Capitolo 5

Realizzazione software

5.1 Introduzione ai PIC

Elenco istruzioni

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)

Istruzioni & loro descrizioni

Somma la costante K a W

Sintassi:

addlw k

Operazione equivalente:

$W + k \rightarrow W$

Descrizione:

Somma la costante **k** al valore memorizzato nell'accumulatore **W** e mette il risultato nell'accumulatore.

Note:

Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 0 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

Somma il valore in W con quello in F

Sintassi:

addwf f,d

Operazione equivalente:

W + f $\rightarrow$ destinazione (dove d può essere W o f)

Descrizione:

Questa istruzione somma il valore contenuto nell'accumulatore **W** con il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Note:

Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 0 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

Esegue l'AND tra una costante K e W

Sintassi:

andlw k

Operazione equivalente:

W = W AND k

Descrizione:

Effettua l'AND tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**. Il risultato viene memorizzato nell'accumulatore.

Note:

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

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

Esegue l'AND tra una variabile in F e W

Sintassi:

andwf f,d

Operazione equivalente:

W AND f → destinazione (dove d può essere W o f)

Descrizione:

Questa istruzione effettua l'AND logico tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Esempio:

Spesso l'AND logico viene utilizzato per mascherare il valore di alcuni bit all'interno di un registro. Se ad esempio volessimo estrarre dal numero binario 01010101B i quattro bit meno significativi al fine di ottenere il seguente valore 00000101B, basterà preparare una maschera del tipo 00001111B e farne l'AND con il nostro valore di partenza, vediamo come:

```
movlw  01010101B ;Memorizza nel registro W il valore
binario
movwf  0CH       ;Metti all'indirizzo 0CH il valore
                  iniziale da mascherare

movlw  00001111B ;Prepara la maschera di bit
andwf    0CH,W   ;Effettua l'AND e memorizza il risultato
                  nell'accumulatore W
```

Il risultato in W sarà 00000101B come richiesto.

W = 00001111 AND

f = 01010101 =

W = 00000101

La **ANDWF** influenza il bit **Z** del registro **STATUS** che varrà 1 se il risultato dell'operazione è 0.

Azzera un bit nel registro F

Sintassi:

bcf f,b

Operazione equivalente:

f(b) = 0

Descrizione:

Questa istruzione azzera il **bit b** del registro all'indirizzo **f**.

Note:
Questa istruzione non influenza alcun bit di stato

Mette a uno un bit del registro F

Sintassi:

bsf f,b

Operazione equivalente:

f(b) = 1

Descrizione:

Questa istruzione mette a uno il **bit b** del registro all'indirizzo **f**.

Note:
Questa istruzione non influenza alcun bit di stato

Salta l’istruzione successiva se un bit nel registro F vale 0

Sintassi:

btfsc f,b

Operazione equivalente:

f(b) = 0 ? Si, salta una istruzione

Descrizione:

Testa il bit b contenuto nel registro all'indirizzo f e salta l'istruzione successiva se questo vale 0.

Note:

Questa istruzione non influenza alcun bit di stato

Salta l’istruzione successiva se un bit nel registro F vale 1

Sintassi:

btfss f,b

Operazione equivalente:

f(b) = 1 ? Si, salta una istruzione

Descrizione:

Testa il bit b contenuto nel registro all'indirizzo f e salta l'istruzione successiva se questo vale 1.

Note:

Questa istruzione non influenza alcun bit di stato

Chiamata a subroutine

Sintassi:

call k

Descrizione:

Richiama in esecuzione una subroutine memorizzata all'indirizzo **k**. Il parametro k può essere specificato utilizzando direttamente il valore numerico dell'indirizzo oppure la relativa label.

Esempio:

```
        org     00H

        call    ledOn
        ...
        ;Subroutine di accensione di un led
ledOn
        bsf     PORTB,LED1
        return
```

Quando la CPU del PIC incontra una istruzione CALL, memorizza nello STACK il valore del registro PC + 1 in modo da poter riprendere l'esecuzione dall'istruzione successiva alla CALL, quindi scrive nel PC l'indirizzo della subroutine saltando all'esecuzione di quest'ultima.

Il valore originale del PC viene ripristinato all'uscita della subroutine con l'esecuzione dell'istruzione di ritorno RETURN o RETLW.

Nel PIC16C74 sono disponibili 8 livelli di stack, per cui il numero massimo di CALL rientranti, ovvero di istruzioni CALL all'interno di subroutine che a loro volta contengono altre CALL, è limitato ad 8 livelli.

Note:

Questa istruzione non influenza nessun bit di stato.

Azzera il registro F

Sintassi:

clrf f

Operazione equivalente:

f = 0

Descrizione:

Questa istruzione azzera il valore contenuto nel registro indirizzato dal parametro **f**.

Note:

Dopo l'esecuzione di questa istruzione il bit **Z** del registro **STATUS** viene messo a **1**.

Azzera il registro F

Sintassi:

clrw

Operazione equivalente:

W = 0

Descrizione:

Azzera il valore contenuto nel registro **W**.

Note:

Dopo l'esecuzione di questa istruzione il bit **Z** del registro **STATUS** viene messo a **1**.

Azzera il timer Watchdog

Sintassi:

clrwdt

Descrizione:

Questa istruzione deve essere utilizzata quando programmiamo il PIC con l'opzione Watchdog abilitata (fusibile WDTE). In questa modalità il PIC abilita un timer che, una volta trascorso un

determinato tempo, effettua il reset del PIC. Per evitare il reset il nostro programma dovrà eseguire ciclicamente l'istruzione CLRWDT per azzerare il timer prima di detto tempo. Se non azzeriamo il timer in tempo, la circuiteria di watchdog (dall'inglese cane da guardia) interpreterà questo come un blocco del programma in esecuzione ed effettuerà il reset per sbloccarlo.

Note:

Questa istruzione non influenza nessun bit di stato.

Esegue il complemento del registro F

Sintassi:

comf f,d

Operazione equivalente:

d = NOT f (dove d può essere W o f)

Descrizione:

Questa istruzione effettua il complemento del valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** determina la destinazione del valore ottenuto.

Per **d = W** il valore viene memorizzato nel **registro W**
Per **d = F** il valore viene lasciato nel **registro f**.

Note:

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

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

Decrementa il contenuto del registro F

Sintassi:

decf f,d

Operazione equivalente:

d = f -1 (dove d può essere W o f)

Descrizione:

Questa istruzione decrementa il contenuto del registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Note:

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

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

Decrementa il contenuto del registro F e salta l'istruzione successiva se il risultato vale 0

Sintassi:

decfsz f,b

Operazione equivalente:

d = f -1 (dove d può essere W o f) se il risultato = 0 salta

Descrizione:

Decrementa il valore del registro all'indirizzo **f** e se il risultato vale zero salta l'istruzione successiva. Il risultato del decremento può essere memorizzato nello stesso registro **f** oppure nell'accumulatore **W** in base al valore del flag **d**.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Note:

Questa istruzione non influenza alcun bit di stato.

Va in esecuzione all'indirizzo K

Sintassi:

goto k

Descrizione:

Determina un salto del programma in esecuzione all'indirizzo k. Il parametro k può essere specificato utilizzando direttamente il valore numerico dell'indirizzo oppure la relativa label.

Note:

Questa istruzione non influenza nessun bit di stato.

Incrementa il contenuto del registro F

Sintassi:

incf f,d

Operazione equivalente:

d = f +1 (dove d può essere W o f)

Descrizione:

Incrementa il contenuto del registro all'indirizzo **f** e memorizza il risultato nello stesso registro o nell'accumulatore **W** in base al valore del flag **d**:

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nello stesso **registro F**

Note:

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

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

Incrementa il contenuto del registro F e salta l'istruzione successiva se il risultato vale 0

Sintassi:

incfsz f,b

Operazione equivalente:

d = f +1 (dove d può essere W o f) se d = 0 salta

Descrizione:

Incrementa il valore del registro all'indirizzo **f** e se il risultato vale zero salta l'istruzione successiva. Il risultato dell'incremento può essere memorizzato nello stesso registro **f** oppure nell'accumulatore **W** in base al valore del flag **d**.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Note:

Questa istruzione non influenza alcun bit di stato.

Esegue l'OR inclusivo tra W e una costante K

Sintassi:

iorlw k

Operazione equivalente:

W = W OR k

Descrizione:

Effettua l'OR inclusivo tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**.

Note:

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

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

Esegue l'OR inclusivo tra W e il valore contenuto nel registro F

Sintassi:

iorwff,d

Operazione equivalente:

d = f OR W (dove d può essere W o f)

Descrizione:

Questa istruzione effettua l'OR inclusivo tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** determina dove viene memorizzato il risultato dell'operazione:

Per **d = W** il risultato viene memorizzato nell'accumulatore **W**.
Per **d = F** il risultato viene memorizzato nel registro **f**.

Note:

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

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

Muove il contenuto del registro F

Sintassi:

movf f,d

Operazione equivalente:

d = f (dove d può essere W o f)

Descrizione:

Questa istruzione copia il contenuto del registro indirizzato dal parametro **f** o nell'accumulatore **W** o nello stesso **registro F**. Il parametro **d** determina la destinazione.

Per **d = W** il valore viene memorizzato nel **registro W**
Per **d = F** il valore viene lasciato nel **registro f**. In questo caso l'utilità dell'istruzione sta nel fatto che viene alterato il bit Z del flag STATUS in base al valore contenuto nel **registro f**. Se in **f** c'era 0 il flag **Z** viene settato a 1.

Note:

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

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

Assegna a W un valore costante

Sintassi:

movlw k

Operazione equivalente:

W = k

Descrizione:

Assegna all'accumulatore **W** il valore costante **k**.

Note:

Questa istruzione non influenza nessun bit di stato.

Muove il contenuto del registro W nel registro F

Sintassi:

movwf f

Operazione equivalente:

f = W

Descrizione:

Questa istruzione copia il contenuto del **registro W** nel registro indirizzato dal parametro **f**.

Note:

L'esecuzione della **MOVWF** non influenza nessun bit di stato.

Nessuna operazione

Sintassi:

nop

Descrizione:

Questa istruzione non esegue nessuna operazione ma è utile per inserire ritardi pari ad un ciclo macchina .

Note:

La **NOP** non influenza nessun bit di stato.

Assegna il valore presente in W al registro Option

Sintassi:

option

Operazione equivalente:

OPTION = W

Descrizione:

Questa istruzione memorizza nel registro speciale OPTION il valore contenuto nell'accumulatore W.

Note:

Questa istruzione esiste per mantenere la compatibilità con i PIC prodotti finora, la Microchip ne sconsiglia l'uso. In alternativa è consigliabile usare le seguenti istruzioni.

```
        org     00H

start
        bsf     STATUS,RP0    ;Attiva il banco registri 1

        movlw   01000100B
        movwf   OPTION
        ...
```

In pratica si consiglia di scrivere direttamente nel registro OPTION presente nel banco 1 dei registri del PIC utilizzando la MOVWF anziché l'istruzione OPTION che in futuro potrebbe non essere più implementata. Questa istruzione non influenza nessun bit di stato.

Ritorna da una subroutine di interrupt

Sintassi:

retfie

Descrizione:

Questa istruzione deve essere inserita al termine di ogni subroutine di gestione degli interrupt per ridare il controllo al programma principale.

Note:

Questa istruzione non influenza alcun bit di stato.

Ritorna da una subroutine con una costante in W

Sintassi:

retlw k

Descrizione:

Questa istruzione restituisce il controllo, da una subroutine al programma principale. A differenza dell'istruzione RETURN essa consente di passare, tramite l'accumulatore **W**, il valore costante **k** al programma principale.

Note:

Questa istruzione non influenza alcun bit di stato

Ritorna da una subroutine

Sintassi:

return

Descrizione:

Questa istruzione deve essere inserita al termine di ogni subroutine per riprendere l'esecuzione del programma principale.

Note:

Nel PIC16C74 possono essere annidate fino ad 8 chiamate a subroutine.

Questa istruzione non influenza alcun bit di stato.

Ruota a sinistra il contenuto del registro F passando per il Carry

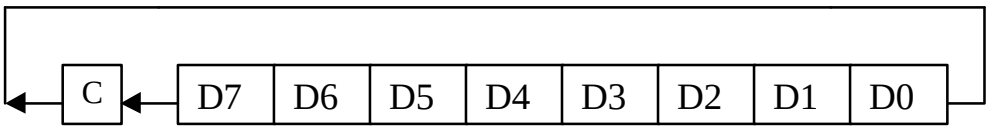
Sintassi:

rlf f,d

Operazione equivalente:

d = f < ruota di 1 < C (dove d può essere W o f)

Descrizione:



Ruota i bit contenuti nel registro all'indirizzo **f** verso sinistra (ovvero dai bit meno significativi verso quelli più significativi) passando per il bit **CARRY** del registro **STATUS** come illustrato in figura:

Il contenuto del bit CARRY del registro status viene spostato nel bit D0 mentre il valore in uscita dal bit D7 viene spostato nel CARRY.

Il valore del parametro **d** determina la destinazione del risultato ottenuto al termine della rotazione:

Per **d = W** il risultato viene memorizzato nel **registro W** lasciando il registro f invariato.
Per **d = F** il risultato viene memorizzato nello stesso **registro f**

Note:

Questa istruzione non influenza nessun altro bit di stato oltre al CARRY.

Ruota a destra il contenuto del registro F passando per il Carry

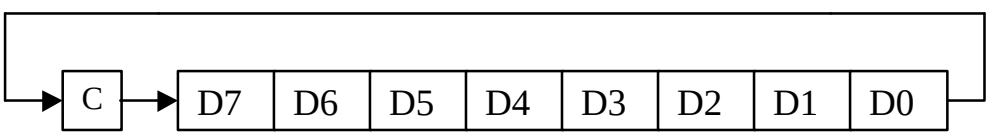
Sintassi:

rrf f,b

Operazione equivalente:

d = f > ruota di 1 > C (dove d può essere W o f)

Descrizione:



Ruota i bit contenuti nel registro all'indirizzo **f** verso destra (ovvero dai bit più significativi verso quelli meno significativi) passando per il bit **CARRY** del registro **STATUS** come illustrato in figura:

Il contenuto del bit CARRY del registro status viene spostato nel bit D7 mentre il valore in uscita dal bit D0 viene spostato nel CARRY.

Il valore del parametro **d** determina la destinazione del risultato ottenuto al termine della rotazione:

Per **d = W** il risultato viene memorizzato nel **registro W** lasciando il registro f invariato.
Per **d = F** il risultato viene memorizzato nello stesso **registro f**

Esempio:

```
parm1 equ     0CH

      org     00H

      clrf    C,STATUS      ;Azzera il CARRY

      movlw   01010101B     ;Valore iniziale
      movwf   parm1

      rrf     parm1,F
```

Al termine del programma il registro **parm1** varrà **00101010B** mentre il CARRY varrà 1.

Note:

Questa istruzione non influenza nessun altro bit di stato oltre al CARRY.

Mette il PIC in standby

Sintassi:

sleep

Descrizione:

Questa istruzione blocca l'esecuzione del programma in corso e mette il PIC in stato di standby (sleep dall'inglese to sleep, dormire).

Note:

Questa istruzione non influenza nessun bit di stato.

Sottrae a k il valore in W

Sintassi:

sublw k

Operazione equivalente:

W = k - W

Descrizione:

Sottra alla costante **k** il valore memorizzato nell'accumulatore **W**.

Note:

Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

- **Z** vale 1 se il risultato dell'operazione vale 0.
- **DC** vale 1 se il risultato dell'operazione è un numero superiore a 15.
- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 0 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

Sottrae il valore contenuto in W al valore contenuto in F

Sintassi:

subwf f,d

Operazione equivalente:

d = f - W (dove d può essere W o f)

Descrizione:

Questa istruzione sottrae il valore contenuto nel **registro W** al valore contenuto nel registro indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Esempio:

Analizziamo un esempio estratto dal datasheet della Microchip:

Se inseriamo l'istruzione:

subwf REG1,F

Dove **reg1** è l'indirizzo di un qualsiasi registro specificato con la direttiva:

REG1 RES 1

Per valori iniziali di REG1=3 e W=2, dopo l'esecuzione avremo REG1=1 e C=1 in quanto il risultato è positivo.

Per valori iniziali di REG1=2 e W=2 dopo l'esecuzione avremo REG1=0 e C=1 perché il risultato è sempre positivo.

Per valori iniziali di REG1=1 e W=2, avremo REG1=FFH ovvero -1 quindi C=0 perché il risultato è negativo.

Note:

Questa istruzione influenza i bit **Z**, **DC** e **C** del registro **STATUS**.

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

- **C** vale 1 se il risultato è positivo ovvero se il bit 7 del registro contenente il risultato vale 0 e 0 se il risultato è negativo ovvero se il bit 7 del registro contenente il risultato vale 1.

Scambia il valore contenuto nei quattro bit meno significativi del registro F con i suoi quattro bit più significativi.

Sintassi:

swap f,d

Operazione equivalente:

f = Swap dei bit 0123 con 4567 di f

Descrizione:

Scambia il valore dei quattro bit più significativi (D7-D4) contenuti nel registro all'indirizzo f con i quattro bit meno significativi (D3-D0) dello stesso. Il risultato viene memorizzato nell'accumulatore **W** o nello stesso registro **f** in base al valore di **d**:

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nello stesso **registro F**

Note:

Questa istruzione non influenza alcun bit di stato

Assegna il valore in W al registro Tris

Sintassi:

tris f

Operazione equivalente:

TRIS di f = W

Descrizione:

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 di I/O denominato TRISA, TRISB, ecc.

Esempio:

```
        org     00H

start
        movlw   11111111B
        tris    PORTA
        ...
```

Note:

Questa istruzione esiste per mantenere la compatibilità con i PIC prodotti finora, la Microchip ne sconsiglia l'uso. In alternativa è consigliabile usare le seguenti istruzioni.

```
        org     00H

start
        bsf     STATUS,RP0    ;Attiva il banco registri 1

        movlw   11111111B
        movwf   TRISA
        ...
```

In pratica si consiglia di scrivere direttamente nei registri TRIS presenti nel banco 1 dei registri del PIC utilizzando la MOVWF anziché l'istruzione TRIS che in futuro potrebbe non essere più implementata.

Note:

Questa istruzione non influenza nessun bit di stato.

Esegue l'OR esclusivo tra W e una costante k

Sintassi:

xorlw k

Operazione equivalente:

W = W XOR k

Descrizione:

Effettua l'OR esclusivo tra il valore contenuto nell'accumulatore **W** ed il valore costante **k**.

Esempio:

```
        org     00H

start
        movlw   00000010B
        xorlw   11110010B
        ...
```

Dopo aver eseguito questo programma l'accumulatore W varrà 11110000B.

Note:

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

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

Esegue l'OR esclusivo tra W e il valore contenuto in F

Sintassi:

xorwf f,d

Operazione equivalente:

d = f XOR W (dove d può essere W o f)

Descrizione:

Questa istruzione effettua l'OR esclusivo (XOR) tra il valore contenuto nell'accumulatore **W** ed il valore contenuto nel registro

indirizzato dal parametro **f**. Il parametro **d** è un flag che indica su quale registro deve essere memorizzato il risultato.

Per **d = W** il risultato viene memorizzato nel **registro W**
Per **d = F** il risultato viene memorizzato nel **registro f**

Questa istruzione influenza i bit **Z** del registro **STATUS** che varrà 1 se il risultato dell'operazione è 0.

Esempio:

Ipotizziamo di dover effettuare lo XOR tra il registro W ed il registro **REG1** da noi definito all'indirizzo **0CH** con la direttiva:

```
REG1    EQU     0CH
```

possiamo utilizzare l'istruzione **IORWF** in due forme a seconda di dove vogliamo mettere il risultato, ovvero:

```
xorwf       COUNTER,F   ;COUNTER = COUNTER XOR W
```

oppure:

```
xorwf   COUNTER,W       ;W = COUNTER XOR W
```

Note:

L'OR esclusivo (XOR) è un'operazione tra due bit in cui il bit risultante vale 0 se i due bit sono uguali.

Spesso lo XOR viene utilizzato nell'assembler del PIC per effettuare la comparazione tra due valori in mancanza di un'istruzione specifica.

Vediamo come:

ipotizziamo di avere un valore nel registro REG1 e di voler verificare se è uguale a 57H. Le istruzioni da eseguire sono le seguenti:

```
movlw 57H           ;W = Valore da comparare = 57H
                    ;Risultato. W = 57H

xorwf  REG1,W       ;W = W XOR REG1 Effettua lo XOR con
                    ;il valore in REG1
btfss  STATUS,Z     ;Salta l'istruzione seguente se il
                    ;risultato dello XOR vale 0, ovvero
                    ;se il valore di REG1 e' pari a 57H
```

```
goto    diverso     ;è diverso da 57H, quindi salta a diverso
goto    uguale      ;è uguale a 57H, quindi salta a uguale
```

5.2 Descrizione del software

Inizialmente riassumo la struttura del programma. Per cominciare si definisce il processore usato, poi si include la libreria dove sono definiti i nomi dei registri speciali, i quali sono utilizzati dalle periferiche del micro. Si configura il funzionamento del micro, dopodiché si comincia ad allocare in ram lo spazio per le variabili necessarie per il funzionamento e successivamente si definiscono i nomi dei bit usati come flag per le variabili che ne fanno uso.
Il programma ha inizio dall'indirizzo H'000', si esegue subito una verifica del tipo di reset avvenuto, perché a secondo del reset sono inizializzate in modo diverso le variabili.
Le istruzioni che seguono sono la routine di interrupt, durante l'avvio sono saltate, perché saranno eseguite solamente da chiamate ad interrupt.
A questo punto ci sono tutte le subroutine utilizzate dal programma principale.
Al termine delle subroutine c'è la label 'Partenza' dove si settano tutti i registri speciali con i valori corrispondenti per una corretta inizializzazione. Prima dell'inizio del programma principale il micro invia al PC il proprio stato in modo da confermare l'avvenuta accensione oppure un diverso tipo di reset.
Inizialmente il micro attende lo start sul BUS I^2C, dopodiché salta alla subroutine "Riconosci_il_comando", che consente di memorizzare i dati nelle locazioni corrette a seconda del tipo di movimento comandato (la descrizione dettagliata verrà condotta nel paragrafo "5.3 Descrizione delle procedure").

Al termine di questa procedura, il micro imposta gli switch esterni in modo da collegare le porte con le relative connessioni, che dipendono dal tipo di movimento che dovrà eseguire.
A questo punto accende i movimenti che dovrà eseguire, cioè setterà le velocità e i flag delle porte relative.
I movimenti implementati sono:

1. Esegui_Curva
2. Esegui_Diritto
3. Esegui_Laterale
4. Esegui_Appoggio
5. Esegui_Diritto_Speciale
6. Esegui_Laterale_Speciale
7. Esegui_Appoggio_Speciale
8. Esegui_Appoggio_Speciale1

5.3 Descrizione delle procedure

A partire dalla label “Partenza” descriverò le procedure in ordine di chiamata.

Inizializza_porte:

Questa procedura consente di configurare tutte le porte come uscite o come ingressi, inserendo i valori iniziali nei data latch.

Invia_stato:

Questa procedura trasmette il byte stato nel bus I^2C.

```
;*****************  Stato        *********************************
Power_on            EQU     H'0007'     ; = 1 è avvenuto un reset di accensione
Brown_out           EQU     H'0006'     ; = 1 "    "    "   "    " abbassamento tensione
Watchdog_Timer      EQU     H'0005'     ; = 1 "    "    "   "    " Watchdog
MCLReset            EQU     H'0004'     ; = 1 "    "    "   "   dall'esterno
Err_movim_impos     EQU     H'0003'     ; = 1 Nessun movimento imposto
Inter_RB0           EQU     H'0002'     ; = 1 è avvenuto un interrupt in RB0
Finito_movim        EQU     H'0001'     ; = 1 è terminato il movimento
;                   EQU     H'0000'     ;
```

Prima di chiamare questa procedura bisogna inserire nel byte SLAVE l'indirizzo da chiamare. All'inizio della procedura si chiama la label

```
Codifica_stato  ;*********************************************************
                ;*********  Tipo di codifica  →  ddddeiii                *******
                ;*********       d = dato trasmesso                      *******
                ;*********       e = errore nella precedente trasmissione *******
                ;*********       i = identificativo del chiamante        *******
                ;*********************************************************
        btfsc   Stato,Power_on
        retlw   B'00000000'
        btfsc   Stato,Brown_out
        retlw   B'00010000'
        btfsc   Stato,Watchdog_Timer
        retlw   B'00100000'
        btfsc   Stato,MCLReset
        retlw   B'00110000'
        btfsc   Stato,Err_movim_impos
        retlw   B'01000000'
        btfsc   Stato,Inter_RB0
        goto    Inter_RB0_1
        goto    Inter_RB0_0
Inter_RB0_1
        btfsc   Stato,Finito_movim
        retlw   B'01010000'             ; RB0 + Finito il movimento
        retlw   B'01100000'             ; RB0 + non Finito il movimento
Inter_RB0_0
        btfsc   Stato,Finito_movim
        retlw   B'01110000'             ; no RB0 + Finito il movimento

        btfsc   Stato,0
        retlw   B'10000000'             ; ********  non è ancora usato
        retlw   B'10010000'             ; ********  non è ancora usato
```

che comprime i dati contenuti in Stato, in modo da lasciare liberi tre bit che serviranno a identificare il dispositivo che sta chiamando, questo perché stiamo lavorando con il multi master, per consentire al dispositivo ricevente di interpretare i dati in modo diverso a seconda del chiamante. Quindi si compone il messaggio da inviare con

```
Aggiungi_identificativo_del_chiamante
        bsf     STATUS,RP0          ; Banco 1
        rrf     SSPADD,W
        bcf     STATUS,RP0          ; Banco 0
        andlw   B'00000111'         ; lascia inalterati i bit dell'indirizzo
        iorwf   Temp,W              ; ci aggiunge i bit della decodifica
        movwf   DATAO
```

A questo punto inizia la vera e propria trasmissione del DATAO, si controlla lo stato del Bus I^2C se libero per la comunicazione, si disabilitano gli interrupt e si chiama la procedura che trasmette il byte indirizzo più il dato. Al ritorno si controlla se sono avvenuti errori di comunicazione, altrimenti si ripete

```
        btfss   FLAG,ERR_1
        goto    Fine_Invia_stato            ; termina
        bsf     DATAO,3                     ; memorizzo e invierò che è avvenuto un errore
        bcf     FLAG,ERR_1
        goto    Attendi_I2C_libero
```

Prima di tornare al programma si riabilitano gli inerrupt.

Riconosci_il_comando:

Questa procedura viene chiamata dopo aver ricevuto un bit di start. Inizialmente attende che il byte sia ricevuto interamente, durante l'attesa cancella il Watchdog timer. Il primo byte ricevuto è il "Primo_DATAI" che deve essere decodificato.

```
;*****************    Primo_DATAI      ***************************
;0000xxxx       Diritto si raccoglie
;0001xxxx       Diritto si estende
;0010xxxx       Laterale si raccoglie
;0011xxxx       Laterale si estende
;0100xxxx       Appoggio si raccoglie
;0101xxxx       Appoggio si estende
;0110xxxx       Curva si raccoglie
;0111xxxx       Curva si estende
;1000xxxx       Diritto si raccoglie usando Delta
;1001xxxx       Diritto si estende usando Delta
;1010xxxx       Laterale si raccoglie usando Delta
;1011xxxx       Laterale si estende usando Delta
;1100xxxx       Appoggio si raccoglie usando Delta
;1101xxxx       Appoggio si estende usando Delta
;1110xxxx       Appoggio si estende si ferma quando tocca terra
;1111xxxx       Altra comunicazione
```

```
;xxxx0xxx        non saranno inviati altri dati
;xxxx1xxx        saranno inviati altri dati

;xxxxx001        è il micro "destro" che sta chiamando il "sinistro"
;xxxxx010        è il PC che sta comunicando
;xxxxx011        è il micro "sinistro" che sta chiamando il "destro"
```

Questo dato contiene dentro di se oltre al comando, anche l'identificativo del dispositivo chiamante (per interpretare i dati in modo diverso a seconda del richiedente). Inizialmente si fa un confronto degli indirizzi per riconoscere il dispositivo.

```
Decifra_Primo_DATAI
        movf    Primo_DATAI,W       ; inizializza W
        andlw   B'00000111'         ; lascia inalterati i bit dell'indirizzo
        movwf   Temp
        rrf     SLAVE_PC,W          ; trasla i bit dell'indirizzo PC e memorizzali in W
        andlw   B'00000111'         ; lascia inalterati i bit dell'indirizzo
        xorwf   Temp,W              ; se i bit sono uguali viene settato il bit Z dentro STATUS
        btfsc   STATUS,Z            ; se = 0 salta
        goto    Ha_chiamato_il_PC

        rrf     SLAVE_Micro,W       ; trasla i bit dell'indirizzo Micro e memorizzali in W
        andlw   B'00000111'         ; lascia inalterati i bit dell'indirizzo
        xorwf   Temp,W              ; se i bit sono uguali viene settato il bit Z dentro STATUS
        btfsc   STATUS,Z            ; se = 0 salta
        goto    Ha_chiamato_il_Micro
```

Notando che ha chiamato il PC si decodifica il comando, che corrisponderà al tipo di movimento da eseguire o altra comunicazione. Per far questo si sfrutta il diverso peso della cifra binaria, per comporre il salto da eseguire dentro una tabella, che contiene il corrispondente dato da inserire in "Movimento". La tabella è contenuta in una zona di memoria che non è raggiungibile con un normale salto, quindi prima di chiamare "Leggi_Tabella_Movimento" bisogna memorizzare in PCLATH la pagina in cui si vorrà saltare.

```
Ha_chiamato_il_PC
        movlw   H'09'               ; inizializzo il PCLATH per il
        movwf   PCLATH              ; salto alla tabella (Pagina 9)
```

```
        clrw                            ; inizializza W
        btfsc   Primo_DATAI,7
        addlw   D'8'
        btfsc   Primo_DATAI,6
        addlw   D'4'
        btfsc   Primo_DATAI,5
        addlw   D'2'
        btfsc   Primo_DATAI,4
        addlw   D'1'                    ; dopo aver impostato W posso chiamare la tabella
        call    Leggi_Tabella_Movimento
        movwf   Movimento
        clrf    PCLATH                  ; Pagina 0      evita successivi salti errati
```

Il salto all'interno della tabella è eseguito aggiungendo il valore contenuto in W nel PCL (Program Counter's Least significant byte).

```
Leggi_Tabella_Movimento                 ; disabilita interrupt per evitare
                                        ; errori di lettura della Tabella_Movimento
        bcf     INTCON,GIE              ; disabilita gli interrupt
        btfsc   INTCON,GIE              ; verifica
        goto    Leggi_Tabella_Movimento ; se non si è disabilitato ci riprova
        call    Tabella_Movimento       ; legge Tabella_movimento
        bsf     INTCON,GIE              ; riabilita gli interrupt
        return

Tabella_Movimento
        addwf   PCL,F           ; carica il salto alla tabella
        retlw   B'10001000'     ; Indirizzo 0 del dato da restituire
        retlw   B'10001010'     ; Indirizzo 1 del dato da restituire
        retlw   B'01001000'     ; Indirizzo 2 del dato da restituire
        retlw   B'01001010'     ; Indirizzo 3 del dato da restituire
        retlw   B'00101000'     ; Indirizzo 4 del dato da restituire
        retlw   B'00101010'     ; Indirizzo 5 del dato da restituire
        retlw   B'10010000'     ; Indirizzo 6 del dato da restituire
        retlw   B'10010010'     ; Indirizzo 7 del dato da restituire
        retlw   B'10000000'     ; Indirizzo 8 del dato da restituire
        retlw   B'10000010'     ; Indirizzo 9 del dato da restituire
        retlw   B'01000000'     ; Indirizzo 10 del dato da restituire
        retlw   B'01000010'     ; Indirizzo 11 del dato da restituire
        retlw   B'00100000'     ; Indirizzo 12 del dato da restituire
        retlw   B'00100010'     ; Indirizzo 13 del dato da restituire
        retlw   B'00101110'     ; Indirizzo 14 del dato da restituire
        retlw   B'00000001'     ; Indirizzo 15 del dato da restituire
```

Una volta decodificato il Byte "Movimento" che contiene i flag del comando da eseguire.

```
;*****************    Movimento    **********************************
Diritto             EQU     H'0007'
Laterale            EQU     H'0006'
Appoggio            EQU     H'0005'
Curva               EQU     H'0004'
```

```
Speciale          EQU   H'0003'        ; = 1 non considera (e non richiede in I²C)
                                       ;     il Delta tra le zampe
Speciale1         EQU   H'0002'        ; = 1 si ferma quando tocca terra
Speciale2         EQU   H'0001'        ; = 1 estende       = 0 raccoglie (questo movimento)
Altra_Comunicaz   EQU   H'0000'        ; = 1 se è una comunicazione di altro tipo
```

Bisognerà riconoscere il comando e fare delle scelte.

```
Riconosci_Movimento
                  clrf    Temp
                  btfss   Primo_DATAI,3
                  goto    Setta_i_movimenti

                  btfsc   Movimento,Curva
                  goto    Attendi_Curva

                  btfsc   Movimento,Diritto
                  goto    Attendi_Diritto

                  btfsc   Movimento,Laterale
                  goto    Attendi_Laterale

                  btfsc   Movimento,Appoggio
                  goto    Attendi_Appoggio

                  btfsc   Movimento,Altra_Comunicaz
                  goto    Attendi_Altra_Comunicaz

                  goto    Fine_Riconosci_il_comando
```

Il secondo dato da ricevere dipende dal quarto bit “Primo_DATAI,3”, se questo bit è uguale a 0, non si riceverà più niente e si setteranno i parametri manipolando i dati presenti in memoria in funzione del movimento comandato; se invece uguale a 1, si riceveranno i parametri tramite I^2C in funzione del movimento.

Caso Movimento,curva

Velocità_2

Fine_tras_ant

Fine_tras_post

Fine_tras_cent

Velocità_1

Fine_later_ant
Fine_later_post
Fine_later_cent

Caso Movimento,Diritto
Velocità_2
Fine_tras_ant (questo stesso valore viene inserito anche nel “post” e “cent”)

Caso Movimento,Diritto (speciale)
Velocità_2
Fine_tras_ant (per posizionare le zampe dove si vuole)
Fine_tras_post
Fine_tras_cent

Caso Movimento,Laterale
Velocità_1
Fine_later_ant (questo stesso valore viene inserito anche nel “post” e “cent”)

Caso Movimento,Laterale (speciale)
Velocità_1
Fine_later_ant (per posizionare le zampe dove si vuole)
Fine_later_post

Fine_later_cent

Caso Movimento,Appoggio
Velocità_1
Fine_Appog_ant (questo stesso valore viene inserito anche nel “post” e “cent”)

Caso Movimento,Appoggio (speciale)
Velocità_1
Fine_Appog_ant (per posizionare le zampe dove si vuole)
Fine_Appog_post
Fine_Appog_cent

Dopo queste procedure ci sono le procedure che setteno i dati ricavandoli da quelli già in memoria senza dover comunicare in I^2C.

Un’altra procedura che si deve considerare per l’I^2C è
Attendi_Altra_Comunicaz
Nella quale per ora si invia il nuovo valore di “Delta” (che rappresenta la massima distanza consentita dai tre movimenti) se l’ottavo bit dentro la variabile temporanea “Temp,7” è settato.

Tornando al principale si incontra la label “Imposta_switch” che serve per comandare i multiplexer esterni.

Poi si passano i parametri delle velocità da caricare nel file register “CCPRxL” per la generazione del “ Duty cycle” imposto all’inizio del movimento. Si attiva il timer 2 che funziona da contatore di clock per essere confrontato con “CCPRxL”+”CCPxCON,<5,4>” e con “PR2”, il risultato di questi confronti genera il PWM.

Dopo aver fatto partire il generatore PWM si abilitano i TLE5205 (drivers di potenza) attraverso la porta A.

A questo punto si è pronti a iniziare il movimento, quindi si fa il salto alla procedura corrispondente.

All’inizio di tutte le procedure dei singoli movimenti si fa un salto alla stessa subroutine “Controlla_tutti_i_passi” che compie più funzioni, per aggiornare le variabili che corrispondono ai vari movimenti:

- Controlla se i contatori degli interrupt sono diversi da zero.
- Nel caso negativo, salta al successivo controllo di contatore di interrupt.
- Nel caso affermativo:
    - Chiama una subroutine relativa al cambiamento di un pin della porta B.
    - Controlla quale movimento si stava eseguendo.
    - Incrementa o decrementa la relativa variabile, e se ha raggiunto la fine del movimento comandato, spegne il bit che memorizza il motore acceso.

    - Al ritorno, decrementa il contatore dell'interrupt, se non ancora zero, ripete questo ciclo.
- Terminato il controllo degli interrupt, salta al controllo dei "Fine_corsa" fisici del movimento.
- Controlla sulla porta D se c'è qualche pin a zero, se negativo controlla il successivo pin, mentre se affermativo:
    - Chiama una subroutine relativa al fine corsa
    - Controlla quale movimento si stava eseguendo, perché a seconda del movimento ha differenti ingressi.
    - Se l'ingresso è il fine corsa, prima di spegnere il bit che memorizza il motore acceso, controlla se ha superato la metà del movimento. Questo perché entrambi i segnali delle due estremità di fine corsa entrano nello stesso ingresso senza distinzione, ma il microcontrollore non deve usare quella all'inizio del movimento per spegnere il motore.
    - Se l'ingresso è l'appoggio del piede, setta il flag relativo.
- I pin RD1 e RD0 hanno come ingresso una porta or a 3 ingressi, in RD1 entrano i tre piedi appoggiati (= 0) del passo sinistro (basta che uno solo dei tre piedi sia alzato perché l'ingresso sia = 1), mentre in RD0 entrano i segnali del passo destro.
- Termina l'esecuzione della subroutine "Controlla_tutti_i_passi"

Al ritorno da questa procedura, se i movimenti lo richiedono, si controlla l'errore della posizione rispetto alle altre zampe, usando come massimo valore di errore il "Delta".

La curva presenta una differenza sostanziale nel modo in cui ricava i dati da confrontare per determinare l'errore. Infatti lo spostamento laterale è confrontato con dei valori caricati nella ROM in una tabella, quest'ultima in corrispondenza biunivoca con lo spostamento di traslazione diritto anteriore. Una volta ottenute le variabili per eseguire il confronto si chiama "Controlla_Curva" che restituisce una variabile di stato del moto e gli errori tra le posizioni vere e quelle in tabella. A questo punto si inizializza una variabile temporanea per memorizzare i motori accesi e il verso di rotazione, chiamata "Flags_Motori". Poi chiamando "Diverso_x_x" agirà sui Flags_motori in funzione dei dati precedentemente ricavati da Controlla_Curva, spegnendo se necessario alcuni flags. Prima di procedere al controllo della traslazione, se i motori dello spostamento laterale sono rimasti indietro rispetto alla traslazione, si fa un salto in "Pausa_Trasla" che spegne temporaneamente i bit della traslazione. Altrimenti, se i laterali sono corretti, si ricavano i dati della traslazione in valore assoluto (perché durante la rotazione ci sono motori che vanno avanti e altri che vanno indietro, ma il valore assoluto tra loro non può essere più discorde di Delta). Avendo i dati da confrontare si chiama "Controlla_Passi", come in "Controlla_Curva" restituisce una

variabile di stato del moto e gli errori tra le posizioni, questa volta, relative. A questo punto si inizializza una variabile temporanea per memorizzare i motori accesi e il verso di rotazione (questa volta imposto perché sto lavorando con dei valori assoluti), chiamata "Flags_Motori". Poi chiamando "Diverso_x_x" agirà sui Flags_motori in funzione dei dati precedentemente ricavati da Controlla_Passi, spegnendo se necessario alcuni flags. I dati contenuti in Flags_Motori in aggiunta con i dati contenuti in "Motori_accesi" e "Motori_acc_ce" che sono relativi ai fine corsa, consentono di accendere o spegnere i motori, complementando il valore prima di uscire in porta A (porta per l'abilitazione dei TLE5205). Prima di ripetere ciclicamente Esegui_Curva, si controlla se tutti i motori sono spenti, condizione necessaria per uscire dal ciclo. Uscendo da questo ciclo, prima di tornare al programma principale, si spegne il timer 2 relativo al generatore PWM (pulse width modulated) , si cancella la variabile "Movimento" per inizializzarla per un successivo comando.

Gli altri movimenti sono più semplici della curva, "Esegui_Diritto" e "Esegui_Laterale" non hanno la parte relativa a Controlla_Curva, in "Esegui_Appoggio" in più rispetto a questi due, si memorizza il valore iniziale delle posizioni, in modo che durante il movimento si conservi l'inclinazione.

Per quanto riguarda i movimenti speciali è eliminata la parte relativa al controllo delle posizioni tra le zampe, per consentire il posizionamento, quando necessario, diverso dallo standard. "Esegui_Appoggio_Speciale_1" è chiamato dall'interno di "Esegui_Appoggio_Speciale" per aggiungere la

condizione di fermare il moto quando tocca terra, non solo quando si raggiunge il fine corsa.

5.4 Differenze tra micro sinistro e destro

Dentro “Inizializza_registri_speciali” i valori degli indirizzi

B’10100x10’

Che sono posti in SLAVE_Micro e in SSPADD, x = 1 corrisponde al micro di sinistra