Edge Select (Figure 9-2, Figure 9-3 and Figure 9-4)
SPI mode:
For CKP = 0
1 = Data transmitted on rising edge of SCK
0 = Data transmitted on falling edge of SCK
For CKP = 1
1 = Data transmitted on falling edge of SCK
0 = Data transmitted on rising edge of SCK
In I^2C Master or Slave mode:
1 = Input levels conform to SMBus spec
0 = Input levels conform to I^2C specs

bit 5 **D/A**: Data/$\overline{\text{Address}}$ bit (I^2C mode only)
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P**: STOP bit
(I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.)
1 = Indicates that a STOP bit has been detected last (this bit is '0' on RESET)
0 = STOP bit was not detected last

bit 3 **S**: START bit
(I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.)
1 = Indicates that a START bit has been detected last (this bit is '0' on RESET)
0 = START bit was not detected last

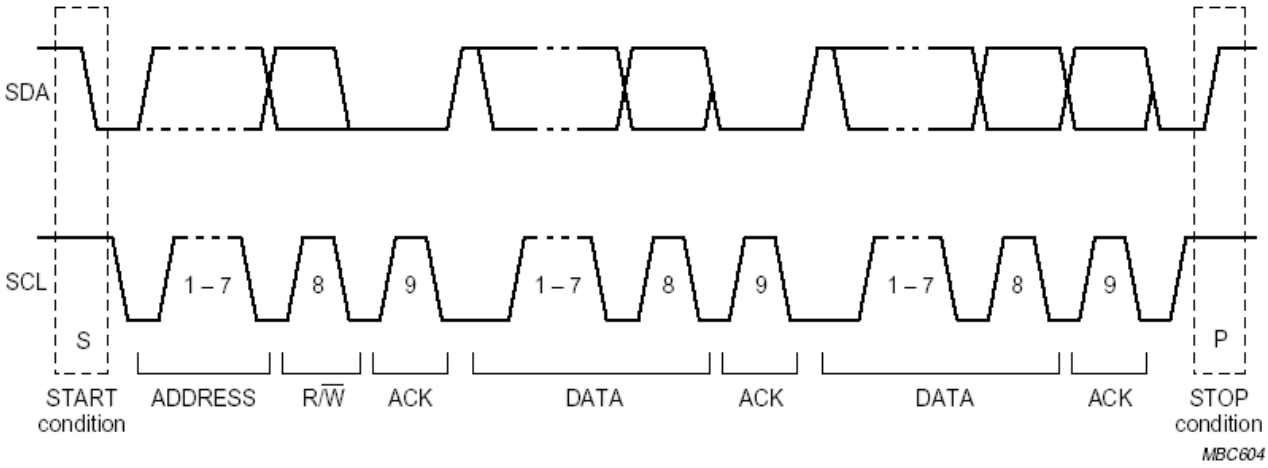
bit 2 **R/W**: Read/Write bit Information (I^2C mode only)
This bit holds the R/W bit information following the last address match. This bit is only valid from the address match to the next START bit, STOP bit or not $\overline{\text{ACK}}$ bit.
In I^2C Slave mode:
1 = Read
0 = Write
In I^2C Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress
Logical OR of this bit with SEN, RSEN, PEN, RCEN, or ACKEN will indicate if the MSSP is in IDLE mode.

bit 1 **UA**: Update Address (10-bit I^2C mode only)
1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit **BF**: Buffer Full Status bit
Receive (SPI and I^2C modes):
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
Transmit (I^2C mode only):
1 = Data transmit in progress (does not include the ACK and STOP bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{\text{ACK}}$ and STOP bits), SSPBUF is empty

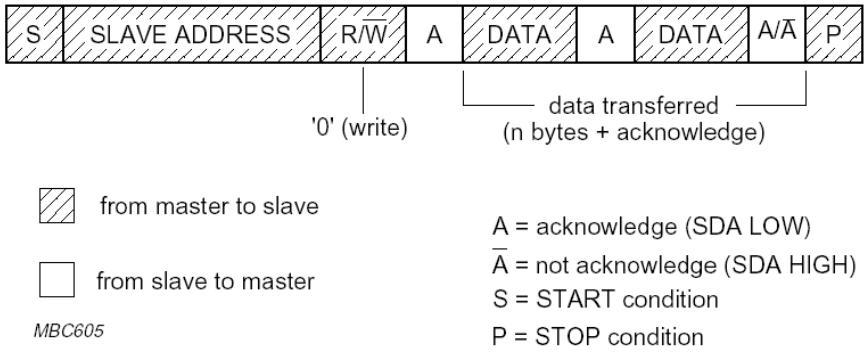
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

IL FORMATO INDIRIZZI A 7BIT

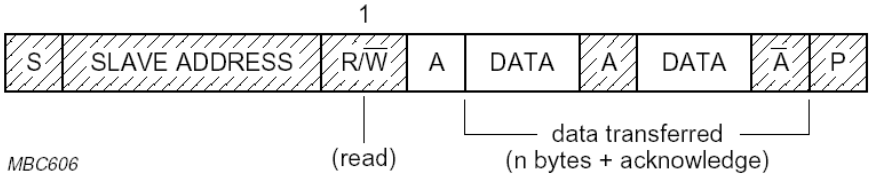


Una trasmissione dati avviene secondo il formato mostrato nella figura qui sopra: dopo la condizione di start (S) è spedito l'indirizzo dello slave. Quest'ultimo è di 7bit seguito da un ottavo indicante la direzione dei dati (R/W): se vale zero indica una trasmissione (write); se vale uno indica una richiesta di dati (read). Ogni trasferimento è terminato sempre da uno Stop generato dal master a meno che il master non intenda continuare a comunicare sul bus con altri slave: genera una condizione di start ripetuto (SR) e indirizza un altro slave senza generare la condizione si Stop.

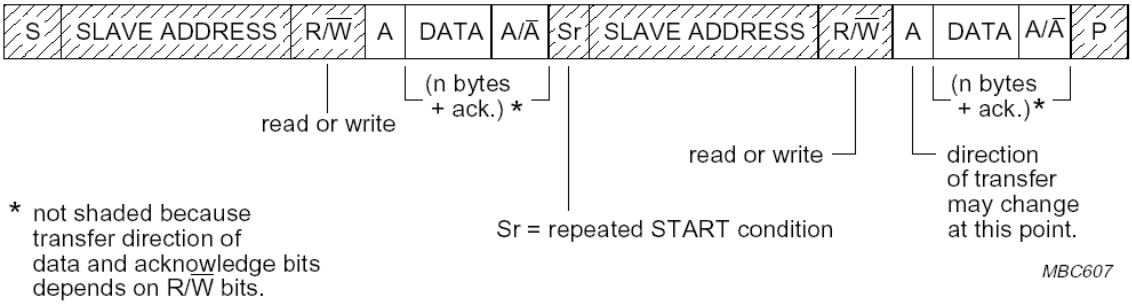
Il master invia dati allo slave: lo indirizza con 7 bit ponendo R/W=0



Il master legge dallo slave: lo indirizza con 7 bit ponendo R/W=1



Il master non termina la comunicazione, ma tiene occupato il bus ripetendo SR



Quando viene spedito un indirizzo, ogni slave lo compara con il proprio; se coincide la periferica viene considerata slave-receiver o slave-transmitter (dipende da R/W) e risponde con una procedura di ACK.
Un indirizzo slave può essere composto da una parte fissa e una programmabile dal progettista tramite i relativi pin previsti sull'integrato slave; ciò determina il numero massimo di periferiche identiche collegabili al bus: ad esempio le memorie seriali EEPROM hanno i primi 4 bit fissi (1010) mentre gli altri 3 bit sono programmabili; questo determina il numero massimo (8) di EEPROM collegabili sullo stesso bus.
Per le periferiche ad alta velocità e per l'indirizzamento a 10bit le procedure di colloquio sono poco differenti dallo standard descritto, per approfondire questo standard seriale riferirsi alla documentazione:
THE I²C BUS SPECIFICATION VERSION 2.1 JANUARY 2000 di PHILIPS SEMICONDUCTORS.

I²C BUS E IL PIC16F87X

Il PIC16F87x è un microcontrollore con l'interfaccia on-chip che può essere programmato per funzionare come dispositivo master o slave. Può essere interrotto nell'elaborazione delle sue routine interne da un interrupt generato dalla richiesta di una periferica collegata al bus.

MODI DI FUNZIONAMENTO DELL'INTERFACCIA I²C BUS INTEGRATA

- Slave mode a 7 bit di indirizzamento
- Slave mode a 10 bit di indirizzamento
- Master mode, clock = OSC/4
- Firmware modes (implementato per la compatibilità con altri periferiche in commercio)
- Velocità a 100kbit/s
- Velocità a 400kbit/s

REGISTRI INTERESSATI

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE	PEIE	T0IE	INTE	RBIE(2)	T0IF	INTF	RBIF(2)	0000 000x	0000 000u
PIR	SSPIF (1)								0	0
PIE	SSPIE (1)								0	0
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPADD	Synchronous Serial Port (I²C mode) Address Register								0000 0000	0000 0000
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
SSPSTAT	SMP	CKE	D/Ā	P	S	R/W	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, – = unimplemented read as '0'.
Shaded cells are not used by SSP in I2C mode.
Note 1: The positions of these bits are device dependent.
2: These bits may also be named GPIE and GPIF.

registro SSPCON: SYNC SERIAL PORT CONTROL (indirizzo 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL**: Write Collision Detect bit
Master mode:
1 = A write to SSPBUF was attempted while the I2C conditions were not valid
0 = No collision
Slave mode:
1 = SSPBUF register is written while still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV**: Receive Overflow Indicator bit
In SPI mode:
1 = A new byte is received while SSPBUF holds previous data. Data in SSPSR is lost on overflow. In Slave mode, the user must read the SSPBUF, even if only transmitting data, to avoid overflows. In Master mode, the overflow bit is not set, since each operation is initiated by writing to the SSPBUF register. (Must be cleared in software.)
0 = No overflow
In I^2C mode:
1 = A byte is received while the SSPBUF is holding the previous byte. SSPOV is a "don't care" in Transmit mode. (Must be cleared in software.)
0 = No overflow

bit 5 **SSPEN**: Synchronous Serial Port Enable bit
In SPI mode,
When enabled, these pins must be properly configured as input or output
1 = Enables serial port and configures SCK, SDO, SDI, and SS as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins
In I^2C mode,
When enabled, these pins must be properly configured as input or output
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP**: Clock Polarity Select bit
In SPI mode:
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level
In I^2C Slave mode:
SCK release control
1 = Enable clock
0 = Holds clock low (clock stretch). (Used to ensure data setup time.)
In I^2C Master mode:
Unused in this mode

bit 3-0 **SSPM3:SSPM0**: Synchronous Serial Port Mode Select bits
0000 = SPI Master mode, clock = Fosc/4
0001 = SPI Master mode, clock = Fosc/16
0010 = SPI Master mode, clock = Fosc/64
0011 = SPI Master mode, clock = TMR2 output/2
0100 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0101 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin.
0110 = I^2C Slave mode, 7-bit address
0111 = I^2C Slave mode, 10-bit address
1000 = I^2C Master mode, clock = Fosc / (4 * (SSPADD+1))
1011 = I^2C Firmware Controlled Master mode (slave idle)
1110 = I^2C Firmware Controlled Master mode, 7-bit address with START and STOP bit interrupts enabled
1111 = I^2C Firmware Controlled Master mode, 10-bit address with START and STOP bit interrupts enabled
1001, 1010, 1100, 1101 = Reserved

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro OPTION_REG (indirizzo 81h, 181h)

E' un registro (di lettura /scrittura) che contiene vari bit di controllo per configurare il prescaler del modulo Timer0/WatchDog, l'interrupt esterno INT, l'interrupt TMR0 e i resistori di pull-up del port B.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro PIR1 (indirizzo 0Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro PIE1 (indirizzo 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro PIE2 (indirizzo 8Dh)

REGISTER 2-6: PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

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

bit 6 **Reserved**: Always maintain this bit clear

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIE**: EEPROM Write Operation Interrupt Enable
1 = Enable EE Write Interrupt
0 = Disable EE Write Interrupt

bit 3 **BCLIE**: Bus Collision Interrupt Enable
1 = Enable Bus Collision Interrupt
0 = Disable Bus Collision Interrupt

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

registro SSPCON2: SYNC SERIAL PORT CONTROL 2 (indirizzo 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN**: General Call Enable bit (In I^2C Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT**: Acknowledge Status bit (In I^2C Master mode only)
In Master Transmit mode:
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT**: Acknowledge Data bit (In I^2C Master mode only)
In Master Receive mode:
Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
1 = Not Acknowledge
0 = Acknowledge

bit 4 **ACKEN**: Acknowledge Sequence Enable bit (In I^2C Master mode only)
In Master Receive mode:
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence idle

bit 3 **RCEN**: Receive Enable bit (In I^2C Master mode only)
1 = Enables Receive mode for I^2C
0 = Receive idle

bit 2 **PEN**: STOP Condition Enable bit (In I^2C Master mode only)
SCK Release Control:
1 = Initiate STOP condition on SDA and SCL pins. Automatically cleared by hardware.
0 = STOP condition idle

bit 1 **RSEN**: Repeated START Condition Enable bit (In I^2C Master mode only)
1 = Initiate Repeated START condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated START condition idle

bit 0 **SEN**: START Condition Enable bit (In I^2C Master mode only)
1 = Initiate START condition on SDA and SCL pins. Automatically cleared by hardware.
0 = START condition idle

Note: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the IDLE mode, this bit may not be set (no spooling), and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

CODICE di ESEMPIO per la GESTIONE DELL'INTERFACCIA I²C

```
        :
RTCaddr     EQU         029H            ;indirizzo dell'RTC
#define     FOSC        D'4000000'      ; define FOSC to PICmicro
#define     I2CClock    D'100000'       ; define I2C bite rate
#define     ClockValue  (((FOSC/I2CClock)/4) -1) ;
        :
        :
        banksel SSPADD      ; BANCO1
        movlw   ClockValue  ; read selected baud rate
        movwf   SSPADD      ; initialize I2C baud rate
        bcf     SSPSTAT,6   ; input levels conform to the I2C spec.
        bcf     SSPSTAT,7   ; enable slew rate?????????????
        movlw   b'00011000' ; RC3 ed RC4 sono le linee dell'I2C
        movwf   TRISC       ; ensure SDA and SCL are inputs
        BCF     STATUS,RP0
        BCF     STATUS,RP1  ;BANCO 0
        movlw   b'00111000' ; WCOL-SSPOV-SSPEN-CKP-M3-M2-M1-M0
        movwf   SSPCON      ; Master mode, SSP enable
        movlw   B'01101000' ; B'01101000'
        movwf   RTCaddr
        :
        :
;.........ROUTINES.....................................................................
;Invio in I2C il Byte contenuto in W
WRT_W
        movwf   SSPBUF      ;initiate I2C bus write condition
        CALL    WaitIF      ;'interrupt è generato al 9° impulso di clock
        banksel SSPCON2     ; select SFR bank
        BTFSC   SSPCON2,ACKSTAT
        BCF     STATUS,RP0
        BCF     STATUS,RP1  ;BANCO 0
        RETURN
;.....................................................................................
;Attesa del flag dell'interrupt dell'I2C
WaitIF
        BCF     PIR1,SSPIF
Wait01
        BTFSS   PIR1,SSPIF  ;attendo l'interrupt
        GOTO    Wait01
        RETURN
;.....................................................................................
RTCWrite
        banksel SSPCON2     ; select SFR bank
        bsf     SSPCON2,SEN ; initiate I2C bus start condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1  ;BANCO 0
        CALL    WaitIF      ;attendo la fine dello start
; Generate I2C address per scrittura (R/W=0)
SendWriteAddr
        bcf     STATUS,C    ; ensure carry bit is clear
        rlf     RTCaddr,W   ; compose 7 bit address
        call    WRT_W       ; invio address
```

```
        movlw  00H
        call    WRT_W          ; indirizzo 00H sul quale avverrà successiva scrittura
        movlw  07H
        movwf  ReadCnt
        movlw  FIRST_BYTE
        movwf  FSR
WriteDT
        movf    INDF,W
        call    WRT_W
        banksel ReadCnt
        INCF    FSR,F          ; Inc pointer
        decfsz  ReadCnt,f      ;VERIFICO SE HO INVIATO TUTTI I BYTE
        goto    WriteDT
        banksel SSPCON2        ; select SFR bank
        bsf    SSPCON2,PEN     ; initiate I2C bus stop condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1     ;BANCO 0
        CALL    WaitIF
        return

;*******************************************************************************
;Lettura
RTC_Read
        ;Occorre un indirizzamento in scrittura sul primo registro (00H)
        ;affinché si posizioni il puntatore per la successiva lettura
        banksel SSPCON2                ; select SFR bank
        bsf    SSPCON2,SEN     ; initiate I2C bus start condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1     ;BANCO 0
        CALL    WaitIF         ;attendo la fine dello start
; Generate I2C address per scrittura (R/W=0)
        bcf     STATUS,C       ; ensure carry bit is clear
        rlf     RTCaddr,W      ; compose 7 bit address
        call    WRT_W
        movlw  00H
        call    WRT_W                  ; indirizzo 00H sul quale avverrà successiva lettura
        banksel SSPCON2                ; select SFR bank
        bsf      SSPCON2,PEN           ; initiate I2C bus stop condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1     ;BANCO 0
        CALL    WaitIF         ; attendo la fine dello stop
;Inizio un nuovo ciclo di start
        banksel SSPCON2                ; select SFR bank
        bsf     SSPCON2,SEN; initiate I2C bus start condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1     ;BANCO 0
        CALL    WaitIF         ; attendo la fine dello start
        ; Generate I2C address per lettura (R/W=1)
SendReadAddr
        banksel RTCaddr                ;
        bsf     STATUS,C       ; ensure carry bit is set
        rlf     RTCaddr,W      ; compose 7 bit address
        Call    WRT_W
        banksel ReadCnt
```

```
        movlw  07H
        movwf  ReadCnt
        movlw  FIRST_BYTE
        movwf  FSR
ReadDT
        banksel SSPCON2
        bsf     SSPCON2,RCEN        ; generate receive condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1  ;BANCO 0
        CALL    WaitIF              ; l'interrupt è generato dal buffer pieno
        movf    SSPBUF,W     ; save of byte into W
        MOVWF       INDF            ;... e lo salvo in memoria
        INCF    FSR,F               ; Inc pointer
        decfsz  ReadCnt,f           ;VERIFICO SE HO MEMORIZZATO TUTTI I BYTE
        goto    Acknow
        goto    NotAck
Acknow
        banksel SSPCON2             ; select SFR bank
        bcf     SSPCON2,ACKDT       ; acknowledge bit state to send
        bsf     SSPCON2,ACKEN       ; initiate acknowledge sequence
WaitACK
        banksel SSPCON2             ; select SFR bank
        btfsc   SSPCON2,ACKEN       ; ack cycle complete?
        goto    WaitACK             ; no, so loop again
        goto    ReadDT
NotAck                  ; Send Not Acknowledge
        banksel SSPCON2             ; select SFR bank
        bsf   SSPCON2,ACKDT         ; acknowledge bit state to send (not ack)
        bsf   SSPCON2,ACKEN         ; initiate acknowledge sequence
WtNACK      btfsc   SSPCON2,ACKEN       ; ack cycle complete?
        goto    WtNACK              ; no, so loop again
        banksel SSPCON2             ; select SFR bank
        bsf   SSPCON2,PEN   ; initiate I2C bus stop condition
        BCF     STATUS,RP0
        BCF     STATUS,RP1  ;BANCO 0
        CALL    WaitIF              ; attendo che termini la stop condition
        return
;.....................................................................
```