

Ricordiamo

Tutte le rappresentazioni (Esadecimale, ASM) sono solo convenzioni create per facilitarci la lettura dei valori e la scrittura del programma, i circuiti del Pic a livello fisico lavorano solo con livelli binari, (livelli di tensione alto o basso).

CARICAMENTO E SPOSTAMENTO DATI

MOVLW	n		W = n
MOVWF	reg		(reg) = W
MOVF	reg,d	Z	d = (reg)
SWAPF	reg,d		d = swap nibbles (reg)
CLRF	reg	Z=1	(reg) = 0
CLRWF		Z=1	W = 0

Come si nota dalla tabella sopra riportata l'attività principale svolta dai circuiti del micro è quella di assegnare bytes (o singoli bit) .

La prima importante categoria di istruzioni sono quelle che permettono l'assegnare di valori ben precisi ai registri, e di spostare questi valori da un registro all'altro.

La prima assegna semplicemente il valore n al registro accumulatore W, per esempio l'istruzione:

```
MOVLW 173
```

fa assumere all'accumulatore W il valore 173 (binario 10101101). Siccome l'assemblatore accetta anche numeri scritti direttamente in binario o in esadecimale è possibile scrivere anche:

```
MOVLW 10101101B
```

```
MOVLW 0xAD
```

```
MOVLW 0ADh
```

La seconda istruzione della tabella trasferisce il contenuto dell'accumulatore in una locazione di memoria RAM di indirizzo reg. Il PIC 16Fxx nel banco RAM 0 dispone di un'area dati liberamente usabile per memorizzare i propri valori, quest'area parte dall'indirizzo 32 (20h). Quindi se supponiamo che sia attivo il banco 0 e vogliamo trasferire il contenuto dell'accumulatore nella cella (registro) 32 dobbiamo scrivere:

```
MOVWF 32
```

Sarebbe molto scomodo però doversi ricordare a memoria gli indirizzi di tutte le celle che ci interessa usare, per facilitare il compito l'assemblatore accetta la definizione di un nome simbolico per i valori e gli indirizzi usati nel programma tramite la "direttiva di compilazione" EQU:

```
PIPPO EQU 32
```

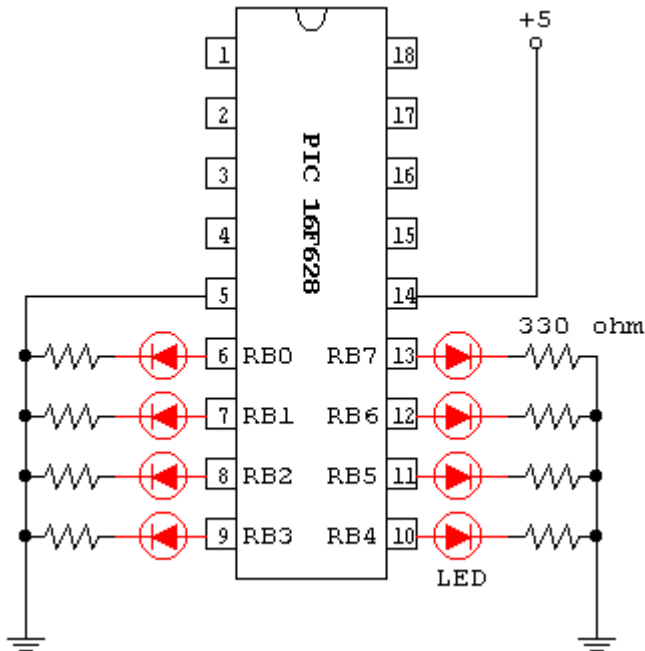
```
MOVWF PIPPO
```

In questo modo è possibile assegnare un nome comodo da ricordare ad ogni cella. Va ricordato che ogni PIC ha un'area RAM usabile dal programmatore, però l'indirizzo a cui inizia varia da un modello all'altro, questo è uno dei motivi principali per cui NON è possibile far eseguire ad un PIC un programma scritto per un altro tipo di PIC senza nessuna modifica.

Come si può vedere non esiste alcun modo per assegnare in un colpo solo un valore ad un registro, ma occorre sempre passare per l'accumulatore usando quindi due istruzioni.

Esperimento: Visualizzare in forma binaria attraverso dei diodi LED il valore dell'accumulatore. Il PIC 16F628 dispone di 16 pin usabili come ingressi/uscite, questi

sono raggruppati in due "porte" da 8 bit chiamate PORTA e PORTB mappate nell'area registri SFR, questo significa che è possibile leggere o scrivere su di esse leggendo o scrivendo un valore al loro indirizzo. I pin corrispondenti ai vari bit sono chiamati RA0..RA7 e RB0..RB7. all'accensione tutti i pin sono configurati come ingressi, perciò le prime istruzioni del programma dovranno configurare come uscite i pin che ci interessano. In questo caso renderemo un'uscita l'intera PORTB (scrivendo 0 nel registro di controllo TRISB).



A fianco è rappresentato il collegamento pratico di 8 diodi LED alla porta B del 16F628. Tutto il necessario è una sorgente di alimentazione stabilizzata a 5V (ma può andare bene anche una comune pila piatta da 4,5V in quanto il PIC può funzionare da 3 a 5,5V). Le resistenze limitano la corrente circolante nei diodi a una decina di mA ciascuno, e sono delle comuni 330 ohm 1/4W.

Riportiamo il programma completo evidenziando le istruzioni sopra citate. i bit caricati nell'accumulatore vengono trasferiti pari pari sui pin RB0..RB7 (bit 0 su RB0 ecc) sotto forma di livelli di tensione, 0V per i bit a zero e +5V per quelli a 1. Risultano quindi accesi i LED corrispondenti ai bit a 1.

```

PROCESSOR 16F628
RADIX    DEC
INCLUDE  "P16F628.INC"
__CONFIG 11110100010000B
ORG      0

BSF      STATUS,RP0 ;Attiva banco 1
CLRF     TRISB      ;Rende PORTB un'uscita
BCF      STATUS,RP0 ;Ritorna al banco 0
MOVLW   10101101B ;Carica 173 nell'accumulatore
MOVWF   PORTB     ;Mandalo sui pin di uscita
SLEEP                    ;Stop programma

END

```

La prima parte del programma è detta header (testata), e contiene informazioni specifiche per l'assemblatore. Il programma vero e proprio è composto solo dalle 6 righe centrali racchiuse tra la testata e l'end finale. Nella testata si indica all'assemblatore il tipo di micro usato, la base di default in cui vanno considerati scritti i numeri, si include un file di definizioni EQU che permette di assegnare automaticamente un nome a tutti i registri di uso comune (come per esempio PORTB, TRISB e STATUS), si definisce la configurazione hardware per il funzionamento del micro (in questo caso per esempio si predispone il funzionamento con clock interno a 4MHz, 16 pin di I/O e WDT disattivato). Org 0 indica l'indirizzo di partenza a cui andranno caricate le istruzioni nella memoria programma (il micro all'accensione inizia ad eseguire le istruzioni partendo dall'indirizzo 0), e l' END finale indica all'assemblatore la fine del testo. Se il programma funziona, appena si fornisce alimentazione i LED devono accendersi coerentemente al valore binario impostato in W, si ricorda che la cifra meno significativa (LSB) si trova sul pin RB0, mentre quella più significativa (MSB) su RB7.

```
MOVF    reg,d    Z    d = (reg)
```

La successiva istruzione della tabella permette invece di spostare il contenuto di un registro nell'accumulatore, o... in se stesso! Questa cosa deriva da una caratteristica di molte istruzioni, che prevedono di specificare la destinazione del risultato (indicato genericamente con d). Il valore d può essere 0 o 1, nel primo caso il risultato viene messo nell'accumulatore W, nel secondo viene messo nel registro stesso chiamato in causa dall'operazione. Per non confondersi nell'indicare 0 o 1 come destinazione, anche questi valori hanno una EQU che ne definisce il nome simbolico W o F. Pertanto è possibile scrivere due forme di questa istruzione:

```
MOVF PIPPO,W ;Carica in W il valore del registro PIPPO
```

```
MOVF PIPPO,F ;Carica nel registro PIPPO il suo stesso valore
```

La seconda forma è del tutto inutile? In realtà no, perché questo tipo di spostamento dati coinvolge il flag Z (flag di zero) come indicato nella colonna centrale. Il flag Z è un bit contenuto nel registro STATUS che viene settato (posto a 1) quando il risultato dell'operazione vale 0. Questo spostamento di un registro in se stesso è quindi un modo rapido per capire se il registro contiene il valore 0 senza dover effettuare sottrazioni o altre operazioni.

Anche in questo caso si vede che non è possibile spostare direttamente un registro in un altro, ma bisogna sempre passare attraverso l'accumulatore usando due istruzioni.

```
SWAPF  reg,d    d = swap nibbles (reg)
```

Un gruppo di 4 bit si chiama nibble. L'istruzione SWAPF scambia tra di loro i 4 bit meno significativi di un registro con quelli più significativi (nella rappresentazione esadecimale questo equivale a scambiare tra loro le due cifre esa che rappresentano i due nibbles). Anche in questo caso il risultato può essere rimesso nel registro di partenza oppure in W a seconda del valore che si dà al parametro d. Questa operazione non altera i flags e può essere perciò usata vantaggiosamente per leggere (e salvare) il valore del registro STATUS, senza alterarlo come avverrebbe invece con una MOVF. Questo salvataggio è necessario per esempio quando si lavora con gli interrupt. Le seguenti tre istruzioni caricano 147 in PIPPO (una generica cella RAM liberamente utilizzabile a cui è stato dato un nome con una EQU, per esempio la cella 32) e ne effettuano uno swap (scambio) dei nibbles:

```
MOVLW 147 ;Carica 147 nell'accumulatore (W=10101110)
```

```
MOVWF PIPPO ;Lo mette nel registro PIPPO (PIPPO=10101110)
```

SWAPF PIPPO,F ;Swappa i nibbles di PIPPO (PIPPO=11101010)

Esperimento: Per sperimentare quanto detto riguardo a MOVF e SWAPF si può sempre usare il circuito visualizzatore con i LED per vedere il valore assunto dal flag Z in due situazioni diverse. Nel primo caso si carica il valore 147 nella cella PIPPO e si esegue una MOVF di PIPPO in se stesso (per cui Z=0). Poi si usa SWAPF per caricare in W il valore di STATUS senza alterarlo ed infine si scrive il valore di W sulla PORTB, ricordando che i nibbles sono stati invertiti. Il flag Z è il bit 2 del registro STATUS, dopo lo swap (scambio) ce lo dobbiamo aspettare nel bit 6 di W, e quindi sul pin RB6 in uscita. Sono date per scontate la presenza della testata, il settaggio della porta B, la definizione di una cella di nome PIPPO con una EQU, e l'istruzione SLEEP finale per fermare il micro, che rimangono comunque sottointese.

```
MOVLW 147 ;Carica 147 nell'accumulatore
MOVWF PIPPO ;Lo mette nel registro PIPPO
MOVF PIPPO,F ;Muove il registro PIPPO in se stesso (Z=0)
SWAPF STATUS,W ;Swappa STATUS mettendolo in W
MOVWF PORTB ;Manda W sui pin di uscita
```

Il secondo programma invece carica 0 in PIPPO, quindi il MOVF deve far settare il flag Z, questa volta il LED su RB6 deve accendersi:

```
MOVLW 0 ;Carica 0 nell'accumulatore
MOVWF PIPPO ;Lo mette nel registro PIPPO
MOVF PIPPO,F ;Muove il registro PIPPO in se stesso (Z=1)
SWAPF STATUS,W ;Swappa STATUS mettendolo in W
MOVWF PORTB ;Manda W sui pin di uscita
```

CLRF reg	Z=1	(reg) = 0
CLRW	Z=1	W = 0

Le ultime due istruzioni di caricamento e spostamento dati sono le CLRF, che permettono di azzerare i bit di un registro qualsiasi e dell'accumulatore. entrambe queste istruzioni impostano il flag Z a 1. L'esempio precedente poteva perciò esser scritto più sinteticamente:

```
CLRW ;Azzerà l'accumulatore
MOVWF PIPPO ;Lo mette nel registro PIPPO
MOVF PIPPO,F ;Muove il registro PIPPO in se stesso (Z=1)
SWAPF STATUS,W ;Swappa STATUS mettendolo in W
MOVWF PORTB ;Manda W sui pin di uscita
```

Oppure, ancora meglio:

```
CLRF PIPPO ;Azzerà il registro PIPPO
MOVF PIPPO,F ;Muove il registro PIPPO in se stesso (Z=1)
SWAPF STATUS,W ;Swappa STATUS mettendolo in W
MOVWF PORTB ;Manda W sui pin di uscita
```

Riepilogo breve: le istruzioni MOVLW e MOVWF non alterano il flag Z. L'istruzione MOVF invece modifica il flag Z, che risulta settato (s=set=1) se il valore caricato è 0, e resettato (c=clear=0) negli altri casi. Una MOVF può trasferire il valore nella stessa locazione da cui viene letto, il suo valore perciò non cambia, ma, visto che il flag Z viene modificato, è un modo rapido per verificare se contiene zero.

L'istruzione SWAPF scambia i nibbles (i 4 bit superiori e i 4 bit inferiori) di un registro, e deposita il risultato nell'accumulatore o nella locazione stessa da cui è stato prelevato. In alcuni casi può essere vantaggioso usarla, oltre che per swappare i nibble, anche come

spostamento perché non altera i flags (per esempio è usata per salvare il registro STATUS durante un interrupt).

Le istruzioni CLRF e CLRW sono un caricamento diretto del valore 0 in una locazione dati o nell'accumulatore. Entrambe impostano il flag Z a 1. L'unica differenza tra usare una MOVLW 0 e una CLRW è data dal flag Z, che resta invariato nel primo caso, mentre viene settato nel secondo.