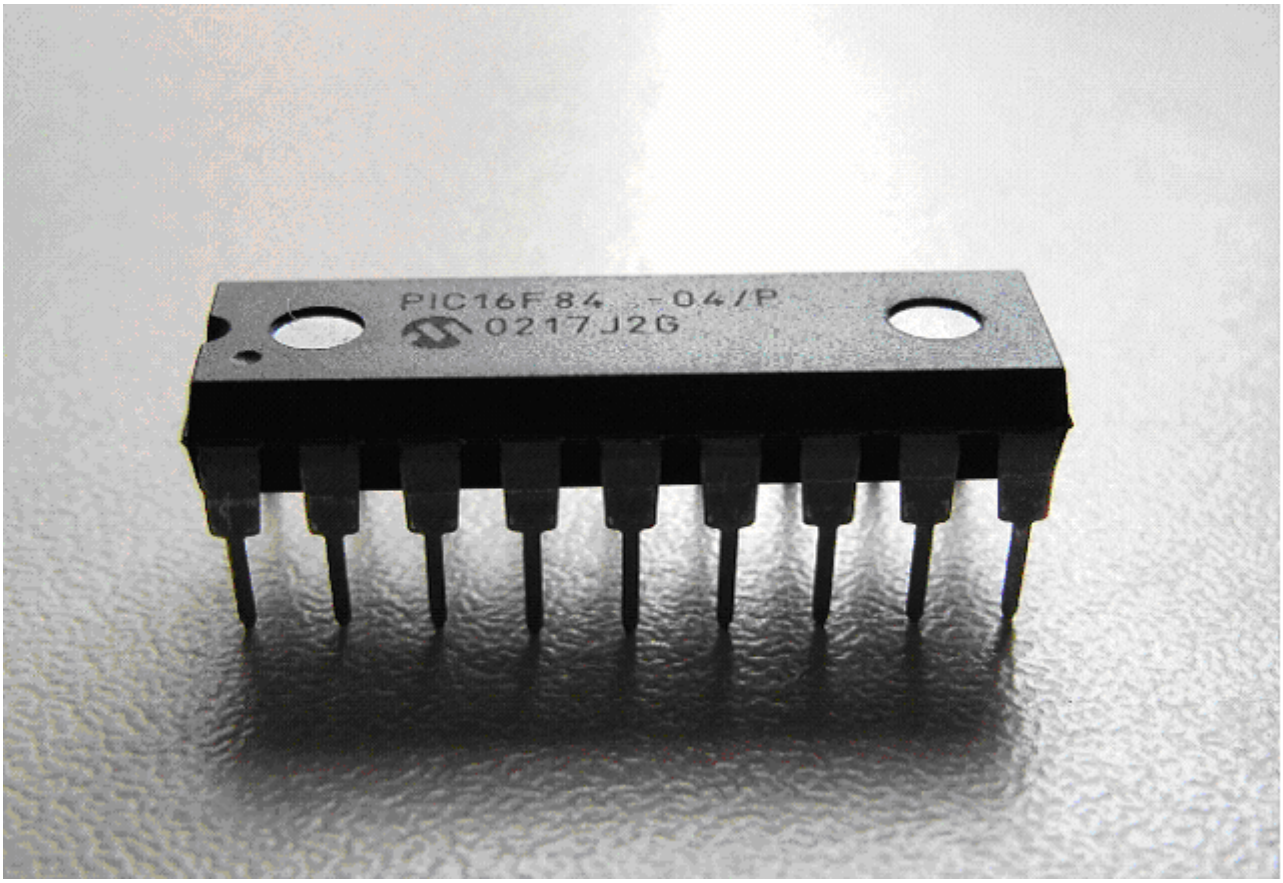
A.s. 2004/05
ROLFO Daniele

5AET



16F84

Principi base per la programmazione

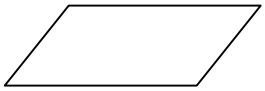
1.1 Introduzione alla programmazione

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

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

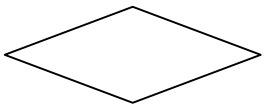
Analisi: l'analisi è una fase molto importante perché si definisce cosa deve fare il progrmmna, il numero delle variabili e tutte le funzioni che deve eseguire il nostro progrmmna. Ciò sembrerà una fase inutile ma per i principianti o per chi realizza un programma complesso sarà molto utile.

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 vero o falso, 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.

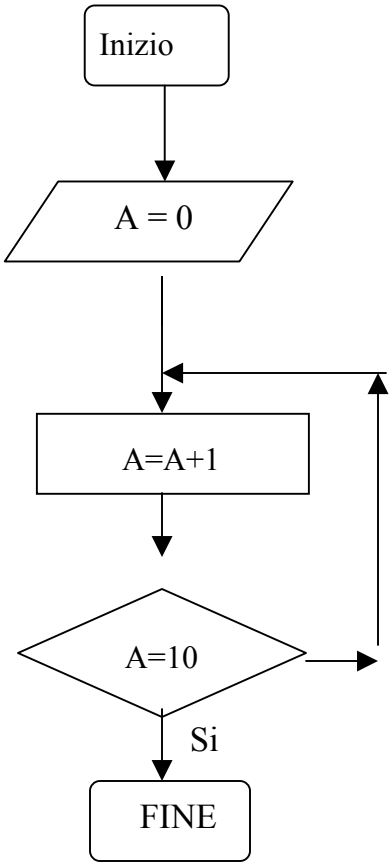
Esempio su come si esegue l'analisi e lo schema a blocchi.

Questo programma può 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:



Con questo blocco si indica l'inizio del programma.

Con questo blocco si crea la variabile A e le si attribuisce il valore 0.

Questo blocco incrementa A di uno e salva il nuovo valore in A.

Con questo blocco si effettua una scelta se A è uguale a 10 allora esce dal ciclo, se no esegue l'istruzione del blocco precedente fino a che non si verifica la condizione presente nel blocco a fianco

Con 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).

PIC 16F84

E' un microcontroller, cioè un circuito che integra in un unico dispositivo (single chip) tutti i componenti necessari per realizzare un completo sistema digitale programmabile; esso risulta più veloce, versatile, affidabile ed economico dei tradizionali sistemi multi chip.

E' essenzialmente composto da:

- CPU (Central Processing Unit), che esegue le istruzioni del programma e, per mezzo della ALU (Arithmetic Logic Unit), svolge i calcoli sui dati durante l'esecuzione del programma operando a 8 bit (quindi con valori numerici non superiori a 255).
- Memorie programma (Program Memory) di tipo flash: si tratta di memoria riscrivibile (più di 1000 volte)in cui vengono memorizzate in maniera permanente le istruzioni del programma da eseguire.
- Memoria dati (Register file) di tipo RAM: è una memoria volatile che contiene le variabili utilizzate dal programma.
- 13 linee di I/O, configurabili singolarmente via software come Input o come Output, utilizzate per i collegamenti a dispositivi esterni.
- Dispositivi ausiliari al funzionamento (generatori di clock, contatore, ecc.).

Si tratta di un sistema programmabile di tipo RISC (Reduced Istructions Set Computer), che utilizza meno di 40 istruzioni software.
Usa un architettura Hardward, utilizza cioè due memorie distinte: una per il programma e l'altra per i dati. In questo modo entrambe le memorie sono accessibili durante lo stesso ciclo macchina, con il vantaggio di essere più veloce dell'architettura standard di Von Neumann (la quale usa invece un'unica memoria sia per il programma che per i dati).

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 clock l, questo segnale può essere usato come sincronismo, il circuito è rappresentato nelle **Figura 1**.

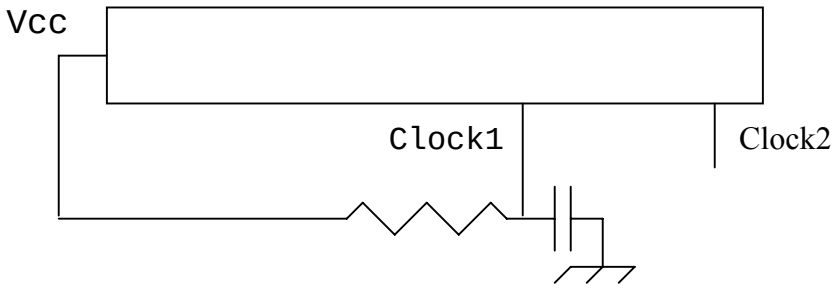


Figure 1. Schema per genera un cIock con una rete RC

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 Clystal).

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 l.

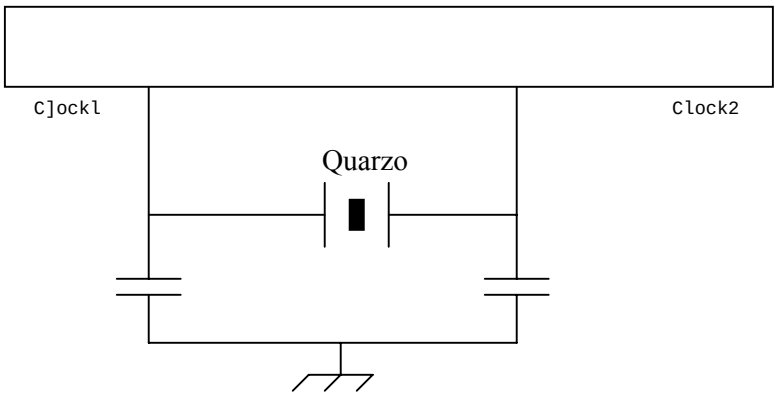


Figura 2. Schema che genera un clock con un quarzo

Mode	Freq	*OSC11C1*	OSC2/C2
LP	32 kHz	68 - 100 pF	68 - 1 O O pF
	200 kHz	15 - 33 pF	15 . 33 pF
XT	100 kHz	100 -150 pF	100 150 pF
	2MHz	15 - 33 pF	15 - 33 pF
	4 MHz	15 - 33 pF	15 - 33 pF
HS	4 MHz	15 - 33 pF	15 - 33 pF
	10M Hz	15 . 33 pF	15 - 33 pF

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

PIEDINATURA

Ovviamente sono presenti due pin per l'alimentazione: Vss e Vdd. Nel primo pin deve arrivare una tensione compresa tra 2V e 6V stabilizzata; nell'altro deve arrivare la massa del circuito. Le linee di I/O prendono due nomi, RA ed RB. Come potete vedere ci sono 4 linee RA che vanno da RA0 a RA3. Le linee RB sono invece 8 e vanno da RB0 a RB7. Le linee RA ed RB hanno una corrispondenza con un registro interno del PIC, più precisamente con i registri PORTA e PORTB, in modo tale che una modifica al registro si ripercuota sulle linee e viceversa. Tutti i pin possono essere configurati singolarmente, via software, sia come entrate che come uscite. Il pin contrassegnato da MCLR è il pin adibito al reset del µC. Quando questo pin è a livello logico basso il PIC si trova in reset e non può eseguire nessun tipo di operazione. Come molti dispositivi di elettronica digitale anche il PIC ha bisogno di un clock che dia gli impulsi necessari al suo funzionamento. Per il PIC16F84 sono previsti diversi tipi di clock (rete RC, quarzo, oscillatori, ...) che vanno collegati in uno solo o tutti e due i pin OSC1/CLKIN e OSC2/CLKOUT.

Il PIC 16F84 i 18 piedini si dividono nel seguente modo:

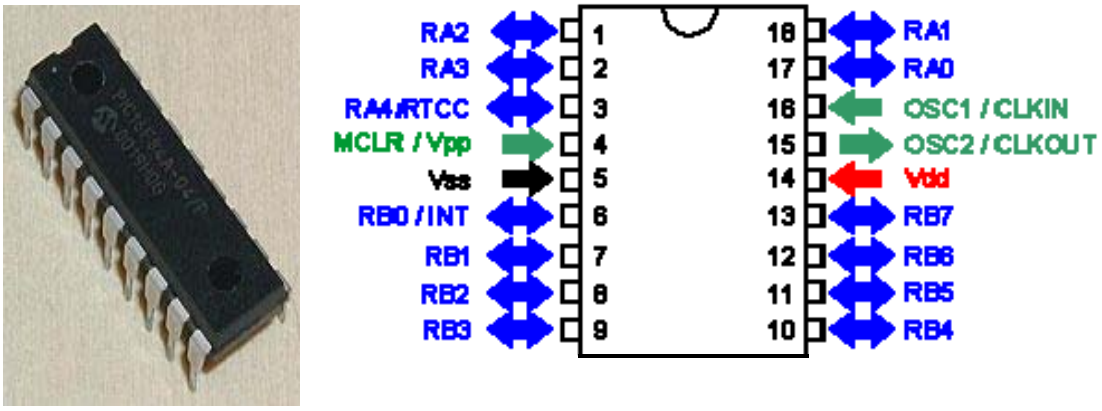


Figura 3. Piedinatura del PIC 16F84

Vcc, GND: alimentazione (da 2 a 6 V)
MCLR: reset attivo basso (riceve dati di programmazione)
OSC1, OSC2: Ingressi del generatore di clock al quarzo (f <= 4MHz)
CLKIN: ingresso per il generatore di clock esterno o con rete RC (f<=4MHz)
CLKOUT: segnale di uscita con frequenza pari a ¼ di quella di clock (eventualmente utilizzabile per sincronizzare altri dispositivi)
Vpp: ingresso utilizzato in fase di programmazione (trasferimento del codice programma all'interno della memoria del PIC)
TOCS: ingresso utilizzabile per inviare un clock esterno al registro contatore TMR0
INT: ingresso utilizzato per la richiesta di interrupt
RA0 – RA4: linee di I/O configurabili singolarmente sia in input che in output via softwere (RA4 solo open collector quindi bisogna mettere pull up)
RB0 – RB7: idem

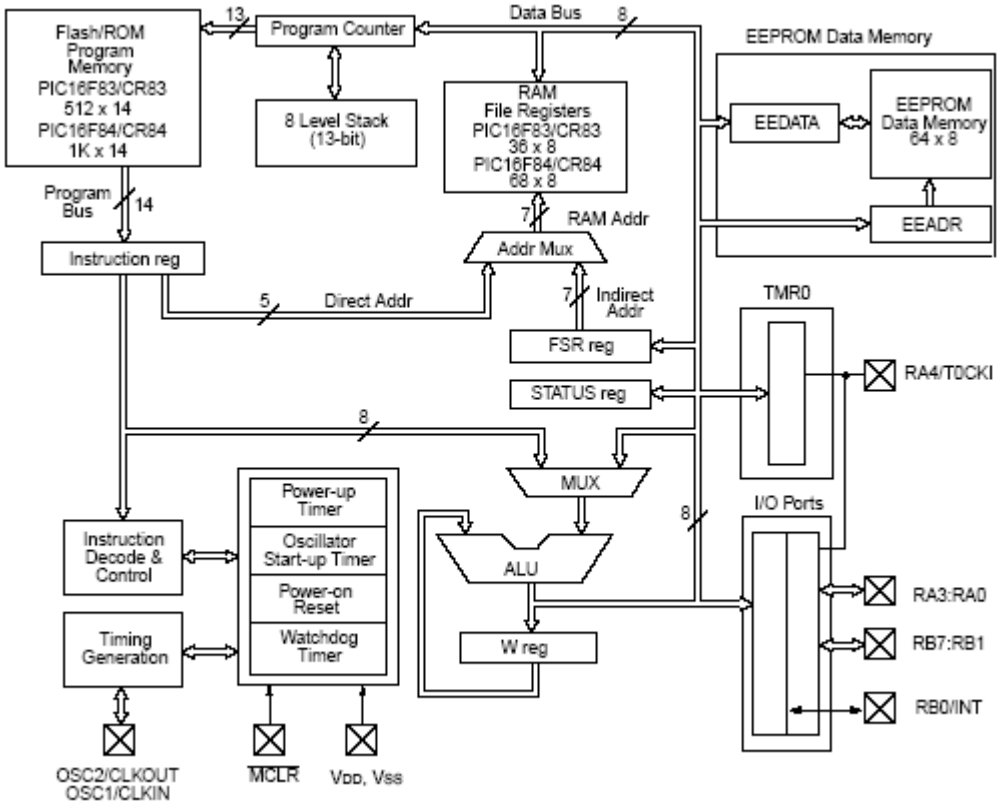
Il bus dei dati (a 8 bit) è distinto dal bus del programma (a 13 bit). A differenza della marea dei microprocessori che per ridurre i costi di produzione adottano l'architettura Von Neumann, i PIC

non condividono la stessa memoria per codice e dati. Ciò gli consente di accedere contemporaneamente a entrambi;
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 che va collegato a Vss con una resistenza da l KΩ.

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 Eprom al16F84.

ARCHITETTURA



Architettura del PIC16F84 (da "Datasheet PIC16x84")

[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 in Figura2

Il PIC16F84 dispone di 3 memorie distinte chiamate Program Memory, Data Memory (o File Register) e EEPROM Data Memory. La prima memoria si trova, nella figura, in alto a sinistra e come vedete viene puntata dal Program Counter con un indirizzo a 13bit. La Program Memory è la memoria adibita al contenimento del programma che il PIC deve eseguire ed è costituita da locazioni da 14bit ciascuna. A sua volta il Program Counter ha un accesso all'area dello Stack per poter recuperare i dati o memorizzarli in questa memoria.
È presente un registro per le istruzioni (l'Instruction Register) con un bus a 5 bit che va ad indirizzare la memoria per permettere di leggere o scrivere dentro la Data Memory. L'instruction Register ha anche un'altro bus a 8 bit che va nel decodificatore di istruzioni; la funzione di quest'ultimo blocco è decodificare ed eseguire le istruzioni.
La Data Memory RAM è la memoria adibita al contenimento dei dati utente e dei registri di configurazione. In questa versione del PIC la DM-RAM è suddivisa in due parti chiamate banchi, accessibili solo uno alla volta. La DM-RAM contiene al suo interno due sezioni. Nella prima sono presenti tutti i registri di configurazione del PIC (Special Function Registers) che permettono di cambiare il suo comportamento e le sue azioni; la seconda sezione contiene registri a 8 bit a libero uso dell'utente chiamati General Purprose Register, in cui è possibile scriverci quello che si desidera. Si hanno a disposizione in totale 68byte di GPR
La Data EEPROM Memory è una memoria EEPROM, quindi, in caso di interruzione dell'alimentazione i dati salvati in questa memoria vengono comunque mantenuti.
Sia la Data Memory RAM che la Data EEPROM Memory confluiscono in un bus, il Data Bus, che serve per far arrivare i dati dalle memorie a tutte le parti del PIC che possono ricevere i dati e viceversa, cioè da tutte le parti che possono salvare i loro dati nella memoria. Vedete che la Program Memory non dispone di un bus come quello descritto sopra, per questo il PIC16F84 non può ne leggere ne scrivere dentro questa memoria.
Il registro delle istruzioni può indirizzare direttamente la memoria RAM, infatti si vede un bus (chiamato "Direct Addr") che va a confluire in un multiplexer il quale indirizza la memoria. Ma è possibile indirizzare la memoria anche tramite il registro FSR, che va anch'esso a confluire nel multiplexer che indirizza la memoria.
Le linee di I/O sono direttamente collegate al data bus, questo perchè modificando particolari registri della Data Memory RAM viene modificato lo stato delle linee tramite il data bus.
La linea RA4, oltre che a essere collegata con il blocco delle linee di I/O è direttamente collegata con TMR0, perchè RA4 è anche la linea di input per poter governare TMR0 se si è in una certa modalità.
Il blocco di decodifica delle istruzioni è collegato con diversi dispositivi, come il Power-Up Timer, il WDT e altri poiché questi ultimi comandano o sono comandati da certe funzioni del PIC che possono essere modificate via software.
Il PIC16F84 possiede una ALU a 8 bit che ha come ingressi i bus e il registro W. Il registro W è un registro molto utilizzato, sia per quanto riguarda le operazioni della ALU che per quando riguarda operazioni con i dati e la memoria.

MEMORIA

PROGRAM MEMORY: memoria flash (riscrivibile) da 1Kx14 (1024 locazione da 14 bit), I cui indirizzi vanno da 000H a 3FFH. La prima locazione di memoria, all'indirizzo 0, è detta Reset Vector, e contiene la prima istruzione che il PC dovrà eseguire al Reset Vector si usa la direttiva ORG 00H. (Si può rendere non cancellabile).

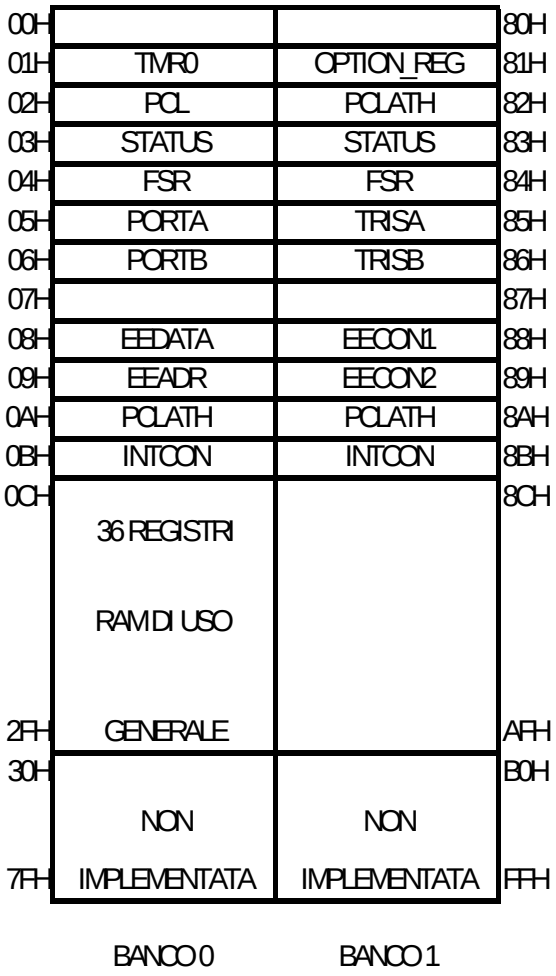
REGISTER FILE: insieme di locazione da 8 bit di memoria RAM (volatile) indirizzabili direttamente, usate per memorizzare le variabili del programma (il cui valore cambia durante l'esecuzione del programma). Il Register File è composto da 48 byte (dall'indirizzo 00H a 2FH per il banco 0) più altri 48 byte (dall'indirizzo 80H a AFH per il banco 1). L'accesso al banco 0 piuttosto che al banco 1 è determinato dai bit RP0 e RP1 del registro STATUS.
Le prime 12 locazioni del banco 0 (dall'indirizzo 00H a 0BH) e del banco 1 (dall'indirizzo 80H a 8BH) sono riservate alle funzioni speciali del PIC.

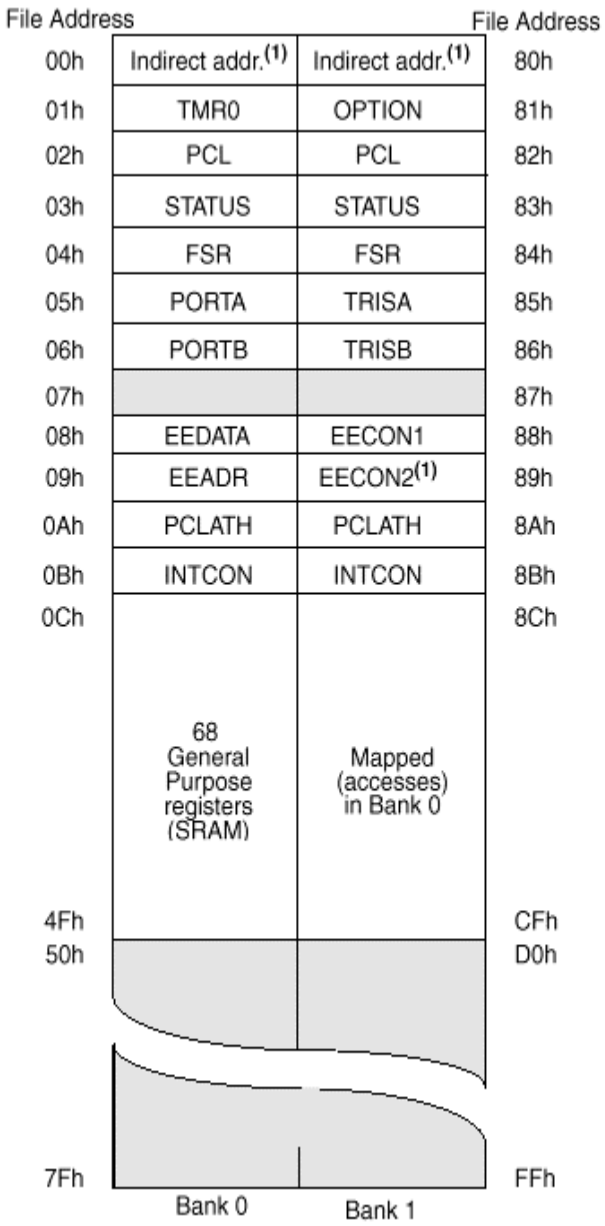
STATUS

Irp	RP1	RP0	NOT_TO	NOT_PD	Z	DC	C

ACCUMULATORE (REGISTRO w): locazione di memoria a 8 bit connessa con l'ALU, che vi può accedere direttamente, senza dover fornire alcun indirizzo di memoria del Register File (es. 0CH) non può essere eseguita con un'unica istruzione (servirebbero 6 bit per il codice, 8 bit per il dato e 6 bit per l'indirizzo, mentre la locazione della program memory può solo contenere solo 14 bit), ma si deve passare attraverso l'accumulatore.

```
Movlw    01H   ; dato
Movwf    0CH   ; indirizzo, prima locazione di memoria
```





GESTIONE I/O

Si hanno a disposizione 13 linee di I/O, configurabili singolarmente via SW, così ripartite:

- **PORTA A :** 5 linee (RA0 – RA4), ognuna delle quali ha una corrente massima di 20 mA in source a 25 mA in sink (la corrente massima complessiva della porta A è di 50 mA in source e 80 mA in sink). La linea R4 come già detto è open collector (occorre collegare una resistenza di pull-up).

- **PORTA B:** 8 linee (RB0 – RB7), ognuna delle quali ha una corrente massima di 20 mA in source e 25mA in sink (la corrente massima complessiva della porta B è di 100 mA in source e 150 mA in sink).

Ad ogni porta sono associati due registri interni (TRISA e PORTA per la porta A, e TRISB e PORTB per la porta B); ad ogni bit del registro corrisponde una linea di I/O.

- I registri TRIS determinano la configurazione come input (bit a 1) o come output (bit a 0) di ogni singola linea.
- I registri PORT contengono lo stato logico (livello alto o basso) di ogni singola linea.

Le linee RB0, RB4, RB5, RB6, RB7, quando sono configurate in input, possono generare (se abilitate via software), in corrispondenza di un cambiamento del loro livello logico, una richiesta di interrupt.

Codice programma per configurare una linea di I/O in input o in output, per assegnarle o per leggerne il livello logico:

```
bsf     STATUS,RP0 ; seleziona il banco 1 di memoria per accedere ai registri TRIS
movlw   00010000B  ; configura la linea RB4
mowf    TRISB      ; come input
bcf     STATUS,RP0 ; seleziona il banco 0 di memoria
bsf     PORTB,2    ;manda a livello logico alto il terzo bit cioè RB2
bcf     PORTB,1    ; manda a livello logico basso il secondo bit, cioè RB1
btfss   PORTB,4    ; leggi il livello logico della linea RB4
goto    LivelloZero ; se è basso salta alla label LivelloZero
goto    LivelloUno  ; altrimenti (se è alto) salta alla label LivelloUno
```

CONTATORE TMR0

Il contatore TMR0 è un particolare tipoi di registro il cui contenuto viene incrementato, con cadenza regolare e programmabile, direttamente dall'hardware del PIC.
Il contatore è presettabile, è cioè possibile stabilire il valore da cui iniziare il conteggio (es.10) caricandolo nel registro TMR0:

```
movlw   10
movwf   TMR0
```

Una volta raggiunto il valore 255, il registro TMR0 si azzera automaticamente e riprende il conteggio da 0.
A seconda del valore del bit TOCS del registro OPTION_REG, la frequenza di conteggio è pari a ¼ di quella di clock (TOCS = 0) oppure a quella di un segnale generato da un oscillatore esterno (TOCS = 1) applicato al pin 3 del PIC.

Tramite il bit TOSE del registro OPTION_REG, si può impostare il conteggio attivo sul fronte di salita (TOSE = 0) o su quello di discesa (TOSE = 1) del segnale applicato esternamente.
PRESCALER: se il bit PSA del registro OPTION_REG è posto a 1, il segnale di cadenza del conteggio (derivato dal clock del PIC o generato esternamente) viene inviato direttamente al contatore TMR0. Se invece viene posto a 0, il segnale di cadenza prima di essere inviato al contatore viene fatto passare in un prescaler (divisore di frequenza) programmabile per mezzo del bit PS0, PS1 e PS2 del registro OPTION_REG.

PS2	PS1	PS0	DIVISORE
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

OPTION_REG

LSB

NOT_RBPU	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**

PROGRAMMAZIONE DEL PIC

Prima di programmare il PIC, inserito nel programmatore collegato al PC attraverso la porta seriale, è necessario settare i parametri di comunicazione (COM1 o COM2); successivamente bisogna configurare il programmatore come LudiPipo, oppure, se questa opzione non è presente come JDM. Infine occorre selezionare il tipo di PIC da programmare (16f84).
A questo punto si può procedere al caricamento del file .hex (che è in sostanza la mappa del program memory) all'interno del PIC e settare i fuses.

SETTAGGIO DEI FUSES

I fuses sono delle celle di memoria che vengono programmate sul PIC, e servono per indicare la modalità di funzionamento del chip.

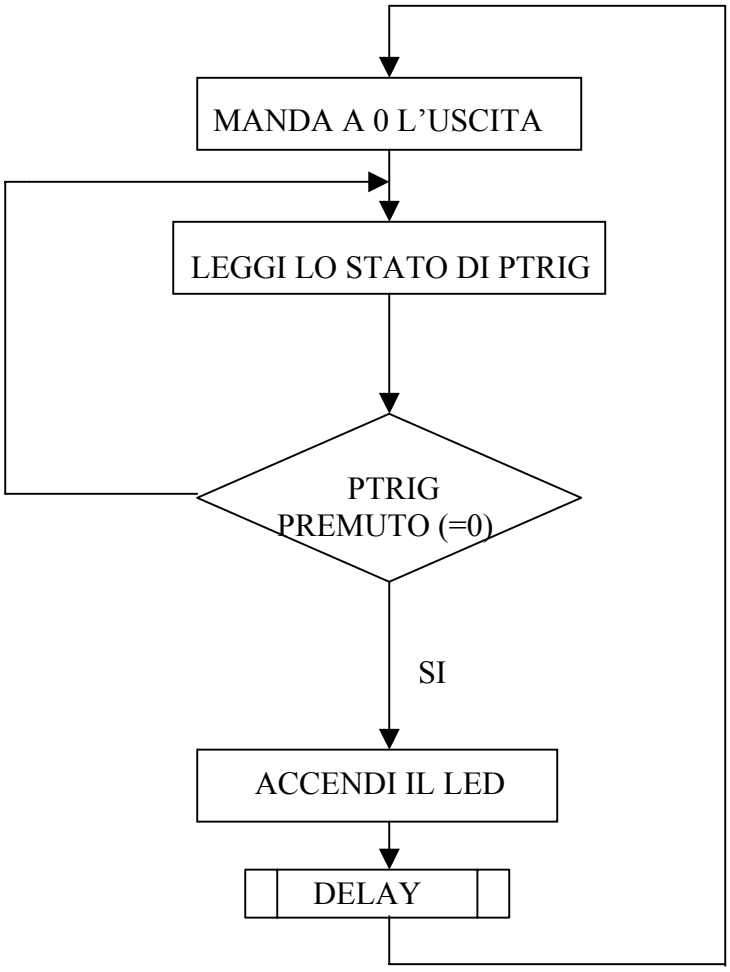
- **CODE PROTECT (default no):** indica se il codice programma del PIC deve essere protetto da lettura (in questo caso però il PIC non sarà più rscrivibile).
- **POWER UP TIMER (default YES) :** attende 50 ms prima di iniziare l'esecuzione del programma, in modo che l'alimentazione si stabilizzi.
- **WATCHDOG TIMER (default NO):** serve per evitare che il programma si pianti (se il programma non da segnali di vita entro un certo tempo, viene resettato il PIC.
- **OSCILLATOR (default XT o RS) :** configura il tipo di oscillatore per il clock, a seconda che si usi un quarzo (XT) o una rete RC (RS).

Di solito si usa la configurazione NO, YES, NO, RS, che corrisponde al codice **3FF3H**.

PROGRAMMI ESEMPIO

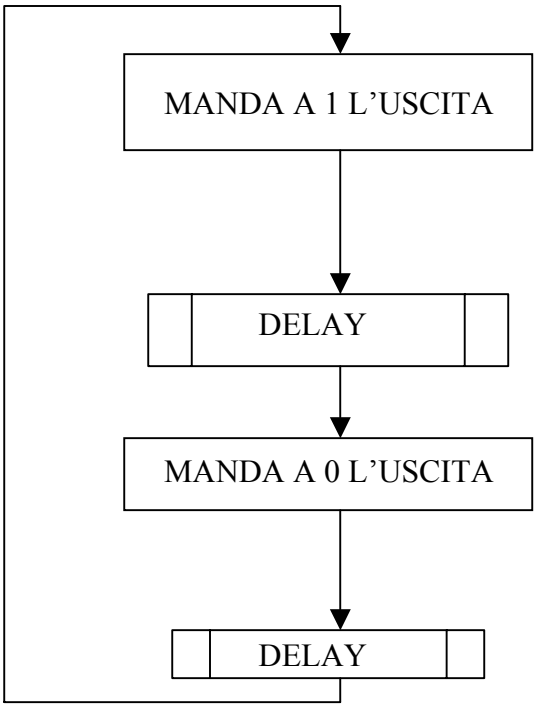
PROGRAMMA MONOSTABILE

```
        PROCESSOR 16F84         ;definisce il tipo di chip usato
        RADIX           DEC     ;i numeri senza notazione si intendono decimali
        INCLUDE         "P16F84.INC"    ;include nel sorgente il file relativo al chip usato
        __CONFIG    3FF3H       ;configura i fuses

PTRIG   EQU  0
OUT     EQU  0

        ORG         0x0C            ;punta al register file
CONT1   RES         1               ;riserva 1 byte per CONT1
CONT2   RES         1               ;e 1 byte per CONT2
        ORG         0x00            ;punta al reset vector
        bsf         STATUS,RP0      ;seleziona il banco1 di memoria
        movlw       00000001B       ;configura le linee di I/O RA
        movwf       TRISA & 0x7F
        movlw       11111110B       ;configura le linee di I/O RB
        movwf       TRISB & 0x7F
        bcf         STATUS,RP0      ;seleziona il banco0 di memoria
start
        bcf         PORTB,OUT       ;seleziona il banco0 di memoria
wait
        btfsc       PORTA,PTRIG      ;leggi lo stato di TRIG e salta se viene premuto
        goto        wait
        bsf         PORTB,OUT       ;manda a 1 l'uscita
        call        Delay           ;segue la subroutine di ritardo
        call        Delay
        call        Delay
        call        Delay
        call        Delay
        call        Delay
        call        Delay
        call        Delay
        call        Delay
        goto        start           ;torna all'inizio
Delay
        clrf        CONT1           ;azzera contatori
        clrf        CONT2
Loop
        decfsz      CONT1,F         ;decrementa cont1 e salta se è 0
        goto        Loop
        decfsz      CONT2,F         ;decdrementa cont2 e salta se è 0
        goto        Loop
        return                      ;fine subroutine
        END                         ;fine programma
```


MANDA A 0 L'USCITA
LEGGI LO STATO DI PTRIG
PTRIG
PREMUTO (=0)
SI
ACCENDI IL LED
DELAY

PROGRAMMA ASTABILE

```
PROCESSOR 16F84
        RADIX           DEC
        INCLUDE         "P16F84.INC"
        __CONFIG    3FF3H


OUT     EQU         0           ;punta al register file




        ORG         0x0C

CONT1   res         1           ;riserva 1 byte
CONT2   res         1           ;riserva 1 byte




        ORG         0x00

        bsf         STATUS,RP0      ;seleziona BANCO 0

        movlw       0FFH            ;metto nell ACC. tutti 1
        movwf       TRISA & 0x7F    ;configuro PORTA A imput
        movlw       00H             ;metto nell ACC. tutti 0
        movwf       TRISB & 0x7F    ;configuro PORTA B output

        bcf         STATUS,RP0




start   bsf         PORTB,OUT   ;manda a 1 uscita
        call        DELAY
        bcf         PORTB,OUT   ;manda a 0 uscita
        call        DELAY
        goto        start
```


MANDA A 1 L'USCITA
DELAY
MANDA A 0 L'USCITA
DELAY

SET DI ISTRUZIONI DEL PIC 16F84

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 O l e indica dove il dato verrà salvato se d=l allora il dato verrà salvato nel registro f, se d = O il dato verrà salvato nel registro w.

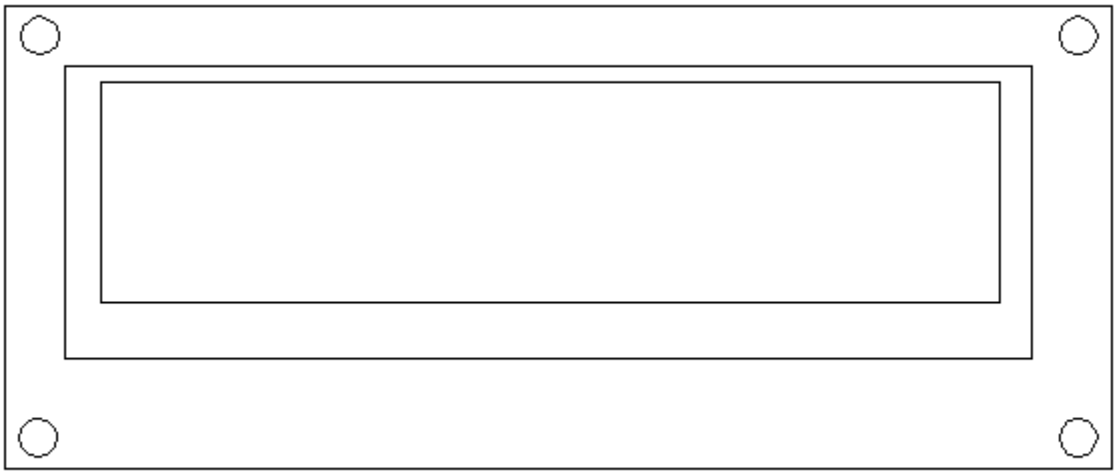
b : Questo parametro definisce il bit su cui deve essere portata a tennine l'operazione, poiché i bit di un registro sono 8 il valore di b varia tra O e 7.

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

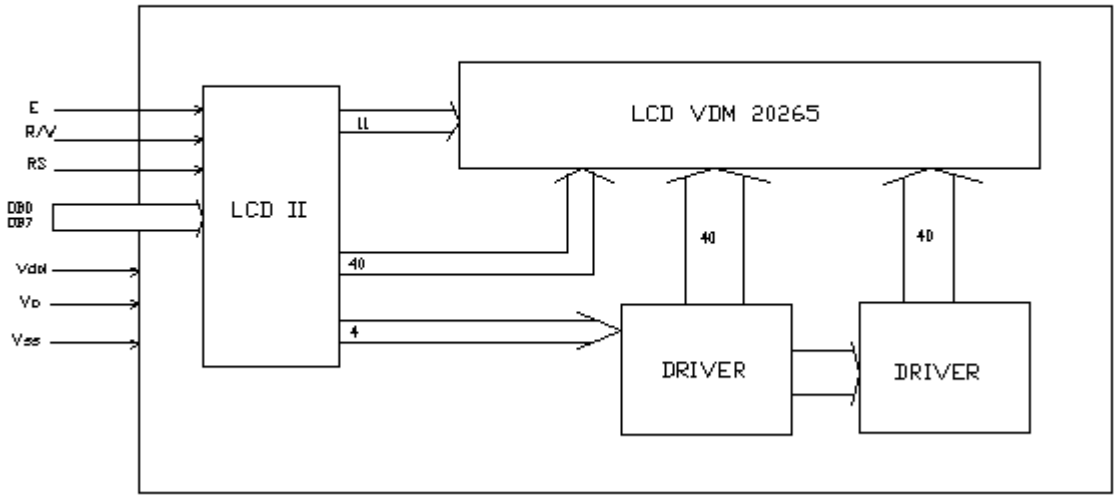
Sintassi	**Descrizione Microchip**	**Operazione equivalente**
ADDLW k	Add Literal and W	W = W + k
ADDWF f,d	Add W and f	d = W + f (dove d può essere W o f)
ANDLW k	AND Literal with W	W = W AND k
ANDWF f,d	AND W with f	d = W AND f (dove d può essere W o f)
BCF f,b	Bit Clear f	f(b) = 0
BSF f,b	Bit Set f	f(b) = 1
BTFSC f,b	Bit Test f, Skip if Clear	f(b) = 0 ? Si, salta una istruzione
BTFSS f,b	Bit Test f, skip if Set	f(b) = 1 ? Si, salta una istruzione
CALL k	Subroutine Call	Chiama la subroutine all'indirizzo k
CLRF f	Clear f	f = 0
CLRW	Clear W Register	W = 0
CLRWDT	Clear Watchdog Timer	Watchdog timer = 0
COMF f,d	Complement f	d = not f (dove d può essere W o f)
DECF f,d	Decrement f	d = f -1 (dove d può essere W o f)
DECFSZ f,d	Decrement f, Skip if 0	d = f -1 (dove d può essere W o f) se d = 0 salta
GOTO k	Go to address	Salta all'indirizzo k
INCF f,d	Increment f	d = f +1 (dove d può essere W o f)

INCFSZ f,d	Increment f, Skip if 0	d = f +1 (dove d può essere W o f) se d = 0 salta
IORLW k	Inclusive OR Literal with W	W = W OR k
IORWF f,d	Inclusive OR W with f	d = f OR W (dove d può essere W o f)
MOVLW k	Move literal to W	W = k
MOVF f,d	Move f	d = f (dove d può essere W o f)
MOVWF f	Move W to f	f = W
NOP	No Operation	Nessuna operazione
OPTION	Load Option Register	OPTION = W
RETFIE	Return from Interrupt	Ritorna da un interrupt handler
RETLW k	Return Literal to W	Ritorna da una subroutine con W = k
RETURN	Return from Subroutine	Ritorna da una subroutine
RLF f,d	Rotale Left f through Carry	d = f << 1 (dove d può essere W o f)
RRF f,d	Rotale Right f through Carry	d = f >> 1 (dove d può essere W o f)
SLEEP	Go into Standby Mode	Mette in standby il PIC
SUBLW k	Subtract W from Literal	W = k - W
SUBWF f,d	Subtract W from f	d = f - W (dove d può essere W o f)
SWAPF f	Swap f	f = Swap dei bit 0123 con 4567 di f
TRIS f	Load TRIS Register	TRIS di f = W
XORLW k	Exclusive OR Literal with W	W = W XOR k
XORWF f,d	Exclusive OR W with f	d = f XOR W (dove d può essere W o f

LCD DYSPLAY



SCHEMA A
BLOCCHI



DISPLAY LCD

La struttura interna del display LCD è così composta: ha al suo interno due registri: un registro per le istruzioni (IR) e un registro per i dati (DR). Il registro delle istruzioni contiene i comandi mandati al display (cancella schermo, muovi cursore, posizione cursore, ecc) e può essere scritto ma non letto dalla porta MPU (porta del microcontrollore o della CPU). Il registro dei dati contiene temporaneamente i caratteri scritti nella DD Ram (memoria caratteri) o nella CG Ram (memoria grafica).
Ogni dato scritto dentro a questo registro viene automaticamente ricopiato nella memoria di destinazione. Questo registro può essere scritto e letto. In fase in fase di lettura viene caricato il dato dalla DD Ram o dalla CG Ram e posto in DR (la posizione di lettura è impostata tramite l'IR). A ogni lettura viene incrementato automaticamente il puntatore AC.

Per rendere il Display pronto a ricevere e mandare in visualizzazione i dati, ogni volta all'accensione occorre effettuare una procedura di inizializzazione:

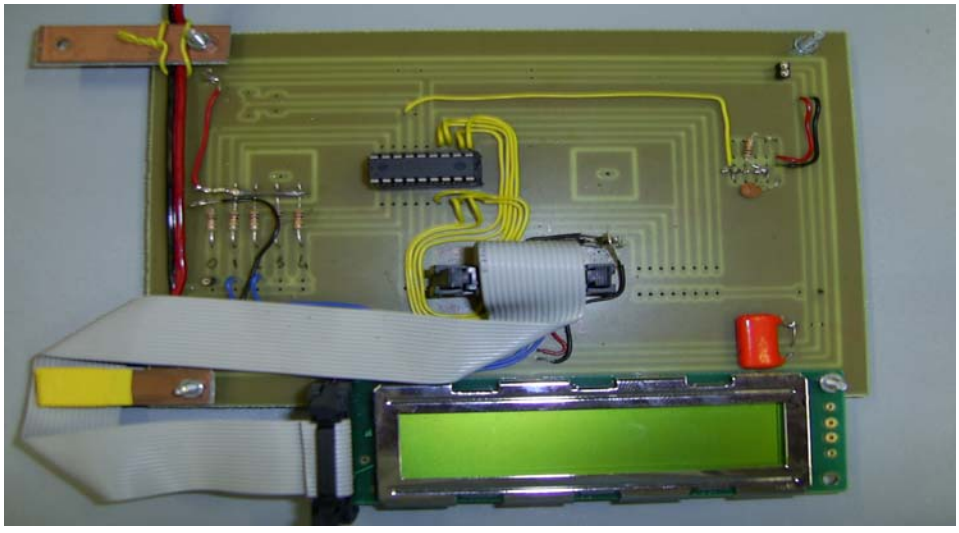
		rs	r/w	
1°	30H	0	0	00110000
2°	30H	0	0	00110000
3°	30H	0	0	00110000
4°	38H	0	0	00111000
5°	08H	0	0	00001000
6°	01H	0	0	00000001
7°	06H	0	0	00000110
8°	0FH	0	0	00001111

NOSTRO UTILIZZO DEL PIC

Per meglio comprendere le potenzialità dei microcontrollori, utilizzando il PIC 16F84 abbiamo costruito 3 schede per visualizzare dei caratteri su un display LCD. La scheda RICEZIONE, riceve i dati in seriale e li visualizza sul display mentre le schede di trasmissione funzionano una con un tastierino 4x4 a matrice, mentre l'altra con un semplice switch a 8 bit.

SCHEDA TASTIERINO

SCHEDA DISPLAY RX



LISTATO DEI PROGRAMMI

PROGETTO TASTIERINO

RICEVITORE DISPLAY

La scheda ricevente, deve inizializzare il display e aspettare il dato fornito dalla scheda trasmettente. Per inizializzare il display bisogna:

```
        ORG         0x0C

BYTE1   RES         1
BYTE2   RES         1
BYTE3   RES         1
BYTE4   RES         1
BYTE5   RES         1
BYTE6   RES         1
BYTE7   RES         1
BYTE8   RES         1
```

Riservare delle variabili per poi caricare nel display i codici esadecimali per l'inizializzazione;

```
        ORG         0x00

        bsf         STATUS,RP0 ;BANCO       1
        movlw       11001B
        movwf       TRISA & 0x7F
        movlw       00000000B
        movwf       TRISB & 0x7F
        bcf         STATUS,RP0 ;BANCO  0
```

Impostiamo il PORTA come ingresso per i pin RA0 RA3 RA4 e come uscita i pin RA1 RA2, e tutto il PORTB come uscita

```
        movlW       30H
        movwf       BYTE1
        movlW       30H
        movwf       BYTE2
        movlW       30H
        movwf       BYTE3
        movlW       38H
        movwf       BYTE4
        movlW       08H
```

```
movwf   BYTE5
movlW   01H
movwf   BYTE6
movlW   06H
movwf   BYTE7
movlW   0FH
movwf   BYTE8
```

Carico le variabili riservate in precedenza con i codici per l'inizializzazione

```
clrf    PORTB       ;METTO TUTTO A 0 IL PORTB
bcf     PORTA,1     ;RS = 0  PER INIZIALIZZARE
bcf     PORTA,2     ;ENABLE = 0



bsf     PORTA,2     ;ENABLE = 1
movf    BYTE1,W     ;1°  BYTE
movwf   PORTB       ;1°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE2,W     ;2°  BYTE
movwf   PORTB       ;2°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE3,W     ;3°  BYTE
movwf   PORTB       ;3°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE4,W     ;4°  BYTE
movwf   PORTB       ;4°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE5,W     ;5°  BYTE
movwf   PORTB       ;5°  BYTE
bcf     PORTA,2     ;ENABLE = 0
```

```
call        DELAY1
call        DELAY1

bsf         PORTA,2         ;ENABLE = 1
movf        BYTE6,W         ;6°  BYTE
movwf       PORTB           ;6°  BYTE
bcf         PORTA,2         ;ENABLE = 0
call        DELAY1
call        DELAY1

bsf         PORTA,2         ;ENABLE = 1
movf        BYTE7,W         ;7°  BYTE
movwf       PORTB           ;7°  BYTE
bcf         PORTA,2         ;ENABLE = 0
call        DELAY1
call        DELAY1

bsf         PORTA,2         ;ENABLE = 1
movf        BYTE8,W         ;8°  BYTE
movwf       PORTB           ;8°  BYTE
bcf         PORTA,2         ;ENABLE = 0
call        DELAY1
call        DELAY1
```

carico dal microcontrollore al display i primi 8 byte che servono a portare il cursore del display nella posizione iniziale in alto a sinistra

```
bsf         PORTA,1         ; RS = 1 per scrivere
```

Abilito il display per ricevere i codici per visualizzare i caratteri.

MAIN LOOP DEL PROGRAMMA PER LA RICEZIONE

```
RICE        clrw
            clrf        STATUS
            clrf        VISUAL
            clrf        TOGLI
            clrf        PORTB
```

Inizializzo il PIC pulendo Status,Visual, Togli, e il PORTB

```
ARRIVA      btfss       PORTA,0         ;Verifico che sul pin 0 del PORTA arrivino i
            goto        ARRIVA          ; segnali che indicano di iniziare la ricezione
            call        CENTRO
            call        DELAY
            btfsc       PORTA,0
            call        DELAY
            call        DELAY           ; → la chiamo 16 volte
```

RICEVO IL DATO SERIALE E LO SALVO IN MEMORIA

```
            movf        PORTA,w

            movwf       VISUAL          ; 1^ bit
            bcf         VISUAL,1
            bcf         VISUAL,2
            bcf         VISUAL,3
            bcf         VISUAL,4
            bcf         VISUAL,5
            bcf         VISUAL,6
            bcf         VISUAL,7
            rlf         VISUAL          ; DA 0 VADO A 1

            call        DELAY           ; →a chiamo 32 volte



            movf        PORTA,w         ; 2^ bit
            movwf       TOGLI
            bcf         TOGLI,1
            bcf         TOGLI,2
            bcf         TOGLI,3
            bcf         TOGLI,4
```

```
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F    ; DA 1 VADO A 2

call        DELAY           ; →la chiamo 32 volte

call        DELAY
call        DELAY



movf        PORTA,w         ; 3^ bit
movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F

call        DELAY           ; → la chiamo 32 volte



movf        PORTA,w         ; 4^ bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F
```

```
call        DELAY           ; →la chiamo 32 volte



movf        PORTA,w         ; 5^ bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F

call        DELAY           ; → la chiamo 32 volte


movf        PORTA,w         ; 6^ bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F

call        DELAY           ; → la chiamo 32 volte


movf        PORTA,w         ; 7^ bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
```

```
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F    ; DA 6 VADO A 7

call        DELAY           ; →la chiamo 32 volte


movf        PORTA,w         ; 8^ bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F


call        DELAY           ; → la chiamo 32 volte




call        DELAY           ; →la chiamo 32 volte
```

SCRITTURA DEL DATO NEL DISPLAY

```
bsf         PORTA,2         ; ENABLE = 1
call        DELAY
movf        VISUAL,W
movwf       PORTB           ; visualizza il byte
call        DELAY
bcf         PORTA,2         ; ENABLE = 0


call        DELAY           ; → la chiamo 32 volte

goto        RICE
```

SUBROUTINE

```
DELAY1      movlw       11111111B
            movwf       CONT
QUI         decfsz      CONT,F
            goto        QUI
            return

FREQ        movlw       00001000B
            movwf       CONT2
QUII        decfsz      CONT2,F
            goto        QUII
            return

DELAY       movlw       00010100B
            movwf       CONT1
QUIII       decfsz      CONT1,F
            goto        QUIII
            return

CENTRO      movlw       00000011B
            movwf       CONT3
QUIIII      decfsz      CONT3,F
            goto        QUIIII
            return

        END
```

TX TASTIERINO

Questa scheda serve per rilevare il tasto premuto su un tastierino 4x4 tasti a matrice.
Rilevato quale tasto è stato premuto, fornisce in uscita il valore associato al tasto e salvato in memoria, sul RB0 in seriale.

```
ORG         0X0C
CONT1       RES         1
CONT8       RES         1
DATOB       RES         1
```

Riservo le variabili che mi servono durante il programma

```
      ORG       0x00

        bsf     STATUS,RP0 ; banco  1
        movlw   11111111B    ; PORTA  in
        movwf   TRISA
        movlw   00000000B    ; PORTB  out
        movwf   TRISB
        bcf     STATUS,RP0 ; banco  0
```

Imposto il PortB come uscita, e il PortA come ingresso

MAIN LOOP DEL PROGRAMMA TX-TASTIERINO########

```
INIZIO      clrw                    ;
        clrf        DATOB           ; RESET INIZIALI
        clrf        PORTA           ;
        clrf        PORTB           ;
        movlw       8               ; metto 8 nell' accumulatore
        movwf       CONT8           ; metto 8 in    CONT3
        bcf         PORTB,0         ; bit PORTA1 basso
        clrf        STATUS
```

Con le prime righe di codice si inizializza il microcontrollore e poi si inizia a testare se un qualsiasi tasto è stato premuto. Questo è il corpo centrale del programma, e quando riconosce che un tasto è stato premuto carica in una variabile temporanea il dato chiamando la subroutine relativa al tasto premuto. Quando lo stack-pointer ritorna dalla subroutine si trova un'altra chiamata che punta alla routine che mi serializza e mette in uscita il dato della variabile.

```
TEST    bsf         PORTB,1         ; T E S T   T A S T I    L O O P
        btfss       PORTA,0
        goto        a
        call        GH
        call        OUT
        goto        INIZIO
a       btfss       PORTA,1
        goto        bp
        call        GB
        call        OUT
        goto        INIZIO
bp      btfss       PORTA,2
        goto        c
        call        GE
        call        OUT
```

```
        goto    INIZIO
c       btfss   PORTA,3
        goto    e
        call    GI
        call    OUT
        goto    INIZIO
e       bcf     PORTB,1

        bsf     PORTB,2
        btfss   PORTA,0
        goto    f
        call    CH
        call    OUT
        goto    INIZIO
f       btfss   PORTA,1
        goto    g
        call    CB
        call    OUT
        goto    INIZIO
g       btfss   PORTA,2
        goto    h
        call    CE
        call    OUT
        goto    INIZIO
h       btfss   PORTA,3
        goto    l
        call    CI
        call    OUT
        goto    INIZIO
l       bcf     PORTB,2


        bsf     PORTB,3
        btfss   PORTA,0
        goto    m
        call    DH
        call    OUT
        goto    INIZIO
m       btfss   PORTA,1
        goto    n
        call    DiB
        call    OUT
        goto    INIZIO
n       btfss   PORTA,2
        goto    o
        call    DiE
        call    OUT
        goto    INIZIO
o       btfss   PORTA,3
        goto    q
```

```
        call    DI
        call    OUT
        goto    INIZIO
q       bcf     PORTB,3

        bsf     PORTB,4
        btfss   PORTA,0
        goto    r
        call    FH
        call    OUT
        goto    INIZIO
r       btfss   PORTA,1
        goto    s
        call    FB
        call    OUT
        goto    INIZIO
s       btfss   PORTA,2
        goto    t
        call    FE
        call    OUT
        goto    INIZIO
t       btfss   PORTA,3
        goto    v
        call    FI
        call    OUT
        goto    INIZIO
v       bcf     PORTB,4
        goto    TEST    ; SE NN HA TROVATO TASTI PREMUTI ---> TORNA A
                          TESTARE DALL'INIZIO
```

SUBROUTINE

CARICO IN DATOB IL VALORE DEL TASTO PREMUTO

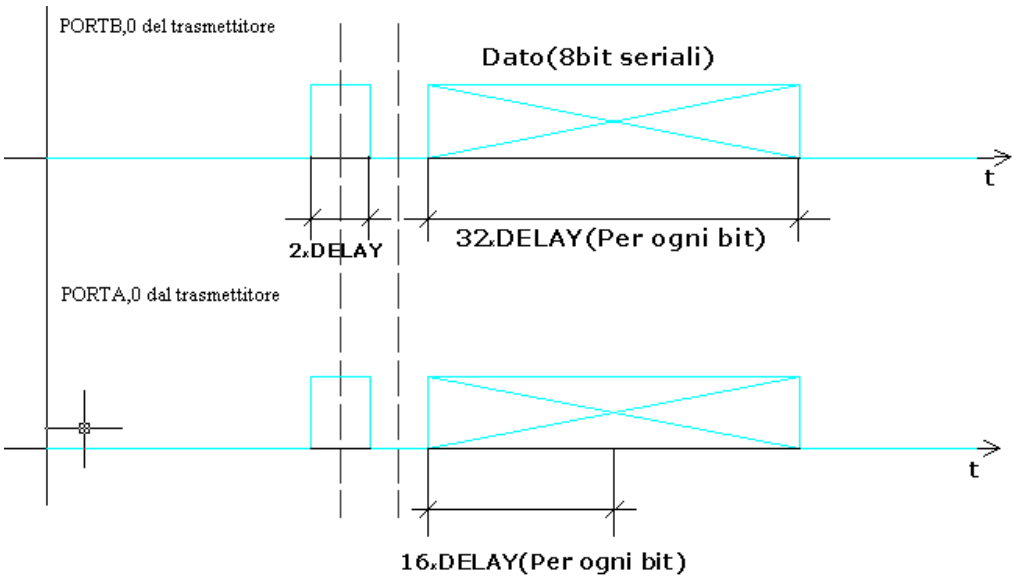
```
CI      movlw   01001100B   ; 2
        movwf   DATOB
        return

CH      movlw   01101100B   ; 6
        movwf   DATOB
        return

CE      movlw   10000010B   ; A
        movwf   DATOB
        return

CB      movlw   10100010B   ; E
        movwf   DATOB
```

```
        return

GI      movlw   11001100B  ; 3
        movwf   DATOB
        return

GH      movlw   11101100B  ; 7
        movwf   DATOB
        return

GE      movlw   01000010B  ; B
        movwf   DATOB
        return

GB      movlw   01100010B  ; F
        movwf   DATOB
        return


FI      movlw   00001100B  ; 0
        movwf   DATOB
        return

FH      movlw   00101100B  ; 4
        movwf   DATOB
        return

FE      movlw   00011100B  ; 8
        movwf   DATOB
        return

FB      movlw   11000010B  ; C
        movwf   DATOB
        return


DI      movlw   10001100B  ; 1
        movwf   DATOB
        return

DH      movlw   10101100B  ; 5
        movwf   DATOB
        return

DiE     movlw   10011100B  ; 9
        movwf   DATOB
        return
```

```
DiB     movlw       00100010B    ; D
        movwf       DATOB
        return

;-------------------------------------------------------------------------------------------
```

SERIALIZZO E METTO IN USCITA SUL PIN RB0

```
OUT     bsf         PORTB,0      ;genera il bit di riconoscimento per il ricevitore
        call        DELAY
        call        DELAY


        bcf         PORTB,0
        call        DELAY
        call        DELAY
```



Segnali di temporizzazione del programma del tastierino

```
; T R A S M I S S I O N E
; I N I Z I O   A   T R A S M E T T E R E   G L I   8   B I T


SALT    movf        DATOB,w
        movwf       PORTB

        rrf         DATOB,F          ; ruoto a sinistra
        call        DELAY            ; →la chiamo 32 volte
```

```
        decfsz      CONT8,F             ; dec 8 poi torno a INIZIO
        goto        SALT
        bcf         PORTB,0
        call        DELAY
        return

;-------------------------------------------------------------------------------

SUBROUTINE DEL RITARDO

DELAY   movlw       11111111B
        movwf       CONT1
QUI     decfsz      CONT1,F
        goto        QUI
        return
;-------------------------------------------------------------------------------
END
```

PROGETTO WIRELESS

RICEVITORE DISPLAY WIRELESS

Utilizzando la scheda Rx-display e collegandogli un modulo RX senza fili si possono sincronizzare la scheda TX con quella RX per trasmettere i dati in seriale via etere. Solitamente questi moduli in ingresso devono avere un'onda quadra di frequenza max 4Khz. Il PIC del RX deve sincronizzarsi sulla frequenza dell'onda quadra che riceve dal TX, e aspettare fino a che non riceve il segnale di SOT (start of transmition). Ricevere il Dato in seriale, salvarlo in una variabile temporanea e inviarlo ad display.

```
        PROCESSOR 16F84
        RADIX           DEC
        INCLUDE         "P16F84.INC"
        __CONFIG    3FF3H



        ORG         0x0C

BYTE1       RES         1
BYTE2       RES         1
BYTE3       RES         1
BYTE4       RES         1
BYTE5       RES         1
BYTE6       RES         1
BYTE7       RES         1
BYTE8       RES         1
```

```
CONT        RES         1
CONT1       RES         1
CONT2       RES         1
CONT3       RES         1
TOGLI       RES         1
VISUAL      RES         1
```

Mi riservo le varabili che mi serviranno lungo lo svolgimento del programma

```
        ORG         0x00

        bsf         STATUS,RP0 ;BANCO       1
        movlw       11001B
        movwf       TRISA & 0x7F
        movlw       00000000B
        movwf       TRISB & 0x7F
        bcf         STATUS,RP0 ;BANCO  0
```

Imposto i porti del PIC

```
        movlW               30H
        movwf               BYTE1
        movlW               30H
        movwf               BYTE2
        movlW               30H
        movwf               BYTE3
        movlW               38H
        movwf               BYTE4
        movlW               08H
        movwf               BYTE5
        movlW               01H
        movwf               BYTE6
        movlW               06H
        movwf               BYTE7
        movlW               0FH
        movwf               BYTE8
```

```
        clrf        PORTB
        bcf         PORTA,1         ;RS = 0  PER INIZIALIZZARE
         bcf        PORTA,2         ;ENABLE = 0
```

```
        bSf         PORTA,2         ;ENABLE = 1
        movf        BYTE1,W         ;1°  BYTE
        movwf       PORTB           ;1°  BYTE
```

```
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE2,W     ;2°  BYTE
movwf   PORTB       ;2°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE3,W     ;3°  BYTE
movwf   PORTB       ;3°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE4,W     ;4°  BYTE
movwf   PORTB       ;4°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE5,W     ;5°  BYTE
movwf   PORTB       ;5°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE6,W     ;6°  BYTE
movwf   PORTB       ;6°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE7,W     ;7°  BYTE
movwf   PORTB       ;7°  BYTE
bcf     PORTA,2     ;ENABLE = 0
call    DELAY1
call    DELAY1

bsf     PORTA,2     ;ENABLE = 1
movf    BYTE8,W     ;8°  BYTE
movwf   PORTB       ;8°  BYTE
bcf     PORTA,2     ;ENABLE = 0
```

```
        call    DELAY1
        call    DELAY1
```

carico dal microcontrollore al display i primi 8 byte che servono a portare il cursore del display nella posizione iniziale in alto a sinistra

```
        bsf     PORTA,1         ; RS = 1 per scrivere
```

Abilito il display per ricevere i codici per visualizzare i caratteri.

######## MAIN LOOP DEL PROGRAMMA PER LA RICEZIONE ########

```
RICE    clrw
        clrf    STATUS
        clrf    VISUAL
        clrf    TOGLI
        clrf    PORTB
```

Inizializzo il PIC pulendo Status,Visual, Togli, e il PORTB

```
ARRIVA  btfss   PORTA,0         ; MaInLooP
        goto    BASSO
```

Con queste 2 righe di codice, il Pic si accorge se l'onda quadra che riceve è a livello logico alto o basso, per poi sincronizzarsi, e controllare se arriva un codice di SOT.

```
ALTO    call    CENTRO          ; mi porto al centro del _-_-_
        btfss    PORTA,0
        goto    ARRIVA
        call    FREQ
        btfsc   PORTA,0
        goto    TEST
        call    FREQ
        goto    ALTO
```

Utilizzando delay con valori uguali a quelli usati nella scheda TX controllo che l'onda quadra che ricevo non cambi frequenza, e la controllo al centro del semiperiodo, se trovo che la frequenza è cambiata salto TEST

```
BASSO   call    CENTRO          ;mi porto al centro del _-_-_
        btfsc   PORTA,0
        goto    TEST
        call    FREQ
        call    CENTRO
        btfss   PORTA,0
```

```
        goto    ARRIVA
        call    FREQ
        call    CENTRO
        goto    BASSO
```

Utilizzando delay con valori uguali a quelli usati nella scheda TX controllo che l'onda quadra che ricevo non cambi frequenza, e la controllo al centro del semiperiodo, se trovo che la frequenza è cambiata salto TEST

```
TEST    call    DELAY
        call    DELAY
        btfsc   PORTA,0
        goto    ARRIVA
```

Ulteriore test per verificare che il segnale è un SOT e non magari un disturbo elettrico

```
        call    DELAY
        call    DELAY           ; →la chiamo 16 volte
```

RICEVO IL DATO SERIALE E LO SALVO IN MEMORIA

```
    movf    PORTA,w

    movwf   VISUAL          ; 1 bit
    bcf     VISUAL,1
    bcf     VISUAL,2
    bcf     VISUAL,3
    bcf     VISUAL,4
    bcf     VISUAL,5
    bcf     VISUAL,6
    bcf     VISUAL,7
    rlf     VISUAL          ; DA 0 VADO A 1

    call    DELAY           ; → la chiamo 32 volte


    movf    PORTA,w         ; 2 bit

    movwf   TOGLI
    bcf     TOGLI,1
    bcf     TOGLI,2
    bcf     TOGLI,3
    bcf     TOGLI,4
    bcf     TOGLI,5
    bcf     TOGLI,6
    bcf     TOGLI,7
    movf    TOGLI,W
```

```
iorwf   VISUAL,F
rlf     VISUAL,F        ; DA 1 VADO A 2

call    DELAY           ; →la chiamo 32 volte


movf    PORTA,w         ; 3 bit
movwf   TOGLI
bcf     TOGLI,1
bcf     TOGLI,2
bcf     TOGLI,3
bcf     TOGLI,4
bcf     TOGLI,5
bcf     TOGLI,6
bcf     TOGLI,7
movf    TOGLI,W

iorwf   VISUAL,F
rlf     VISUAL,F

call    DELAY           ; → la chiamo 32 volte


movf    PORTA,w         ; 4 bit

movwf   TOGLI
bcf     TOGLI,1
bcf     TOGLI,2
bcf     TOGLI,3
bcf     TOGLI,4
bcf     TOGLI,5
bcf     TOGLI,6
bcf     TOGLI,7
movf    TOGLI,W

iorwf   VISUAL,F
rlf     VISUAL,F

call    DELAY           ; → la chiamo 32 volte


movf    PORTA,w         ; 5 bit

movwf   TOGLI
bcf     TOGLI,1
bcf     TOGLI,2
bcf     TOGLI,3
bcf     TOGLI,4
bcf     TOGLI,5
```

```
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F

call        DELAY           ; → la chiamo 32 volte



movf        PORTA,w         ; 6 bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F

call        DELAY           ; →la chiamo 32 volte



movf        PORTA,w         ; 7 bit

movwf       TOGLI
bcf         TOGLI,1
bcf         TOGLI,2
bcf         TOGLI,3
bcf         TOGLI,4
bcf         TOGLI,5
bcf         TOGLI,6
bcf         TOGLI,7
movf        TOGLI,W

iorwf       VISUAL,F
rlf         VISUAL,F    ; DA 6 VADO A 7

call        DELAY           ; → la chiamo 32 volte


movf        PORTA,w         ; 8 bit
```

```
        movwf   TOGLI
        bcf     TOGLI,1
        bcf     TOGLI,2
        bcf     TOGLI,3
        bcf     TOGLI,4
        bcf     TOGLI,5
        bcf     TOGLI,6
        bcf     TOGLI,7
        movf    TOGLI,W

        iorwf   VISUAL,F


        call    DELAY           ; →la chiamo 32 volte


        call    DELAY           ; → la chiamo 32 volte
```

SCRITTURA DEL DATO NEL DISPLAY

```
        bsf     PORTA,2         ; ENABLE = 1
        call    DELAY
        movf    VISUAL,W
        movwf   PORTB           ; visualizza il byte
        call    DELAY
        bcf     PORTA,2         ; ENABLE = 0



        call    DELAY           ; →la chiamo 32 volte

        goto    RICE
```

SUBROUTINE

```
DELAY1  movlw   11111111B
        movwf   CONT
QUI     decfsz  CONT,F
        goto    QUI
        return

FREQ    movlw   00001000B       ;ritardo software per verificare la frequenza dell'
        movwf   CONT2           ;onda quadra
QUII    decfsz  CONT2,F
```

```
            goto        QUII
            return

DELAY       movlw       00010100B
            movwf       CONT1
QUIII       decfsz      CONT1,F
            goto        QUIII
            return


CENTRO      movlw       00000011B       ;ritardo software per centrare il controllo sulla
            movwf       CONT3           ;frequenza dell'onda quadra
QUIIII      decfsz      CONT3,F
            goto        QUIIII
            return

            END
```

TX WIRELESS

```
        PROCESSOR 16F84
        RADIX           DEC
        INCLUDE         "P16F84.INC"
        __CONFIG    3FF3H


        ORG         0X0C

CONT1   RES         1
CONT2   RES         1
CONT3   RES         1
DATOB   RES         1
```

Mi riservo le varabili che mi serviranno lungo lo svolgimento del programma

```
    ORG         0x00

      bsf       STATUS,RP0          ; banco  1
      movlw     00010B              ; PORTA0 out  PORTA1/2/3/4  in
      movwf     TRISA
      movlw     11111111B           ; PORTB1/2/3/4/5/6/7/8      in
      movwf     TRISB
      bcf       OPTION_REG,7        ; pull up  porto B
      bcf       STATUS,RP0          ; banco  0
```

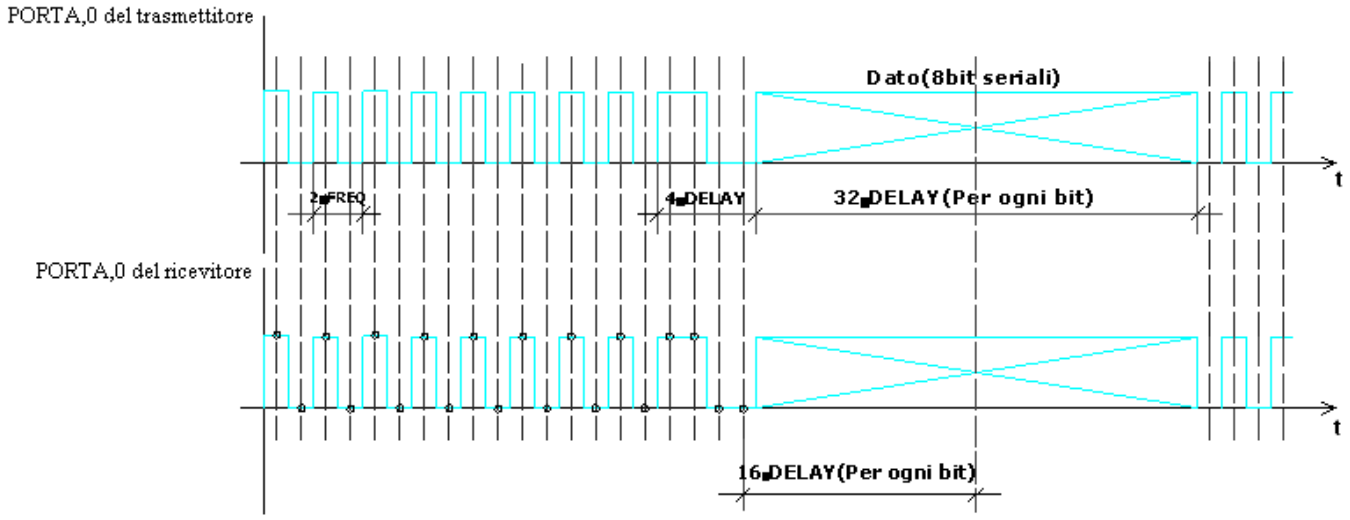
Imposto i porti del Pic

MAIN LOOP DEL PROGRAMMA TX-WIRELESS########

```
INIZIO      bsf         PORTA,1
            bcf         PORTA,0

            clrf        STATUS
            clrf        DATOB

            movlw       8               ; metto 8 nell' accumulatore
            movwf       CONT3           ; metto 8 in    CONT3
```

Ciclo che genera l'onda quadra

```
TRAS        bsf         PORTA,0         ;metto a 1 il PortA0
            call        FREQUE          ;chiamo subroutine ritardo
            bcf         PORTA,0         ;metto a 0 il PortA0
            call        FREQUE          ;chiamo subroutine ritardo
            btfsc       PORTA,1         ;se premo il tasto per inviare il dato salto 1-istr.
            goto        TRAS            ; se non premo il tasto salta a TRAS


            bsf         PORTA,0         ;    ###### codice   SOT ######
            call        DELAY
            call        DELAY




            bcf         PORTA,0

            call        DELAY
            call        DELAY
```



Segnali di temporizzazione del progetto wireless

```
        movf    PORTB,W         ; leggo PORTB in accumulatore
        movwf   DATOB           ; da W  a DATOB

SALT    movf    DATOB,w
        movwf   PORTA

        rrf     DATOB,F         ; ruoto a sinistra
        call    DELAY           ; → la chiamo 32 volte


        decfsz  CONT3,F         ; dec 8 poi torno a INIZIO
        goto    SALT
        bcf     PORTA,0
        call    DELAY
        goto    INIZIO
```

SUBROUTINE

```
FREQUE  movlw   00001000B       ;ritardo per generare onda quadra
        movwf   CONT2
QUII    decfsz  CONT2,F
        goto    QUII
        return

DELAY   movlw   00010100B       ;ritardo per SOT
        movwf   CONT1
QUI     decfsz  CONT1,F
        goto    QUI
        return
        END
```