Microntrollori: il PIC16F84

Cercando su Internet si trovano numerose notizie che trattano dei microcontrollori in generale. Ad esempio, cercando nell'autorevole sito di Wikipedia è possibile trovare la generica definizione:

Un microcontrollore o microcontroller, detto anche computer single chip è un sistema a microprocessore completo, integrato in un solo chip, progettato per ottenere la massima autosufficienza funzionale ed ottimizzare il rapporto prezzo-prestazioni per una specifica applicazione, a differenza, ad esempio, dei microprocessori impiegati nei personal computer, adatti per un uso più generale.
I microcontroller sono la forma più diffusa e più invisibile di computer. Comprendono la CPU, un certo quantitativo di memoria RAM e memoria ROM (può essere PROM, EPROM, EEPROM o FlashROM) e una serie di interfacce di I/O (input/output) standard, fra cui molto spesso bus (I^2C,SPI,CAN,LIN). Le periferiche integrate sono la vera forza di questi dispositivi: si possono avere convertitori ADC e convertitori DAC multicanale, timer/counters, USART, numerose porte esterne bidirezionali bufferizzate, comparatori, PWM.

Un interessante e diffusa categoria di microcontrollori è la famiglia dei PIC (programmable interrupt controller). Questi, come tutti i microcontrollori del resto, si presentano esternamente come dei circuiti integrati ma, come già letto in precedenza, il package ingloba al suo interno diversi dispositivi tipici di un sistema a microprocessore come:

- una CPU (central process unit) per l'interpretazione dei programmi utenti;
- un'area di memoria di tipo PROM (programmable read only memory) per memorizzare il programma da eseguire;
- un'area di memoria di tipo RAM (random acces memory) per memorizzare le variabili del programma e le elaborazioni della CPU;
- una linea per i segnali di I/O per ricevere in ingresso i segnali provenienti dai sensori e produrre in uscita i segnali diretti agli attuatori (le linee di I/O possono essere dedicate a segnali analogici e/o binari);
- altri dispositivi vari tra cui contatori, bus e porte per la comunicazione (quest'ultima può essere implementata nel chip sia nella forma parallela per la comunicazione, come ad esempio la PIA, che nella forma seriale più comune, come ad esempio la USART);

Detto ciò, se non specifichiamo subito le caratteristiche interne di un microcontrollore, sia esse in termini di architettura interna che in termini di hardware rischiamo davvero di confonderlo con un vero e proprio microprocessore. Nel corso degli anni si sono succedute

diverse famiglie di microcontrollori, ognuna di queste con una propria struttura organizzativa interna che tuttavia possiamo ritenere pressapoco simile. Per questo motivo è possibile associare ad un microcontrollore un architettura interna di riferimento che ci permetta di iniziarne ad apprezzare le potenzialità:

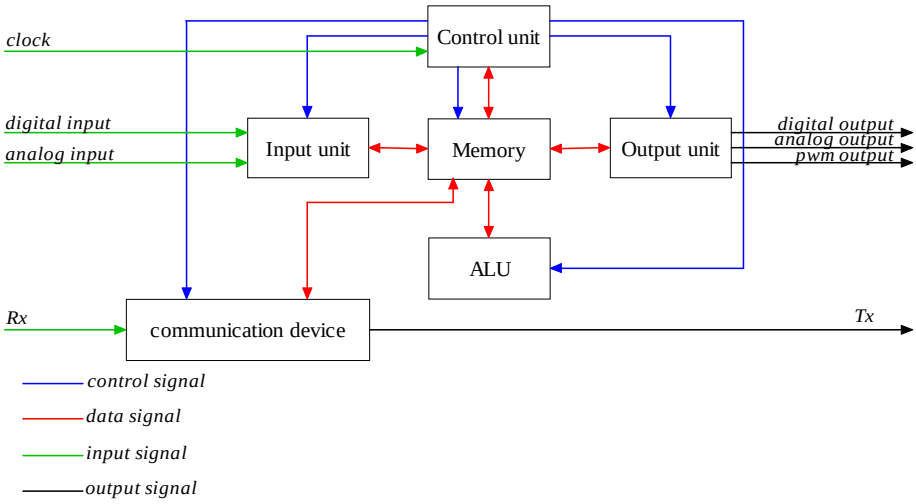


Fig.1: architettura generale di un microcontrollore

Tutti i dispositivi elencati come periferiche dei microcontrollori possono trovare una loro collocazione in uno dei blocchi illustrati in figura. Ogni linea ha il suo significato. La disponibilità di così tanti dispositivi in uno spazio estremamente contenuto offre al progettista di sistemi per il controllo un valido strumento per la realizzazione della strategia di controllo. Un microcontrollore può essere caratterizzato internamente dal numero di bit che è capace di trattare (esistono microcontrollori da 4,8,16 e 32 bit) mentre si caratterizza, esternamente, dal numero di pin che il package offre al mondo estreno per l'interconnessione verso altri componenti elettrici. I modelli più piccoli sono dotati di soli 8 pin (come ad esempio la famiglia PIC12c5xx) quelli più grandi, invece, possono avere anche 40 pin (coma la famiglia PIC17Cxx). Vista la generalità dell'argomento ho deciso di esaminare in queste note uno specifico tipo di microcontrollore. La scelta è ricaduta sul PIC16F84, si tratta di un modello di microcontrollore di fascia media sufficientemente semplice da poter essere studiato. Il packake è dotato di 18 pin di cui 13 pin dedicati alle linee di I/O. Ed ancora, come tutti i microcontrollori, anche il PIC16F84 dispone di una memoria di tipo PROM (programmable read only memory) che in questo caso è anche detta EEPROM (elettrical eraseble programmable read only memory) poichè può essere riscritta tutte le volte che vogliamo a patto di cancellare il precedente contenuto.

Fig.2: PIC16F84

I 18 pin del PIC16F84 (che da qui in avanti per brevità chiameremo semplicemente PIC) sono disposte su due file di 9 pin ciascuna. Ogni pin ha il proprio significato:

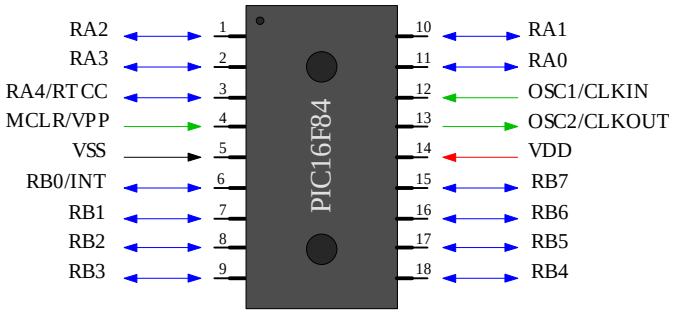


Fig.3: significato dei pin del PIC16F84

tra questi distinguiamo in blu i pin appartenenti alle linee di I/O; in verde i pin utili al funzionamento del PIC come ad esempio quello relativo al segnale di clock (pin 12) ed alla funzione di reset (pin 4); in rosso e nero i pin relativi alla tensione di alimentazione che per il PIC in oggetto è compresa tra 2.0 e 6.0 V.

Pin:	Descrizione:
RA0	Linea programmibile per l'I/O, è il bit 0 della porta A.
RA1	Linea programmibile per l'I/O, è il bit 1 della porta A.
RA2	Linea programmibile per l'I/O, è il bit 2 della porta A.
RA3	Linea programmibile per l'I/O, è il bit 3 della porta A.
RA4	Pin multifunzione, può rappresentare il bit 4 della porta A oppure un clock esterno per il contatore interno RTCC.
RB0	Linea programmibile per l'I/O, è il bit 0 della porta B.
RB1	Linea programmibile per l'I/O, è il bit 1 della porta B.
RB2	Linea programmibile per l'I/O, è il bit 2 della porta B.
RB3	Linea programmibile per l'I/O, è il bit 3 della porta B.
RB4	Linea programmibile per l'I/O, è il bit 4 della porta B.
RB5	Linea programmibile per l'I/O, è il bit 5 della porta B.
RB6	Linea programmibile per l'I/O, è il bit 6 della porta B.
RB7	Linea programmibile per l'I/O, è il bit 7 della porta B.
OSC1/CLKIN	Qualora il clock interno da 4MHZ non sia sufficiente alla strategia di controllo costituisce l'ingresso per un oscillatore esterno.
OSC2/CLKOUT	Qualora sia necessario costituisce il segnale di clock in uscita dal PIC, può essere usato come ingresso per altri microcontrollori.

VDD	Terminale positivo per l'alimentazione del PIC, può assumere valori compresi tra 2.0 e 6.0V.
VSS	Terminale negativo per l'alimentazione del PIC.
MCLR/VPP	Svolge la funzione di reset (master clear) mentre in fase di programmazione costituisce l'ingresso per la tensione di programmazione VPP.

Fig.4: tabella descrittiva dei pin del PIC16F84

Ai pin OSC1/CLKIN e OSC2/CLKOUT sono collegati i segnali di clock per la temporizzazione dei cicli di funzionamento interni al chip. Da tale frequenza dipende la velocità con cui il PIC esegue le istruzioni del programma. Per il PIC in questione tale frequenza di clock può essere al massimo di 10MHz, nel package è già presente un clock da 4MHz.
La scrittura del programma da inserire nel PIC avviene in linguaggio assembler. Un programma per il PIC è una sequenza di istruzioni di 14 bit ciascuna. Ogni istruzione ha il proprio codice, detto opcode (operative code). Tuttavia, nella programmazione è possibile fare riferimento alle istruzioni mediante codici mnemonici piuttosto che ricordare gli opcode di ogni singola istruzione.
La memoria del PIC è di 1024 (1K) locazioni da 14 bit, ogni locazione di memoria può contenere sia una singola istruzione che la coppia istruzioni/dato. Ad esempio, l'opcode 0100H (in notazione esadecimale) è ricordata mnemonicamente dall'utente dalla stringa CLRW (azzera il registro W). Altre sigle mnemoniche, la definizione di variabili e di etichette permettono all'utente di ordinarle per formare complessi programmi utente, i cosiddetti file sorgenti.
Il compilatore assembler, in seguito, trasforma il file sorgente contenente il programma utente in file binario (il PIC nella realtà comprende il solo linguaggio binario e/o comunque un linguaggio ad esso somigliante). Durante la compilazione, l'assemblatore per il PIC produce come output diversi file:

- un file con estensione .ERR contenente la lista degli errori riscontrati nel programma utente e quindi nel file sorgente;
- un file con estensione .LST contenente il codice sorgente del programma con a seguito la relativa traduzione (fino al primo errore riscontrato) fatta da soli codici opcode;
- un file con estensione .HEX contenente la sola traduzione del programma sorgente scritto dall'utente;

Il file .HEX è il programma da caricare nella memoria del PIC. Come l'estensione lascia intendere si tratta di un file che nella realtà non è ancora in formato binario, l'unico conosciuto dal PIC. Quest'ultimo, infatti, costituisce l'input per il programmatore del PIC. Sul programmatore viene inserito mediante zoccolo il PIC da programmare. In fase di programmazione il file .HEX è tradotto dal programmatore in una sequenza di opcode che sono quindi trasferiti nella memoria EEPROM del PIC.

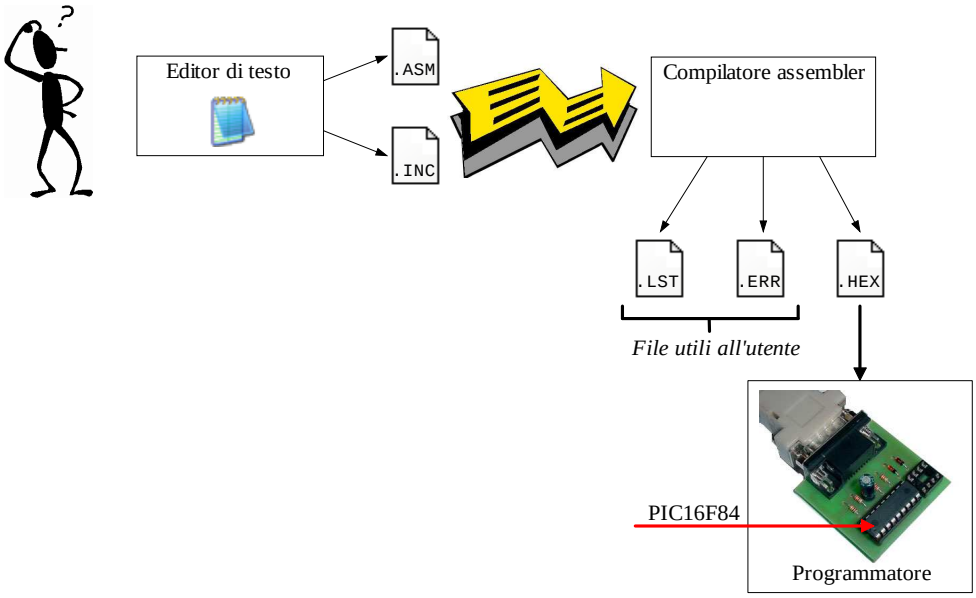


Fig.5: descrizione dei passi da eseguire per la programmazione del PIC

Per comodità faremo riferimento ad un architettura semplificata del PIC che tratta solo i registri ed i dispositivi principali:

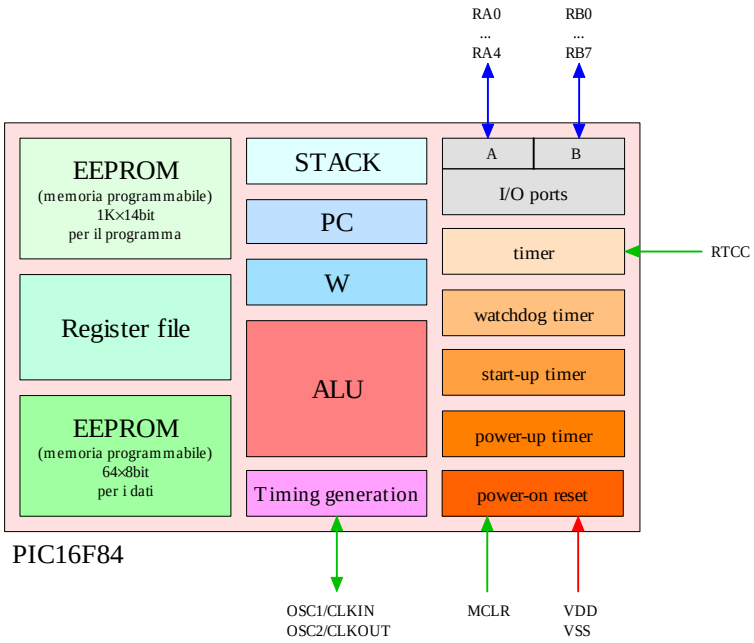


Fig.6: architettura semplificata del PIC16F84

L'area di memoria è suddivisa in due aree di memoria: un'area di memoria per il programma utente ed una, invece, per i dati e le variabili a runtime. Entrambe le due aree di memoria sono di tipo EEPROM. Quella dedicata al programma è di 1K locazioni di 14 bit, pertanto il programma più lungo che possiamo inserire nel PIC può avere al massimo 1024 istruzioni. Gli indirizzi di memoria della EEPROM sono contenuti nell'intervallo 0000H/03FFH. La prima locazione di memoria, quella all'indirizzo 0000H, deve contenere la

prima istruzione che il PIC esegue ogni volta che subisce un reset ed è per questo motivo detta reset vector. La prima direttiva `ORG indirizzo` di un programma occupa sempre il reset vector ed indica l'indirizzo iniziale del programma qualora questo debba necessariamente partire da un indirizzo diverso da 0000H.

Il register file è un area di memoria RAM visibile dal programma che quindi può scriverne e leggerne il contenuto. Non tutti i registri dell'area RAM register file possono essere usati liberamente, alcuni di questi sono infatti riservati al PIC che qui scrive le proprie cose, ad esempio variabili temporanee. Questo spazio di memoria è suddiviso in due pagine: la pagina 0 (da 00H a 2FH) e la pagina 1 (da 80H a AFH). Le prime 12 locazioni della pagina 0 (quindi da 00Ha 0BH) e della pagina 1 (quindi da 80H a 8BH) sono riservate, come già detto, per il funzionamento del PIC e non possono essere adoperate per altri scopi. Pertanto, della pagina 0 e della pagina 1 sono utilizzabili solo 36 locazioni di memoria per memorizzare variabili, contatori e ogni altra cosa utile al programma. Alcuni registri interni al PIC fotografano lo stato di alcuni dispositivi e registrano il risultato delle recenti operazioni aritmetiche e logiche (è questo il caso del registro di stato di soli 8 bit). Ciascuno di questi registri è poi mappato in un indirizzo di memoria sicchè in ogni momento se ne può apprenderne il valore e/o modificarlo se il PIC lo prevede.

L'ALU (arithmetic and logic unit, unità logico aritmetica) è delegata alla manipolazione dei dati durante l'esecuzione del programma. Da questo componente scaturisce la potenza di calcolo del microcontrollore. La ALU del PIC, ad esempio, è in grado di operare su operandi di 8 bit al massimo. Il registro W è direttamente connesso alla ALU che quindi vi accede senza fornire alcun indirizzo di memoria come invece accade per tutte le altre periferiche che, ricordiamo, sono mappate in determinati indirizzi. Questo registro è molto usato nei programmi (alcune istruzioni si riferiscono infatti al registro W che ne leggono e/o scrivono il contenuto). Supponiamo di voler inserire nella locazione di memoria 0CH del register file il valore 01H. Cercando tra le istruzioni messe a disposizione dal PIC ci accorgiamo subito che non esiste un unica istruzione in grado di effettuare questa operazione ma dobbiamo ricorrere necessariamente al registro W (per questo motivo detto anche accumulatore) ed usare due istruzioni in sequenza. Ecco spiegato il motivo: l'opcode di una istruzione non può essere più grande di 14 bit (ricordate le 1024 locazioni di 14 bit nella memoria EEPROM?) mentre a noi, in questo caso, servirebbero 8 bit per indicare il valore da inserire nella locazione di memoria (01H), 7 bit per specificare in quale locazione di memoria scrivere il suddetto valore (che per noi è l'indirizzo 0CH) e 6 bit per indicare al PIC quale istruzione usare dal set di istruzioni disponi della ALU (il PIC da noi preso in considerazione ha 35 istruzioni). Detto ciò, quello che solitamente si fa, come già detto, è l'uso consecutivo di due istruzioni assembler:

```
MOVLW   01H
MOVWF   0CH
```

La prima di queste inserisce nell'accumulatore (il regisro W) il valore `01H` (istruzione `movlw`, move literal to W) mentre la seconda copia il contenuto dell'accumulatore alla locazione di memoria `0CH` (istruzione `movwf`, move W to f).

Quando un programma inizia a partire dal vettore di reset (che a sua volta può puntare all'indirizzo della prima istruzione) l'ALU provvede ad aggiornare l'indirizzo della prossima istruzione che è contenuto nel program counter. Se scorrendo il programma utente non interviene nessun salto condizionato a qualche etichetta il PIC arriverebbe presto ad eseguire tutte le istruzioni presenti in memoria ed appartenenti al programma caricato. Tuttavia un programma utente difficilmente è dotato di tale lineatità e disporrà sicuramente di istruzioni di salto, ovvero di istruzioni capaci di modificare il flusso di esecuzione del programma in base alle esigenze modellate dal programmatore. Ad esempio, una di queste istruzioni di salto è l'istruzione `GOTO etichetta` (go to etichetta) che permette al flusso di esecuzione del programma di saltare in un punto qualsiasi del programma in memoria e di continuare da lì l'esecuzione. Ad esempio:

```
ORG 00H
Punto1
      MOVLW 10
      GOTO Punto1
```

Ogni volta che il PIC è azzerato o riavviato perchè precedentemente spento esso esegue dapprima l'istruzione `MOVLW 10` (collocata proprio all'indirizzo 00H della memoria) che inserirà, quindi, il valore decimale 10 nell'accumulatore W. Il program counter verrà fatto puntare alla prossima istruzione e l'ALU esguirà nel successivo ciclo di clok l'istruzione `GOTO Punto1`. Si tratta di una istruzione che determina un salto incondizionato alla locazione di memoria puntata dall'etichetta `Punto1`. Durante questo ciclo di clock, per stabilire quale sarà la successiva istruzione, si utilizza ancora una volta il program counter (che in questo caso è stato fatto puntare all'indirizzo 00H). Ad ogni istruzione eseguita corrisponde un incremento del program counter. Quando il PIC viene azzerato il program counter viene fatto puntare all'indirizzo 0000H. Un altra istruzione che influenza il valore del program counter è invece l'istruzione `CALL` (effettua una chiamata a subroutine). La differenza sostanziale con l'istruzione `GOTO` è che prima di scrivere nel program counter l'indirizzo della label della subroutine l'istruzione `CALL` memorizza in un altro registro detto STACK l'indirizzo di quella che sarebba dovuta essere la successiva istruzione da eseguire se non si fosse incontrata la `CALL`:

```
ORG 00H
Punto1
      MOVLW 10
      CALL  Punto2
      GOTO  Punto1
Punto2
      MOVLW 11
      RETURN
```

L’area STACK di un PIC è gestita secondo una politica LIFO (last input first output), grazie a questa caratteristica è possibile effettuare più `CALL` annidate e tenere traccia del punto in cui riprendere il flusso al momento in cui si incontra una istruzione `RETURN`.

Il PIC mette a disposizione 13 linee per l’I/O suddivise in due porte: la PORT A e la PORT B. La porta A dispone di 5 linee di I/O (da RA0 ad RA4), la PORTA B di 8 linee di I/O (da RB0 ad RB7).
Per la gestione delle linee di I/O da programma, il PIC dispone di due registri interni per ogni porta denominati TRISA e PORTA per la porta A e TRISB e PORTB per la porta B. I registri TRIS A e B, determinano il funzionamento in ingresso o in uscita di ogni singola linea, i registri PORT A e B determinano lo stato delle linee in uscita o riportano lo stato delle linee in ingresso. Ognuno dei bit contenuti nei registri menzionati corrisponde univocamente ad una linea di I/O. Ad esempio il bit 0 del registro PORTA e del registro TRISA corrispondono alla linea RA0, il bit 1 alla linea RA1 e così via. Se il bit 0 del registro TRISA viene messo a zero, la linea RA0 verrà configurata come linea in uscita, quindi il valore a cui verrà messo il bit 0 del registro PORTA determinerà lo stato logico di tale linea (0 = 0V, 1 = 5V). Se il bit 0 del registro TRISA viene messo a uno, la linea RA0 verrà configurata come linea in ingresso, quindi lo stato logico in cui verrà posta dalla circuiteria esterna la linea RA0 si rifletterà sullo stato del bit 0 del registro PORTA. Facciamo un esempio pratico, ipotizziamo di voler collegare un led sulla linea RB0 ed uno switch sulla linea RB4, il codice da scrivere sarà il seguente:

```
MOVLW 00010000B
TRIS  B
```

in cui viene messo a 0 il bit 0 (linea RB0 in uscita) e ad 1 il bit 4 (linea RB4 in ingresso). Si ricorda a tale proposito che nella notazione binaria dell'assembler il bit più a destra corrisponde con il bit meno significativo e quindi con il bit 0. Per accendere il led dovremo scrivere il seguente codice:

```
BSF   PORTB,0
```

Per spegnerlo:

```
BCF   PORTB,0
```

Per leggere lo stato dello switch collegato alla linea RB4, il codice da scrivere sarà:

```
BTFSS PORTB,4
GOTO  TastoNonPremuto
```

Il registro TIMER è un contatore incrementato con una certa cadenza che è programmabile direttamente dall’hardware del PIC. Si tratta di un registro di 8 bit, quando il valore massimo è stato raggiunto (il

valore massimo in questo caso è di 255) il conteggio riprende da 0. La frequenza di conteggio è proporzionale alla frequenza di clock. Se ad esempio al PIC è dato in ingresso un clock esterno di 6MHZ se ne può generare da questo un altro, mediante il contatore TIMER, saltando un campione (in tal caso il clock generato è di 3MHz poichè prendiamo dal clock di 6Mhz un campione e l'altro no).

Set di istruzioni del PIC16F84:

Istruzione	Descrizione	Operazione
ADDLW K	Add literal and W	W=W+K
ADDWF f,d	Add W and f	d=W+f
ANDLW K	AND literal with W	W=W AND K
ANDWF f,d	AND W with f	d=W AND f
BCF f,d	Bit clear f	f(b)=0
BSF f,d	Bit set f	f(b)=1
BTFSC f,d	Bit test f, skip if clear	f(b)=0?
BTFSS f,d	Bit test f, skip if clear	f(b)=1?
CALL K	Call subroutine K	
CLRF f	Clear f	f=0
CLRW	Clear W register	W=0
CLRWDT	Clear watchdog timer	WDT=0
COMF f,d	Complement f	d=NOT f
DECF f,d	Decrement f	d=f-1
DECFSZ f,d	Decrement f, skip if clear	(d=f-1)=0?
GOTO K	Go to address of K	
INCF f,d	Increment f	d=f+1
INCFSZ f,d	Increment f, skip if clear	(d=f+1)=0?
IORLW K	Inclusive OR literal with W	W=W OR K
IORWF f,d	Inclusive OR with f	d=f OR W
MOVLW K	Move literal to W	W=K
MOVF f,d	Move f	d=f
MOVWF f	Move W to f	f=W
NOP	No operation (delay of 1μs)	
OPTION	Load option register	OPTION=W
RETFIE	Return from interrupt	
RETLW K	Return literal to W	
RETURN	Return from subroutine	
RLF f,d	Rotate left through carry	d=f<<1
RRF f,d	Rotate right through carry	d=f>>1
SLEEP	Go to into sleep mode	
SUBLW K	Subtract W from literal	W=K-W
SUBWF f,d	Subtract W from f	d=f-W
SWAPF f	Swap of 012 with 4567 bits	Swap of bits
TRIS f	Load TRIS register	TRIS of f=W
XORLW f	Exclusive OR literal with W	W=W XOR K
XORWF f,d	Eclusive OR with f	d=f XOR W

ADDLW K

Sintassi: `DDLW K`

Questa istruzione somma la costante `K` al valore presente nel registro W. Il risultato è messo nel registro `W`. Ad esempio:

```
            ORG   00H
START       MOVLW 20
            ADDLW 10
```

Al termine dell'ultima istruzione nell'accumulatore troveremo il valore 30. Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;
- **DC=1** se il risultato dell'operazione è superiore a 15;
- **C=1** se il risultato è un numero positivo. Oppure, se il bit 7 del registro contenente il risultato, quello relativo segno vale 0;

ADDWF

Sintassi: `DDLWF f,d`

Questa istruzione somma il valore contenuto all'indirizzo `f` con quello contenuto nel registro `W`. Ad esempio:

```
addendo1    EQU   0CH
addendo2    EQU   0DH
            ORG   00H
            MOVLW 10
            MOVWF adendo1
            MOVLW 20
            MOVWF addendo2
            MOVF  addendo1,W
            ADDWF addendo2,W
```

Il parametro `d` indica il registro in cui destinare l'operazione. Se `d=W` il risultato viene messo nel registro `W`. Se `d=F` il risultato viene messo nel registro `f`. Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;
- **DC=1** se il risultato dell'operazione è superiore a 15;
- **C=1** se il risultato è un numero positivo. Oppure, se il bit 7 del registro contenente il risultato, quello relativo segno vale 0;

ANDLW

Sintassi: `ANDLW K`

Questa istruzione effettua l'operazone di `AND` logico tra il valore memorizzato nell'accumulatore e la costante `K`. Ad esempio:

```
            ORG   00H
START       MOVLW 11110000B
```

```
ANDLW 00001111B
```

Al termine dell'ultima istruzione nell'accumulatore troveremo il valore binario `00000000B`. Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

ANDWF

Sintassi: `ANDWF f,d`

Questa istruzione effettua l'operazone di `AND` logico tra il valore memorizzato nell'accumulatore ed il valore all'indirizzo `f`. Il parametro `d` indica il registro in cui destinare l'operazione. Se `d=W` il risultato viene messo nel registro `W`. Se `d=F` il risultato viene messo nel registro `f`. Ad esempio:

```
MOVLW 01010101B
MOVWF 0CH
MOVLW 00001111B
ANDWF 0CH,W
```

Al termine dell'ultima istruzione nell'accumulatore troveremo i primi 4 bit della stringa binaria `01010101B` (avendoli selezionati con una maschera di bit usata in fase di `AND`). Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

BCF

Sintassi: `BCF f,d`

Questa istruzione imposta a `0` il bit `d` del registro `f`. Non inflenza alcun bit del registro di **STATUS**. Ad esempio:

```
REG1    EQU     0CH
        ORG     00H
        MOVLW   11111111B
        MOVWF   REG1
        BCF     REG1,0
```

Al termine dell'ultima istruzione nel registro `REG1` (precedentemente impostato alla stringa binaria `11111111B`) troveremo il valore binario `11111110B`.

BSF

Sintassi: `BSF f,d`

Questa istruzione imposta a `1` il bit `d` del registro `f`. Non inflenza alcun bit del registro di **STATUS**. Ad esempio:

```
REG1    EQU     0CH
```

```
        ORG     00H
        MOVLW   00000000B
        MOVWF   REG1
        BCF     REG1,0
```

Al termine dell'ultima istruzione nel registro `REG1` (precedentemente impostato alla stringa binaria `00000000B`) troveremo il valore binario `00000001B`.

BTFSC

Sintassi: `BTFSC f,b`

Questa istruzione testa il bit `b` del registro `f` e salta la prossima istruzione se `f(b)=0`. Non inflenza alcun bit del registro di **STATUS**. Ad esempio:

```
REG1    EQU     0CH
        ORG     00H
        MOVLW   11111111B
        MOVWF   REG1
LOOP    BTFSC   REG1,0
        GOTO    LOOP
SKIP    ...
```

Le prime istruzioni dell'esempio inizializzano il registro `REG1` alla stringa binaria `11111111B`. Quindi, con l'istruzione `BTFSC` viene testato il bit `0` del registro `REG1`. Quest'ultimo essendo pari ad `1` provocherà un ciclo (salto incondizionato all'etichetta `LOOP`) infinito di testing del suddetto bit. Non appena il bit da testare diventa `0` si procede a saltare la prossima istruzione, in tal caso è possibile uscire dal ciclo di testing.

BTFSS

Sintassi: `BTFSS f,b`

Questa istruzione testa il bit `b` del registro `f` e salta la prossima istruzione se `f(b)=1`. Non inflenza alcun bit del registro di **STATUS**. Ad esempio:

```
REG1    EQU     0CH
        ORG     00H
        MOVLW   00000000B
        MOVWF   REG1
LOOP    BTFSS   REG1,0
        GOTO    LOOP
SKIP    ...
```

Le prime istruzioni dell'esempio inizializzano il registro `REG1` alla stringa binaria `00000000B`. Quindi, con l'istruzione `BTFSS` viene testato il bit `0` del registro `REG1`. Quest'ultimo essendo pari ad `0` provocherà un ciclo (salto incondizionato all'etichetta `LOOP`) infinito di testing del suddetto bit. Non appena il bit da testare diventa `1` si

procede a saltare la prossima istruzione, in tal caso è possibile uscire dal ciclo di testing.

CALL

Sintassi: `CALL K`

Questa istruzione effettua una chiamata alla subroutine memorizzata all'indizirro `K`. Il parametro `K` può simboleggiare sia l'indirizzo della subroutine che una sua etichetta. Quando la CPU del PIC incontra questa istruzione, memoriza nel registro di STACK il valore del registro PC+1 (indirizzo dell'istruzione successiva) cosicchè, al termine della chiamata alla subroutine è possibile riprendere dall'istruzione successiva. A tale indirizzo si accede al termine della chiamata che avviene con una delle due istruzioni `RETURN` o `RETLW`. Ad esempio:

```
              ORG   00H
              CALL  MIXER-ON
              ...
MIXER-ON      BSF   PORTA,0
              RETURN
```

Non inflenza alcun bit del registro di **STATUS**.

CLRF

Sintassi: `CLRF f`

Questa istruzione azzera l'intero contenuto del registro all'indirizzo `f`. Ad esempio:

```
              CLRF  01H
```

Al termine di questa istruzione il registro all'indirizzo `01H` (registro `TMR0`) sarà azzerato. Se nel programma sorgente è poi incluso il file PIC16F84.inc è anche possibile usare le etichette dei registri piuttosto che i loro indirizzi, ad esempio:

```
              CLRF  TMR0
```

Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** dopo l'esecuzione di questa istruzione;

CLRW

Sintassi: `CLRW`

Come l'istruzione precedente, il registro interessato alle operzioni è tuttavia il registro `W`.

CLRWDT

Sintassi: `CLRWDT`

Questa istruzione va usata solo quando il PIC viene programmato con l'opzione watchdog abilitata. L'abilitazione a tale modalità avviene mediante un opportuno fusibile al pin `WDTE` del PIC. In questa modalità il PIC fa partire un timer che quando raggiunge un valore limite provoca il reset del PIC. Il programma scritto che opererà nella suddetta modalità dovrà periodicamente azzerare il timer con l'istruzione CLRWDT, se ciò non avviene il PIC blocca il programma in esecuzione ed effettuerà un reset per sbloccarlo. Non inflenza alcun bit del registro di **STATUS**.

COMF

Sintassi: `COMF f,d`

Questa istruzione effettua il complemento del valore contenuto nel registro indirizzato dal parametro f. Il parametro d determina, invece, la destinazione del risultato ottenuto. Ancora una volta, il parametro `d` può assumere valore `W` (in tal caso il registro `W` è la destinazione del risultato) oppure il valore `F` (in tal caso il risultato è messo nello stesso registro). Ad esempio:

```
REG1  EQU   0CH
      ORG   00H
      MOVLW 00001111B
      MOVWF REG1
      COMF  REG1,F
```

Al termine dell'ultima istruzione nel registro `REG1` (inizializzato alla stringa binaria `00001111B`) troveremo il valore binario `11110000B`. Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

DECF

Sintassi: `DECF f,d`

Questa istruzione decrementa di una unità il valore contenuto nel registro all'indirizzo `f`. Ancora una volta, il parametro `d` può assumere valore `W` (in tal caso il registro `W` è la destinazione del risultato) oppure il valore `F` (in tal caso il risultato è messo nello stesso registro). Ad esempio:

```
MOVLW 31H
MOVWF 0CH
DECF  0CH,F
```

Al termine dell'ultima istruzione nel registro all'indirizzo `0CH` troveremo il valore esadecimale `30H`. Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

DECFSZ

Sintassi: `DECFSZ f,d`

Questa istruzione decrementa di una unità il valore contenuto all'indirizzo `f` e salta la prossima istruzione se quest'ultimo vale `0`. Ancora una volta, il parametro `d` può assumere valore `W` (in tal caso il registro `W` è la destinazione del risultato) oppure il valore `F` (in tal caso il risultato è messo nello stesso registro). Ad esempio:

```
COUNTER     EQU   0CH
            ORG   00H
            MOVLW 30
            MOVWF COUNTER
LOOP        DECFSZ      COUNTER,F
            GOTO  LOOP
SKIP        ...
```

Il ciclo è effettuato 30 volte di seguito prima di arrivare a saltare l'istruzione successiva. Non influenza alcun bit di stato.

GOTO

Sintassi: `GOTO K`

Questa istruzione provoca un salto incondizionato all'indirizzo `K` oppure alla label `K`. Ad esempio:

```
            ORG   00H
      LOOP  GOTO  LOOP
```

provoca un ciclo infinito. Non influenza alcun bit di stato.

INCF

Sintassi: `INCF f,d`

Questa istruzione incrementa il contenuto del registro all'indirizzo `f` e memorizza il risultato nello stesso registro (in tal caso `d=F`) oppure nell'accumulare (in tal caso `d=W`). Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

INCFSZ

Sintassi: `INCFSZ f,d`

Questa istruzione incrementa il contenuto del registro all'indirizzo `f` e memorizza il risultato nello stesso registro (in tal caso `d=F`) oppure nell'accumulare (in tal caso `d=W`) e salta la prossima istruzione se il risultato è `0`. Ad esempio:

```
COUNTER     EQU   0CH
            ORG   00H
            MOVLW 250
            MOVWF COUNTER
```

```
LOOP        INCFSZ      COUNTER,F
            GOTO        LOOP
```

Il ciclo è eseguito fintanto che `COUNTER` non è uguale a 0. Non inflenza alcun bit del registro di **STATUS**.

IORLW

Sintassi: `IORLW K`

Questa istruzione effettua l'OR inclusivo tra il valore contenuto nell'accumulatore ed il valore `K`. Ad esempio:

```
            ORG   00H
START       MOVLW 00001111B
            IORLW 11110000B
```

Al termine dell'ultime istruzione il registro `W` conterrà il valore binario `11111111B`. Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

IORWF

Sintassi: `IORWF f,d`

Questa istruzione effettua l'operazione logica di OR inclusivo tra il valore contenuto nell'accumulatore ed il registro all'indirizzo `f`. L'operazione risultante può essere memorizzata nel registro all'infirizzo `f` (in tal caso `d=F`) oppure nel registro `W` (in tal caso `d=F`). Ad esempio:

```
REG1  EQU   0CH
      ORG   00H
      MOVLW 00001111B
      MOVWF REG1
      MOVLW 11111111B
      IORWF REG1,F
```

Al termine dell'ultima istruzione il valore nel registro `REG1` sarà `11111111B`. Questa istruzione influenza il bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

MOVLW

Sintassi: `MOVLW K`

Questa istruzione assegna al registro `W` il valore costante `K`. Ad esempio:

```
      ORG   00H
START MOVLW 20
```

Al termine dell'ultima istruzione nel registro `W` ci sarà il valore `K`. Non influenza alcun bit del registro **STATUS**.

MOVF
Sintassi: `MOVF f,d`

Questa istruzione effettua la copia del contenuto del tegistro all'indirizzo `f` nell'accumulatore `W` (`d=W`) oppure nello stesso registro `f` (`d=F`). Ad esempio:

```
MOVF  0CH,W
```

Al termine dell'istruzione nell'accumulatore troveremo il valore esadecimale `0CH`. Non influenza alcun bit del registro **STATUS**.

MOVLWF
Sintassi: `MOVEWF f`

Questa istruzione copia il contenuto del registro W nel registro indirizzato dal parametro `f`. Ad esempio:

```
MOVLWF      10H
MOVWF       01H
```

Al termine dell'ultima istruzione nel registro all'indirizzo `01H` (`TMR0`) troveremo il valore precedentemente caricato nell'accumulatore e quindi `10H`. Non influenza alcun bit del registro **STATUS**.

NOP
Sintassi: `NOP`

Questa istruzione non produce alcun lavoro se non un ritardo pari ad un ciclo macchina. Utilizzando un quatzo da 4 MHz potremo ottenere un ritardo pari ad 1 μs per ogni istruzione `NOP`. Non influenza alcun bit del registro **STATUS**.

OPTION
Sintassi: `OPTION`

Questa istruzione memorizza nel registro speciale `OPTION` (da 8 bit) il valore scritto/presente nel registro `W`. Ad esempio:

```
            ORG   00H
START       MOVLW 01000100B
            OPTION
```

Non influenza alcun bit del registro **STATUS**.

RETFIE
Sintassi: `RETFIE`

Questa istruzione va usata al termine di ogni chiamata a subroutine per la gestione degli interrupte e permette di riassegnare il controllo del

flusso di esecuzione al programma principale. Non influenza alcun bit del registro **STATUS**.

RETLW

Sintassi: `RETLW K`

Questa istruzione assume lo stesso significato di quella precedente con l'aggiunta di passare, tramite l'accumulatore `W`, un valore di ritorno al programma principale, il valore `K`. Ad esempio:

```
REG1  EQU   0CH
      ORG   00H
      CALL  SUB1
      MOVWF REG1
      ...
SUB1  NOP
      RETLW 10
```

Al termine dell'ultima istruzione nel registro `REG1` sarà memorizzato il valore 10, passato come parametro di ritorno all'accumulatore dalla chiamata a subroutine mediante istruzione `RETLW`. Non influenza alcun bit del registro **STATUS**.

RETURN

Sintassi: `RETURN`

Questa istruzione va usata al termine di ogni chiamata a subroutine e permette di riassegnare il controllo del flusso di esecuzione al programma principale. Non influenza alcun bit del registro **STATUS**.

RLF

Sintassi: `RLF f,d`

Questa istruzione ruota verso sinistra i bit del registro all'indirizzo `f`. Il bit appena mosso è riflesso nel bit `C` del registro **STATUS**. Il parametro `d` stabilisce la destinazione dell'operazione, se `d=W` la destinazione è l'accumulatore; se `d=F` la destinazione della rotazione è il registro all'indirizzo `f`. Non influenza altri bit del registro **STATUS** se non il solo bit `C` nella maniera sopra descritta.

RRF

Sintassi: `RRF f,d`

Svolge gli stessi compiti dell'istruzione precedenta, la rotazione dei bit, tuttavia, avviene verso destra.

SLEEP

Sintassi: `SLEEP`

Questa istruzione arresta ilflusso di esecuzione del programma principale conducendo il PIC in uno stato di attesa. Non influena alcun bit del registro **STATUS**.

SUBLW
Sintassi: `SUBLW K`

Questa istruzione sottrae alla costante `K` il valore memorizzato nell'accumulatore. Il nuovo valore è messo nell'accumulatore. Ad esempio:

```
            ORG   00H
START       MOVLW 40
            SUBLW 10
```

Al termine dell'ultima istruzione il valore nell'accumulatore sarà 30. Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;
- **DC=1** se il risultato dell'operazione è superiore a 15;
- **C=1** se il risultato è un numero positivo. Oppure, se il bit 7 del registro contenente il risultato, quello relativo segno vale 0;

SUBWF
Sintassi: `SUBWF f,d`

Questa istruzione sottrae il valore contenuto nel registro `W` al valore contenuto nel registro all'indirizzo `f`. Il parametro `d` stabilisce la destinazione dell'operazione. Se `d=W` il risultato è messo nell'accumulatore; se `d=F` il risultato è messo nel registro `f`. Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;
- **DC=1** se il risultato dell'operazione è superiore a 15;
- **C=1** se il risultato è un numero positivo. Oppure, se il bit 7 del registro contenente il risultato, quello relativo segno vale 0;

SWAPF
Sintassi: `SWAPF f,d`

Questa istruzione scambia i bit 7,6,5,4 con i bit 0,1,2,3 del registro all'indirizzo `f`. Il parametro `d` stabilisce la destinazione dell'operazione. Se `d=W` il risultato è messo nell'accumulatore; se `d=F` il risultato è messo nel registro `f`. Non influenza alcun bit del registro **STATUS**.

TRIS
Sintassi: `TRIS f`

Questa istruzione memorizza in uno dei registri speciale TRIS il valore contenuto nell'accumulatore W. I registri TRIS determinano il funzionamento in ingresso e uscita delle linea di I/O del PIC. Esiste un registro TRIS per ogni porta A e B di I/O denominato TRISA e TRISB. Ad esempio:

```
            ORG   00H
START       MOVLW 11111111B
            TRIS  PORTA
```

Se il bit è alto la linea è impegnata in operazioni di imput, se è basso in operazioni di output.

XORLW

Sintassi: `XORLW K`

Questa istruzione effettua l'operazione logica di `OR` esclusivo tra il valore scritto/presente nell'accumulatore e la costante `K`. Ad esempio:

```
            ORG   00H
START       MOVLW 00000000B
            XORLW 11110000B
            ...
```

Al termine dell'ultima istruzione il valore nell'accumulatore sarà `11110000B`. Questa istruzione influenza i bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

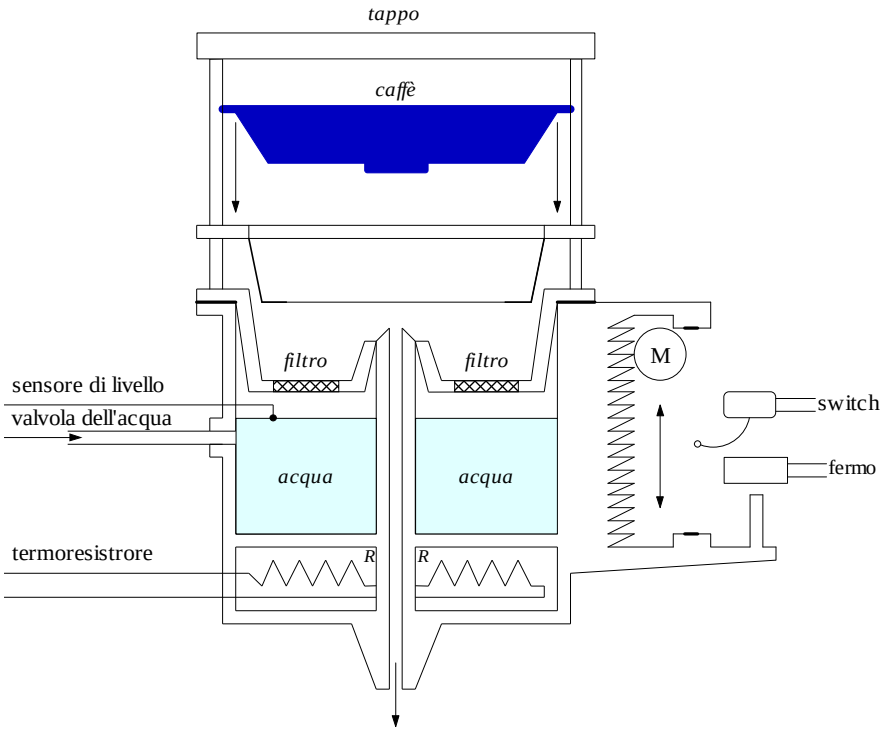
XORWF

Sintassi: `XORWF f,d`

Questa istruzione effettua l'operazione logica di `OR` esclusivo fra il valore presente/caricato nell'accumulatore ed il valore nel registro all'indirizzo `f`. Il parametro `d` stabilisce la destinazione dell'operazione. Se `d=W` il risultato è messo nell'accumulatore; se `d=F` il risultato è messo nel registro `f`. Questa istruzione influenza i bit **Z** del registro **STATUS**:

- **Z=1** se il risultato dell'operazione vale 0;

Un esempio: facciamo un caffè...



Lo schema sopra illustrato schematizza una moderna macchina da caffè elettronica le cui sequenze saranno programmate con un PIC. Quando un tasto viene premuto la valvola dell'acqua è aperta ed il bollitore inizia a riempirsi. Il sensore di livello indica il raggiungimento del livello ottimale dell'acqua (la valvola dell'acqua viene chiusa). Quindi la macchina (presupponendo la presenza della cialda da caffè) aziona il motore elettrico che solleva il bollitore con la cialda fino a preme quest'ultima contro il tappo collocato all'estremità della corsa. Uno switch segnala la fine della corsa (il motore elettrico viene fermato). A questo punto viene acceso un termoresistrore per 50 secondi, l'acqua nel bollitore aumenta la sua temperatura e la sua pressione ed attraversando il filtro e la cialda precedentemte inserita fuoriesce dal bollitore ed è raccolto nella tazza. Al termine dei 50 secondi il termoresistore viene spento e la molla che bloccava il fermo viene rilasciata facendo così abbassare il bollitore nella posizione di riposo.

Immaginando di mappare i bit di `PORTA` nel seguente modo:

- bit 0: tasto START (input);
- bit 1: indicatore di livello dell'acqua (input);
- bit 2: switch di fine corsa (input);
- bit 3: valvola dell'acqua (output);
- bit 4: motore elettrico (output);
- bit 5: termoresistore (output);
- bit 6: elettromagnete per la funzione di release (output);
- bit 7: non usato;

un esempio di programma da inserire nel PIC potrebbe essere il seguente:

```
COUNT EQU   05H          //REGISTRO PER CONTEGGIARE
      ORG   00H          //INIZIO PROGRAMMA
INIT  MOVLW 0000111B     //INIZIALIZZAZIONE
      TRIS  PORTA        //IMPOSTA I BIT DI 1/0
START BTFSS PORTA,0      //ATTENDI IL SEGNALE START
      GOTO  START        //VAI SU START
ACQUA BSF   PORTA,3      //APRILA VALVOLA DELL'ACQUA
      BTFSS PORTA,1      //CONTROLLA IL LIBELLO
      GOTO  ACQUA        //LIVELLO NON RAGGIUNTO
      BCF   PORTA,3      //CHIUDI LA VALVOLA
      BSF   PORTA,4      //AZIONA IL MOTORE
TEST  BTFSS PORTA,2      //TESTA LO SWITCH
      GOTO  TEST         //ASPETTA ANCORA
      BCF   PORTA,4      //SPEGNI IL MOTORE
HEAT  CALL  R            //CHIAMATA A SUBROUTINE
REL   BSF   PORTA,6      //RILASCIA IL BOLLITORE
      GOTO  START        //ATTENDI IL SEGNALE START


R     MOVLWF 50          //SUBROUTINE R, DURA 50S
      BSF    PORTA,5     //ACCENTI LA TERMORES.
      MOVLW  50          //CARICA 50 IN W
      MOVWF  COUNT       //CARICA 50 IN COUNT
WAIT1 CALL   SEC         //CHIAMATA A SEC
WAIT2 DECFSZ COUNT,F     //DECREMENTA DI 1
      GOTO   WAIT1       //SALTA A WAIT1
      BCF    PORTA,5     //SPEGNI LA TERMORES.
      RETURN


SEC   MOVLWF      50     //SUBROUTINE SEC, DURA 1S
      MOVWF       01H    //CARICA 50 IN TMR0
LOOP  DECFSZ      01H,F  //DECREMENTA IL CONTATORE
      GOTO        LOOP   //RIPETI ANCORA
      RETURN             RITORNA AL CHIAMANTE
```