NOZIONI BASE
DI
ASSEMBLER PER PIC16C84

Aggiornato al 09 settembre 2005

Ermes ZANNONI
(ermes@zannoni.to.it)
(http://www.zannoni.to.it)

Indice :

1. Introduzione

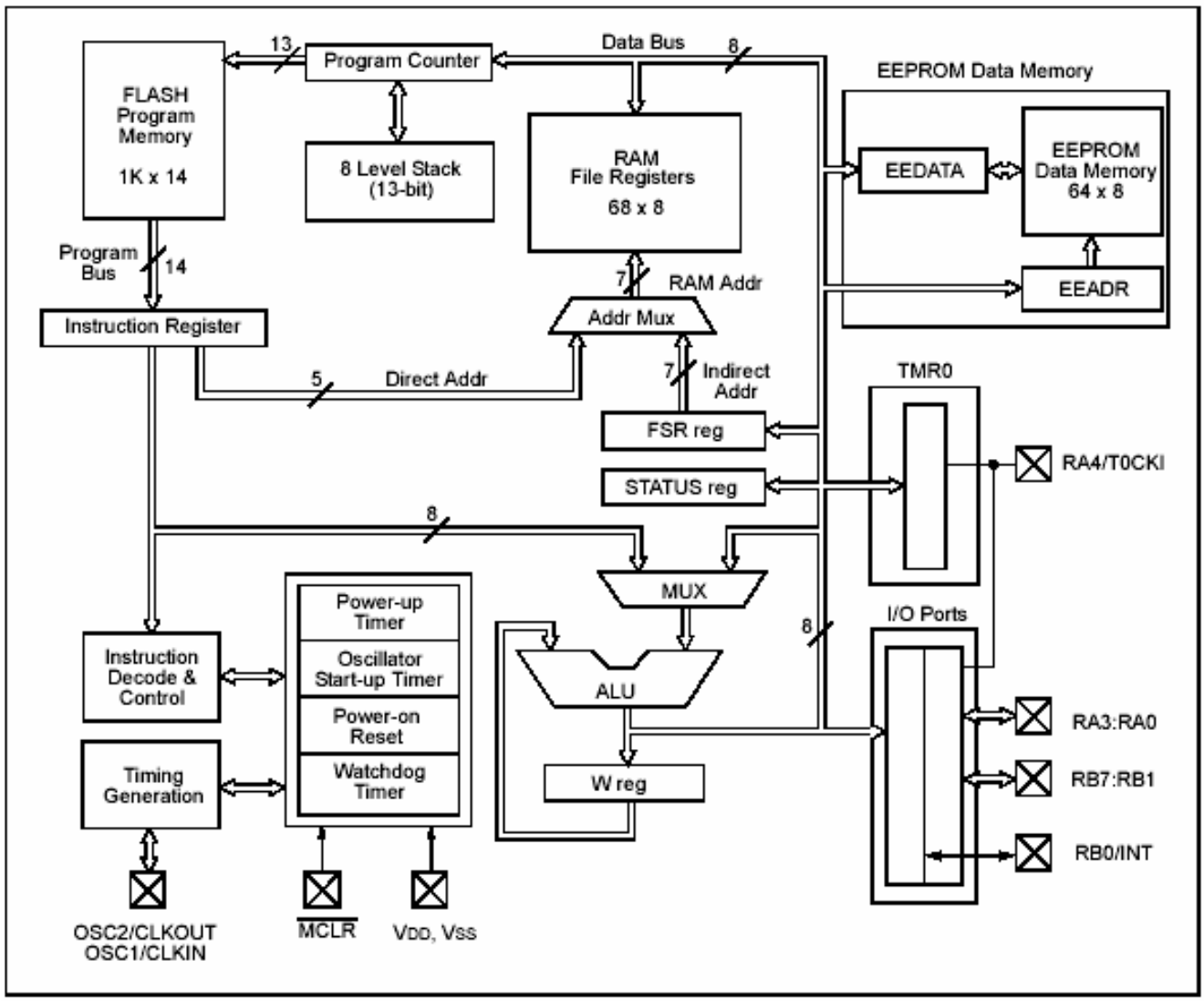
Dispensa introduttiva al linguaggio Assembler per PIC16C84. Per esercitarvi vi consiglio di usare i seguenti programmi: MPLAB IDE (per la compilazione), ICPROG (caricare il programma nel microcontrollore) e il PIC SIMULATOR IDE (simulazione), l'ultimo è a pagamento.

Questa dispensa è di distribuzione gratuita per cui non può essere messa in vendita.

2. Cosa è un PIC

Sono dei circuiti integrati che fan parte dei microcontrollori costruiti dalla Microchip Technology Inc.
Al suo interno troviamo una CPU, una memoria EEPROM (memoria a sola lettura), una memoria RAM (memoria ad accesso casuale), delle linee I/O e dei dispositivi ausiliari (es. clock).
Con questo manuale non scendo nel dettaglio di come è fatto all'interno un PIC perché ci sarebbero da scrivere centinaia di pagine, per cui per ulteriori approfondimenti vi mando al data sheet della Microchip scaricabile il formato pdf da internet.

3. Architettura interna



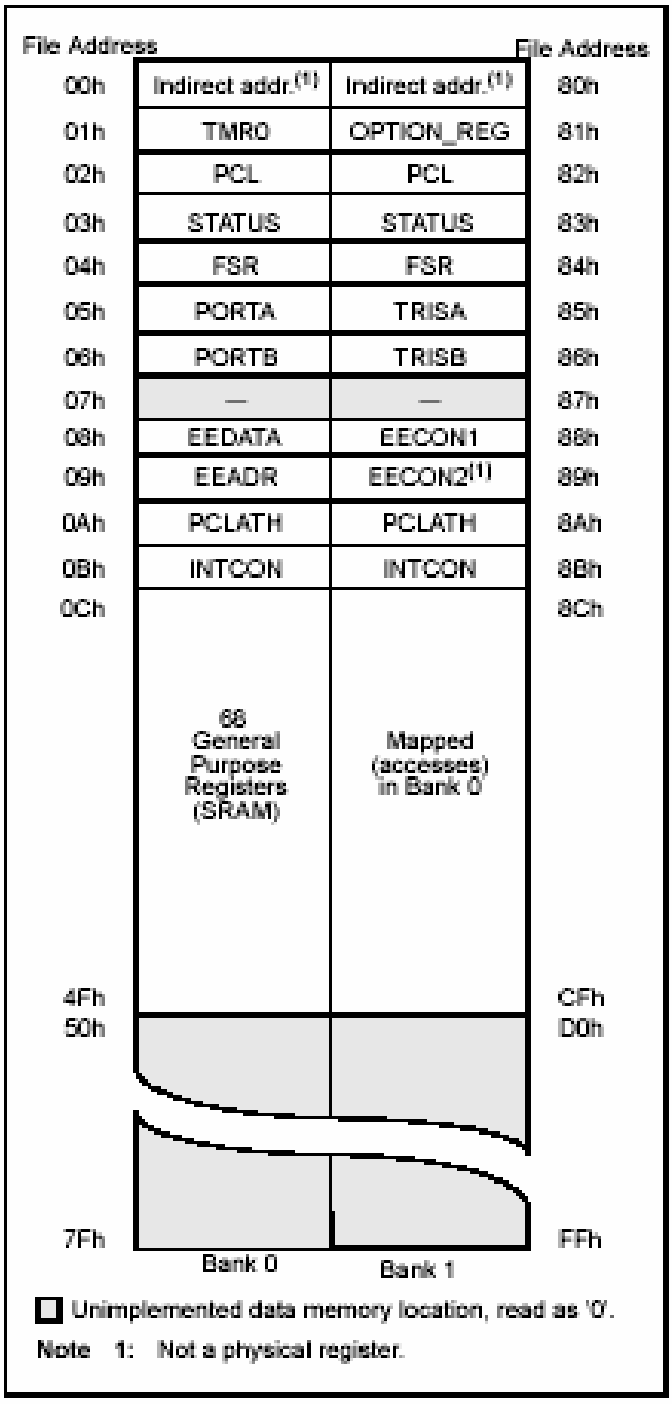
Schema a blocchi estratto dal data sheet della Microchip

EEPROM Area di programma

Questa è una memoria non cancellabile non volatile, per cui non cancellabile elettricamente, in questa area viene caricato il nostro programma.
La sua capienza è di 1024 locazioni ognuna può contenere un'istruzione, per cui possiamo scrivere programmi non superiori alle 1024 istruzioni. Gli indirizzi vanno dalla locazione 0000H alla 03FFH.

Register file

E' una memoria RAM, per cui possiamo scrivere, leggere e modificare ogni locazione.
Questa memoria e divisa in due banchi, il banco zero (0) che va da 00H a 2FH e il banco uno (1) da 80H a 8BH.
Tra queste abbiamo le locazioni speciali che sono nel banco 0 da 00H a 0BH e nel banco 1 da 80H a 8BH.
Le locazioni da 0CH e 2FH possono essere usate per memorizzare delle variabili.



ALU (Arithmetric and Logic Unit)

Come dice la parola questa ha la funzione di calcolo e manipolazione dei dati. Ha la possibilità di operare con valori di massimo 8 bit, per cui valori non superiori a 255.

Registro W

Questo è un accumulatore in grado di contenere un solo valore di 8 bit.

Program Counter e lo stack

Queste vengono utilizzate dalle istruzioni di salto (GOTO) e di chiamata a una subroutine (CALL).
La funzione di program counter è quella di mantenere la traccia dell'indirizzo che contiene la prossima istruzione da eseguire.
Lo Stack serve a memorizzare l'istruzione successiva di un salto eseguito nel programma, come se questo salto non ci fosse, per renderlo più chiaro faccio un esempio:

```
            istruzione1
            istruzione2
            call SALTO
            istruzione3
            …

SALTO
            Istruzione4
            …
```

in questo caso nello stack memorizza l'istruzione istruzione3.

Linee I/O

In questo microcontrollore abbiamo a disposizione 2 porte (A e B). La porta A ha 5 linee (RA0, RA1, RA2, RA3 e RA4) utilizzabili sia in ingresso che uscita e la porta B ha 8 linee (RB0, RB1, RB2, RB3, RB4, RB5, RB6 e RB7) anche queste utilizzabili sia in ingresso che uscita.
Per comandare queste 2 porte abbiamo due registri per ogni porta, TRISA e TRISB, PORTA e PORTB.
I registri nel banco 1 TRISA e TRISB determinano ogni singola linea se è in ingresso o uscita e i registri nel banco 0 PORTA e PORTB determinano o riportano lo stato logico delle linee.

Esempio:
Se mettiamo a zero il bit 0 del registro TRISA, abbiamo che la linea RA0 sarà un'uscita, viceversa se lo mettiamo a 1 avremmo un ingresso. Lo stato logico della linea di uscita potrà essere 1 (5 Volt) o 0 (0 Volt).

Stadio d'uscita della linea RA0, RA1, RA2 e RA3

Faccio un breve esempio prendendo come riferito la linea RA0, le successive linee si comportano nello stesso modo tranne la RA4 che vedremo in seguito il motivo.

Funzionamento in uscita delle linee RA0, RA1, RA2 e RA3

- Configurazione della linea RA0 in uscita, per cui impostiamo il bit 0 del registro TRISA a zero;

```
bcf       TRISA,0
```

- Ora possiamo impostare la linea a zero, 0 volt;

```
bcf       PORTA,0
```

o a 1, 5 volt;

```
bsf       PORTA,0
```

Funzionamento in ingresso

- Configurazione della linea RA0 in ingresso, per cui impostiamo il bit 0 del registro TRISA a uno;

```
bsf       TRISA,0
```

- Applicando una tensione in ingresso 0 o 5 volt nella linea RA0, possiamo leggere il suo stato;

```
btfss     PORTA,0        ; se vale 1 esegue l'istruzione1
Istruzione1
btfsc     PORTA,0        ; se vale 0 esegue l'istruzione2
Istruzione2
```

Stadio d'uscita della linea RA4

In questo caso abbiamo lo stesso pin condiviso alla porta I/O e il TOCK1, per cui abbiamo un circuito d'uscita con collettore aperto.
Ciò comporta utilizzando questa uscita mettendola a 1 (5 volt) nella realtà questa impostazione non avviene, per eliminare questo problema dobbiamo collegare esternamente una resistenza di pull-up al positivo dell'alimentazione (5 volt).

Stadio d'uscita delle linee RB0, RB1, RB2 e RB3

Lo stato degli ingressi dipende esclusivamente dalla circuiteria esterna, ad esempio se abbiamo un pulsante collegato all'ingresso, conviene che quando viene premuto la linea venga collegata a massa e quando viene rilasciato ci sia uno stato logico 1 costante (5 volt), per far ciò utilizzeremo una resistenza di pull-up verso il positivo.
Se vogliamo evitare l'utilizzo di resistenze esterne di pull-up abbiamo la possibilità di utilizzare la circuiteria interna di weak pull-up abilitando o disabilitando il bit RBPU del registro OPTION.
Inoltre la linea RB0 configurata come linea di ingresso, può generare un interrupt, un'interruzione del programma in esecuzione ed una chiamata ad una subroutine (chiamata interrupt handler).

Stadio d'uscita delle linee RB4, RB5, RB6 e RB7

Queste linee sono uguali alle precedenti ma l'unica differenza e nel avere uno stadio in grado di rilevare variazioni di stato, generando un interrupt.

TMR0 Registro contatore e il prescaler

Questo registro è un contatore, questo valore viene incrementato con cadenza regolare, ad esempio se impostiamo il registro TMR0 a 3:

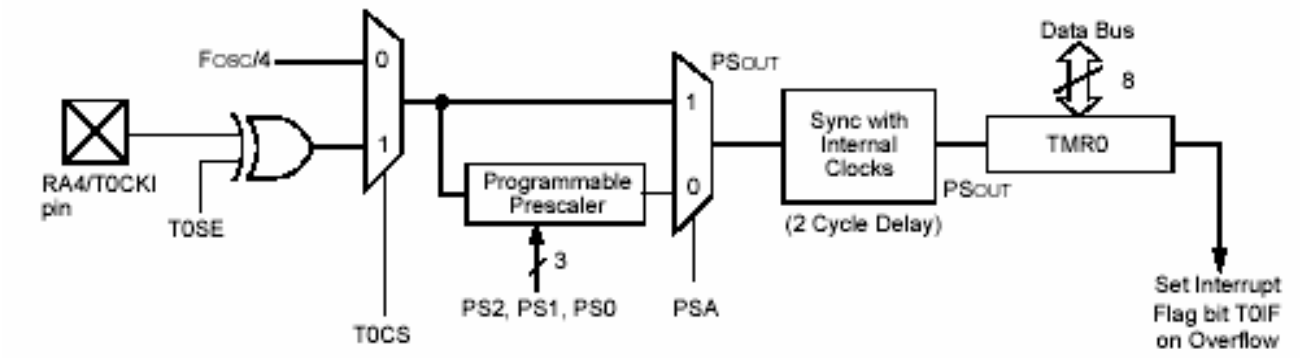
```
movlw       3
movwf       TMR0
```

ogni 4 ciclo di macchina incrementa il valore del registro, 4, 5, 6, … una volta arrivato a 255 il registro viene azzerato.

Le sorgenti di segnale per il contatore possono essere 2:

- Fosc/4
- T0CK1

Fosc/4 è un segnale generato internamente ed è pari alla frequenza di clock divisa per 4.
T0CK1 è un segnale generato da un circuito esterno ed applicato al pin 3 del microcontrollore.



Schema a blocchi estratto dal data sheet della Microchip

T0CS e PSA sono due commutatori di segnale comandati dai bit T0CS e PSA del registro OPTION.

La porta XOR all'uscita del TOCK1 serve a configurare il metodo di incrementazione del contatore TMR0 comandato dal bit TOSE se in fronte di salita (TOSE=0) o discesa (TOSE=1).

Il PRESCALER è un divisore programmabile a 8 bit utilizzabile in caso la frequenza di conteggio del contatore TMR0 sia troppo elevata.

Utilizzando un quarzo da 4 MHz la frequenza del contatore TMR0 è pari a 1 MHz.

I bit che possiamo configurare sono PS0, PS1 e PS2 del registro OPTION. Configurando uno o più bit PS0, PS1 e PS2 possiamo avere le seguenti frequenze:

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

Freq. ingresso / Divisore = Freq. Uscita

Esempio:

1 MHz / 256 = 3.906 Hz

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h	TMR0	Timer0 Module Register							
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
85h	TRISA	—	—	—	PORTA Data Direction Register				

Interrupt

Permette di interrompere il programma in esecuzione tramite un evento esterno.

Abilitando (mettendo a 1) uno dei seguenti bit nel registro INTCON il microcontrollore può gestire quattro eventi:

- cambiando di stato la linea RB0 (INTE bit 4);
- cambiando una delle linee da RB4 ad RB7 (RBIE bit 3);
- fine del conteggio del registro TMR0 (T0IE bit 5);
- fine scrittura su una locazione EEPROM (EEIE bit 6).

Oltre a questi bit da abilitare c'è ne uno generale chiamato GIE bit 7 del registro INTCON.

Al verificarsi un dei eventi configurati nel registro INTCON, interrompe immediatamente il programma in esecuzione , memorizza automaticamente nello stack il valore del program counter e salta all'istruzione situata nel locazione di memoria 0004H (Interrupt vector).
Da questa locazione di memoria dobbiamo scrivere la subroutine di come deve agire al verificarsi dell'evento (Interrupt Handler).

Nel caso attivassimo più di un interrupt, per verificare quale a causato l'evento abbiamo le seguenti configurazioni da abilitare nel registro INTCON:

- INTF bit 1 l'evento è stato causato dal cambio di stato della linea RB0;
- RBIF bit 0 l'evento è stato causato dal cambio di stato di una delle linee RB4-5-6-7;
- T0IF bit 2 l'evento è stato causato dal termine del conteggio del contatore TMR0.

Per ritornare da un interrupt handler bisogna utilizzare l'istruzione RETFIE, questo comando riabilita il bit GIE del registro INTCON, perché ad ogni evento che causa un interrupt automaticamente disabilita questo bit per dare la possibilità di terminare senza problemi la subroutine.

Sleep mode

Questo stato viene utilizzato per economizzare i consumi passa da un consumo di 2mA a circa 2uA, il microcontrollore è in attesa di un interrupt. Fin quando non succede un evento il PIC non esegue alcuna istruzione.
Per ridurre ulteriormente i consumi si consiglia di collegare tutte le linee non utilizzate al positivo o al negativo dell'alimentazione.

Per il risveglio del microcontrollore si può:

- Resettare tramite il pin 4 (MCRL) mettendolo a 0;
- Timeout del timer del Watchdog (abilitandolo);
- Causando un interrupt;

Watchdog

Questa funzione è un timer che controlla se abilitato il blocco del programma in esecuzione, in tal caso resetta e parte dalla prima locazione del programma.

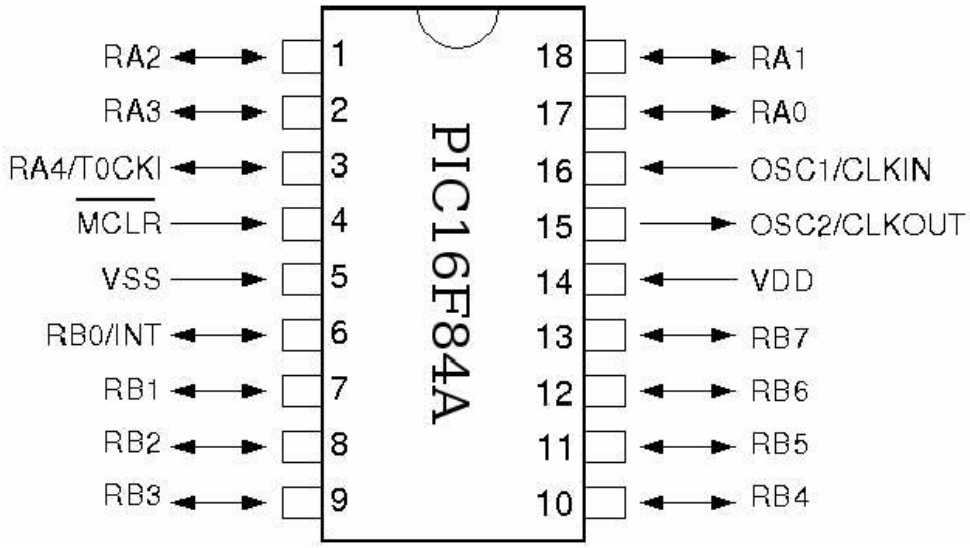
Per evitare un reset involontario bisogna inserire nel nostro programma l'istruzione CLRWDT, questa resetta il contatore watchdog.

Per avere questa funzione bisogna abilitarla in fase di programmazione il flag WDTE della word di configurazione, questo dipende dal programmatore a voi utilizzato.

Agendo sul prescaler visto precedentemente (PSA,PS0,PS1 e PS2 del registro OPTION_REG) possiamo variare il tempo per il reset del PIC da circa 18 ms a 2 s.

4. Approfondimento sul PIC16C84

E' un microcontrollore a 8 bit dotato di 2 porte I/O digitali con una frequenza massima di 20 MHz.



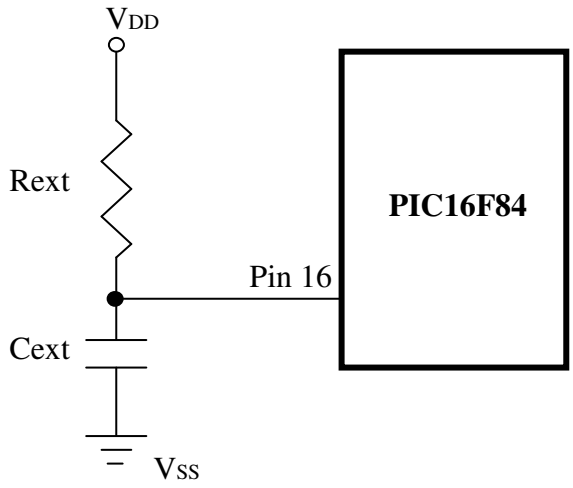
Descrizione dei piedini:

- da RA0 al RA4 (5 pin) identificano la porta di comunicazione A;
- da RB0 a RB7 (8 pin) identificano la porta di comunicazione B;
- pin 15 e 16 l'oscillatore (clock);
- pin 14 ed 5 (massa) alimentazione da 2,2 a 5 V dc;
- pin 4 reset del sistema attivo se messo a massa, ciò significa che in normali condizioni deve essere tenuto tramite una resistenza da 10kΩ al positivo dell'alimentazione (da 2,2 a 5 V dc)

Possono essere utilizzati 4 tipi di clock: RC, LP, HS e XT:

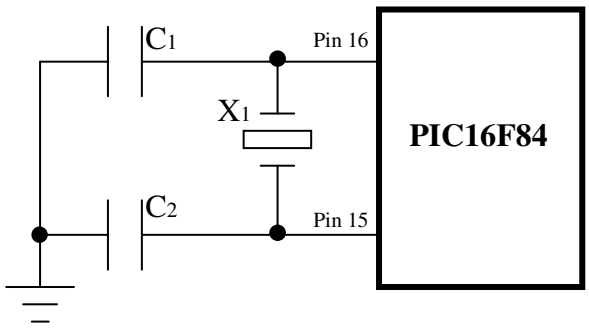
- Configurazione **RC**

  Composta da una resistenza e da un condensatore, come vedete in figura va collegata nel pin 16, volendo per ulteriori utilizzi, possiamo prelevare dal pin 15 la frequenza divisa per 4 (clock/4). La frequenza, utilizzando il clock RC è definita da diversi fattori: valore Rext (5kΩ≤Rext≤100kΩ) e Cext (Cext>20pF), tensione di alimentazione e dalla temperatura.



- Configurazione con cristallo esterno LP, HS e XT

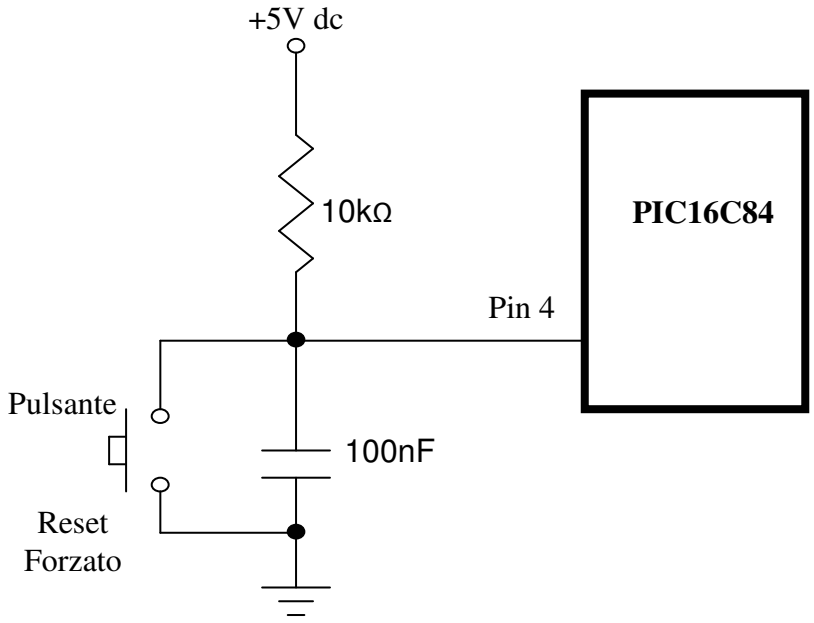
  Composta da un quarzo e due condensatori di sfasamento, risulta essere la configurazione più diffusa se si ha bisogno di un clock estremamente preciso e sincronizzato.



L'assorbimento del microchip è proporzionale alla frequenza di lavoro.

MODO	FREQUENZA	C1	C2
LP	32 kHz	68-100 pF	68-100 pF
	200 kHz	15-33 pF	15-33 pF
XT	100 kHz	100-150 pF	100-150 pF
	2 MHz	15-33 pF	15-33 pF
	4 MHz	15-33 pF	15-33 pF
HS	4 MHz	15-33 pF	15-33 pF
	10 MHz	15-33 pF	15-33 pF

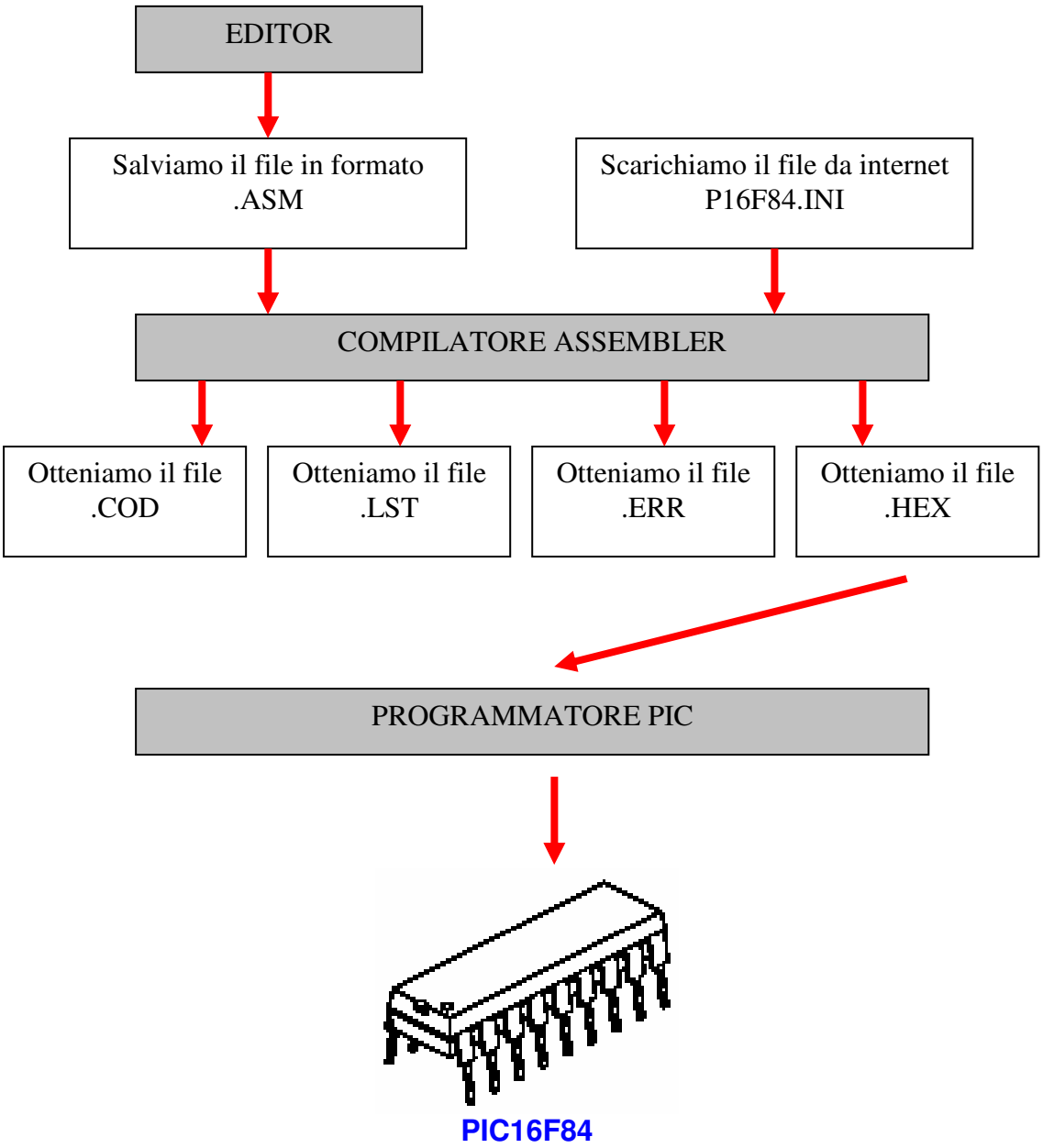
Come abbiamo detto precedentemente per quanto riguarda il **reset del microchip** bisogna collegare attraverso una resistenza da 10kΩ il pin 4 con il positivo dell'alimentazione (da 2,2 a 5 V dc), ma si consiglia sempre di aggiungere un condensatore collegato a massa e un interruttore (in fase di riposo aperto).
Il condensatore ha la funzione di tenere il livello del reset sempre alto nel caso ci siano sbalzi di tensione che ciò potrebbero causare uno scorretto svolgimento del programma.
Premendo l'interruttore ha la funzione di forzare il reset, in questo modo obblighiamo il microchip di riavviare il programma dall'inizio.



5. Compilazione di un programma assembler

Una volta scritto il nostro codice con un normale editor di testo (per windows notepad.exe) dobbiamo trasformare il nostro file ad esempio prova.asm in un formato accettato da nostro microcontrollore (.HEX). Per far ciò possiamo utilizzare il programma MPLAB IDE scaricabile gratuitamente dal sito della Microchip.

In seguito ho schematizzato le operazioni per avere un PIC programmato:

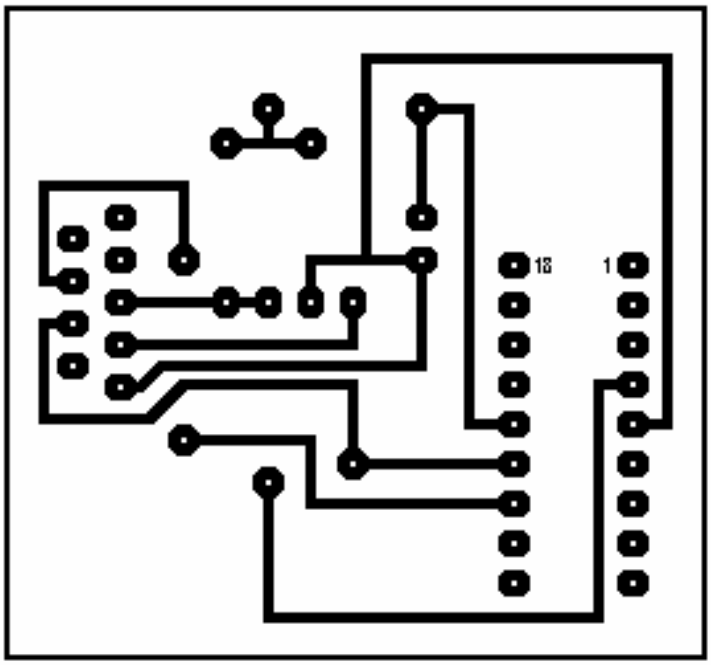


Come possiamo notare il compilatore genera più file:

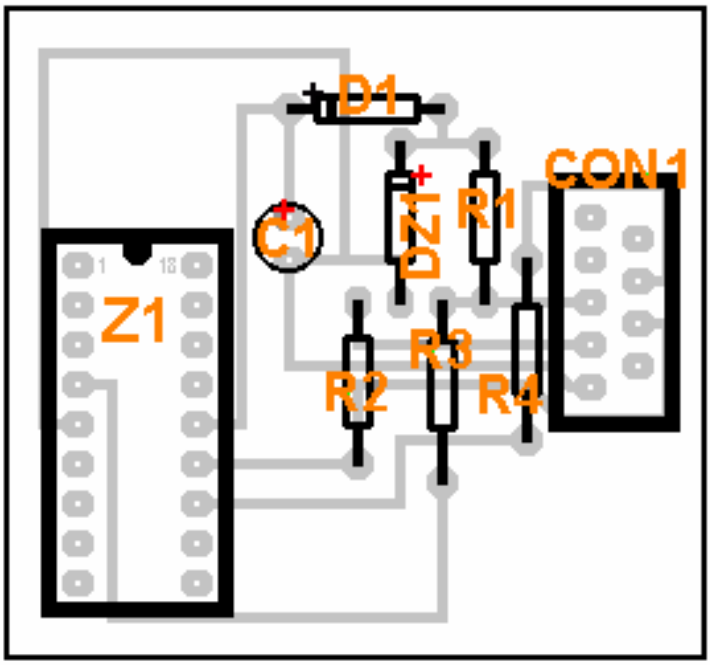
.HEX file contenente gli opcode da caricare nel microcontrollore
.LST file dove viene riportato l'interno source assembler gli opcode
.ERR file contenente la lista degli errori con la relativa riga

6. Programmatore del PIC16C84

Questo circuito è indispensabile per caricare il nostro programma prima scritto in assembler e successivamente compilato nel formato opcode.



Lato piste della basetta



Lato componenti della basetta

I Componenti da utilizzare sono i seguenti:

Z1	Zoccolino 18 pins + PIC16C84
R1 – R2	2,2 kohm ¼ W
R3	10 kohm ¼ W
R4	22 kohm ¼ W
C1	10 uF elettrolitico (verificare la polarizzazione prima di saldare)
D1	1N4148 ½ W (verificare la polarizzazione prima di saldare)
DZ1	5,6 Volt ½ W diodo zener
CON1	Connettore seriale per PC

Per caricare il file opcode nel microcontrollore possiamo utilizzare una delle tante soluzioni freeware che possiamo scaricare da internet, tra cui vi consiglio ic-prog.

7. Istruzioni in assembler

Ora che abbiamo le basi riguardante il microchip da noi preso in considerazione, passiamo alle 35 istruzioni in assembler che il nostro PIC16C84 conosce.

Le istruzioni si dividono in tre gruppi:

- Riferite al byte;
- Riferite al singolo bit;
- Controllo.

Oltre a queste abbiamo:

INCLUDE "P16F84.INC"
Serve a includere nel nostro programma un file con tutte le definizioni standard (Se abbiamo installato MPLAB IDE v7.20 possiamo trovare il file P16F84.INC nella patch "c:\Programmi\Microchip\MPASM Suite", questo dipende anche dal percorso dove l'avete installato).

EQU
Serve a dare un nome ad una locazione di memoria utente.

```
VARIABILE1      EQU      0
```

ORG
Si applica per due scopi:

- Nell'area programma indica all'assemblatore l'indirizzo dove devono essere caricate le istruzioni (ORG 00H);
- Nell'area dati specifica l'indirizzo fisico di partenza (ORG 0CH).

#DEFINE
Si ha la possibilità di dare un nome per ridefinire dei comandi più usati, esempio:

```
#define     Spegni_Led1      bcf  PORTB,LED1
```

In questo modo nelle istruzioni successive possiamo solamente scrivere il nome che abbiamo definito per chiamare l'istruzione, in questo caso:

```
Spegni_Led1
```

__CONFIG
Questa direttiva nel PIC16F84 si trova nell'indirizzo 2007H. Prima di specificare questa istruzione nel file source bisogna prima verifica che il programmatore che si sta utilizzando supporti questa funzione (leggere la configurazione tramite file source). Altrimenti bisogna settarli direttamente tramite il programmatore.

Questa word costituita da 14 bit è suddivisa nella seguente struttura:

Bit 1 e 0	FOSC1 – FOSC2	(Selezionare il tipo di oscillatore utilizzato)
	1 – 1	Oscillatore RC
	1 – 0	Oscillatore HS
	0 – 1	Oscillatore XT
	0 – 0	Oscillatore LP
Bit 2	WDTE	(Abilitare il Watchdog Timer)
	1	Abilitao
	0	Disabilitato
Bit 3	PWRTE	(Abilitare il Power-up Timer)
	1	Disabilitato
	0	Abilitato
Bit 13 al 4	CP	Codice di protezione
	1	OFF
	0	Tutta la memoria è protetta da codice

Esempio di configurazione:	oscillatore HS	10
	WDTE abilitato	1
	PWRTE Disattivato	0
	Codice di protezione disabilitato	1111111111

Per cui otterremo: __CONFIG 11111111110110B

7.1 Riferite al byte

Queste istruzioni vengono eseguite sull'intero byte (8 bit):

ADDWF f,d

Somma il valore contenuto in W con il valore contenuto nel registro f.

Sintassi:

```
addwf     f,d
```

Esempio:

```
primo_dato     equ       0CH
secondo_dato   equ       0DH

               org       00H

               movlw     30
               movwf     primo_dato
               movlw     10
               addwf     primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 40, se avessimo messo d=0 il risultato verrebbe messo in W.

ANDWF f,d

Esegue l'operazione booleana AND tra il registro W ed il registro f.

Sintassi:

```
andwf     f,d
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     11110011B
               movwf     primo_dato
               movlw     00010010B
               andwf     primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 00010010B, se avessimo messo d=0 il risultato verrebbe messo in W.

CLRF f

Azzera il registro f.

Sintassi:

```
clrf      f
```

Esempio:

```
clrf      TMR0
```

Azzera il registro TMR0.

CLRW

Azzera il registro W.

Sintassi:

```
clrw
```

Esempio:

```
clrw
```

COMF f,d

Effettua il complemento del registro f.

Sintassi:

```
comf      f,d
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     11110011B
               movwf     primo_dato
               comf      primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 00001100B, se avessimo messo d=0 il risultato verrebbe messo in W.

DECF f,d

Sottrae 1 al registro f.

Sintassi:

```
decf      f,d
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     9H
               movwf     primo_dato
               decf      primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 8H, se avessimo messo d=0 il risultato verrebbe messo in W.

DECFSZ f,d

Decrementa di 1 il valore del registro f successivamente controlla se tale valore è arrivato a 0, in tal caso salta all'istruzione successiva.

Sintassi:

```
decfsz          f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       9H
                movwf       primo_dato
loop
                decfsz      primo_dato,1
                loop
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà al primo loop 8H fino ad arrivare a 0H dopo di cui esce dal loop, se avessimo messo d=0 il risultato verrebbe messo in W.

INCF f,d

Somma 1 al registro f.

Sintassi:

```
incf        f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       2H
                movwf       primo_dato
                incf        primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 3H, se avessimo messo d=0 il risultato verrebbe messo in W.

INCFSZ f,d

Incrementa di 1 il valore del registro f successivamente controlla se tale valore è arrivato a 0, in tal caso salta all'istruzione successiva.

Sintassi:

```
incfsz          f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       F0H                 ; F0H = 240 dec
                movwf       primo_dato
loop
                incfsz      primo_dato,1
                loop
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà al primo loop F1H fino ad arrivare a FFH dopo di cui essendo un registro a 8 bit al loop successivo si azzera il registro causando l'uscita dal loop, se avessimo messo d=0 il risultato verrebbe messo in W.

IORLWF f,d

Esegue l'operazione booleana OR tra il registro W ed il registro f.

Sintassi:

```
iorlwf          f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       11110011B
                movwf       primo_dato
                movlw       11110110B
                iorlwf      primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 11110111B, se avessimo messo d=0 il risultato verrebbe messo in W.

MOVF f,d

Copia il valore del registro f in se stesso se d=1 ed in W se d=0.

Sintassi:

```
movf            f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       2H
                movwf       primo_dato
                movlw       4H
                movf        primo_dato,0
```

In questo caso d=0 il risultato viene messo in W che varrà 2H, se avessimo messo d=1 il risultato verrebbe messo in primo_dato.

MOVWF f

Copia il valore del registro W nel registro f.

Sintassi:

```
movwf           f
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       2H
                movwf       primo_dato
```

In questo caso primo_dato prende il valore di 2H.

NOP

Non esegue alcuna operazione, serve a far fare un ciclo a vuoto al microprocessore. Ad esempio se utilizziamo un quarzo da 4MHz abbiamo un ritardo di 1 µs ad ogni istruzione nop utilizzata

Sintassi:

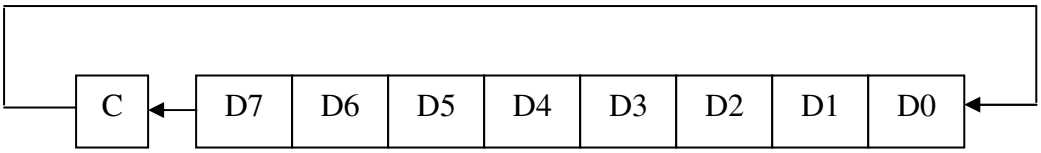
```
nop
```

Esempio:

```
primo_dato     equ       0CH
secondo_dato   equ       0DH

               org       00H

               movlw     2H
               movwf     primo_dato
               nop
               movlw     4H
               movwf     secondo_dato
```

RLF f,d

Ruota a sinistra il contenuto del registro f passando dal carry.



Sintassi:

```
rlf            f,d
```

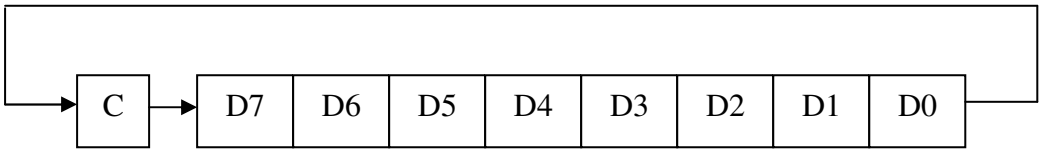
Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     00110011B
               movwf     primo_dato
               rlf       primo_dato,F
```

In questo caso primo_dato prende il valore di 01100110B

RRF f,d

Ruota a destra il contenuto del registro f passando dal carry.



Sintassi:

```
rrf             f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       00110011B
                movwf       primo_dato
                rrf         primo_dato,F
```

In questo caso primo_dato prende il valore di 10011001B

SUBWF f,d

Sottrae il valore del registro W al valore del registro f.

Sintassi:

```
subwf           f,d
```

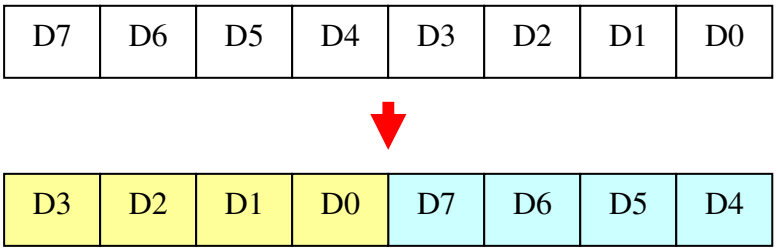
Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       8H
                movwf       primo_dato
                movlw       3H
                subwf       primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 5H, se avessimo messo d=0 il risultato verrebbe messo in W.

SWAPF f,d

Scambia il valore dei quattro bit più significativi (da D7 a D4) contenuti nel registro f con i quattro bit meno significativi (da D3 a D0) dello stesso registro.



Sintassi:

```
swapf           f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       10110010B
                movwf       primo_dato
                swapf       primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 00101011B, se avessimo messo d=0 il risultato verrebbe messo in W.

XORWF f,d

Esegue l'operazione booleana EXOR tra il registro W ed il registro f.

Sintassi:

```
xorwf           f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       11100011B
                movwf       primo_dato
                movlw       10110110B
                xorwf       primo_dato,1
```

In questo caso d=1 il risultato viene messo in primo_dato che varrà 0101010B, se avessimo messo d=0 il risultato verrebbe messo in W.

7.2 Riferite al singolo bit

BCF f,d

Pone a 0 il valore del bit d del registro f.

Sintassi:

```
bcf             f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       11111111B
                movwf       primo_dato
                bcf         primo_dato,1
```

In questo caso primo_dato varrà 11111101B.

BSF f,d

Pone a 1 il valore del bit d del registro f.

Sintassi:

```
bsf             f,d
```

Esempio:

```
primo_dato      equ         0CH

                org         00H

                movlw       00000000B
                movwf       primo_dato
                bsf         primo_dato,1
```

In questo caso primo_dato varrà 00000010B.

BTFSC f,d

Controlla il valore del bit d del registro f, se vale 0 salta all'istruzione successiva.

Sintassi:

```
btfsc          f,d
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     11111110B
               movwf     primo_dato
loop
               btfsc     primo_dato,0
               goto      loop
```

In questo caso essendo il primo bit del registro f uguale a zero non esegue nessun ciclo di loop.

BTFSS f,d

Controlla il valore del bit d del registro f, se vale 1 salta all'istruzione successiva.

Sintassi:

```
btfss          f,d
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               movlw     00000001B
               movwf     primo_dato
loop
               btfss     primo_dato,0
               goto      loop
```

In questo caso essendo il primo bit del registro f uguale a uno non esegue nessun ciclo di loop.

7.3 Controllo

ADDLW k

Somma la costante k al valore del registro W.

Sintassi:

```
addlw          k
```

Esempio:

```
                org       00H

                movlw     2H
                addlw     4H
```

In questo caso il registro W varrà 6H.

ANDLW k

Esegue l'operazione booleana AND tra la costante k e il registro W.

Sintassi:

```
andlw          k
```

Esempio:

```
                org       00H

                movlw     11100011B
                andlw     10000001B
```

In questo caso il registro W varrà 10000001B.

CALL k

Effettua una chiamata alla subroutine k.

Sintassi:

```
call            k
```

Esempio:

```
                org         00H

                call        somma
                …

somma

                movlw       11100011B
                andlw       10000001B
                return
```

CLRWDT

Azzera il registro Watchdog. Questa istruzione viene utilizzata quando abbiamo l'opzione Watchdog abilitata, consiste in un contatore che dopo un determinato tempo effettua un reset del microchip. Per evitare un reset non aspettato bisogna ciclicamente eseguire l'istruzione CLRWDT (azzera il contatore). Questa funzione serve a evitare un blocco nel programma in esecuzione.

Sintassi:

```
clrwdt
```

Esempio:

```
                org         00H

                call        reset
                …

reset

                clrwdt
                return
```

GOTO k

Effettua un salto alla label k.

Sintassi:

```
goto            k
```

Esempio:

```
                org         00H

infinito

                movlw       11100011B
                andlw       10000001B
                goto        infinito
```

In questo esempio abbiamo un ciclo infinito che esegue sempre le stesse due istruzione movlw e andlw.

IORLW k

Esegue l'operazione booleana OR tra la costante k e il registro W.

Sintassi:

```
iorlw           k
```

Esempio:

```
                org         00H

                movlw       11100011B
                iorlw       10000001B
```

In questo caso il registro W varrà 11100011B.

MOVLW k

Mette nel registro W il valore della costante k.

Sintassi:

```
movlw           k
```

Esempio:

```
                org         00H

                movlw       9H
```

In questo caso il registro W varrà 9H.

RETFIE

Ritorna da una subroutine/interrupt.

Sintassi:

```
retfie
```

Esempio:

```
                org         00H
loop
                goto loop
                org         04H
evento
                retfie
```

In questo caso il programma principale esegue un ciclo infinito (loop), ma abilitando un interrupt del microchip e eseguendo il determinato evento il controllo passera al programma 04H e con l'istruzione retfi passera nuovamente al programma principale, cioè il ciclo continuo (loop).

RETLW k

Ritorna dopo una chiamata (call) e copia il valore della costante k nel registro W.

Sintassi:

```
retlw          k
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               call      dato_sub
               movwf     primo_dato

dato_sub

               retlw     9H
```

In questo caso il primo_dato varrà 9H.

RETURN

Ritorna dopo una chiamata (call).

Sintassi:

```
return
```

Esempio:

```
primo_dato     equ       0CH

               org       00H

               call      dato_sub
               movwf     primo_dato

dato_sub

               movlw     9H
               return
```

In questo caso il primo_dato varrà 9H.

SLEEP

Mette il microchip in modalità basso consumo (standby).

Sintassi:

```
sleep
```

Esempio:

```
                org       00H

                sleep
```

SUBLW k

Sottrae a k il valore del registro W.

Sintassi:

```
sublw           k
```

Esempio:

```
                org       00H

                movlw     4H
                sublw     6H
```

In questo caso il registro W varrà 2H.

XORLW k

Esegue l'operazione booleana EXOR tra la costante k e il registro W.

Sintassi:

```
xorlw           k
```

Esempio:

```
                org       00H

                movlw     11100011B
                xorlw     10000001B
```

In questo caso il registro W varrà 01100010B.

OPTION

Copia il valore del registro W nel registro speciale OPTION.

Sintassi:

```
option
```

Esempio:

```
org       00H

movlw     11100011B
option
```

Si sconsiglia di utilizzare l'istruzione option, ma in alternativa si consiglia:

```
org       00H

bsf       STATUS,RP0
movlw     11100011B
movwf     OPTION_REG
```

TRIS f

Copia il valore del registro W nel registro TRIS (porte A e B), i registri TRIS (TRISA e TRISB) determinano il funzionamento delle I/O del microchip.

Sintassi:

```
tris      f
```

Esempio:

```
org       00H

movlw     11100011B
tris      PORTA
```

Si sconsiglia di utilizzare l'istruzione tris, ma in alternativa si consiglia:

```
org       00H

bsf       STATUS,RP0
movlw     11100011B
movwf     TRISB
bcf       STATUS,RP0
```

L'istruzione "bsf STATUS,RP0" serve a passare dal banco 0 al banco 1, viceversa l'istruzione "bcf STATUS,RP0".

Nell'esempio precedente abbiamo impostato il valore di TRISB in 11100011B, per cui abbiamo settato la porta B del microcontrollore nel seguente modo:

N° Bit registro TRISB	Linea Porta B	Valore	Stato
0	RB0	1	Ingresso
1	RB1	1	Ingresso
2	RB2	0	Uscita
3	RB3	0	Uscita
4	RB4	0	Uscita
5	RB5	1	Ingresso
6	RB6	1	Ingresso
7	RB7	1	Ingresso

Nello stesso identico modo possiamo impostare anche la porta A, ricordandosi che le linee non sono 8 ma 5, vediamo un esempio:

```
org       00H

bsf       STATUS,RP0
movlw     00000011B
movwf     TRISA
bcf       STATUS,RP0
```

Per cui la porta A sarà nel seguente modo:

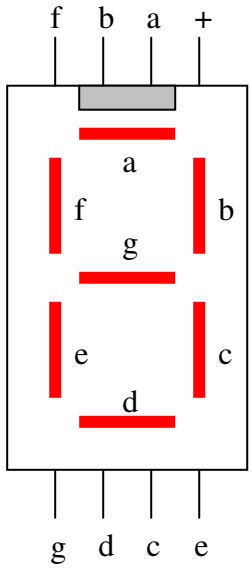
N° Bit registro TRISA	Linea Porta A	Valore	Stato
0	RA0	1	Ingresso
1	RA1	1	Ingresso
2	RA2	0	Uscita
3	RA3	0	Uscita
4	RA4	0	Uscita
5	-	0	-
6	-	0	-
7	-	0	-

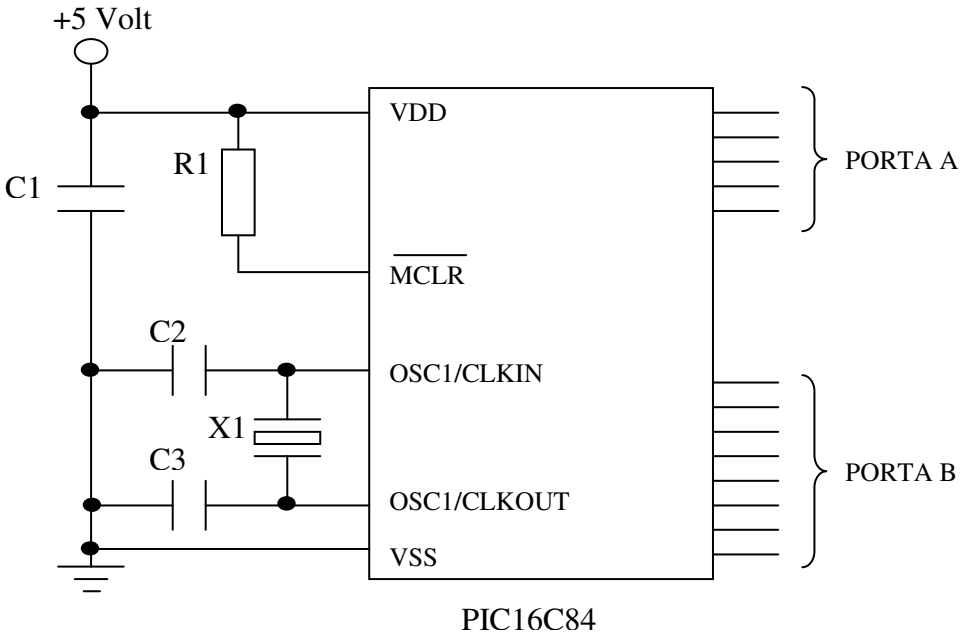
8. Programma di esempio con display 7 segmenti

Ora che abbiamo finito di trattare tutte le istruzioni che possiamo utilizzare per il miglior funzionamento del nostro microcontrollore, passiamo a scrivere un semplice programma che conta da 0 a 9 utilizzando un display a 7 segmenti ad anodo comune, una volta arrivato a contato fino a 9 si azzererà e rincomincia contare, questo ciclo girerà all'infinito.

Per prima cosa dobbiamo progettare il circuito che soddisfi la nostra richiesta collegando i pin del display con la porta B del microcontrollore nel seguente modo:

Pin micro	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Display	-	g	f	e	d	c	b	a





Seconda operazione da eseguire è scaricare da internet il file P16F84.INC (cercare con un motore di ricerca), questo ci permette di avere meno complicazioni.
Terza, scrivere il programma con un normalissimo editor di testo (notepad per windows) e salvarlo con estensione .asm (es. contatore.asm).
Tutte le righe che iniziano con punto virgola (**;**) sono commenti per cui non sono presi in considerazione dal compilatore, consiglio di mettere sempre i commenti perché ci possono dare un aiuto alla lettura del programma, per evidenziarli li ho colorati di blu.

```
;*****************************************
;               contatore.asm
;
;           scritto da Ermes Zannoni
;*****************************************

; Le tre righe seguenti sono direttive per il compilatore
; PROCESSOR gli dice il tipo di CPU
; RADIX dove i numeri non sono specificati sono in decimale
; INCLUDE copia il contenuto del file P16F84.INC nel nostro progr.

                PROCESSOR       16F84
                RADIX           DEC
                INCLUDE         P16F84.INC

; ORG 0CH specifica che le direttive successive saranno dislocate
; nell'area dati.
; NUMERO vanno dall'indirizzo 0CH a 16H.
; INDICE va all'indirizzo 17H.
; CONT_CICLI va all'indirizzo 18H.
; RIT_CONT va all'indirizzo 19H e 1AH.

                ORG             0CH

NUMERO          RES             10
INDICE          RES             1
CONT_CICLI      RES             1
RIT_CONT        RES             2

; ORG 00H specifica che le successive istruzioni vanno nell'area
; programma.

                ORG             00H
```

```
; Le seguenti istruzioni hanno la funzione simile ad un array,
; inserisco un numero in ogni locazione di memoria contigue.

                movlw           11000000B
                movwf           NUMERO+0
                movlw           11111001B
                movwf           NUMERO+1
                movlw           10100100B
                movwf           NUMERO+2
                movlw           10110000B
                movwf           NUMERO+3
                movlw           10011001B
                movwf           NUMERO+4
                movlw           10010010B
                movwf           NUMERO+5
                movlw           10000011B
                movwf           NUMERO+6
                movlw           11111000B
                movwf           NUMERO+7
                movlw           10000000B
                movwf           NUMERO+8
                movlw           10010000B
                movwf           NUMERO+9

; Abilito nel banco 1 le uscite I/O della porta B.

                bsf             STATUS,RP0
                movlw           00000000B
                movwf           TRISB
                bcf             STATUS,RP0

; condizioni iniziali del programma.

Resetta

                movlw           10
                movwf           CONT_CICLI
                movlw           0H
                movwf           INDICE


; Con queste istruzioni all'interno della label Restar sarà il
; nostro ciclo infinito, contatore da 0 a 9 successivamente si
; riavvia partendo nuovamente da 0.

Restart

; Chiamata ad una subdirettiva Ritardo, questo ci permette di
; avere il tempo per la visualizzazione nel display.

                call            Ritardo
```

```
; 0CH è la locazione di memoria chiamata da noi precedentemente
; NUMERO+0, per cui il nostro zero visualizzato nel display.

                movlw           0CH
                addwf           INDICE,0

; Carica l'indice in un puntatore;

                movwf           FSR

; Mette nell'acculumalore il valore puntatore

                movf            INDF,0

; PORTB accende gli elementi del display a seconda del valore nel
; registro W.

                movwf           PORTB

; incf INDICE,1 incrementa il valore di INDICE, questa locazione
; di memoria viene utilizzata da me come se fosse un puntatore
; in un array, nel nostro caso nell'array NUMERO.

                incf            INDICE,1

; decfsz CONT_CICLI,1 decrementa il valore di CONT_CICLI, questo
; serve ad avere un contatore per il controllo di un ciclo
; completo da 0 a 9, quando questo contatore a ritroso prende il
; valore di 0 resetta il tutto.

                decfsz          CONT_CICLI,1

; Rincomincia dalla label Restart.

                goto            Restart

; Rincomincia dalla label Resetta.

                goto            Resetta
```

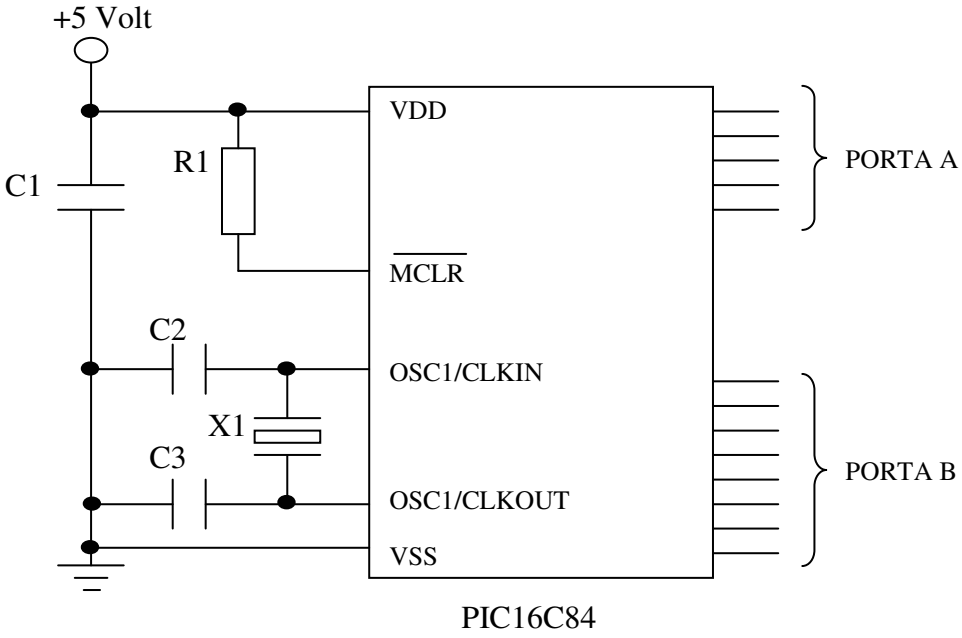
```
; Queste ultime istruzioni servono ad avere il tempo per la
; visualizzazione del numero nel display, conta 255 per 255 volte.
; Impostando le locazioni di memoria RIT_CONT e RIT_CONT+1 a zero,
; al primo decremento passa da 0 a FFH.

Ritardo

                clrf            RIT_CONT
                clrf            RIT_CONT+1

Ritardo_Loop

                decfsz          RIT_CONT,1
                goto            Ritardo_Loop
                decfsz          RIT_CONT+1,1
                goto            Ritardo_Loop
                return

; indica al compilatore che il programma è terminato

END
```

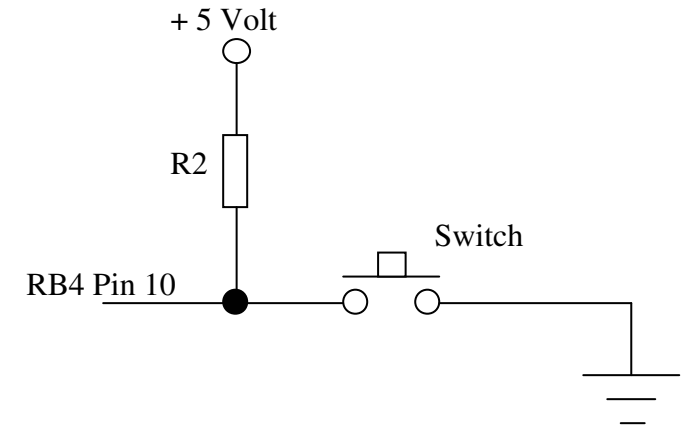
9. Programma di esempio con input dal pulsante

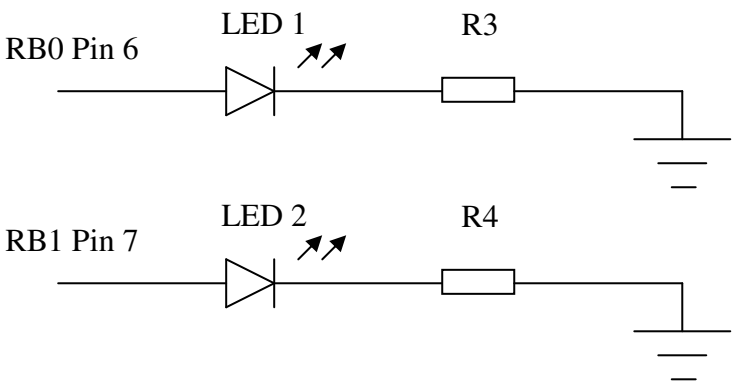
In questo esempio trattiamo come possiamo interagire con un evento esterno.

Utilizzando la porta B del nostro PIC progettiamo un circuito che con un pulsante in posizione on (premuto) accenda il led 1 e spenga il led 2, viceversa con il pulsante il posizione off (non premuto).

I circuito di base è sempre identico:







Ora passiamo a scrivere le istruzioni che servono a soddisfare le nostre esigenze.

```
; Setup del PIC

          PROCESSOR      16F84
          RADIX          DEC
          INCLUDE        P16F84.INC

; Variabili

LED1      EQU            0
LED2      EQU            1
SW        EQU            4

          ORG            0CH

; Carica le seguenti istruzioni nell'area programma

          ORG            00H

; Passa nel banco 1

          bsf            STATUS,RP0

; Setta le prime 4 linee in uscita e le altre 4 in ingresso

          movlw          11110000B
          movwf          TRISB

; Passa nel banco 0

          bcf            STATUS,RP0

; Label per l'istruzione goto (ciclo infinito)

Start

; Se lo switch è premuto accende il led 1

          btfss          PORTB,SW
          bsf            PORTB,LED1

; Se lo switch è premuto spegne il led 2

          btfss          PORTB,SW
          bcf            PORTB,LED2

; Se lo switch non è premuto spegne il led 1

          btfsc          PORTB,SW
          bcf            PORTB,LED1

; Se lo switch non è premuto accende il led 2
```

```
          btfsc           PORTB,SW
          bsf             PORTB,LED2

; Torna all'istruzione Start, inizia il ciclo infinito

          goto            Start

; indica al compilatore che il programma è terminato

END
```

10. Programma di esempio con input da pulsante e interrupt

In questo esempio trattiamo come possiamo interagire con un evento esterno utilizzando gli interrupt.

Utilizzando la porta B del nostro PIC progettiamo un circuito che premendo il pulsante , permette al led 1 di lampeggiare 3 volte.

Il circuito utilizziamo quello precedente, per cui passiamo subito a scrivere il codice da eseguire.

```
; Setup del PIC

          PROCESSOR      16F84
          RADIX          DEC
          INCLUDE        P16F84.INC

; Variabili

LED1      EQU            0

          ORG            0CH

; Questa variabile serve per avere un ritardo da un lampeggio
all'altro

Contatore RES            2

; Questa variabile serve a contare i cicli del lampeggio

Control   RES            1

; Carica le seguenti istruzioni nell'area programma

          ORG            00H

; Salta alla label Start

          goto           Start

; inizio di tutte le istruzioni riguardanti l'interrupt

          ORG            04H

; Imposta a 3 la variabile Control

          movlw          2H
          movwf          Control
```

```
Programma

; accende il led 1

          bsf            PORTB,LED1

; chiama subroutine del ritardo

          call           Ritardo

; spegne il led 1

          bcf            PORTB,LED1

; Chiama la label Ritardo

          call           Ritardo

; decrementa di 1 la variabile Control e successivamente controlla
se è uguale a zero, se è a zero esegue l'istruzione retfie invece
se è diverso da zero esegue un nuovo ciclo

          decfsz         Control,1

; Control <> 0 salta alla label Programma (loop)

          goto           Programma

; Control = 0, resetta il bit RBIF ed esce dal interrupt

          bcf            INTCON,0
          retfie

; Ciclo di ritardo

Ritardo

          clrf           Contatore
          clrf           Contatore+1

LoopRitardo

          decfsz         Contatore,1
          goto           LoopRitardo

          decfsz         Contatore+1,1
          goto           LoopRitardo

          return
```

```
Start

; Passa nel banco 1

          bsf            STATUS,RP0

; Setta le prime 4 linee in uscita e le altre 4 in ingresso

          movlw          11110000B
          movwf          TRISB

; Passa nel banco 0

          bcf            STATUS,RP0

; Azzera i due bit che controllano i led

          bcf            PORTB,LED1

; impostiamo nel registro INTCON a 1 il bit GIE (abilita gli
interrupt) e a 1 il bit RBIE (in specifico abilita quello di stato
delle linee RB4-5-6-7)

          movlw          10001000B
          movwf          INTCON

; Ciclo infinito, interrotto da un interrupt

Inizio    goto           Inizio

; indica al compilatore che il programma è terminato

          END
```

11. Operazioni booleane

AND

dato 1	dato 2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Esempio:

```
10101010    AND
00001111     =
00001010
```

OR

dato 1	dato 2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Esempio:

```
10101010    OR
00001111     =
10101111
```

XOR

dato 1	dato 2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Esempio:

```
10101010    XOR
00001111     =
10100101
```

NOT

dato 1	NOT
0	1
1	0

Esempio:

```
10101010        NOT
01010101
```

12. Notazione decimale, binaria e esadecimale

DECIMALE	BINARIA	ESADECIMALE
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

- Rappresentazione in notazione esadecimale:

  1) H'<numero>' esempio: H'A1'
  2) 0x<numero> esempio: 0xA1
  3) <numero>H esempio: A1H

  Personalmente consiglio la terza opzione

- Rappresentazione in notazione binaria:

  1) B'<numero>' esempio: B'11110000'
  2) <numero>B esempio: 11110000B

  Personalmente consiglio la seconda opzione

13. Riassunto istruzioni

13.1 Istruzione riguardanti il byte

`movlw`	`n`	Copia il valore n sul registro W **(n -> w)**.
`movwf`	`loc`	Copia il valore del registro W nella locazione indicata **(w -> loc)**.
`movf` `movf`	`loc,0` `loc,w`	Copia il valore della locazione indicata nel registro W **(loc -> w)**.
`movf` `movf`	`loc,1` `loc,f`	Copia il valore della locazione indicata in se stesso **(loc -> loc)**.
`swapf` `swapf`	`loc,0` `loc,w`	Inverte i 4 bit meno significativi con quelli più significativi e il risultato viene messo nel registro W.
`swapf` `swapf`	`loc,1` `loc,f`	Inverte i 4 bit meno significativi con quelli più significativi e il risultato viene messo nella locazioni di memoria.
`clrf`	`loc`	Azzera la locazione **(loc = 0)**.
`clrw`		Azzera il registro W **(w = 0)**.

13.2 Istruzioni riguardanti il bit

`rlf` `rlf`	`loc,0` `loc,w`	Ruota a sinistra il contenuto della locazione indicata mettendo il risultato sul registro W.
`rlf` `rlf`	`loc,1` `loc,f`	Ruota a sinistra il contenuto della locazione indicata mettendo il risultato sulla locazione.
`rrf` `rrf`	`loc,0` `loc,w`	Ruota a destra il contenuto della locazione indicata mettendo il risultato sul registro W.
`rrf` `rrf`	`loc,1` `loc,f`	Ruota a destra il contenuto della locazione indicata mettendo il risultato sulla locazione.
`bcf`	`loc,d`	Pone a 0 il valore del bit d della locazione indicata.
`bsf`	`loc,d`	Pone a 1 il valore del bit d della locazione indicata.

13.3 Istruzioni logiche

andlw	n	**w = w AND n**
andwf andwf	loc,0 loc,w	**w = w and loc**
andwf andwf	loc,1 loc,f	**loc = w and loc**
iorlw	n	**w = w OR n**
iorwf iorwf	loc,0 loc,w	**w = w OR loc**
iorwf iorwf	loc,1 loc,f	**loc = w OR loc**
xorlw	n	**w = w XOR n**
xorwf xorwf	loc,0 loc,w	**w = w XOR loc**
xorwf xorwf	loc,1 loc,f	**loc = w XOR loc**
comf comf	loc,0 loc,w	**w = NOT loc**
comf comf	loc,1 loc,f	**loc = NOT loc**

13.4 Istruzioni aritmetiche

addlw	n	Somma il registro W con il valore n e mette il risultato nel registro W **(w = w + n)**.
addwf addwf	loc,0 loc,w	Somma il registro W con la locazione indicata e mette il risultato nel registro W **(w = w + loc)**.
addwf addwf	loc,1 loc,f	Somma il registro W con la locazione indicata e mette il risultato nel locazione indicata **(loc = w + loc)**.
sublw	n	Sottrae a n il valore del registro W e mette il risultato nel registro W **(w = n – w)**.
subwf subwf	loc,0 loc,w	Sottrae alla locazione indicata il valore del registro W e mette il risultato nel registro W **(w = loc – w)**.
subwf subwf	loc,1 loc,f	Sottrae alla locazione indicata il valore del registro W e mette il risultato nel locazione **(loc = loc – w)**.
incf incf	loc,0 loc,w	Incrementa di 1 la locazione indicata e mette il risultato nel registro W **(w = loc + 1)**.
incf incf	loc,1 loc,f	Incrementa di 1 la locazione indicata e mette il risultato nella locazione **(loc = loc + 1)**.
decf decf	loc,0 loc,w	Decrementa di 1 la locazione indicata e mette il risultato nel registro W **(w = loc - 1)**.
decf decf	loc,1 loc,f	Decrementa di 1 la locazione indicata e mette il risultato nella locazione **(loc = loc - 1)**.

13.5 Chiamate alla Subroutine

`btfsc loc,d`	Salta all'istruzione successiva se il bit d della locazione è uguale a 0.
`btfss loc,d`	Salta all'istruzione successiva se il bit d della locazione è uguale a 1.
`incfsz loc,0` `incfsz loc,w`	Mette sul registro W il valore della locazione indicata incrementato di 1, successivamente controlla se il risultato sia a 0. Se il risultato è a 0 salta all'istruzione successiva.
`incfsz loc,1` `incfsz loc,f`	Mette sulla locazione il valore della locazione indicata incrementato di 1, successivamente controlla se il risultato sia a 0. Se il risultato è a 0 salta all'istruzione successiva.
`decfsz loc,0` `decfsz loc,w`	Mette sul registro W il valore della locazione indicata decrementato di 1, successivamente controlla se il risultato sia a 0. Se il risultato è a 0 salta all'istruzione successiva.
`decfsz loc,1` `decfsz loc,f`	Mette sulla locazione il valore della locazione indicata decrementato di 1, successivamente controlla se il risultato sia a 0. Se il risultato è a 0 salta all'istruzione successiva.
`goto loc`	Salta alla locazione richiesta.
`call loc`	Chiamata di una subroutine.
`return`	Ritorno da una subroutine.
`retlw n`	Ritorno da una subroutine con il valore di W.
`retfie`	Ritorno da un interrupt.

13.6 Istruzioni di controllo

`nop`	Interruzione fa un ciclo a vuoto del microcontrollore
`clrwdt`	Azzera il watch dog
`sleep`	Standby mode, risparmio energetico

14. Confronto tra due valori

Uguaglianza (W = f)

```
movlw     90              n -> W
sublw     90              W = n - W
btfss     STATUS,Z
```

valore A = valore B se il flag Z vale 1.

Disuguaglianza (W <> f)

```
movlw     90              n -> W
sublw     45              W = n - W
btfsc     STATUS,Z
```

valore A <> valore B se il flag Z vale 0.

Maggioranza (W > f)

```
movlw     90              n -> W
sublw     40              W = n - W
btfss     STATUS,C
```

valore A > valore B se il flag C vale 0 (90 > 40).

Minoranza (W < f)

```
movlw     40              n -> W
sublw     90              W = n - W
btfss     STATUS,C
```

valore A < valore B se il flag C vale 0.

Minoranza o Uguaglianza (W <= f)

```
movlw     40              n -> W
sublw     90              W = n - W
btfsc     STATUS,C
```

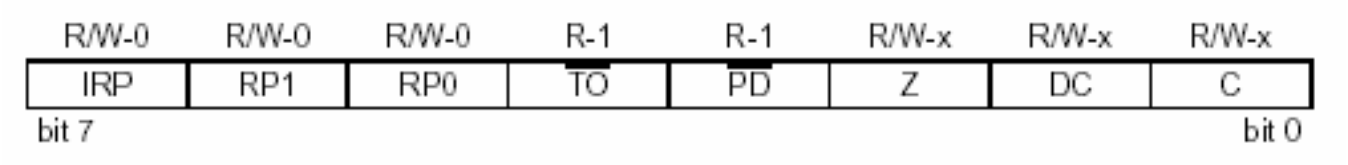
valore A <= valore B se il flag C vale 1.

Minoranza o Uguaglianza (W >= f)

```
movlw     90              n -> W
sublw     40              W = n - W
btfsc     STATUS,C
```

valore A >= valore B se il flag C vale 1.

Sintassi del registro STATUS



RPO Seleziona il banco 0 o 1
1 = Banco 1
0 = Banco 0

TO Time out
1 = Dopo l'accensione, all'istruzione CLRWDT o SLEEP
0 = WDT Time out

PD Power-down
1 = Dopo l'accensione o all'istruzione CLRWDT
0 = All'istruzione SLEEP

Z Zero
1 = Zero
0 = No zero

Z Carry
1 = Carry
0 = No carry