Capitolo X del Testo
Ettore Panella – Giuseppe Spalierno
Corso di Elettronica 3
Edizioni Cupido

MICROCONTROLLORE PIC 16F84

1. Generalità

I microcontrollori sono dei circuiti integrati programmabili a larghissima scala di integrazione strutturalmente analoghi ai microprocessori. Tali dispositivi sono utilizzati nell'automazione industriale di piccola e media complessità come ad esempio nei sistemi di controllo degli elettrodomestici, nei dispositivi mixer audio e video, nei decoder per la TV satellitare, negli impianti di antifurto nonché per il comando di display a diodi LED e LCD, tastiere, relè ecc.

La programmabilità e il basso costo rendono questi dispositivi competitivi rispetto ad analoghi sistemi in logica cablata. La funzione svolta dal circuito può essere modificata variando semplicemente il software di gestione che si deve memorizzare nel microcontrollore. In commercio sono distribuiti da diversi produttori che forniscono vari modelli con caratteristiche più o meno spinte. Alcuni modelli posseggono al loro interno un convertitore analogico-digitale altri sono dotati anche di un generatore di segnale PWM (Pulse Width Modulation) per il controllo della potenza di un carico e così via.

Le società più importanti che producono microcontrollori sono.

- SGS con la famosa serie ST6TXX
- Siemens con la serie di microcontrollori 80C165/6/7
- Motorola con la serie 68HCXX
- Hitachi con la serie H8/300H
- Microchip

In questo capitolo ci occuperemo del microcontrollore PIC16F84 della Microchip. Tale componente risulta interessante poiché unisce varie caratteristiche come l'ottimo rapporto costo/prestazioni, una facile programmazione e non ultima una vasta documentazione disponibile su Internet. A tale proposito si consiglia di consultare i seguenti siti Internet da cui è possibile scaricare vari software ed esempi di circuiti applicativi per la programmazione e l'utilizzo del PIC.

- http://www.microchip.com/
- http://www.tanzilli.com/
- http://www.picpoint.com/
- http://www.ic-prog.com/

Per programmare il microcontrollore si deve disporre di un opportuno hardware da collegare ad un Personal Computer o alla porta seriale o a quella parallela. Il programma che si intende eseguire è scritto in *linguaggio assembler* e successivamente trasferito nella memoria del microcontrollore.

Il dispositivo così programmato viene tolto dal circuito programmatore e inserito nella scheda elettronica che deve comandare.

2. Caratteristiche del PIC 16F84

E' un microcontrollore a 8 bit della Microchip che si presenta in un contenitore a 18 piedini. In fig.1 si riporta la piedinatura dell'integrato e lo schema a blocchi per la descrizione delle caratteristiche di funzionamento del dispositivo.

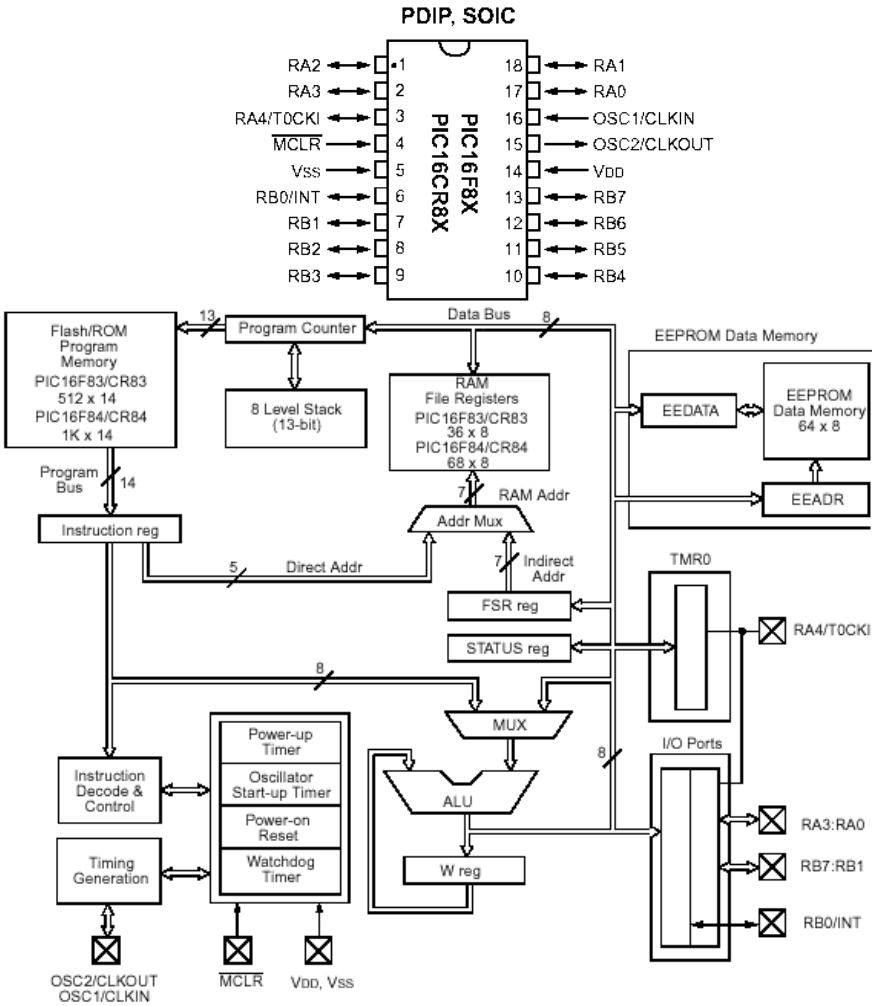


Fig. 1 Schema a blocchi funzionale e piedinatura del PIC 16F84

L'integrato richiede una tensione di alimentazione compresa tra 2V e 6V (valore tipico di 5V) da applicare ai terminali indicati con V_{DD} (terminale positivo) e Vss (terminale di massa GND). I pin OSC1/CLKIN e OSC2/CLKOUT devono essere collegati ad un circuito oscillatore, ad esempio un quarzo o un a rete RC, per generare il clock di sistema. La frequenza massima di tale segnale è di 10 MHz (20 MHz per il modello 16F84A). Nelle applicazioni che seguono si farà uso di un circuito oscillante che utilizza un quarzo di 4MHz.

L'unità centrale di elaborazione CPU è di tipo RISC (Reduced Instruction Set Computer) con un set di istruzioni ridotto a sole 35 istruzioni in grado di elaborare dati a 8 bit con già inclusi i metodi di indirizzamento immediato, diretto ed indiretto. Il modo di operare sfrutta la tecnica denominata *pipeline a 2 stati* per cui mentre viene eseguita un'istruzione, contemporaneamente viene caricata l'istruzione successiva nel registro delle istruzioni per la decodifica. Ciò si traduce in una riduzione del tempo di esecuzione di ciascuna istruzione.

In particolare, se si escludono le istruzioni di salto che richiedono due cicli macchina, il tempo di esecuzione di una istruzione ha una durata pari ad un ciclo macchina che corrisponde a 4 impulsi di clock. Pertanto, se la frequenza di clock è f_{CK} = 4MHz il tempo richiesto per eseguire un'istruzione vale $4T_{CK}$ = 1μs.

La CPU (Central Processing Unit) è in diretta connessione con **l'unità aritmetica e logica ALU** per svolgere tutte le operazioni imposte dal programma.

Il PIC si avvale di 15 registri speciali. In particolare i**l registro accumulatore W** è un registro interno privilegiato ampiamente utilizzato dal microcontrollore per svolgere moltissime operazioni come somma, sottrazione, memorizzazione temporanea dei dati, ecc.

Un registro fondamentale per il funzionamento del microcontrollore è il **Program Counter PC.** E' un registro contatore che si incrementa automaticamente durante l'esecuzione di un programma in modo da contenere l'indirizzo della successiva istruzione che deve essere eseguita. Solo nel caso di istruzioni di salto il suo valore viene modificato in modo da contenere l'indirizzo dell'istruzione relativa al salto. Dopo un comando di RESET il registro PC è azzerato e punta alla locazione 0000H della memoria di programma che deve, pertanto, contenere la prima istruzione che deve essere eseguita. Il RESET del PIC si realizza ad hardware portando la linea MCLR (pin 4) al livello basso. Tale linea deve essere tenuta, normalmente, al livello alto V_{DD}.

L'integrato dispone di **13 linee bidirezionali di I/O**; 5 indicate con RA0…RA4 costituiscono la PORTA A e 8 con RB0…..RB7 la PORTA B. Ciascuna linea può essere programmata come Input o come Output ed è in grado di assorbire (sink current) fino a 25mA ed erogarne (source current) fino a 20mA.

3. Organizzazione della memoria

La memoria del PIC è logicamente divisa in due blocchi**:** *memoria di programma* e *memoria dei dati*. Ciascun blocco ha un proprio bus per il trasferimento dati e in tal modo si aumenta la velocità operativa del sistema.

La **memoria di programma** è una EEPROM (Electrically Erasable PROM) con una capacità di 1024 locazioni di memoria (word), ciascuna a 14, bit cancellabili e programmabili elettricamente più di 1000 volte. In fig. 2 si riporta la mappa della memoria di programma.

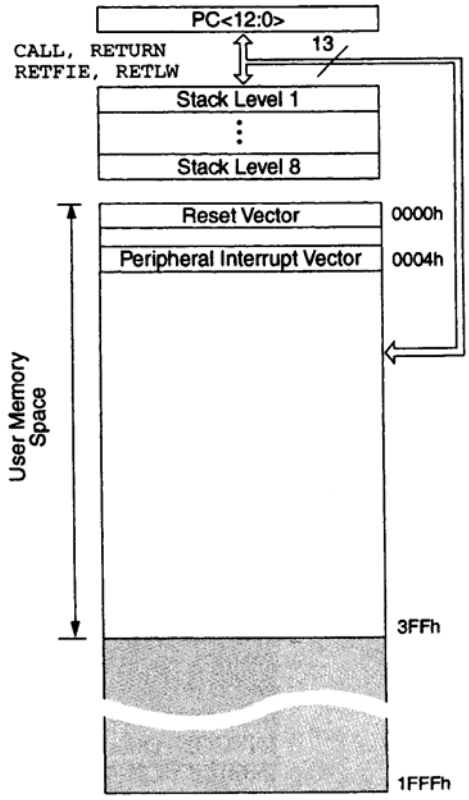


Fig.2 Mappa della memoria di programma del PIC16F84.

Il Program Counter PC è a 13 bit e quindi ha una capacità di indirizzare fino a 8Kword ma negli attuali PIC della serie 16F8X è fisicamente indirizzabile solo 1Kword. Pertanto, il programma più esteso non può superare le 1024 istruzioni. Il byte basso del Program Counter (bit 0-7) è indicato con PCL mentre i 5 bit di ordine superiore (bit 8-12) sono caricati in un registro interno indicato con

PCLATH. Questa particolare divisione del Program Counter consente una migliore gestione delle istruzioni di salto e delle chiamate alle subroutine. Gli indirizzi

riservati alla memoria di programma, espressi in formato esadecimale, vanno da 0000H a 03FFH. In particolare 8 locazioni di memoria, ciascuna 13 bit, sono direttamente connesse con il Program Counter per costituire **l'AREA di STACK** del microcontrollore. Lo STACK presenta una struttura a registro LIFO (Last In First Out) per cui l'ultimo elemento inserito è il primo ad essere prelevato.

Lo STACK è utilizzato dal microcontrollore nelle istruzioni di chiamata (CALL) a subroutine o nella gestione degli interruput. In questi casi il microcontrollore, prima di interrompere il normale flusso del programma, memorizza nello STACK il contenuto del Program Counter e salta ad eseguire la subroutine di servizio. Al termine di tale subroutine viene ripristinato nel Program Counter il valore memorizzato nello STACK e quindi si riprende il programma principale. Se ad esempio nel programma si trova una istruzione CALL, il microcontrollore inserisce nello STACK il valore corrente del Program Counter e in esso inserisce l'indirizzo a cui saltare. Al termine della subroutine una istruzione RETLW (ritorna da subroutine) riporta nel Program Counter il valore presente nello STACK. Essendo lo STACK ad 8 livelli sono possibili al più 8 annidamenti (nesting) di subroutine. Si fa notare che il PIC non prevede istruzioni sullo STACK come quelle di Push e Pop tipiche del microprocessori. La fig.2 mostra che la locazione di indirizzo 0000H è riservata al RESET del microcontrollore, mentre nella locazione 0004H deve essere posto l'indirizzo della subroutine per la gestione di un eventuale interrupt. Per interrupt si intende la possibilità di interrompere il normale flusso del programma per eseguire particolari sottoprogrammi. L'interrupt può essere generato da eventi esterni al microcontrollore o interni. Il cambiamento di stato logico della linea RB0/INT è un esempio di interruput esterno mentre un fine conteggio del TIMER interno TMR0 può essere gestito come interrupt interno. Il PIC può operare con quattro diversi tipi di interrupt della cui gestione si parlerà più diffusamente nel seguito del capitolo.

Il PIC 16F84 dispone di due aree per la **memoria dati** una di tipo EEPROM e l'altra di tipo RAM.

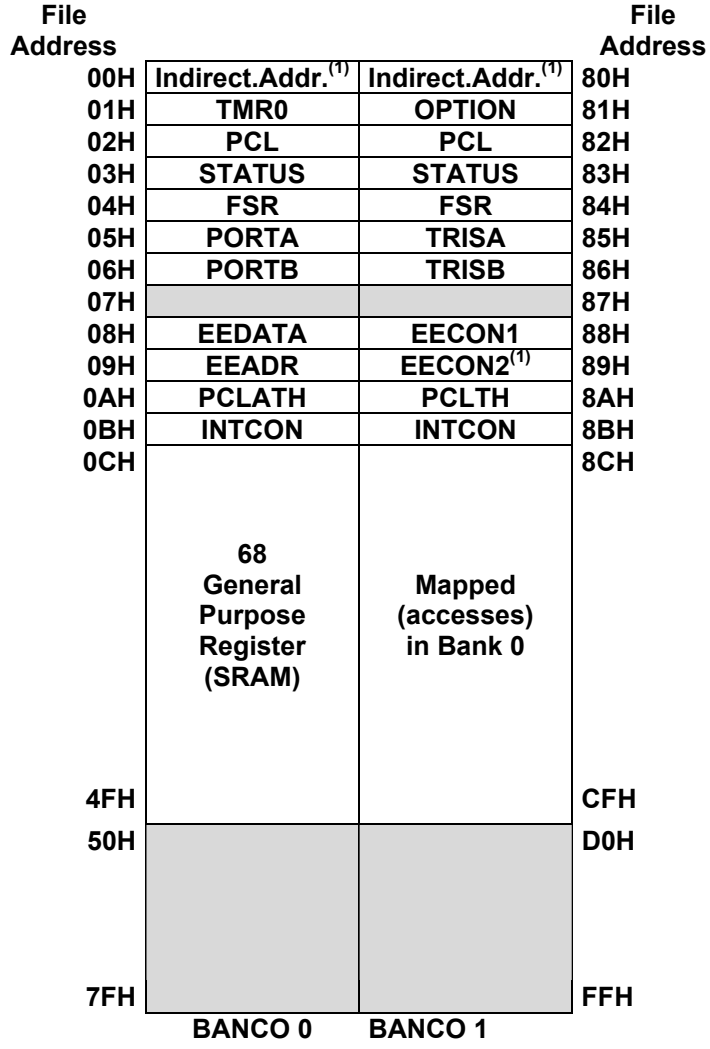
La memoria dati EEPROM ha una capacità di 64 byte ed è impiegata per la memorizzazione di dati che si desidera che non vengano persi in caso di mancanza di alimentazione. Ciascuna locazione della memoria EEPROM dati può essere letta o scritta durante il normale funzionamento del PIC senza necessità di utilizzare il circuito di programmazione del microcontrollore. La memorizzazione di dati nella EEPROM è utile in tutte quelle applicazioni, come antifurti, chiavi elettroniche, che necessitano della memorizzazione dei dati numerici anche se viene a mancare l'alimentazione all'integrato. Allorquando si ripristina l'alimentazione sarà possibile far ripartire il programma e leggere i dati dalla EEPROM. Nel seguito del capitolo si descriverà il software per la gestione della EEPROM dati.

La memoria dati di tipo RAM è nota come **File Register** ed è utilizzata durante l'esecuzione di un programma per la memorizzazione temporanea dei dati e per l'esecuzione delle istruzioni. Il File Register è diviso in due parti denominate *Banco 0* e *Banco 1* e ciascuna locazione di memoria è denominata *Registro*. In fig.

3 si mostra la mappa della memoria dati RAM. Ciascun banco è costituito da 128 byte (7FH). Le prime 12 locazioni di ciascun banco sono riservate ai registri

speciali SFR (Special Function Register). Le 68 locazioni di memoria del banco 0 che vanno da 0CH a 4FH sono denominate GPR (General Purpose Register) e individuano i registri generali del PIC.

Si può accedere all'intera area di memoria utilizzando l'indirizzamento assoluto o quello indiretto tramite il registro FSR (File Select Register).



Unimplemeted data memory location; read as '0'
Note (1) Not a physical register

Fig.3 Mappa della memoria RAM Dati.

4. Il set di istruzioni

In questo paragrafo saranno analizzate le istruzioni mnemoniche in *linguaggio assembly* che il programmatore deve utilizzare per scrivere il programma di gestione del microcontrollore. Un programma assembler può essere scritto utilizzando un qualunque editor di testi purché sia salvato con l'estensione **.ASM**. In pratica è più conveniente utilizzare dei programmi dedicati come MPLAB scaricabile gratuitamente dal sito della Microchip.

Il programma MPLAB contiene oltre all'editor anche l'assemblatore e consente inoltre, il debug e la simulazione del programma. Il set di istruzioni utilizza un formato a 14 bit diviso in un *codice operativo* che individua il tipo d'istruzione e da un campo *operando* che indica il registro o il dato su cui operare. Il PIC divide le istruzioni in tre gruppi:

- Orientate al byte. *Byte Oriented Operation*
- Orientate al bit. *Bit Oriented Operation*
- Di controllo e su dati letterali o numerici. *Literal and Control Operation*

Nel seguito si impiegheranno le seguenti convenzioni.

Campo	Descrizione
f	Indirizzo di un registro nel File Register
W	Registro accumulatore
b	Ordine, da 0 a 7, del bit nel registro
d	Destinazione. Il risultato di una operazione è posto: nell'accumulatore se d = 0; nel File Register f se d = 1 (default)
K	Campo letterale, dato costante o label
X	Valore "non importa" (0 o 1)

L'esecuzione di una istruzione può modificare lo stato logico di alcuni bit del Registro di Stato (STATUS) posto all'indirizzo 03H del File Register.

In particolare sono interessati i bit:

-C Bit di Carry: C=1 se si verifica un riporto dal bit più significativo
-Z Bit di Zero: Z=1 se il risultato dell'operazione è zero
-DC Bit Digit Carry: DC=1 se si verifica un riporto dal 4°bit al 5°

Si riporta il formato generale delle istruzioni e i relativi codici menmonici.

Byte Oriented Operation

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Codice Operativo							d	File Register f					

Codice	Descrizione	Bit di stato
ADDWF f,d	Somma il valore di W al valore del registro f.	C, Z, DC
ANDWF f,d	Esegue l'AND tra W e f bit a bit.	Z
CLRF f	Azzera il registro f.	Z
CLRW	Azzera l'accumulatore.	Z
COMF f,d	Complementa bit a bit il registro f.	Z
DECF f,d	Decrementa di 1 il registro f.	Z
DECFSZ f,d	Decrementa di 1 il registro f. Se il contenuto di f diventa 0 salta l'istruzione successiva.	
INCF f,d	Incrementa di 1 il registro f.	Z
INCFSZ f,d	Incrementa di 1 il registro f. Se il contenuto di f diventa 0 salta l'istruzione successiva.	
IORWF f,d	Esegue l'OR tra W e f bit a bit.	Z
MOVF f,d	Legge il valore di f e lo copia in W se d=0 o in f stesso se d=1.	Z
MOVWF f	Legge il valore di W e lo copia in f.	
NOP	Non esegue nessuna operazione. Produce un ciclo a vuoto.	
RLF f,d	Ruota a sinistra di una posizione attraverso il carry il registro f.	C
RRF f,d	Ruota a destra di una posizione attraverso il carry il registro f.	C
SUBWF f,d	Sottrae W da f.	C, Z, DC
SWAPF f,d	Scambia I due nibble del registro.	
XORF f,d	Esegue l'XOR tra W e f bit a bit.	Z

Bit Oriented Operation

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cod. Operativo				b			File Register						

Codice	Descrizione	Bit di stato
BCF f,b	Pone a 0 il bit di ordine b del registro f	
BSF f,b	Pone a 1 il bit di ordine b del registro f	
BTFSC f,b	Testa il bit di ordine b del registro f. Se vale 0 salta l'istruzione immediatamente successiva	
BTFSS f,b	Testa il bit di ordine b del registro f. Se vale 1 salta l'istruzione immediatamente successiva	

Literal and Control Operation

Generale

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Codice Operativo						Campo K							

Istruzioni GOTO e CALL

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cod. Operativo			Campo K										

Codice	Descrizione	Bit di stato
ADDLW K	Somma all'accumulatore W il valore K	C, Z, DC
ANDLW K	Esegue l'AND tra l'accumulatore W e K	Z
CALL K	Effettua una chiamata alla subroutine K	
CLRWDT	Azzera il registro Watchdog timer	TO, PD
GOTO K	Salta alla label K	
IORLW K	Esegue l'OR tra l'accumulatore W e K	Z
MOVLW K	Pone nell'accumulatore il valore k	
RETFIE	Ritorna al programma sospeso dopo un interrupt	
RETLW K	Ritorna al programma sospeso e copia in W il valore K	
RETURN	Ritorna al programma sospeso da una subroutine	
SLEEP	Pone il PIC in modalità basso consumo (standby mode)	TO, PD
SUBLW K	Sottrae il valore K dall'accumulatore W	C, Z, DC
XORLW K	Esegue l'XOR tra l'accumulatore W e K	Z

5. I Registri speciali del PIC

In questo paragrafo si darà una descrizione generale del File Registrer del PIC facendo riferimento alla mappa di memoria di fig.3. La conoscenza dello stato logico e della funzione assegnata a ciascun bit risulta fondamentale per la programmazione del componente. Anche se alcune definizioni sulle funzioni svolte dai registri possono al momento non essere chiare, se ne comprenderà meglio il significato durate la scrittura dei programmi.

5.1 Registro Status

Il Registro di Stato **(**STATUS**)** si trova all'indirizzo 03H del banco 0 e all'indirizzo 83H del banco 1. Contiene 8 flag che indicano lo stato logico della **ALU** , quello del PIC al RESET e ulteriori flag che consentono l'indirizzamento al banco 0 o al banco 1. In particolare si ha:

Bit 7	6	5	4	3	2	1	0
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

Bit 7 – 6: IRP e RP1 non sono supportati dai modelli PIC 16F8X e devono essere tenuti al livello logico basso;

Bit 5: RP0 seleziona il banco del File Register RP0 = 0 Banco 0 (00H – 7FH); RP0 = 1 Banco1 (80H – FFH):

Bit 4: $\overline{TO}$ è il bit di time out : $\overline{TO}$ = 0 se il Watchdog va in time out. $\overline{TO}$ = 1 dopo che viene fornita alimentazione all'integrato o se viene eseguita l'istruzione SLEEP;

Bit 3: $\overline{PD}$ è il bit di Power Down: $\overline{PD}$ = 0 dopo che viene eseguita l'istruzione SLEEP. $\overline{PD}$ = 1 dopo che viene fornita alimentazione all'integrato o dopo che viene eseguita l'istruzione CLRWDT;

Bit 2: Z. Zero bit. Indica se il risultato di una operazione è zero (Z = 1) o non è zero (Z = 0);

Bit 1: DC. Digit Carry. Indica se, dopo un'operazione, c'è un riporto dal 4° al 5° bit (DC = 1). In caso contrario DC = 0;

Bit 0: C. Bit di Carry**.** Indica se, dopo un'operazione, c'è riporto dal bit più significativo (C = 1**)** oppure non c'è riporto (C = 0).

Tutti i bit del registro di stato sono di lettura/scrittura ad eccezione dei bit $\overline{TO}$ e $\overline{PD}$ che sono di sola lettura.

5.2 Registro Option

Il Registro Option (OPTION) si trova all'indirizzo 81H del banco1 ed è utilizzato per configurare il Prescaler del registro di temporizzazione TMR0, il Watchdog e l'interrupt esterno.

Bit 7	6	5	4	3	2	1	0
$\overline{RBPU}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Bit 7 : $\overline{\text{RBPU}}$ Abilita ($\overline{\text{RBPU}}$ = 0) oppure disabilita ($\overline{\text{RBPU}}$ = 1) le resistenze interne di pull-up della porta B;

Bit 6: INTDG. Interrupt Edge. Seleziona il fronte del segnale di interruput al pin RB0/INT. INTDG = 0 fronte di discesa; INTDG = 1 fronte di salita;

Bit 5: TOCS**.** Seleziona la sorgente del clock per il timer TMR0. TOCS = 0 clock interno; TOCS = 1 clock esterno su RA4/TOCK1;

Bit 4: TOSE. Seleziona il fronte del segnale di clock esterno RA4/TOCK1. TOSE = 0 fronte di salita; TOSE = 1 fronte di discesa;

Bit 3: PSA. Prescaler Assignment. PSA = 0 pescaler assegnato a TMR0; PSA = 1 prescaler assegnato al sistema Whatchdog WDT;

Bit 2-1-0: Seleziona il fattore di divisione della frequenza secondo la seguente tabella 1.

Tabella 1

PS2-PS1-PS0	TMR0	WDT
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

5.3 Registro contatore TMR0

Il Registro Contatore TMR0 è un registro che si autoincrementa con una cadenza programmabile. Esso è allocato all'indirizzo 01H del File Register.

Quando incrementandosi raggiunge il valore massimo 255 (FFH in esadecimale) si azzera e riprende il conteggio. Inizialmente può essere caricato anche con un valore diverso da zero. La rapidità del conteggio dipende dalla frequenza di clock con cui lavora il PIC o dalla frequenza di un segnale esterno. In fig.4 si riporta lo schema funzionale del sistema di gestione del segnale di clock per il TMR0.

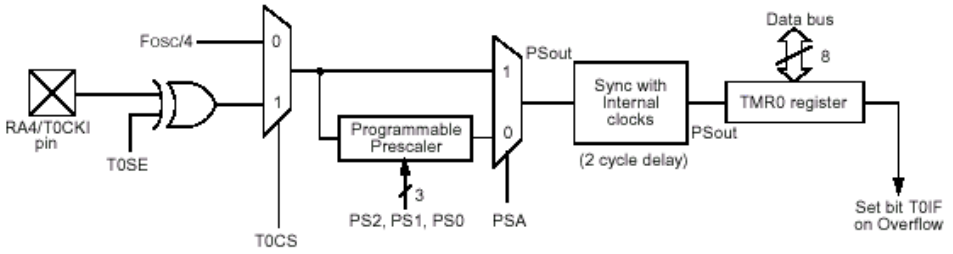


Fig.4 Sistema di gestione per il comando del TMR0.

Il segnale di comando del contatore TMR0 indicato con PSout può provenire o dal segnale esterno RA4/TOK1 (pin3) oppure dal circuito oscillatore del PIC. In questo caso la frequenza del clock dell'oscillatore è divisa per 4 ed è indicata con Fosc/4. Se il segnale di comando proviene da TOK1 è possibile attivare il conteggio sul fronte di salita (TOSE = 0) o su quello di discesa (TOSE = 1). Ciò deriva dalla proprietà della porta XOR per cui:

$$TOK1 \oplus 0 = TOK1; \qquad TOK1 \oplus 1 = \overline{TOK1}$$

I bit TOCS e PSA comandano due multiplexer che selezionano diverse sorgenti di segnale di clock. I bit PS2, PS1, PS0 pilotano un circuito di *prescaler* che divide la frequenza di clock secondo la precedente tabella 1.

Ad esempio, ponendo TOCS = 0 e PSA = 1 il clock del timer TMR0 è pari a Fosc/4, mentre se TOCS = 1 e PSA = 1 la sorgente del segnale di conteggio proviene da TOK1,

5.4 Registro INTCON

Il registro INTCON è un registro che si trova all'indirizzo 0BH. E' utilizzato per la gestione degli interrupt. Se il bit è 0 la funzione associata è disabilitata.

Bit 7	6	5	4	3	2	1	0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Bit 7 : GIE. Global Interrupt Enable. GIE = 1 abilita tutti gli interrupt;

Bit 6: EEIE. EEPROM Write Complet Interruput. EEIE = 1 abilitata l'interruput alla fine della scrittura in una locazione EEPROM dati;

Bit 5: TOIE. TMR0 Overfow Interrupt. TOIE = 1 abilita l'interrupt alla fine del conteggio del timer TMR0;

Bit 4: INTE. RB0/INT Interrupt Enable. INTE = 1 abilita l'interrupt sul cambio di stato logico della linea RB0/INT (pin 6);

Bit 3: RBIE. RB Port Change Interruput. RBIE = 1 abilita l'interrupt sul cambio di stato logico su una delle linee da RB4 a RB7;

Bit 2: TOIF. TMR0 Overflow Interrupt Flag. TOIF diventa 1 se l'interrupt è stato generato dalla fine del conteggio del timer TMR0;

Bit 1: INTF. RB0/INT Interrupt Flag. INTF diventa 1 se l'interrupt è stato generato dal cambio di stato logico della linea RB0/INT;

Bit 0: RBIF. RB Port Change Interrupt Flag. RBIF diventa 1 se l'interrupt è stato generato da un cambio di stato logico su una delle linee da RB4 a RB7.

Quando viene generato un interrupt il microcontrollore disabilita gli interrupt ponendo GIE = 0. Per poter tornare al programma principale si deve terminare la routine di servizio dell'interrupt con l'istruzione RETFIE che porta GIE = 1.

5.5 Registro EECON1

Il registro EECON1 si trova all'indirizzo 88H. E'un registro di controllo utilizzato per gestire le operazioni di lettura e scrittura nella EEPROM dati.

Bit 7	6	5	4	3	2	1	0
--	--	--	EEIF	WRERR	WREN	WR	RD

Bit 7 - 5 : Non utilizzati nella serie 16F8X

Bit 4: EEIF. EEPROM Interrupt Flag. EEIF = 1 indica che la scrittura nella EEPROM è stata completata;

Bit 3: WRERR. Error Flag. WRERR = 1 indica che l'operazione di scrittura è stata interrotta per una causa esterna come ad esempio un RESET del PIC;

Bit 2: WREN Write Enable. WREN deve essere posto a 1 prima di iniziare una operazione di scrittura. Se WREN = 0 la EEPROM diventa a sola lettura;

Bit 1: WR. Write Control. WR deve essere posto a 1 prima di iniziare un ciclo di scrittura. Al termine del ciclo di scrittura è posto automaticamente a 0.

Bit 0: RD. Read Control. RD deve essere posto a 1 prima di iniziare un ciclo di lettura. Al termine del ciclo di lettura è posto automaticamente a 0.

6. Le porte di I/O

Si è detto che il PIC 16F84 dispone di 13 linee di I/O programmabili individualmente come Input o come Output denominate Porta A (5 linee) e PORTA B (8 linee). La gestione delle porte di I/O è affidata ai registri TRISA (85H) e PORTA (05H) per la porta A e TRISB (86H) e PORTB (06H) per la porta B.

I registri TRISA e TRISB configurano ciascuna linea come Input o come Output. Se un bit del registro TRIS è posto al livello alto, il corrispondente pin della porta è configurato come input. Se, invece, si azzera un bit del registro TRIS il corrispondente pin della porta è configurato come output.

I registri PORTA e PORTB contengono il dato numerico di input o di output associato alla porta di I/O. Su ciascun pin della Porta B è possibile inserire a software una resistenza di pull-up. Ciò si ottiene azzerando il bit $\overline{\text{RBPU}}$ del registro OPTION. Si ricordi, inoltre, che i 4 bit che vanno da RB4 a RB7 possono gestire anche un interrupt abilitato dal bit RBIE del registro INTCON.

Analizziamo le istruzioni fondamentali per la gestione dell'I/O sviluppando un esempio.

Si supponga di voler configurare i 4 bit più significativi della Porta B (da RB7 a RB4) come uscita e quelli meno significativi (da RB3 a RB0) come entrata.

Si utilizzano le seguenti istruzioni:

```
BSF  STATUS, RP0    ;Pone a 1 il bit RP0 (5° bit) del Registro di Stato (indirizzo
                    ;03H) per selezionare il banco 1 del File Register
MOVLW 0FH           ;Carica nell'accumulatore W il numero 0FH (00001111)
MOVWF  TRISB        ;Carica nel registro TRISB (86H del banco 1) il contenuto
                    ;di W
```

In questo modo nel registro TRISB si è caricato il numero binario 00001111
Le tre precedenti istruzioni si potrebbero sostituire con le seguenti due:

```
MOVLW 0FH
TRIS  PORTB
```

In questo caso si utilizza l'istruzione speciale TRIS che automaticamente carica nel registro TRISB il contenuto dell'accumulatore. Tale istruzione è al momento ancora supportata dal PIC per mantenere la compatibilità con software precedenti ma il costruttore ne sconsiglia l'uso poiché si prevede che nelle prossime versioni della serie 16F8X tale istruzione non sarà più presente.

Scrittura

Continuando nell'esempio, supponiamo di voler inviare in uscita alla porta B sui pin RB7 e RB6 il livello 1 e sulle linee RB5 e RB4 il livello 0. Si deve scrivere il seguente codice:

```
BCF  STATUS, RP0    ;Pone a 0 il bit RP0 (5° bit) del Registro di Stato (indirizzo
                    ;03H) per selezionare il banco 0 del File Register
MOVLW  B0H          ;Carica in W il numero B0H (11000000)
MOVWF  PORTB        ;Carica il registro PORTB (06H del banco 0) con W
```

Lettura

Supponiamo di voler leggere lo stato logico delle linee da RB3 a RB0, programmate come input. Si deve scrivere il seguente codice:

```
BCF STATUS, RP0     ;Pone a 0 il bit RP0 del Registro di Stato ( Banco 0)
MOVF PORTB, 0       ;Copia il contenuto del registro PORTB in W
```

Dopo che è stata eseguita quest'ultima istruzione, i 4 bit meno significativi dell'accumulatore W assumeranno uno stato logico coincidente con le 4 linee di I/O che vanno da RB3 a RB0.

E' molto importate osservare che nelle precedenti istruzioni si sono utilizzate delle ***parole chiave*** come **STATUS** per indirizzarsi al registro di stato allocato all'indirizzo 03H del File Register.

Analogamente **RP0** per individuare il valore 05H, ovvero il 5° bit del registro di stato, mentre TRISB corrisponde a 86H e PORTB a 06H. Queste parole chiave sono tradotte automaticamente nel corrispondente valore numerico dal *programma assemblatore* grazie alla direttiva INCLUDE "P16F84.INC". L'uso delle parole chiave al posto del corrispondente valore numerico non è obbligatorio ma, ovviamente, migliora la leggibilità del programma.

7. La EEPROM DATI

Si è detto che il PIC possiede una memoria EEPROM di 64 byte per la memorizzazione di valori numerici che devono essere conservati anche se l'integrato non è alimentato. Il costruttore garantisce che sulla memoria EEPROM DATI si possono effettuare fino a 10 milioni di scritture/letture e che i dati possono essere mantenuti in memoria per oltre 40 anni. In questo paragrafo si vogliono analizzare le procedure software relative alla scrittura e alla lettura nella EEPROM DATI.

Il registro di controllo EECON1 insieme all'altro registro EECON2, gestito dal PIC, consentono di controllare le operazioni di scrittura e di lettura nella EEPROM DATI. Tali operazioni richiedono anche l'utilizzo di due registri speciali per la memorizzazione dei dati:

Registro EEADR, allocato all'indirizzo 09H del File Register, contiene l'indirizzo di una delle 64 locazioni di memoria EEPROM su cui operare;

Registro EEDATA, allocato all'indirizzo 08H del File Register, è impiegato per inviare alla EEPROM DATI il byte da scrivere oppure per ricevere un byte in una operazione di lettura dalla EEPROM DATI.

Si riportano le istruzioni necessarie per scrivere e leggere nella EEPROM DATI.

Scrittura nella EEPROM DATI

Supponiamo di voler scrivere nella locazione 00H della EEPROM DATI il valore 0AH.

Si deve scrivere il seguente codice:

```
MOVLW   00H          ;Pone 00H nell'accumulatore
MOVWF   EEDR         ;Carica nel registro EEDR il valore 00
MOVLW  0AH           ;Pone 0AH nell'accumulatore
MOVWF  EEDATA        ;Carica nel registro EEDATA il valore 0AH
BSF  STATUS, RP0     ; Seleziona il banco 1
BSF  EECON1, WREN    ;Abilita la EEPROM alla scrittura
```

```
;Le 4 istruzioni che seguono sono suggerite dal costruttore.
;Utilizzano il registro EECON2 e servono per eliminare errori in scrittura.

MOVLW  55H          ;Poni 55H nell'accumulatore
MOVWF  EECON2       ;Scrive 55H nel registro EECON2
MOVLW  AAH          ;Poni AAH nell'accumulatore
MOVWF  EECON2       ;Scrive AAH nel registro EECON2
;Avvio della scrittura in EEPROM
BSF  EECON1 ,WR     ;Inizio della scrittura
;L'Hardware impiega un certo tempo per scrivere in EEPROM
;Si inserisce un loop di attesa che termina quando il flag WR = 0.
 ATTENDI:
BTFSC  EECON1, WR    ;Testa il bit WR e salta l'istruzione successiva se 0
GOTO  ATTENDI        ;Resta in loop se WR =1.
Prosecuzione del programma……..
```

Lettura della EEPROM DATI

Supponiamo di voler leggere il contenuto della locazione 00H della EEPROM DATI e trasferire il contenuto nell'accumulatore.

Si deve scrivere il seguente codice:

```
BCF  STATUS, RP0    ;Seleziona il banco 0.
MOVLW  00H          ;Pone 00H nell'accumulatore.
MOVWF  EEADR        ;Carica nel registro EEADR il valore 00H.
BSF  STATUS, RP0    ;Seleziona il banco 1.
BSF  EECON1, RD     ;Pone a 1 il bit RD. Inizio lettura.
BCF  STATUS, RP0    ;Seleziona il banco 0.
MOVF  EEDATA, W     ;Trasferisce il contenuto di EEDATA
                    nell'accumulatore.
```

L'accumulatore W contiene il dato della locazione 00H della EEPROM DATI

8. Il Power Down Mode

In alcune situazioni operative, come ad esempio nei sistemi di allarme, può essere utile tenere il microcontrollore in uno stato di basso consumo (Standby Mode). Questa caratterista di funzionamento è gestita dall'istruzione SLEEP. Quando il programma incontra tale istruzione blocca l'esecuzione del programma in corso e si porta nello stato di basso consumo con tutti i circuiti interni che assorbono il minimo di corrente; circa mille volte meno di quella di normale funzionamento.

Per riprendere il normale ciclo operativo si possono attivare i seguenti eventi:

- Portare al livello basso la linea MCLR (pin 4). Si realizza un RESET del PIC;
- Time out del timer Watchdog WDT, se abilitato;

- Verificarsi di un interrupt dal pin RB0/INT o dalle linee RB4-RB7 oppure se si completa un ciclo di scrittura nella memoria EEPROM Dati.

Nel primo caso il programma riparte dalla locazione 0000H, negli altri riprende la normale esecuzione.

9. Temporizzatore Watchdog WDT

Il timer Watchdog, letteralmente "cane da guardia", è un oscillatore interno al PIC che non richiede componenti esterni. E' indipendente dal sistema di clock che si applica al microcontrollore per il normale funzionamento. E' utilizzato per rilevare eventuali stati di blocco del sistema e consentire il RESET del microcontrollore. Ad esempio, se il programma entra in un ciclo senza uscita, il WDT, trascorso un tempo prefissato dal programmatore, attiva un RESET del PIC. Se il dispositivo è in SLEEP mode, un time out WDT "risveglia" il PIC che riprende l'esecuzione del programma interrotto. Il sistema WDT può essere permanentemente disattivato mediante l'istruzione CLRWDT (Clear WDT) che azzera il timer WDT.

Se invece il sistema WDT è abilitato è necessario utilizzare ciclicamente la precedente istruzione prima che sia trascorso il tempo di programmazione del WDT, altrimenti il microcontrollore intenderà che il PIC è bloccato e quindi attiverà il RESET del sistema.

Il periodo minimo di time out del WDT è di circa 18ms. Tale periodo può essere incrementato fino a circa 2.3sec (rapporto 1:128) programmando i bit PS0**,** PS1 e PS2 del Prescaler del WDT nel registro OPTION, come riportato nella precedente tabella 1. I tempi di programmazione non sono rigorosi e dipendono dalla temperatura e dalla tensione di alimentazione.

Per rendere attivo il sistema WDT nel microcontrollore è necessario selezionare a software una opportuno flag durante la fase di programmazione dell'integrato. La procedura di attivazione dipende dal tipo di programmatore che si utilizza.

10. La programmazione del microcontrollore

Per poter utilizzare un microcontrollore come il PIC 16F84 si deve:

- Scrivere il programma in linguaggio Assembler;

- Compilare il programma per ottenere il file eseguibile
- Programmare il PIC

Le diverse fasi operative possono essere svolte da un solo software o da software distinti. La Microchip mette a disposizione il programma MPLAB per Windows, scaricabile gratuitamente da Internet, che realizza tutte le fasi di programmazione (www.microchip.com).

In questo caso si deve disporre di un opportuno hardware per la programmazione del PIC come il PICSTART Plus o il PROMATE II distribuiti dalla stessa Microchip.

Nel seguito si descriveranno le operazioni fondamentali per scrivere il programma in linguaggio assembler e generare il file eseguibile utilizzando MPLAB.

Per la programmazione del PIC sarà descritto un semplice circuito da collegare alla porta seriale del Computer. Come software per la programmazione dell'integrato si descriverà il **programma IC-prog** scaricabile gratuitamente dal sito **www.ic-prog.com.**

Il programma in assembler si può scrivere anche con un normale editor di testi purché il file sia salvato con il formato **nome.asm**.

Il file **nome.asm** è detto *file sorgente* e deve, in tutti i casi, terminare con l'istruzione END. Il compilatore assembler carica tale file insieme alla libreria P16F84.INC che contiene le informazioni necessarie per la corretta compilazione del programma. Al termine della compilazione vengono generati i file:

.HEX; Contiene il programma tradotto in un formato idoneo per essere trasferito nella memoria EEPROM di programma del PIC.

.LST è un file di testo che contiene la traduzione dei codici operativi delle istruzioni del programma.

.ERR contiene la lista degli eventuali errori.

Per garantire un certo ordine nella gestione dei diversi file è consigliabile creare una cartella entro cui inserire tutti i file generati dall'assemblatore.

Prima di analizzare i software di programmazione è opportuno tener presente che quando si scrive un programma in linguaggio assembly oltre alle istruzioni relative al programma vero e proprio che si intende eseguire è necessario inserire nel programma sorgente delle **direttive** che consentono all'assemblatore di generare correttamente il file eseguibile. Alcune direttive utilizzate sono:

- **PROCESSOR 16F84:** Specifica il tipo di microcontrollore
- **ORG 00H:** Specifica la locazione da cui parte il programma. Per il PIC16F84 il Reset è all'indirizzo 00H.
- **RADIX XXX:** Definisce il formato numerico.Per default è l'esadecimale (HEX). Il formato può essere decimale (DEC) o ottale (OCT) .
- **INCLUDE "P16F84.INC":** Consente di associare al file sorgente una libreria per la conversione dei codici mnemonici delle istruzioni.

- **EQU:** Consente di assegnare ad una label un valore numerico. Ad esempio: Pippo EQU 0CH indica che nel programma la label Pippo equivale al numero esadecimale 0C.

- **RES XX:** Indica al compilatore il numero di byte riservati ad una label. Ad esempio: Conta RES 2 indica che alla label Conta sono riservati 2 byte.
- **MACRO:** Consente di associare una label ad un'insieme di istruzioni.

E' sostanzialmente equivalente alla chiamata (CALL) di una subroutine, anche se quest'ultima procedura è da preferire poiché consente di gestire meglio lo spazio di memoria. Si riporta un esempio di una semplice MACRO costituita da due sole istruzioni:

```
IMPOSTA  MACRO
BCF      PORTB,  0      ;Pone a 0 la linea RB0
BSF      PORTB,1        ;Pone a 1 la linea RB1
ENDM                    ; Fine Macro
```

Ogni volta che durante lo svolgimento del programma si inserisce la label IMPOSTA sarà richiamata la MACRO che svolge la funzione programmata. Le Macro consentono di definire delle "*nuove istruzioni"* richiamabili semplicemente con una label.

- **CONFIG XX:** Alcune caratteristiche del PIC, come la funzione WDT o la scelta del tipo di oscillatore, si possono impostare sia dal programma assemblatore sia utilizzando la direttiva CONFIG che agisce sul registro a 14 bit di configurazione del microcontrollore. Per il PIC16F84 il registro di configurazione si trova in un'area di memoria riservata all'indirizzo 2007H. E'opportuno sottolineare che alcuni software per la programmazione del microcontrollore non sono in grado di riconoscere la direttiva CONFIG per cui è inutile inserirla nel programma sorgente ma è opportuno agire direttamente sui comandi di impostazione del software del programmatore.

Di seguito si mostra la struttura della word di configurazione per il PIC 16F84

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0

Bit 13 - 4 : CP. Code Protection. CP = 1 protezione OFF; CP = 0 protezione ON. Se CP = 0 il codice del programma è protetto e se si tenta di leggere la EEPROM di programma si ottiene sempre il codice 7FH.

Se la protezione è attivata il PIC potrà essere *programmato una sola volta*. In tal caso si dice che il microcontrollore opera in **modalità OTP** (One Time

Programmable). Questa modalità operativa è utilizzata allorquando si desidera che il programma sia protetto e non modificabile.

Bit 3: $\overline{\text{PWRTE}}$. Power-up Timer Enable. $\overline{\text{PWRTE}}$ = 1 Disabilitato; $\overline{\text{PWRTE}}$ = 0 Abilitato. La funzione provvede a tenere all'accensione il chip

azzerato per un tempo fisso di 72ms in modo da consentire alla tensione di alimentazione V_{DD} di raggiungere il valore di regime.

Bit 2: WDTE. Watchdog Timer Enable. WDTE = 1 Abilitato; WDTE = 0 Disabilitato.

Bit 1 - 0: FOSC1-FOSC0. Oscillator Select :

00 – LP. Oscillatore con cristallo a bassa Potenza
01 – XT. Oscillatore al quarzo
10 – HS. Oscillatore al quarzo ad alta velocità
11 – RC. Oscillatore a componenti RC

Ad esempio, la direttiva:

CONFIG 3FF9H

Indica che il registro di configurazione è caricato con il codice: 11111111111001.

Il PIC è programmato con Protezione del Codice disabilitata (CP = 1), Power-up disabilitato ($\overline{\text{PWRTE}}$ = 1), WDT disabilitato (WDTE = 0) e come oscillatore impiega un quarzo (FOSC1 = 0; FOSC0 = 1).

10.1. Software di programmazione MPLAB

Il programma MPLAB della Microchip consente di sviluppare applicativi in linguaggio assembler per numerosi modelli di microcontrollori. Il programma opera in ambiente Windows e dispone di diversi *Tools* per il *debug* e la simulazione del programma assembler. Esso integra i programmi: Assembler MPASMWIN, Linker MPLINK e le librerie MPLIB, della stessa Microchip, per generare i diversi file di lavoro. Per approfondimenti sull'uso del programma si rimanda alla guida in linea. In fig. 5 si riporta la schermata di avvio del programma.

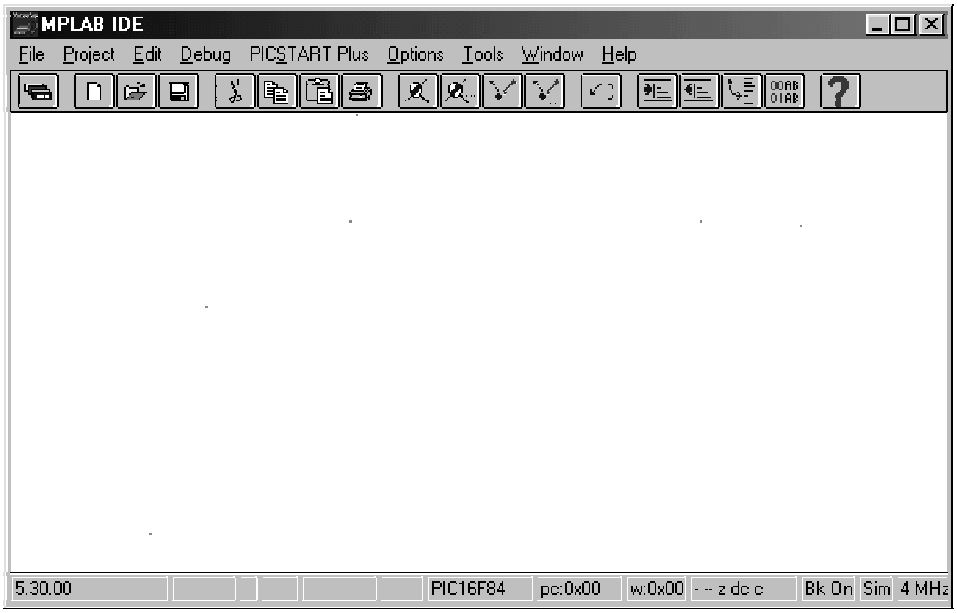


Fig. 5 Schermata del programma MPLAB

Si descrivono i passi fondamentali per l'impostazione del programma, l'editing del file da generare e la procedura per la creazione del file .HEX. Per rendere più chiara la procedura si svilupperà un semplice esempio di programmazione del PIC che consiste nel produrre un programma che faccia lampeggiare un diodo LED collegato alla linea RB0/INT.

Prima di editare il file in linguaggio assembler è necessario impostare alcune opzioni. In particolare dai menu:

- **Project>InstallLanguage Tool** scegliere come linguaggio Microchip e come Tool Name MPASM, come in fig. 6.

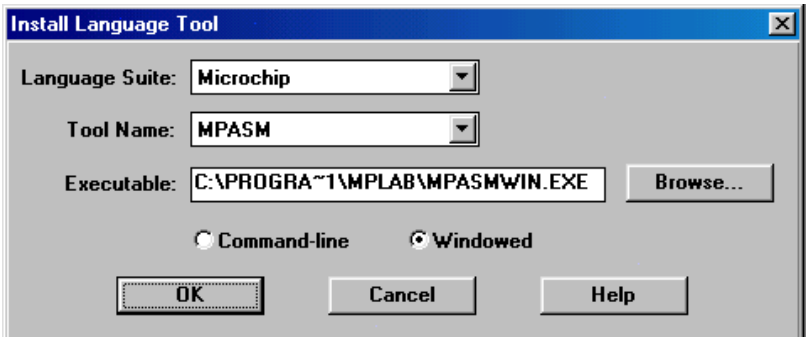


Fig. 6

- **Option>Development Mode>Tools** scegliere come processore PIC16F84 e MPLABSIM Simulator se si desidera oltre che editare il programma anche poter utilizzare il simulatore, come in fig. 7.

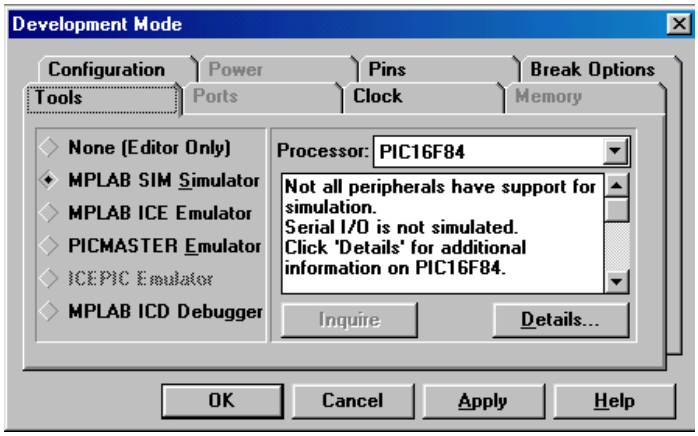


Fig. 7

- **Project>New Projct** scegliere la directory nella quale si vuole salvare il nuovo progetto e assegnare un nome al progetto. E' consigliabile avere una cartella nella quale salvare tutti i file del progetto. Nell'esempio si è assegnato al progetto il nome **ledlam.pjt** nella **Cartella lamp,** come in fig.8.

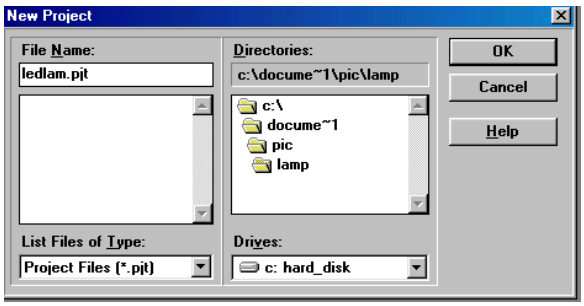


Fig. 8

Dopo aver dato OK si apre la finestra Edit Project. fig. 9.

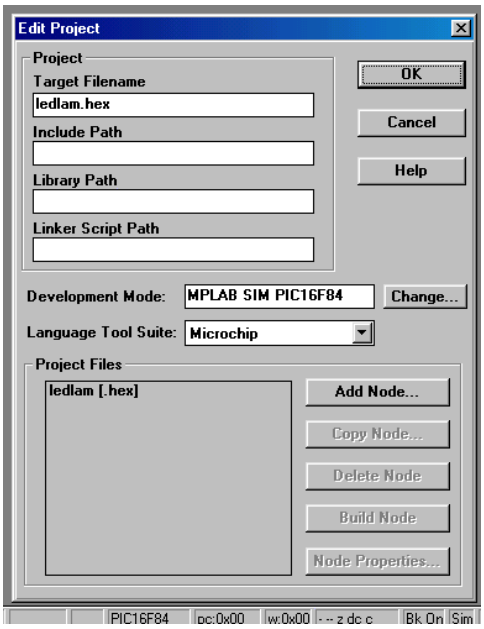


Fig. 9

Controllare che le impostazioni Development Mode e Language Tool Suite siano corrette. Dare OK per tornare nella finestra di lavoro di MPLAB.

Per editare il programma sorgente in linguaggio Assembler si deve aprire un nuovo file dal menu:

File >New . Si apre la finestra dell'editor. Si scrive il programma in assembler e si salva il lavoro assegnando un nome con estensione **.asm.**

Si riporta il listato di un programma che genera un'onda quadra sulla linea RB0/INT collegata ad un diodo LED. L'effetto è quello di avere il LED lampeggiante sulla linea RB0. Nella stesura del programma le *label* devono essere al massimo di 32 caratteri e devono iniziare dalla prima colonna, pena un errore nella compilazione del programma. I commenti devono essere preceduti dal ;.

```
;*************************************************************************
;********LED LAMPEGGIANTE SULLA LINEA RB0 **************
;*******************************************************
;*********Definizione delle Direttive per l'assemblatore************

        PROCESSOR       16F84
       INCLUDE         "P16F84.INC"
       ORG     00H     ;Indirizzo di Reset del PIC
       ;Definizione di due registri utilizzati per generare il ritardo
Count1  EQU     0CH         ;Registro alla locazione 0CH in RAM
Count2  EQU     0DH         ;Registro alla locazione 0DH in RAM

       bsf     STATUS,RP0      ;Selezione del Banco 1
       movlw   00h             ;Azzera l'accumulatore W
       movwf   TRISB           ;Imposta i pin della porta B come OUT
       bcf     STATUS,RP0      ;Seleziona il Banco 0
       bsf     PORTB,0         ;Pone 1 il bit RB0 della Porta B (LED acceso)

MAIN:
       call    DELAY           ;Chiama la subroutine di ritardo
       btfsc   PORTB,0         ;Salta la successiva istruzione se il bit 0 del
                               ;registro PORTB è zero
       goto    LEDOFF
       bsf     PORTB,0         ;Pone a 1 il bit RB0 della Porta B (LED acceso)
       goto    MAIN            ;Salta alla label MAIN

LEDOFF:

       bcf     PORTB,0         ;Pone a 0 il bit 0 della Porta B (LED spento)
       goto    MAIN            ;Salta alla label MAIN

       ;La frequenza del lampeggio è ottenuta generando un ritardo mediante
       ;il decremento dei registri Conta1 e Conta2
       ;Modificando i valori che si caricano inizialmente in Conta1 e Conta2
       ;è possibile cambiare il ritardo. In questo caso si carica 00H.

DELAY:
       movlw   00H             ;Carica 0 nell'accumulatore W
       movwf   Count1          ;Poni il contenuto di W in Conta1
       movlw   00H             ;Carica 0 nell'accumulatore W

movwf          Count2          ;Poni il contenuto di W in Conta2

RIPETI:
       decfsz  Count1,1
       goto    RIPETI
       decfsz  Count2,1
       goto    RIPETI
       return
       END
*************************************************************************
```

Il programma deve essere salvato. Dal menu:
File > Save as assegnare il nome **ledlam.asm**.

Per generare il file eseguibile si deve aprire il menu **Project>Edit Projct .** Si torna alla finestra precedente. Selezionare il file **ledlam[.hex]** nella scheda **Project Files** e cliccare sul pulsante **Add Node…**Nella finestra che si apre selezionare il file **ledlam.asm** e dare OK. Nella scheda **Project File** della finestra di **Edit Project** verrà aggiunto il file **ledlam.asm**., come in Fig.10.

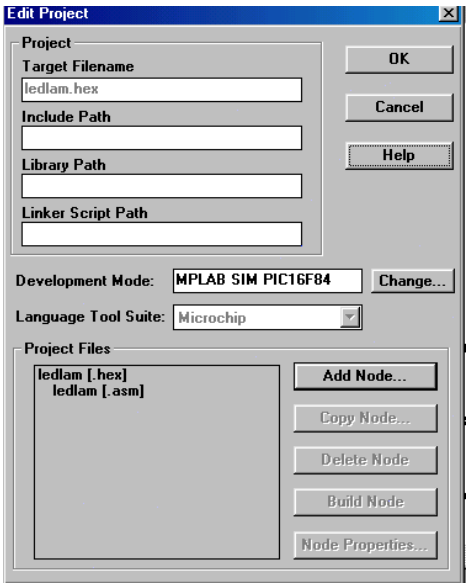


Fig. 10

Se si seleziona il file **ledlam[.asm**] e si attiva il pulsante **Delete Node** si può eliminare il file.

Per assemblare il sorgente **ledlam.asm** e ottenere il file oggetto **ledlam.hex** insieme agli altri file di lavoro, è necessario attivare il menu:

Project>Make Project

Parte la compilazione e se non vi sono errori si ottiene la seguente schermata (fig.11) che ci informa che la procedura è stata completata con successo.

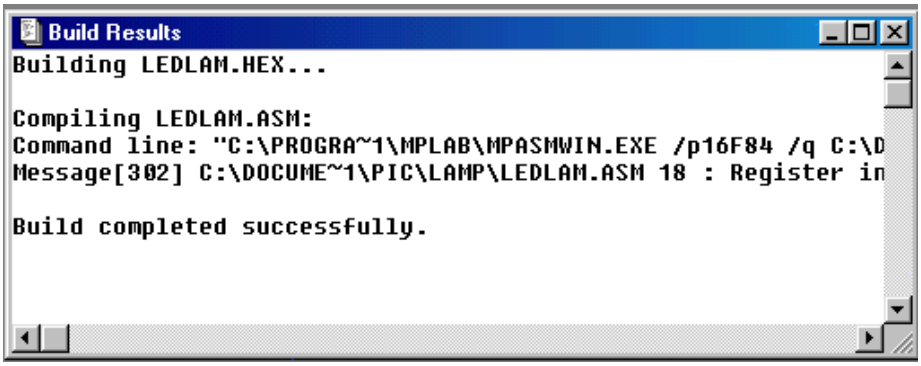


Fig. 11

Dal menu **Windows** si possono analizzare, in particolare, il Programm Memory che fornisce gli indirizzi di memoria e l'Absolute List che fornisce il file eseguibile.

Nella **Cartella lamp** si troveranno i diversi file generati nella compilazione ed in particolare il file **ledlam.hex.**

MPLAB consente, tra l'altro, anche la simulazione (**Debug>Simulator Stimulus**) e il debug del programma assembler (**Debug>Run**). Si consulti la guida in linea.

10.2 La programmazione del PIC

Per trasferire il file eseguibile nella memoria del microcontrollore è necessario disporre di un software di programmazione e di una interfaccia al computer costruita in base alle specifiche fornite dal costruttore. Per quanto concerne il software sarà analizzato il programma **ic-prog** mentre per l'interfaccia al Computer si utilizzerà il circuito mostrato in fig. 12.

Per la realizzazione pratica è necessario disporre di un connettore DB9 a 9 poli da collegare alla porta seriale RS232 del Computer e di uno zoccolo a 18 piedini su cui alloggiare il PIC16F84 da programmare.

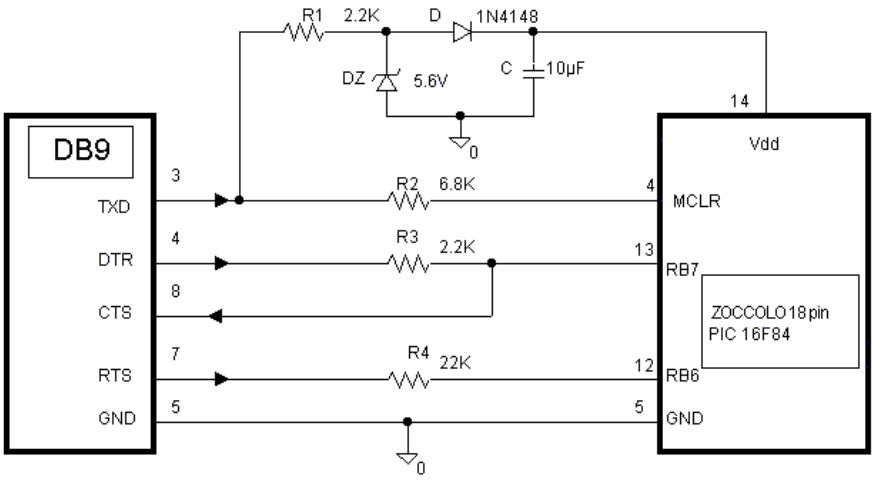


Fig. 12 Interfaccia al Computer tramite connettore seriale RS232.

L'interfaccia proposta è autoalimentata dalla porta seriale del Computer. Per alcune schede madri di computer può accadere che i livelli di tensione non siano sufficienti per la programmazione del PIC e ciò produce un errore. A tale inconveniente si rimedia interponendo tra il pin TXD della porta seriale e il nodo delle resistenze R1-R2 lo stabilizzatore di tensione 7805 alimentato da una batteria alcalina da 9V come in fig. 13 . Quest'ultima soluzione è, comunque, sempre consigliabile.

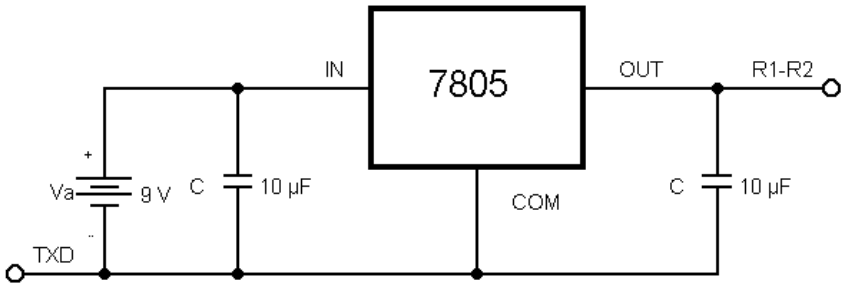


Fig.13 Circuito di alimentazione per l'interfaccia seriale al PIC.

Dopo aver inserito il PIC 16F84 nello zoccolo a 18 pin dell'interfaccia si lancia il **programma ic-prog.** Appare la schermata di Fig.14.

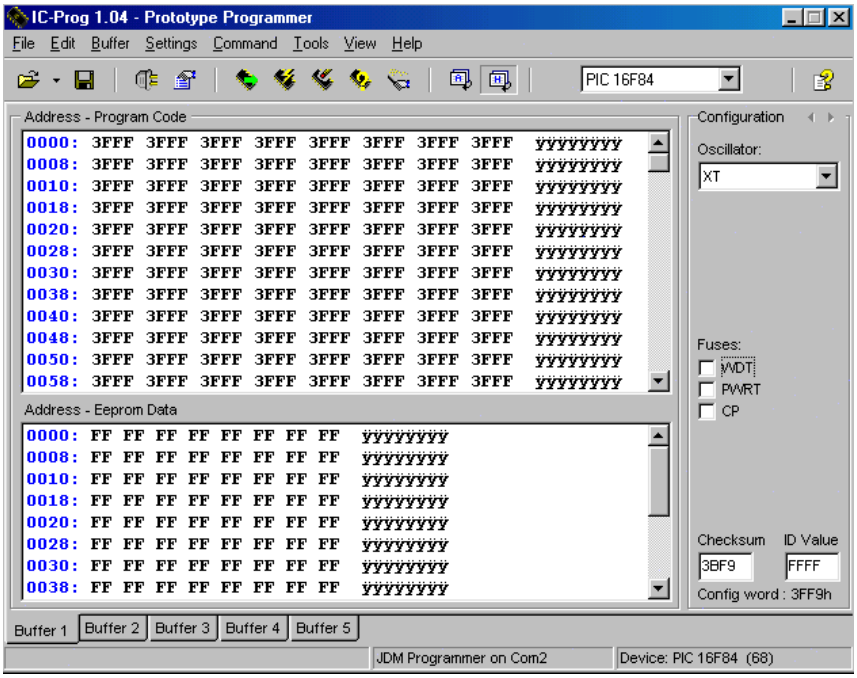


Fig. 14 Schermata del programma ic-prog.

Si procede innanzitutto alle impostazioni preliminari che dipendono da come si intende operare con il microcontrollore. Nel caso generale:

- Si Imposta **Oscillator** su XT (Oscillatore al quarzo)
- Si disattivano i **Fuses** WDT, PWRT e CP (si veda quanto detto a proposito della direttiva CONFIG)

Dal **Menu Setting** o dai pulsanti di scelta rapida attivare:

- **Setting >Device>Microchip PIC** e scegliere il PIC 16F84;
- **Setting** >**Hardware**. Impostare Programmer su JDM, la Porta seriale (Com1 o Com 2) e fissare I/O delay al minimo (1 –2), come in Fig. 15.
  Se si hanno problemi aumentare I/O delay. In ambiente Win 2000-NT usare Windows API.

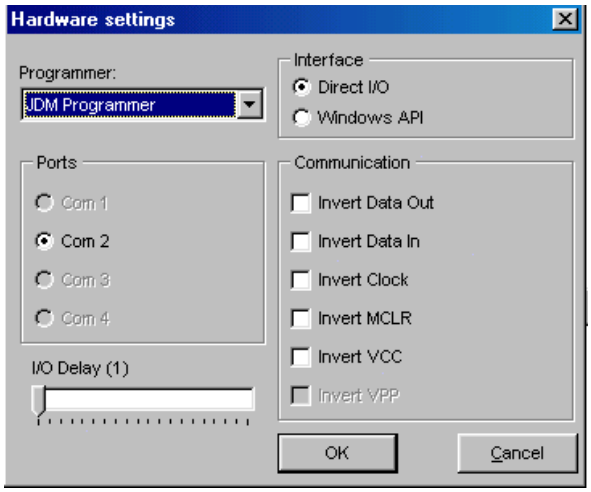


Fig. 15

- **Setting>Option** selezionare le voci Enable MCLR as Vcc e Enable PAGE-WRITE, come mostrato nella seguente fig.16.

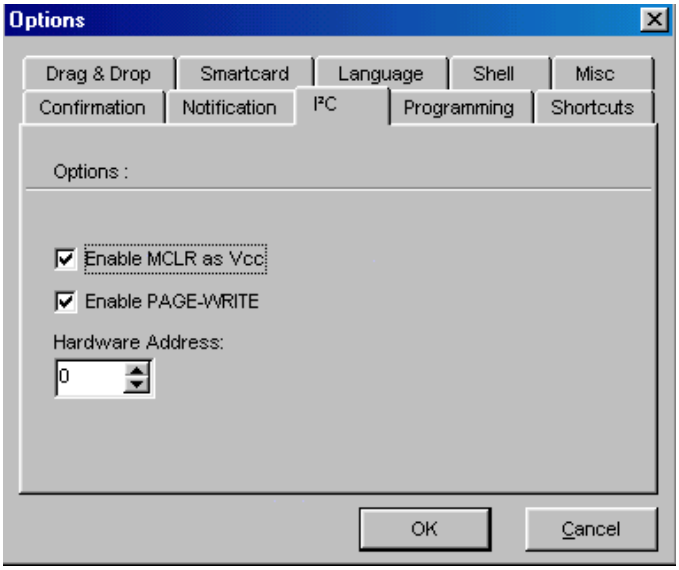


Fig. 16

Dal **Menu File** aprire il programma che si vuole memorizzare nel PIC. Ad esempio il file **ledlam.hex** precedentemente generato con il programma MPLAB:

File>Open File>ledlam.hex

Dal **Menu Command** o di bottoni di scelta rapida, attivare

- **Command>Erase All** per cancellare la memoria del PIC da un eventuale programma precedente;
- **Command>Programm All** per programmare il PIC

Si attiva la procedura di programmazione e se non ci sono errori il PIC è finalmente programmato.

Dal menu Wiew si possono analizzare i listati del programma in assembler e il codice esadecimale. Nell'ambiente ic-prog è possibile effettuare anche piccole modifiche direttamente nelle locazioni di memoria.

Il microcontrollore può essere estratto dalla scheda d'interfaccia e inserito nel circuito in cui deve operare. Nell'esempio il circuito di fig. 17 che può essere montato e provato anche su breadbord.

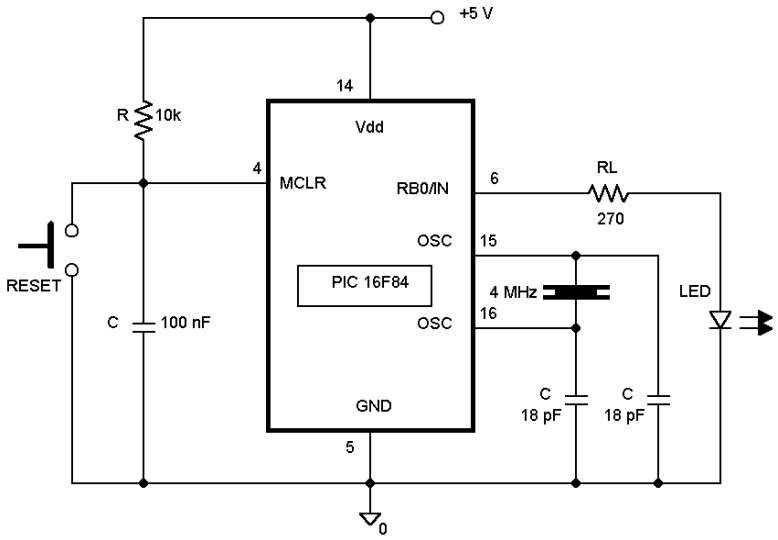


Fig. 17 Schema elettrico del circuito di lampeggio di un LED con PIC 16F84.

In questa semplice applicazione il PIC presenta come unico componente esterno il diodo LED collegato al pin 6 (RB0/INT).

Gli altri componenti sono necessari per il funzionamento del microcontrollore. Il pulsante di RESET consente di far partire il programma memorizzato dalla locazione 00H, mentre l'oscillatore al quarzo da 4MHz genera il segnale di clock necessario al funzionamento dei circuiti interni al microcontrollore.

11. Programmi applicativi per il PIC 16F84

Si riportano alcuni esempi di circuiti in logica programmata che utilizzano come microcontrollore il PIC 16F84. Per ciascun esempio si fornisce il listato del programma in linguaggio assembly e il circuito applicativo. Per poter verificare il funzionamento dei circuiti proposti si deve programmare il microcontrollore seguendo la procedura descritta nel precedente paragrafo.

Esempio n°1

Si riporta il listato di un programma che consente di accendere in sequenza 8 diodi LED collegati alla porta B. Il ritardo tra ciascuna accensione è di circa 1 sec. ed è ottenuto dalla programmazione del timer interno TMR0. In fig. 18 si mostra lo schema elettrico del circuito applicativo.

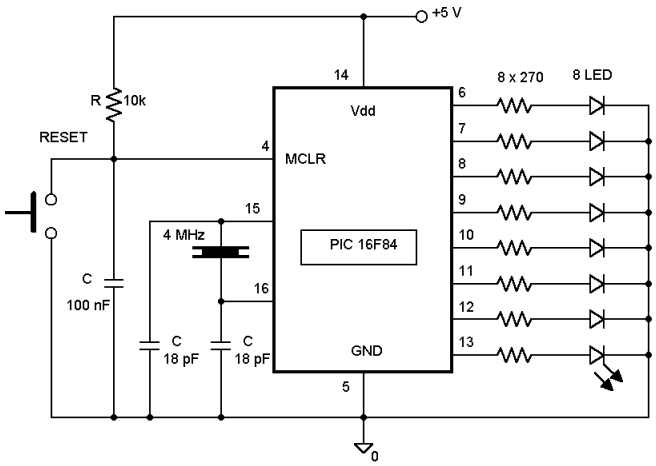


Fig. 18 Generatore di luci sequenziali con ritardo di 1 sec.

```
;*********ACCENSIONE DI 8 LED IN SEQUENZA*****
;********** *****USO DEL TIMER TMR0******************
      PROCESSOR      16F84
      RADIX          DEC
      INCLUDE        "P16F84.INC"
      ;DIRETTIVA CONFIG
       ;Codice di protezione disabilitato
       ;Power-up abilitato
      ;WDT Disabilitato
      ;Oscillatore al quarzo
       __CONFIG       3FF1H
      ;Alla Label Count si riserva un byte nella locazione 0CH
      ORG    0CH
Count          RES    1
Conta          RES    1
```

```
      ORG    00H                ;Indirizzo di RESET del PIC
      bsf    STATUS,RP0         ;Selezione del Banco 1
      movlw  00H                ;Azzera W
      movwf  TRISB              ;Porta B come Output
      ;Impostazione TMR0 con divisione di frequenza 1:32
      ;Con quarzo a 4MHz
      ;la frequenza del clock su TMR0 è f = 1MHz/32 = 31250 Hz
      movlw  00000100B
      movwf  OPTION_REG
      bcf    STATUS,RP0         ;Selezione del Banco 0
      movlw  00H                ;Azzera W
      movwf  Conta              ;Azzera il Registro Conta
MAIN:
      incf   Conta              ;Incrementa Conta
      movf   Conta,W;Trasferisci Conta in W
      movwf  PORTB              ;Trasferisci W in uscita
      ;Chiama la routine di RITARDO e torna a MAIN
      call   Delay
      goto   MAIN
Delay
      ;Si carica in TMR0 6 in modo che si dovranno avere 250 (256-6) impulsi
      ;per azzerare TMR0. La frequenza di conteggio diventa
      ;31250Hz/250 = 125 Hz
      movlw  6
      movwf  TMR0
      ;Si pone Count = 125. Si hanno 125 loop
      ;La frequenza di ripetizione del loop vale f = 125Hz/125 = 1 sec.
      movlw  125
      movwf  Count
DelayLoop
      ;Controlla se TMR0 = 0
      movf   TMR0,W
      btfss  STATUS,Z ;Test bit Z
      goto   DelayLoop
      movlw  6                  ;Carica TMR0 con 6
      movwf  TMR0
      decfsz Count,1
      goto   DelayLoop
      return
      END
```

Esempio n° 2

Si riporta il listato di un programma che consente di gestire un Interrupt proveniente dalle linee RB4…RB7. Il programma è una variante di quello descritto nel paragrafo 10.1. Il LED inserito sulla linea RB0 lampeggia continuamente. Se si porta al livello basso una qualunque delle linee RB4….RB7, si genera un interrupt che provoca l'accensione del LED collegato alla linea RB1. Le linee RB2…RB7, programmate come ingresso, se lasciate aperte sono riconosciute come livello alto poiché connesse all'alimentazione dalle resistenze di pull-up interne.

In fig. 19 si mostra lo schema elettrico del circuito applicativo.

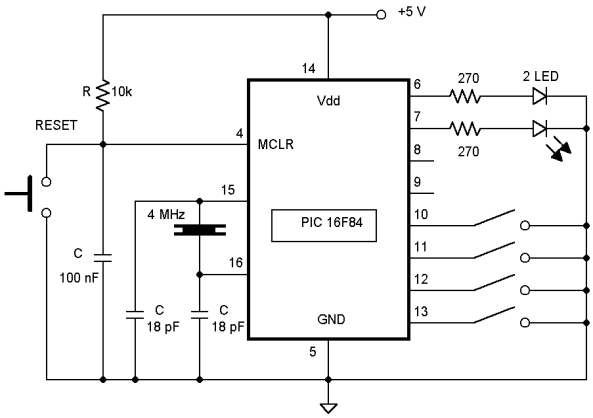


Fig. 19

```
;********LED LAMPEGGIANTE SULLA LINEA RB0 *************
;********ACCENSIONE LED RB1 CON INTERRUPT************
       PROCESSOR      16F84
       RADIX              DEC
       INCLUDE        "P16F84.INC"
       ;Definizione di due registri utilizzati per generare il ritardo
Count1   EQU    0CH             ;Registro alla locazione 0CH in RAM
Count2   EQU    0DH             ;Registro alla locazione 0DH in RAM
       ORG    00H                         ;Indirizzo di Reset del PIC
       goto INIZIO
;**********************ROUTINE DI INTERRUPT***************
       ORG     04H
       bcf    STATUS,RP0        ;Seleziona il Banco 0
       bsf       PORTB,1        ;Pone a 1 il Bit RB1
       bcf       INTCON, RBIF
       retfie
;*****************************************************************************
INIZIO:
       bsf    STATUS,RP0        ;Selezione del Banco 1
       movlw   00H
       movwf   OPTION_REG       ;Inserisce le resistenze di pull-up interne su
                                ;RB4…..RB7
       movlw   11111100B        ;Pone 11111100 nell'accumulatore W
       movwf   TRISB            ;Imposta il pin RB0 e RB1 come OUT
       bcf    STATUS,RP0        ;Seleziona il Banco 0
       bsf    PORTB,0   ;Pone a 1 il bit RB0 della Porta B (LED acceso)
       bcf       PORTB,1        ;Pone a 0 il bit RB1 della PortaB  (LED spento)
       ;****Abilitazione Interruput******
       movlw   10001000B
       movwf   INTCON           ;Abilita l'interrupt sulle linee RB4….RB7
       ;************************************
MAIN:
      call    DELAY             ;Chiama la subroutine di ritardo
      btfsc   PORTB,0  ;Salta la successiva istruzione se il bit 0 del
                                ;Registro PORTB è zero
```

```
        goto    LEDOFF
        bsf     PORTB,0    ;Pone 1 il bit RB0 della Porta B (LED acceso)
        goto    MAIN               ;Salta alla label MAIN
LEDOFF:
        bcf     PORTB,0    ;Pone 0 il bit 0 della Porta B (LED spento)
        goto    MAIN               ;Salta alla label MAIN
        ;La frequenza del lampeggio è ottenuta generando un ritardo mediante
        ;il decremento dei registri Conta1 e Conta2
        ;Modificando i valori che si caricano inizialmente in Conta1 e Conta2
        ;è possibile modificare il ritardo. In questo caso si carica 00H.
DELAY:
        movlw   00H                ;Carica 0 nell'accumulatore W
        movwf   Count1             ;Poni il contenuto di W in Conta1
        movlw   00H                ;Carica 0 nell'accumulatore W
        movwf   Count2             ;Poni il contenuto di W in Conta2
RIPETI:
        decfsz  Count1,1
        goto    RIPETI
        decfsz  Count2,1
        goto    RIPETI
        return
        END
```

Esempio n3

Si riporta il listato di un programma che consente di gestire il Power Down Mode del PIC.

Il LED inserito sulla linea RB0 lampeggia continuamente. Se si porta la linea RA2 al livello basso, il programma si ferma e il PIC entra nella condizione di basso consumo. Per risvegliare il PIC e riprendere il funzionamento del programma è sufficiente attivare il pulsante di RESET. In fig. 20 si riporta lo schema elettrico del circuito applicativo.

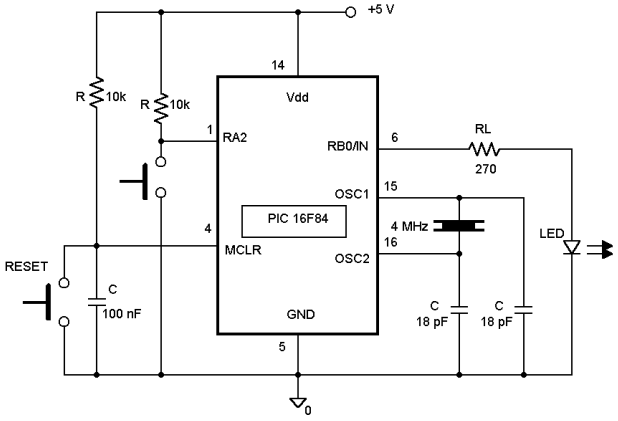


Fig. 20

```
;**********GESTIONE DEL POWER DOWN MODE*******************
;********LED LAMPEGGIANTE SULLA LINEA RB0 ********************
;********RA2 AL LIVELLO BASSO IL PIC SI PORTA IN PWD**********

        PROCESSOR       16F84
        RADIX               DEC
        INCLUDE         "P16F84.INC"
        ;Definizione di due registri utilizzati per generare il ritardo
    Count1   EQU     0CH             ;Registro alla locazione 0CH in RAM
    Count2   EQU     0DH             ;Registro alla locazione 0DH in RAM
        ORG     00H                 ;Indirizzo di Reset del PIC
        bsf     STATUS,RP0          ;Selezione del Banco 1
        movlw   00H                 ;Pone 00H nell'accumulatore W
        movwf   TRISB               ;Imposta la Porta B come OUT
        movlw    00011111B          ;Carica 00001111 in W
        movwf    TRISA              ;Imposta la Porta A come IN
        bcf     STATUS,RP0          ;Seleziona il Banco 0
        bsf     PORTB,0             ;Pone a 1 il bit RB0 della Porta B (LED acceso)
    MAIN:
        ;******USO DELL'ISTRUZIONE SLEEP ******************************
        btfss       PORTA, 2        ;Se RA2 = 0 il PIC si porta in Power Down Mode
        sleep                       ;Se RA2 = 1 il funzionamento è normale
        ;Per risvegliare il PIC premere il tasto RESET
        ;*****************************************************************************
        call    DELAY               ;Chiama la subroutine di ritardo
        btfsc   PORTB,0   ;Salta la successiva istruzione se il bit 0 di PORTB è zero
        goto    LEDOFF
        bsf     PORTB,0   ;Pone 1 il bit RB0 della Porta B (LED acceso)
        goto    MAIN                ;Salta alla label MAIN
    LEDOFF:
        bcf     PORTB,0   ;Pone 0 il bit 0 della Porta B (LED spento)
        goto    MAIN                ;Salta alla label MAIN
        ;La frequenza del lampeggio è ottenuta generando un ritardo ;mediante il
        decremento dei registri Conta1 e Conta2
        ;Modificando i valori che si caricano inizialmente in Conta1 e Conta2
        ;è possibile modificare il ritardo. In questo caso si carica 00H.
    DELAY:
        movlw   00H                 ;Carica 0 nell'accumulatore W
        movwf   Count1              ;Poni il contenuto di W in Conta1
        movlw   00H                 ;Carica 0 nell'accumulatore W
        movwf   Count2              ;Poni il contenuto di W in Conta2
    RIPETI:
        decfsz  Count1,1
        goto    RIPETI
        decfsz  Count2,1
        goto    RIPETI
        return
        END
```

12. Il simulatore di MPLAB

Se il programma eseguibile generato da MPLAB è stato costruito correttamente vuol dire che esso non contiene errori di sintassi ma potrebbero essere presenti errori di logica. Per testare il programma si può caricare l'eseguibile nel PIC e verificarne il funzionamento sul circuito applicativo. MPLAB dispone di un simulatore software che consente di analizzare lo stato logico di tutti i registri. In tal modo è molto più semplice il debug del programma.

Per poter utilizzare il simulatore si deve attivare MPLAB SIM Simulator dal menu **Option/Development Mode/Tools** come descritto nella fig. 7.

In fase di simulazione si può:

1) Definire e/o commutare lo stato logico delle linee di ingresso. Ciò si ottiene attivando il menu **Debug/Simulator Stimulus**. In particolare attivando Asynchronous Stimulus si apre una finestra che permette di assegnare i pin di ingresso e di definirne lo stato logico. (Si veda l'help in linea).
2) Analizzare lo stato logico delle linee di uscita. Dal menu Windows si possono aprire varie finestre tra cui **Special Function Register** che mostra lo stato di tutti i registri del PIC. Per attivare la simulazione si deve aprire la finestra **Debug/Run.** In particolare se si sceglie l'opzione **step** è possibile far eseguire il programma un'istruzione alla volta ed analizzare l'evoluzione dello stato logico di tutti i registri.
3) Inserire dei Break Point. È possibile simulare parti limitate del programma inserendo dei punti d'arresto (Break Point). Ciò si ottiene dal menu **Debug/Break Point.** L'uso dei Break Point si rivela utile soprattutto in programmi complessi costituiti da un numero notevole di istruzioni.

Se nel programma sono presenti lunghi cicli di ritardo è opportuno ridurli o addirittura eliminarli altrimenti diventa praticamente impossibile la simulazione a step.

In fig. 21 si mostra la schermata di lavoro utilizzata per la simulazione dell'esempio n°3. Si noti l'inserimento dei due **;** posti davanti alle istruzioni di **goto RIPETI** per evitare l'esecuzione del loop di ritardo.

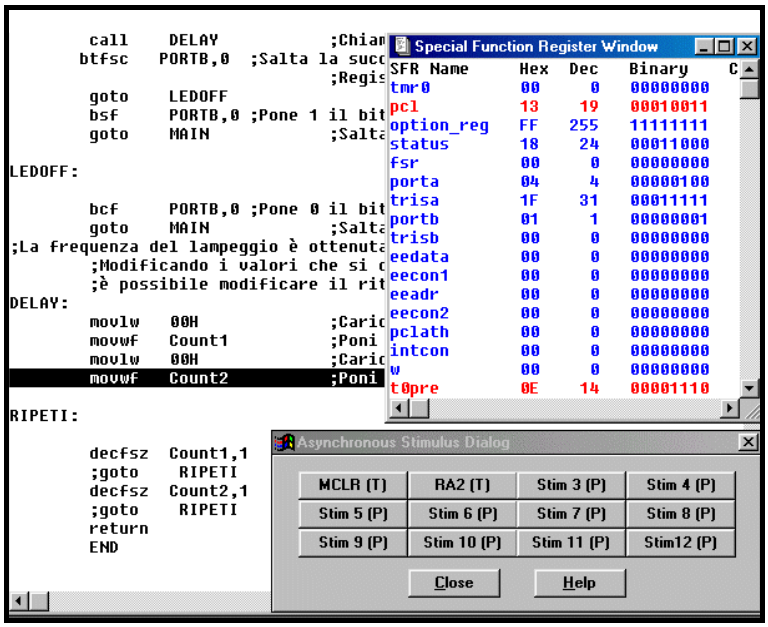
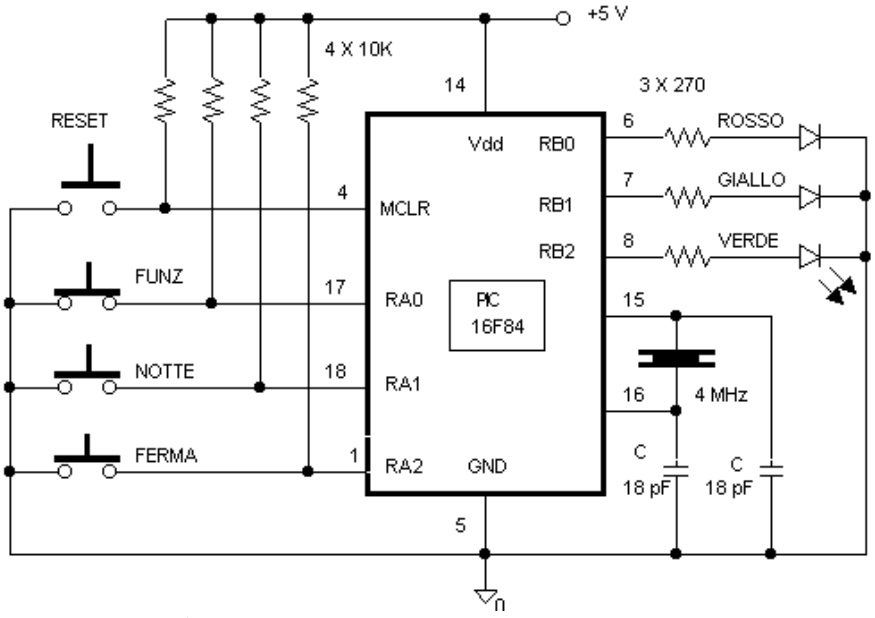


Fig. 21 Schermata relativa all'utilizzo del simulatore di MPLAB.

ESERCITAZIONE

Gestione di un impianto semaforico con microcontrollore PIC 16F84.

Schema elettrico



Elenco componenti

3 diodi LED di colore Rosso, Verde e Giallo. 3 Resistenze da 270Ω. 4 Resistenze da 10KΩ. 3 Pulsanti normalmente chiusi (NC), 1 Pulsante normalmente aperto (NA) per il RESET.

Descrizione

Si riporta il listato del programma in linguaggio assembly per il PIC 16F84.
La porta B è impostata come uscita.. Le linee RB0 RB1 RB2 pilotano, rispettivamente, i LED ROSSO , GIALLO e VERDE.
La porta A è impostata come ingresso. La subroutine SCELTA seleziona la modalità di funzionamento:

- Un impulso positivo su RA0 (RA2RA1RA0 = 001) attiva la sequenza del normale funzionamento del semaforo (subroutine FUNZ);
- Un impulso positivo su RA1 (RA2RA1RA0 = 010) attiva il ciclo notte per cui si ha il giallo lampeggiante (subroutine NOTTE);
- Un impulso positivo su RA2 (RA2RA1RA0 = 100) pone il PIC in modalità SLEEP (subroutine FERMA).

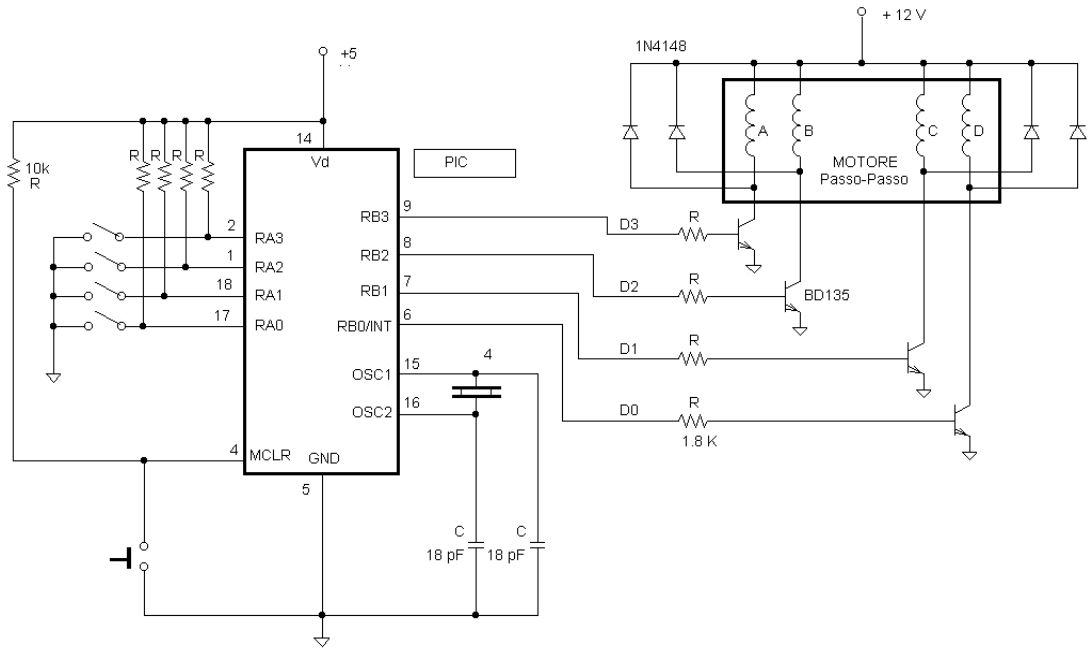
Per riprendere il funzionamento è necessario fornire un impulso di RESET sul piedino MCLR.
La subroutine DELAY imposta la durata dell'accensione dei diodi LED.

```
; **SEMAFORO**

PROCESSOR 16F84
RADIX DEC
INCLUDE "P16F84.INC"
ORG 0CH
Sema          RES 1
Moda          RES 1
Count         RES 1
Count1        RES 1

INIZIO:
 ORG 00H
 bsf STATUS,RP0
 movlw 00H
 movwf Sema
 movlw 255
 movwf TRISA
 movlw 00000011B
 movwf OPTION_REG
 bcf STATUS,RP0
 movlw 00H
 movwf Sema
 movlw 00H
 movwf Moda
 call SCELTA:
 goto 00H
SCELTA:
 bcf STATUS,RP0
 movf PORTA,0
 movwf Moda
 BTFSC Moda,0
 goto FUNZ
 BTFSC Moda,1
 goto NOTTE
 BTFSC Moda,2
 goto FERMA
 return
FUNZ:
 bsf STATUS,RP0
 movlw 00H
 movwf TRISB
 bcf STATUS,RP0
;Accende VERDE
 movlw 100B
 movwf Sema
movwf PORTB

call DELAY
call DELAY
call DELAY
call SCELTA:
;Accende GIALLO
 movlw 010B
 addwf Sema
 movf Sema,w
 movwf PORTB

 call DELAY
 call SCELTA:
;Accende ROSSO
 movlw 001B
 movwf Sema
 movf Sema,w
 movwf PORTB
 call DELAY
 call DELAY
 call SCELTA:
 goto FUNZ

NOTTE:
 bsf STATUS,RP0
 movlw 00H
 movwf OPTION_REG
 movlw 11111101B
 movwf TRISB
 bcf STATUS,RP0
 bsf PORTB,1
GA ; Giallo Acceso
 call DELAY1
 btfsc PORTB,1
 goto GS
 bsf PORTB,1
 call SCELTA:
 goto GA
GS ; Giallo Spento
 bcf PORTB,1
 call SCELTA:
 goto GA
DELAY:
 movlw 00000101B
 movwf OPTION_REG
 movlw 6
 movwf TMR0

movlw 125
 movwf Count
DelayLoop
 movf TMR0,W
 btfss STATUS,Z
 goto DelayLoop
 movlw 6
 movwf TMR0
 decfsz Count,1
 goto DelayLoop
return

DELAY1:
 movlw 00000011B
 movwf OPTION_REG
 movlw 6
 movwf TMR0
 movlw 125
 movwf COUNT1
DelayLoop1
 movf TMR0,w
 btfss STATUS,Z
goto DelayLoop1
 movlw 6
 movwf TMR0
 decfsz COUNT1,1
 goto DelayLoop1
 return
FERMA:
 BTFSC Moda,2
 sleep
 goto INIZIO
 END
```

ESERCITAZIONE

Gestione di un motore passo-passo con microcontrollore PIC 16F84.

Schema elettrico



Elenco componenti

Motore passo-passo a 4 fasi unipolare a magnete permanete; 5 Resistenze da 10KΩ; 4 Resistenze da 1.8KΩ; 4 Transistor NPN BD135; 4 Diodi 1N4148.

Descrizione

Si riporta il listato del programma in linguaggio assembly per il PIC 16F84.
La porta A è impostata come ingresso. In particolare lo stato logico sulla linea RA3 consente di invertire il verso di rotazione.
Le linee RA2, RA1 e RA0 modificano la frequenza dei segnali di comando del motore e quindi la velocità di rotazione. Si possono selezionare 8 valori di frequenza partendo da quella più elevata, di 80 Hz, ottenuta impostando le linee RA2 RA1 RA0 = 000 e, per successive divisione per 2, si ottiene la frequenza più bassa pari a 0.625 Hz ottenuta per RA2 RA1 RA0 = 111.
La porta B è impostata come uscita. Le linee RB3 RB2 RB1 RB0 pilotano le fasi del motore in modalità Full-Step secondo la nota sequenza:
ABCD = 1001 – 1010 – 0110 – 0101.

```
;*MOTORE PASSOPASSO**
PROCESSOR 16F84
RADIX DEC
INCLUDE "P16F84.INC"
ORG 0CH
Count   RES 1
out     RES 1
choose RES 1
ORG 00H
bsf STATUS,RP0
movlw 11110000B
movwf TRISB
movlw 00001111B
movwf TRISA
bcf STATUS,RP0
movlw 00H
movwf PORTB
call scelta
goto INIZIO
scelta
 bcf STATUS,RP0
 movf PORTA,0
 movwf choose
 bcf choose,3
 movf choose,0
 bsf STATUS,RP0
 movwf OPTION_REG
 return
direction
 bcf STATUS,RP0
 movf PORTA,0
 movwf choose
 btfsc choose,3
 swapf out
 return
INIZIO:
 bcf STATUS,RP0
 movlw 01100110B
 movwf out
 call direction
 movf out,0
 movwf PORTB
 call scelta
call delay
 movlw 01011010B
 movwf out
 call direction
 movf out,0
 movwf PORTB
 call scelta
 call delay
 movlw 10011001B
 movwf out
 call direction
 movf out,0
 movwf PORTB
 call scelta
 call delay
 movlw 10100101B
 movwf out
 call direction
 movf out,0
 movwf PORTB
 call scelta
 call delay
 goto INIZIO

delay
 bcf STATUS,RP0
 movlw 6
 movwf TMR0
 movlw 125
 movwf Count
delayloop
 movf TMR0,W
 btfss STATUS,Z
 goto delayloop
 movlw 6
 movwf TMR0
 decfsz Count,1
 goto delayloop
 return
END
```