# MANUALE PER LA ROGRAMMAZIONE DEL PIC 16F84

A cura del Prof. Puglisi Salvatore

## Principi basi per la programmazione

Introduzione alla programmazione
Strutture basi di programmazione

## Descrizione del microcontrollore 16F84

Descrizione tecnica e architettonica del PIC
Impostazioni per il corretto funzionamento

## Linguaggio assembler

Il linguaggio Assembler
Il source
Le direttive

## Realizzazione e compilazione di source per il PIC

Realizzazione di un listato
Programma compilatore

## Programmazione del PIC

Programmatore
Software per programmare il PIC

## Esempi di source

## Premessa:

Spinto dalla passione per l'elettronica e la programmazione ho voluto realizzare un piccolo e modesto manuale con l'unica intenzione di semplificare la vita a chi vuole avvicinarsi per la prima volta a tale esperienza .
Auguro che questo manuale sia chiaro, considerando che sicuramente per i meno avvezzi ci saranno alcune problematiche legate dalla poca, praticità .
In questo manuale saranno trattati diversi argomenti in modo da poter fornire le funzioni base per la programmazione del PIC 16F84.

Buon lavoro dal Prof. Puglisi Salvatore

# Indice

## Principi basi per la programmazione



## Descrizione del microcontrollore 16F84



## Linguaggio assembler



## Realizzazione e compilazione di source per il PIC



## Programmazione del PIC



## Esempi di source

# Principi base per la programmazione

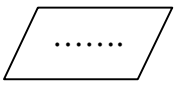
## 1.1 Introduzione alla programmazione

Per creare un programma bisogna seguire una certa scaletta composta da quattro elementi, che sono:

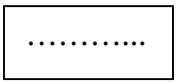
1. Analisi
2. Realizzazione di uno schema a blocchi o flow-chart
3. Scrittura del programma
4. E compilazione

**Analisi:** l'analisi è una fase molto importante perché si definisce cosa deve fare il programma, il numero delle variabili e tutte le funzioni che deve eseguire il nostro programma. Ciò sembrerà una fase inutile ma per i principianti o per chi realizza un programma complesso sarà molto utile.

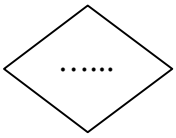
**Realizzazione di uno schema a blocchi o flow-chart:** In questa fase invece si realizza uno schema primitivo del programma che ci permetterà di individuare eventuali errori nel terzo passaggio. Questi flow-chart hanno dei blocchi standard che rappresentano delle operazioni particolari, dentro questi blocchi vanno inserite delle istruzioni o dei valori. I principali blocchi per realizzare uno schema sono:



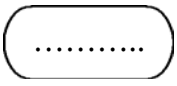

Questo blocco attribuisce dei valori a delle variabili, inizializza ( crea ) le variabili nella ram dell'elaboratore e visualizza il valore di una variabile.




Questo blocco ha invece ha la funzione di eseguire una istruzione, per intenderci le operazioni aritmetiche e altre funzioni.




Questo blocco invece esegue una scelta vera o falsa, se è vero il programma esegue una certa serie d'istruzioni, mentre se è falso ne esegue una serie diversa. Questo comando si usa quando si deve uscire da un ciclo o si deve fare una scelta.




Questo blocco viene posto all'inizio e alla fine dello schema a blocchi e rappresenta soltanto dove il programma inizia e dove finisce.

Questi blocchi sono collegati da delle frecce la cui punta rappresenta il senso in cui i dati vengono trasferiti.

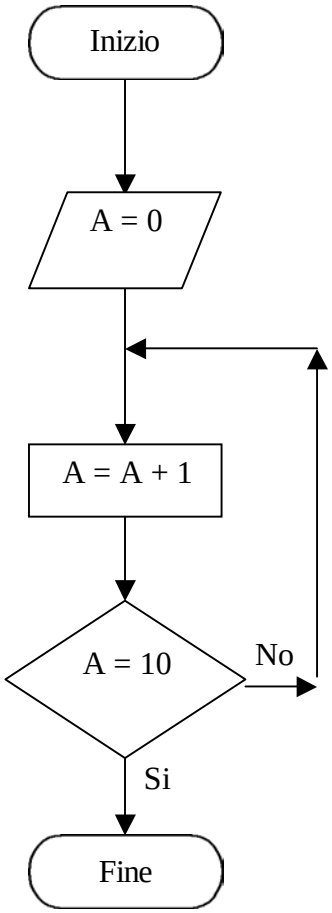
**Esempio su come si esegue l'analisi e lo schema a blocchi.**

Questo programma contare fino a 10.

**Analisi:**

1. Il programma ha bisogno di una variabile per contare fino a 10
2. Il programma deve eseguire un ciclo in modo da poter incrementare la variabile.

**Diagramma a blocchi:**




**C**on questo blocco si indica l'inizio del programma.

**C**on questo blocco si crea la variabile A e le si attribuisce il valore 0.

**Q**uesto blocco incrementa A di uno e salva il nuovo valore in A.

**C**on questo blocco si compie una scelta se A è uguale a 10 allora esce dal ciclo se no esegue l'istruzione che la precede fino a che non si verifica la situazione.

**C**on questo blocco si indica la fine del programma.

**Scrittura del programma:** In questa fase non si fa altro che tradurre lo schema a blocchi nel linguaggio di programmazione in cui si scrive ( C/C++, Assembler, Basic , Pascal, ecc. ).

**Compilazione:** Questa fase solitamente non è influenzabile dall'utente, poiché è il programma stesso che esegue questa operazione che traduce il programma scritto in Alto Livello ( linguaggio umano ) in linguaggio macchina ovvero un linguaggio a Basso Livello ( comprensibile all'elaboratore ).

## 1.2 Strutture basi di programmazioni

A prescindere del linguaggio in cui si scrive ci sono delle strutture canoniche che chiunque deve conoscere, e sono:

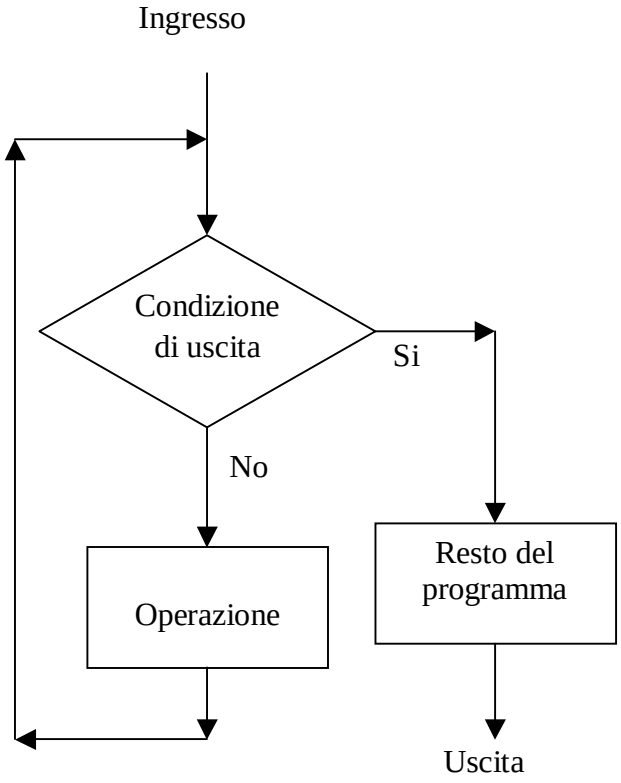
Il ciclo o loop e controllo in testa o in coda
Scelta vera o falsa

Queste strutture sono fondamentali perché i cicli e le scelte sono la parte più consistente di qualsiasi programma, specialmente nel caso nostro che dovendo programmare in Assembler, faremo un grande uso di queste strutture visto che il set d'istruzioni del PIC 16F84 comprende solamente 35 istruzioni.

**Il ciclo o loop:** Questa struttura è di solito accoppiato con un controllo in testa o in coda, con questa operazione si esegue un certo numero di volte una serie di funzioni; fino a che non si verifica una certa condizione. Questa condizione è verificata dal controllo in testa o in coda, la differenza tra queste due opzioni è che nel controllo in testa la condizione è controllato subito, senza quindi eseguire ciò che sta all'interno del ciclo, mentre nel controllo in coda prima di verificare la condizione si esegue ciò che sta all'interno del ciclo.

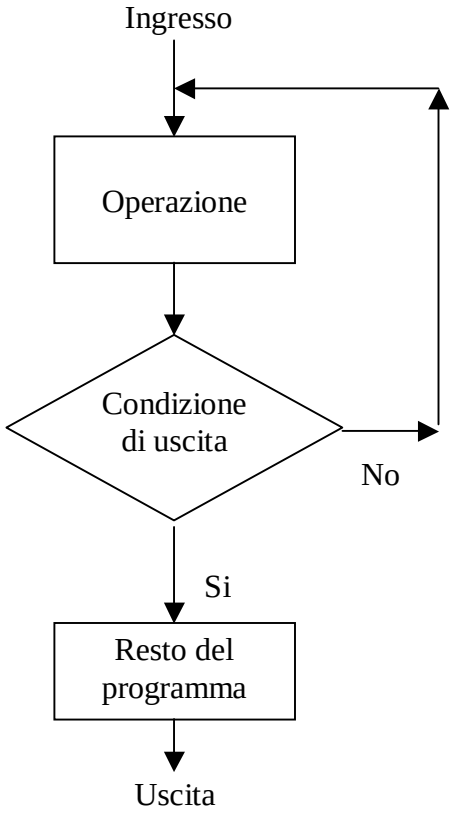
**Esempio di ciclo con controllo in testa o in coda.**

**Controllo in testa.**



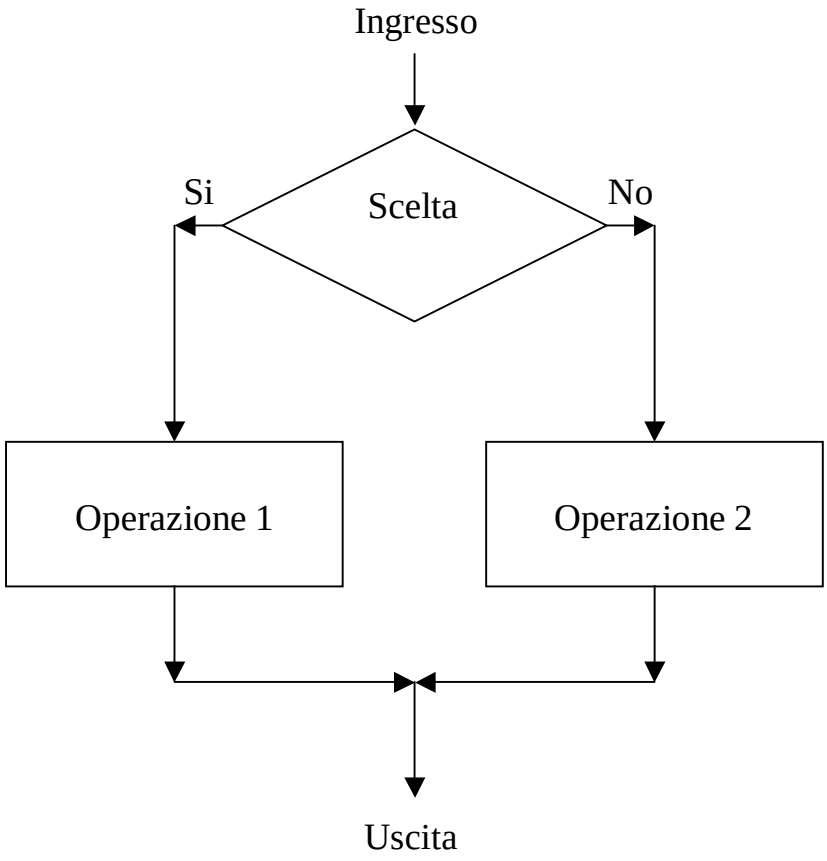

**Q**uesto tipo di ciclo ha un controllo in testa ovvero il dado viene prima controllato nel blocco di scelta se la condizione è verificata il ciclo viene saltato, se la condizione invece non è verificata il programma esegue il ciclo fino a che si crea la condizione di uscita.

**Controllo in coda.**




**Q**uesto ciclo esegue l'operazione che ha nel ciclo, e controlla se la condizione è rispettata, se si esce dal ciclo se no continua l'iterazione.

**La scelta vera o falsa:** Questa struttura invece esegue una scelta, ovvero compara un dato e un base al risultato, vero o falso, esegue una riga d'istruzione anziché un'altra.



# Descrizione del microcontrollore 16F84

## 2.1 Descrizione tecnica e architettonica del PIC

Il PIC 16F84 e un microcontrollore molto versatile, ha una piedinatura 9+9 e ha due porte bi-direzionabili, ovvero possono essere settate a scelta in input o output.
Ha un reset attivo basso, e ha un'alimentazione di 5 volt; la comunicazione interna ha un bus di 8 bit, ciò permette di lavorare con un byte alla volta semplificando parecchio tutte le procedure.
Il 16F84 ha una ram di 68 byte e una Eeprom interna di 64 byte, il clock può essere creato con un quarzo o una rete RC; il clock del PIC può raggiungere i 10 MHz.
Ci sono quattro tipi di clock RC, LP, HS, XT.

**Attenzione:** Questo parametro può essere impostato nel source del programma, o nel programma che programma il PIC.

**RC:**
Il clock RC si ottiene ponendo una rete resistenza condensatore, questa è la soluzione più economica, questa rete va posta tra i due piedini clock. Il clock2 genera una frequenza pari a quella del clock1, questo segnale può essere usato come sincronismo, il circuito è rappresentato nelle **Figura 1**.

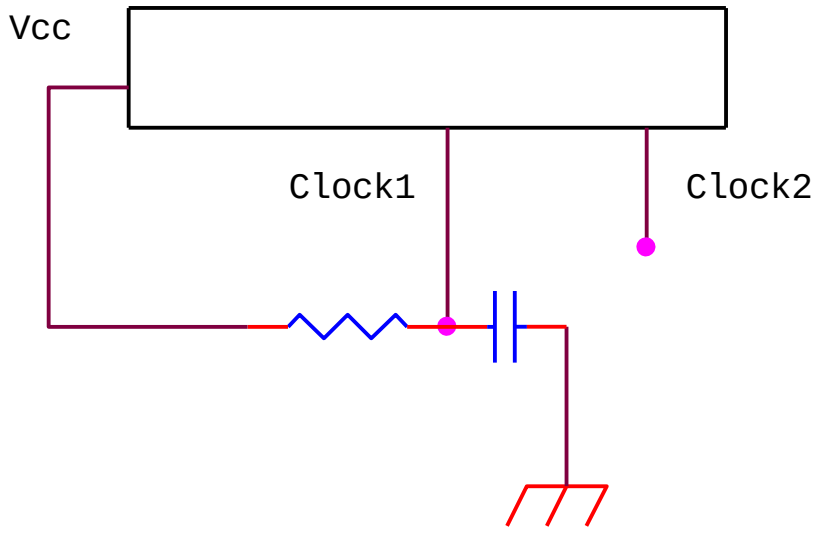



**Figura 1. Schema per generare un clock con una rete RC**

**LP:**
Questa opzione definisce l'uso di un cristallo con basso assorbimento di corrente ( Low power crystal ).

**HS:**
Questa opzione definisce l'uso di un cristallo ad alta frequenza ( High speed crystal ).

**XT:**
Questo tipo di clock si ottiene con un quarzo, e i due condensatori di sfasamento; è il tipo di clock più usato poiché è un parametro standard.

Per ottenere un clock con un quarzo, bisogna aggiungere due condensatori come da **Figura 2**, il valore di questi condensatori variano in base alla frequenza del quarzo, tali valori sono indicati dalla **Tabella 1**.

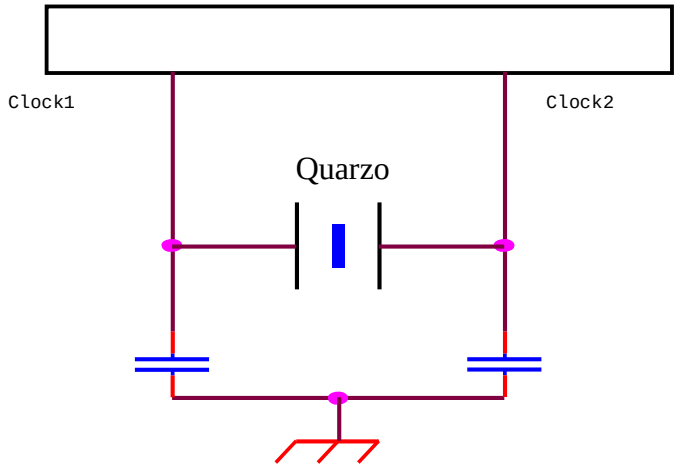



**Figura 2. Schema per generare un clock con un quarzo**

| Mode | Freq | OSC1/C1 | OSC2/C2 |
|---|---|---|---|
| LP | 32 kHz<br>200 kHz | 68 - 100 pF<br>15 - 33 pF | 68 - 100 pF<br>15 - 33 pF |
| XT | 100 kHz<br>2 MHz<br>4 MHz | 100 - 150 pF<br>15 - 33 pF<br>15 - 33 pF | 100 - 150 pF<br>15 - 33 pF<br>15 - 33 pF |
| HS | 4 MHz<br>10 MHz | 15 - 33 pF<br>15 - 33 pF | 15 - 33 pF<br>15 - 33 pF |

**Tabella 1. In questa tabella sono definiti i valori dei condensatori di sfasamento in corrispondenza alla frequenza dei quarzi.**

## 2.2 Architettura

Il PIC 16F84 come abbiamo già detto è un integrato 9+9; questi 18 piedini si dividono nel seguente modo:

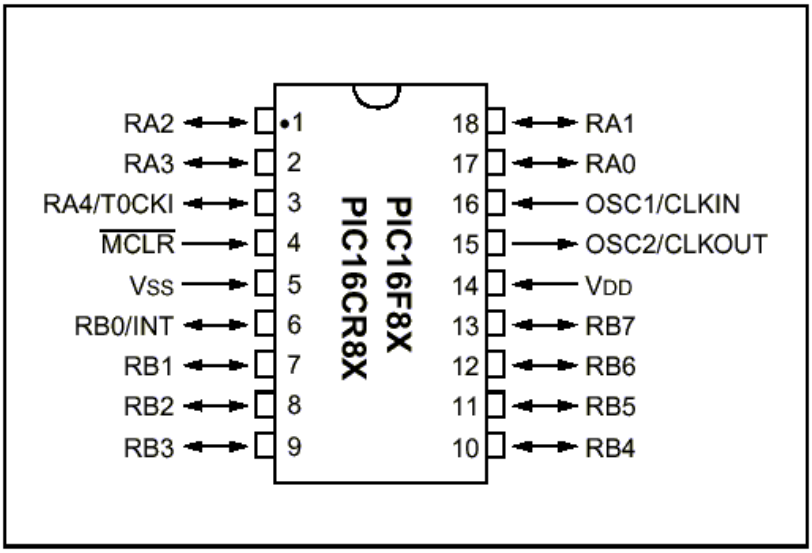



**Figura 3. Piedinatura del PIC 16F84**

13 piedini si dividono nelle due porte bi-direzionabili, 5 compongono la PORTA ( rispettivamente gli RA ) e 8 la PORTB (rispettivamente gli RA e gli RB, come da **Figura 3**). Il piedino 5 è l'alimentazione, mentre il 14 è la massa; i piedini 15 16 sono i due clock[1]. Il piedino 4 è il reset ( vedi **Tabella 2** ) che va collegato a Vss con una resistenza da 1K**Q.**

| Stato reset | Stato programma |
|---|---|
| Alto | Esegue il programma |
| Basso | Non esegue il programma |

**Attenzione:** Un appunto molto importante da fare è che se una delle due porte è settato in output il PIC pilota a livello basso l'eventuale uscita del circuito accoppiato, questa condizione si presenta quando si vuole interfacciare una EEprom al 16F84.

[1] **Quando il clock è generato da una rete RC come ingresso si deve usare solo il clkin, mentre con i quarzi si devono usare tutti e due i clock Figura 2**.

# Linguaggio assembler

## 3.1 Il linguaggio assembler

Il PIC 16F84 è un vero e proprio piccolo computer, dotato di una CPU, una RAM, una ROM e una memoria programma; perciò il suo linguaggio di programmazione è l'assembler. Questo linguaggio è a basso livello, quindi è molto complicato e poco elastico, perciò per eseguire anche solo una semplice operazione come dare un valore a una variabile bisogna usare due istruzioni.
Per programmare il PIC non basta conoscere il set d'istruzioni ma bisogna saper usare anche le direttive, queste istruzioni non vengono eseguite durante il programma ma servono al programma **compilatore** ( vedi pag. 5 ) per creare il file hex [1].

## 3.2 Il source

Per creare un programma bisogna seguire due fasi, la prima è la scrittura del source e la seconda è la compilazione.
Per creare quindi un listato bisogna generare un file txt, questo file può essere generato da Notepad, all'interno noi andremo a scriverci il nostro programma e solo inseguito lo trasformeremo in un file hex.

## 3.3 Le direttive

Come ho gia detto per programmare il PIC bisogna prima impostare le direttive, queste vanno scritte all'inizio del listato e definiscono: il tipo d'integrato, le variabili, le costanti e il tipo clock ( direttiva opzionale ).

PROCESSOR 16F84 = Questa direttiva definisce il tipo d'integrato usato.

__config = 0xFFF = Questa direttiva definisce il tipo di clock il parametro da inserire dopo l'uguale è definito dal checksum del programma programmatore.

RADIX … = Questa direttiva definisce che tutti i numeri senza notazione sono da considerarsi … ; ci sono tre tipi di notazione decimale = DEC, esadecimale = HEX e binario = BIN.

[1] **Il file che andremo a memorizzare dentro il PIC ha un'estensione hex ovvero esadecimale, infatti il PIC capisce solo questo tipo di linguaggio.**

INCLUDE "P16F84.INC" = Questa direttiva dice al programma compilatore quale libreria usare per creare il file hex.

RES … = Questa direttiva dice quanti è lunga una variabile se è uguale a 1 sarà di un byte se 2 sarà di due etc.

#DEFINE variabile  valore = definisce una costante.

variabile EQU valore = definisce una costante.

### 3.4.1 Set d'istruzioni per il PIC 16F84

Come abbiamo detto il PIC ha un set che comprende 35 istruzioni, queste istruzioni si dividono in tre gruppi che sono:

Byte-oriented
Bit-oriented
Literal and control

**Byte-oriented:** questo gruppo d'istruzioni opera su un registro di 8 bit, quindi tutte le operazioni fatte vanno a modificare il contenuto di un registro.

**Bit-oriented:** questo gruppo comprende 4 istruzioni che operano su un singolo bit di un registro.

**Literal and control:** questo gruppo d'istruzioni opera su una costante di 8 bit, questa costante ( literal ) va a modificare un registro in base all'operazione che si deve eseguire.

Questi gruppi d'istruzione hanno in aggiunta una serie di parametri elencati in **Tabella 2**.

| Parametro | Significato |
|---|---|
| f | Indirizzo registro |
| w | Registro di lavoro |
| d | Indirizzo |
| b | Bit |
| k | Costante 8 bit |

**f :** Questo parametro rappresenta un registro, un registro può essere personalizzato in modo da facilitare l'utente poiché al registro possiamo dare un nome qualunque, penserà poi il programma in fase di compilazione a dare un vero indirizzo al registro.

**w :** Questo registro è un accumulatore che usa il PIC per memorizzarci un dato temporaneo.

**d :** Questo parametro può assumere solo i valori 0 1 e indica dove il dato verrà salvato se d = 1 allora il dato verrà salvato nel registro f, se d = 0 il dato verrà salvato nel registro w.

**b :** Questo parametro definisce il bit su cui deve essere portata a termine l'operazione, poiché i bit di un registro sono 8 il valore di b varia tra 0 e 7.

**k :** Questo dato è una costante di 8 bit e lavora solo con le istruzioni del terzo gruppo.

Per il set d'istruzioni completo bisogna scaricare il datasheet direttamente dalla casa produttrice Microchip.
Indirizzo web: www.microchip.it

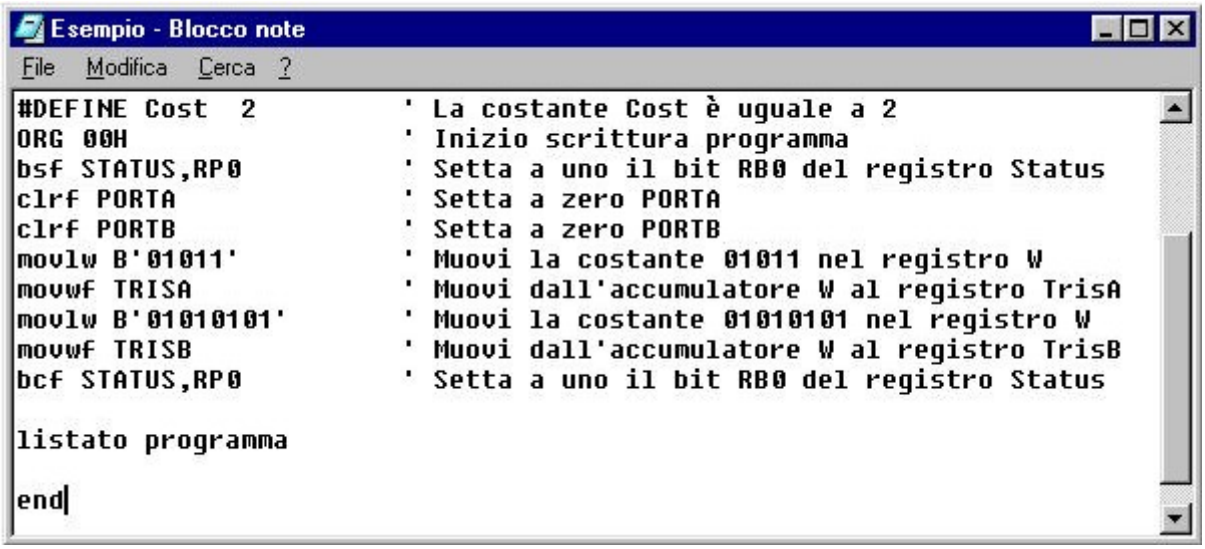
# Realizzazione e compilazione di source per il PIC

## 4.1 Realizzazione di un listato

Per creare un programma hex bisogna avere un file sorgente, questo file si può ottenere con un semplice programma di scrittura come Notepad. Le istruzioni vanno scritte incolonnate poiché il programma compilatore ammette una sola istruzione per riga.
La prima cosa da scrivere sono le direttive, queste dovranno specificare il tipo di integrato, le variabili, le costanti e eventualmente il clock. Bisogna per le variabili e le costanti specificare l'indirizzo dell'area ram in cui verranno create che corrisponde a ORG 0CH.


A questo punto bisogna impostare le porte, per far ciò bisogna modificare i registri, Trisa e Trisb che corrispondono rispettivamente a PortA e PortB.
Il registro TrisA è un registro composto da 5 bit, tale è infatti il numero di pin che corrispondono a questa porta di comunicazione; partendo con RA0 per il bit meno significativo fino a RA4 per il bit più significativo. Impostando i bit o a 0 o a 1 abbiamo rispettivamente delle porte di output e input. La stessa cosa si fa per la PortB con l'unica differenza che il suo registro TrisB è composto da 8 bit, tale infatti è il numero di piedini che la compongono.

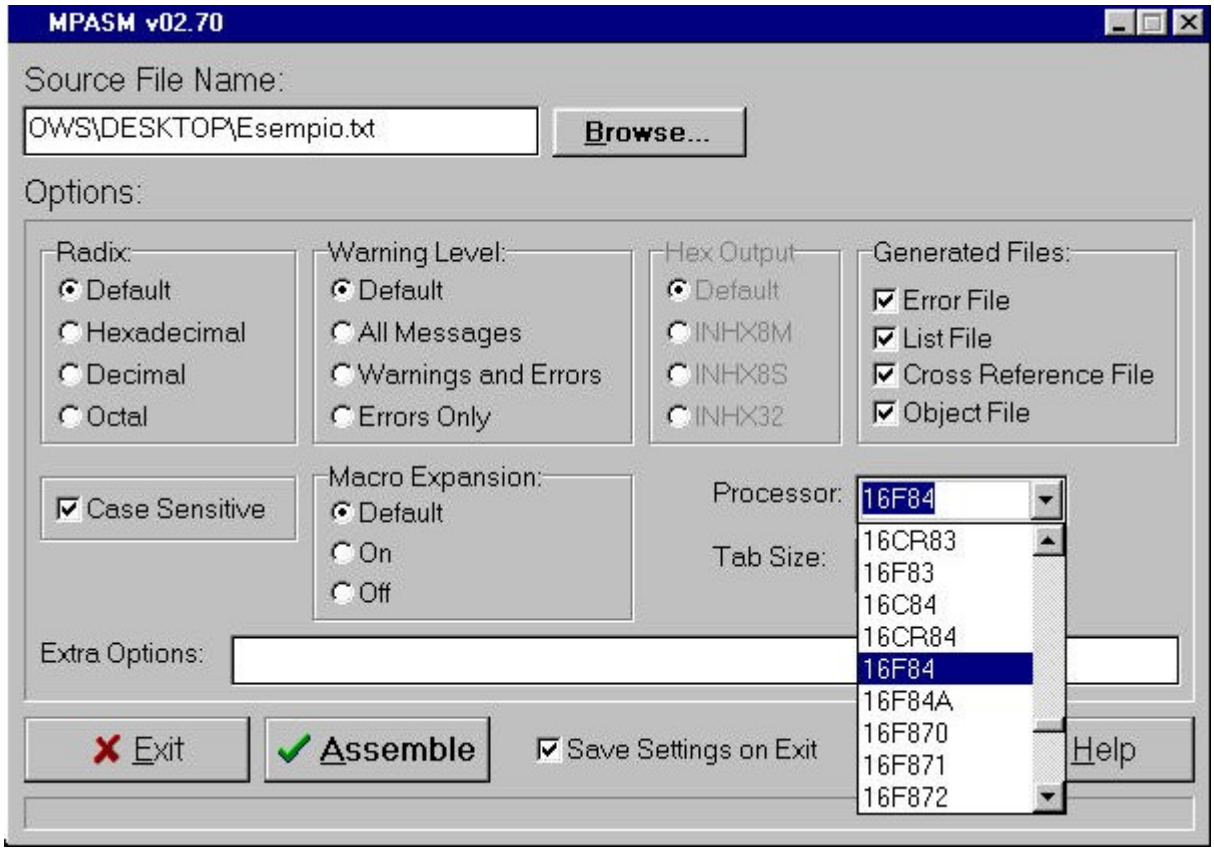
Prima d'impostare le porte di comunicazione bisogna dire al programma compilatore che ora si sta scrivendo nella memoria di programma, ciò si ottiene scrivendo ORG 00h.




Una volta specificato l'area in cui si scrive dobbiamo settare a livello alto il bit RP0 del registro STATUS, queste operazione serve impedire che per sbaglio durante l'esecuzione del programma questi parametri vengano modificati. Ciò non toglie la possibilità di modificare durante il programma l'impostazione delle porte.
A questo punto si muove il valore con notazione binaria dentro i due registri Tris, si setta a zero il bit RB0 e si comincia a scrivere il vero programma. Finito il programma bisogna scrivere **end** per dire al programma compilatore che il listato è terminato.

## 4.2 Programma compilatore

Per generare il file hex bisogna disporre di un programma assemblatore, consiglio il programma **Mpasm** della Microchip che distribuisce freeware. A questo punto bisogna una volta aperto il programma bisogna impostare il tipo d'integrato, in questo caso il PIC 16F84; bisogna anche impostare il percorso del file sorgente.




Una volta stabilito il tipo d'integrato e il percorso della fila sorgente bisogna impostare il Genereted Files, ognuno di questi quattro parametri genera un file, il più importante è il Error file poiché se c'è un errore nel programma Mpasm lo scrive al suo interno. Questo file può essere aperto con Notepad e se avviene un errore, al suo interno troviamo scritta la riga interessate e il tipo di errore.

# Programmazione del PIC

## 5.1 Programmatore

Questa capitolo si divide in due parti, la prima esaminerà la costruzione del hardware per la programmazione del PIC; la seconda riguarderà la gestione del software di programmazione.

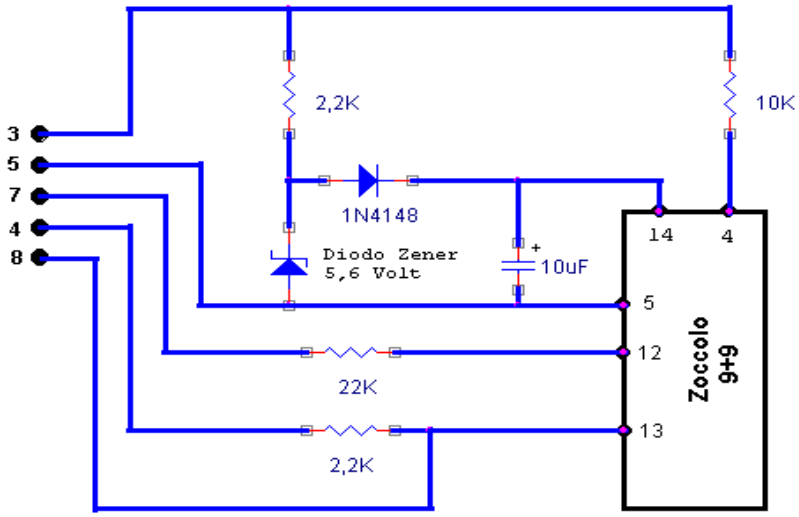
## 5.2 Hardware

Questa parte richiede una buona dose di pratica e di esperienza, poiché per realizzare la basetta su cui andremo a saldare i componenti bisogna avere un po' d'esperienza con acidi e trasferibili.

**Componenti:**

1 resistenze da 22 K
1 resistenza da 10 K
2 resistenza da 2,2 K
1 diodo Zener da 5,6 Volt
1 diodo 1N4148
1 condensatore elettrolitico 10 uF 25 Volt
1 zoccolo porta integrati 9+9
1 spinotto femmina DB9 PC/COM

**Schema:**




**Schema elettrico per il programmatore PIC 16F84**

**Descrizione:**

Questo schema elettrico di principio rappresenta il programmatore di PIC, il circuito è autoalimentao, ovvero l'alimentazione viene creata dal circuito stesso utilizzando la porta COM.
Il programmatore trasmette serialmente i dati al piedino 13 del PIC, al pin 5 è la massa, al pin 14 la Vcc di alimentazione e al piedino 12 il Clock.
Questo circuito genera una corrente bassissima e quindi non rischierete mai di danneggiare il PIC, quindi non preoccupatevi se programmate con il PIC inserito al contrario nello zoccolo. Lo standard di comunicazione tra il PC e il programmatore e il RS232.

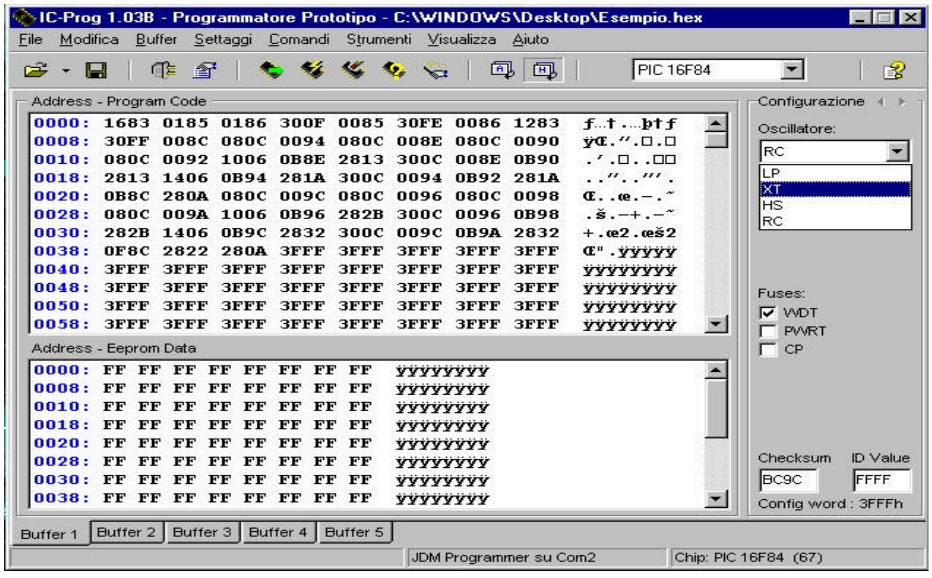
## 5.3 Software per programmare il PIC

### Introduzione

Il software per la programmazione del PIC non fa altro che gestire la porta COM; questi programmi si trovano freeware in rete. Questi programmi sia che lavorino in ambiente Dos o in ambiente Window hanno le stesse opzioni, io consiglio il programmatore Icprog reperibile in rete; questo programma è molto intuitivo e quindi ne rende facile l'uso e inoltre c'è pure la versione italiana, il che non guasta.

### Impostazioni di Icprog

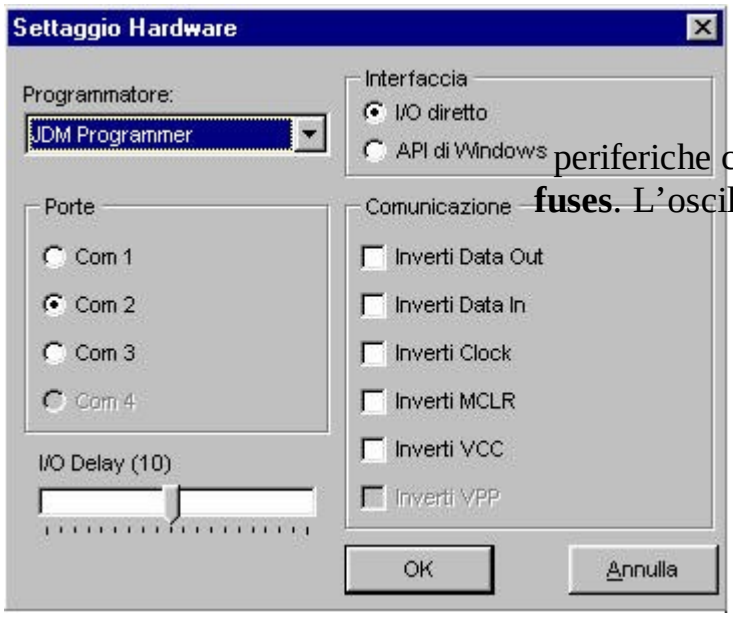
Questo programma come schermata iniziale ha una finestra, che per il PIC saranno due, in questa finestra viene visualizzato il contenuto del file .hex .




**Schermata iniziale di Icprog**

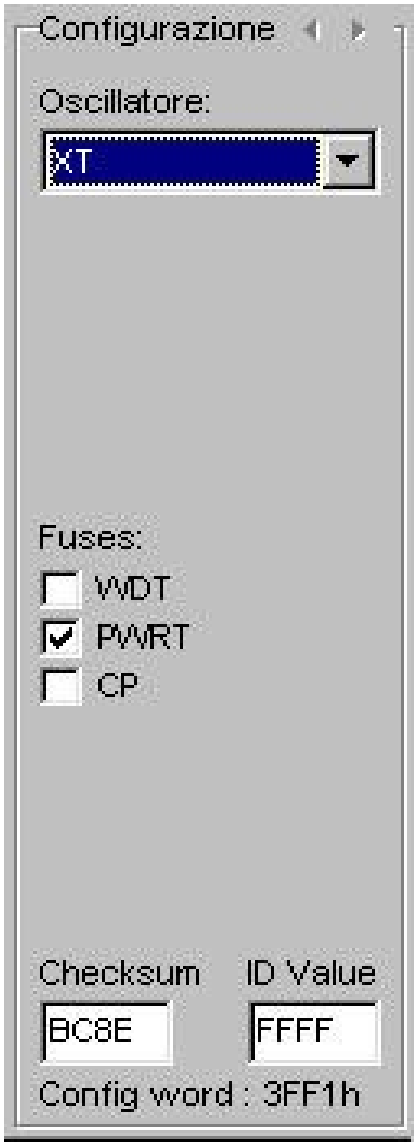
Prima di poter programmare un PIC bisogna impostare la porta di comunicazione ed alcuni parametri importanti. Per impostare la porta bisogna schiacciare i pulsanti settaggi :


Una volta selezionato hardware si aprirà una seconda finestra dove si dovrà impostare al porta :



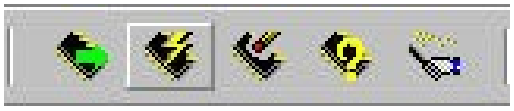

IL programmatore da selezionare nel nostro caso è il JDM Programmer, la porta Com varia a seconda delle periferiche che avete sono, l'**oscillatore** e i **fuses**. L'oscillatore c

| Parametro oscillatore | Tipo di oscillatore |
|---|---|
| RC | Resistenza condensatore |
| XT | Standard quarzo |
| LP | Quarzo con basso assorbimento |
| HS | Quarzo alta frequenza |




Stabilito l'oscillatore, bisogna definire i fuses questi parametri si riferiscono all'alimentazione come da disegno l'unico da selezionare è **PWRT**.
Il Checksum, come si nota nella figura precedente è definito da quattro cifre o lettere; questo parametro è da abbinare alla direttiva config, in questo modo si potrà evitare di dover impostare ogni volta l'oscillatore e i fuses.

Una volta impostati tutti i parametri non rimangono che cliccare con il mouse sul pulsante con sopra disegnato un fulmine.



Dopo aver programmato il PIC vi consiglio di verificare l'integrato perché potrebbero essersi verificati dei problemi nella comunicazione.

# Esempi di source

## 6.1 Esempi

In questa ultima parte parleremo dei source, o meglio faremo alcuni esempi di file sorgente. In questo modo spero di chiarire gli ultimi dubbi che possiate aver riscontrato.
**Importante:** I seguenti programmi non sono completi poiché non sono riportate le direttive.

**Gestione porte di comunicazioni:**

Sulle porte di comunicazione si può lavorare in input e output, ed esistono quattro istruzioni che hanno il compito di testare o impostare i vari livelli logici della porta.

BCF PORTB, …

Questa istruzione mette a zero il bit … ( compreso tra 0 e 7 ) della porta di comunicazione PORTB; una cosa importante da notare è che se il piedino corrispondente a bit scelto è impostato in input all'esterno non si avranno variazioni di livello.

BSF PORTB, …

Questa istruzione è uguale a quella precedente che però setta a uno il bit … della porta di comunicazione.

BTFSC PORTB, …

Questa istruzione legge il bit … all'indirizzo PORTB e se è a livello alto, esegue la prossima istruzione; mentre se è zero, esegue un NOP ( no operation ) che consiste nel stare un fermo un ciclo saltando così l'istruzione che seguiva.

BTFSS PORTB, …

Questa istruzione è uguale a quella precedente con l'unica differenza che esegue l'struzione che segue, se il livello è zero e viceversa salta l'istruzione se il bit è a uno.

**Esempio1:** Questa riga di programma possono essere utilizzate quando si vuole usare un pin del PIC come interruttore, queste istruzioni evitano che il programma sia eseguito più volte mentre l'interruttore è premuto; infatti è facile che il programma sia eseguito più volte in un secondo, quindi se noi vogliamo incrementare di uno una variabile, ma teniamo premuto per troppo tempo l'interruttore, la variabile si incrementa tre o quattro volte.

```
Ciclo                ‘
BTFSC PORTB, 1       ‘ Il bit uno della PORTB viene testato se è a uno viene incrementata
INC Count, 1         ‘ la variabile Count.
Ciclo1               ‘ A questo punto il programma non esce dal ciclo fino a quando il
BTFSS PORTB, 1       ‘ il bit 1 della PORTB non sarà a zero.
goto Ciclo1          ‘
```

**Esempio2:** Questo serie d'istruzioni accende e spegne un led che si trova collegato al bit 0 della PORTB.

```
Ciclo
bcf  PORTB, 0
call Delay
bsf PORTB, 0
call Delay
goto  Ciclo
```

Con questa programmino s'inserisce il concetto subroutine, queste non sono altro che parti di programma che dovendo essere ripetute, per comodità possono essere richiamate con l'istruzione call. Nel nostro caso call richiama un ritardo questo ritardo ci serve per vedere accendere e spegnere il led, in questo ritardo il PIC non fa altro che fare operazioni inutili.

**Esempio3:** Questo programma è un esempio di ritardo, composto da due cicli concatenati.

```
Ritardo
        decfsz Count, 1
       goto  Ritado
decfsz  Count1,  1
goto Ritardo
```

In questo ciclo concatenato la variabile Count è decrementata fino ad arrivare a zero, arrivata a zero il programma esce dal ciclo e decrementa Count1 ritornando poi a Ritardo. In questo modo Count viene decrementata per 256 volte.

**Esempio4:** Questo programma scrive e legge un dato nell'EEprom interna del PIC.

```
Write EEprom
        BCF STATUS,RP0
        MOVLW   1
        MOVWF   EEADR
        MOVLW  0EDh
        MOVWF   EEDATA
        BSF     STATUS, RP0
        BSF     EECON1,WREN
        MOVLW   055h
        MOVWF   EECON2
        MOVLW   0AAh
        MOVWF   EECON2
        BSF     EECON1,WR
        BCF     STATUS,RP0
return
```

Per leggere e scrivere nella EEprom del PIC bisogna lavorare con i due registri EEADR e EEDATA, in questi registri vengono memorizzati l'indirizzo in cui andare a memorizzare il dato e il dato da registrare.
Come si nota nella subroutine Write EEprom nel registro EEADR viene salvato l'indirizzo e nel registro EEDATA viene memorizzato il dato; dopo di che si eseguono una serie d'istruzioni che non vanno modificate; in questo il dato verrà memorizzato.

```
Read EEprom
        BCF STATUS, RP0
        MOVLW 1
        MOVWF EEADR
        BSF STATUS, RP0
        BSF EECON1, RD
        BCF STATUS, RP0
        MOVWF EEDATA
return
```

In questa subroutine invece il dato è letto dalla memoria interna, come prima nel registro EEADR viene memorizzati l'indirizzo in cui andare a leggere; a questo punto come prima si eseguono una serie d'istruzioni da non modificare. Ora il dato viene memorizzato nel registro EEDATA pronto per essere utilizzato.

**Attenzione:** per leggere e scrivere nella EEprom bisogna eseguire una serie d'istruzioni precisa, se non si esegue ciò il dato non verrà letto o salvato.