PIC
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA PER PRINCIPIANTI

By Claudio Fin

[Indice principale]

INTRODUZIONE

Questo testo vuole fornire alcune nozioni base riguardo alla programmazione in linguaggio macchina dei PIC, i piccoli ed economici microcontrollori prodotti dalla Microchip. Intanto va fatta subito una distinzione tra microprocessore e microcontrollore. Il primo è un componente universale che ha bisogno di numerosi integrati esterni aggiuntivi per poter funzionare (memoria, oscillatore di clock, periferiche di ingresso/uscita ecc) e può diventare l'elemento di controllo di un computer molto sofisticato. Un microcontrollore invece racchiude tutti questi elementi all'interno di un unico piccolo contenitore, e ha bisogno di pochissimi (o nessuno) componenti esterni per funzionare. La memoria per il programma, quella per i dati di lavoro (RAM), l'oscillatore, il circuito di reset e le diverse periferiche, sono tutte racchiuse in un unico chip. Le sue capacità di calcolo però sono estremamente ridotte, la memoria RAM per esempio è formata da poche centinaia (se non solo decine) di celle, e solitamente non è espandibile in alcun modo. I microprocessori possono essere usati per effettuare elaborazioni molto complesse su grandi quantità di dati, i microcontrollori sono invece adatti per compiti di controllo hardware a basso livello, che non richiedono grandi quantità di memoria, ma prezzo, consumo e dimensioni ridotti, uniti ad una discreta velocità di esecuzione e ad una nutrita serie di periferiche già pronte come timers, convertitori analogico/digitali (ADC), generatori PWM, porte seriali ecc. Tipiche applicazioni di un microcontroller possono essere automatismi, antifurti, strumenti di misura, regolazione luminosità, caricabatterie, trasmettitori/ricevitori codificati ecc. Da qui in avanti si useranno indifferentemente i termini micro, microprocessore, microcontrollore e microcontroller per indicare la stessa cosa, in particolare con microprocessore si intenderà il piccolo microprocessore contenuto all'interno di ogni microcontroller.

Cos'è il linguaggio macchina? Ogni microprocessore è progettato e costruito per eseguire determinate operazioni in presenza di determinate sequenze binarie che vengono lette una dopo l'altra da un'apposita area di memoria, detta memoria programma. Queste sequenze binarie sono le istruzioni in linguaggio macchina (L.M.), l'unico direttamente «comprensibile» dai circuiti del micro. Una tipica istruzione in linguaggio macchina è la sequenza:

00000100000011

Per rendere più facile la vita al programmatore, ad ogni istruzione L.M. è stato dato un nome simbolico detto codice mnemonico. L'insieme dei codici mnemonici prende il nome di linguaggio assembly. Il programma in assembly si scrive con un qualsiasi editor di testo (come il notepad di Windows). Tramite il un programma assemblatore (assembler) si convertono poi le istruzioni assembly in codice

macchina direttamente eseguibile. Per ogni istruzione assembly esiste naturalmente una e una sola sequenza binaria in codice macchina, per questo motivo i termini linguaggio macchina e assembly indicano spesso la stessa cosa (a dire il vero è diventato anche di uso comune usare la parola assembler per riferirsi all'assembly e non all'assemblatore, per cui è normale sentir parlare di linguaggio assembler).

Perché studiare il linguaggio macchina? I motivi validi sono più di uno. In generale un programma scritto in L.M. è molto compatto, usa poca memoria, è veloce nell'esecuzione, può accedere completamente alle risorse fornite dal micro e ottenere il controllo assoluto delle sue temporizzazioni. Lavora in stretto contatto con l'hardware... una volta si diceva «programmare sul nudo metallo», e perciò si comprende in qualche misura anche il suo funzionamento sottostante. Si comprende inoltre quello che «c'è sotto» ai linguaggi ad alto livello e il modo in cui essi realizzano le loro funzioni complesse partendo dalle istruzioni elementari dell'assembly sottostante. Naturalmente l'assembly ha anche molti svantaggi, che sono di solito quelli che spingono ad utilizzare linguaggi di livello più alto come il C o il BASIC: i programmi sono più difficili da scrivere, interpretare e correggere, ci si impiega molto più tempo per scriverli, richiedono molte istruzioni anche per effettuare operazioni semplici, ed è molto difficoltoso effettuare calcoli.

Ogni micro dispone di un set ben definito di istruzioni elementari, che sono naturalmente dedicate a quel singolo micro, pertanto un programma assembly per Z80 non è compatibile con un programma assembly per PIC o per 8088. In queste pagine si parla dell'assembly dei microcontrollori PIC (famiglie 12F e 16F).

Essendo l'assembly strettamente legato all'hardware, le sue istruzioni risentono dei limiti (o delle potenzialità) offerte dall'hardware stesso. Una minima conoscenza dell'hardware è quindi indispensabile. I PIC di cui si parla qui sono micro con architettura a 8 bit, questo significa che possono lavorare direttamente solo con numeri rappresentabili in 8 bit (valori compresi tra 0 e 255). Hanno una memoria per le istruzioni del programma separata dalla memoria dati (RAM). La prima è di tipo flash, riprogrammabile elettricamente almeno 1000 volte con un apposito programmatore. La RAM invece contiene tutte le informazioni di lavoro necessarie durante l'esecuzione del programma, ma a differenza della prima si cancella ogni volta che viene tolta l'alimentazione. In questi micro ogni cella (o locazione) della RAM può essere pensata (ed effettivamente usata) come un registro a 8 bit in cui salvare o da cui leggere i nostri dati. Le locazioni di memoria, sia programma che dati, hanno un indirizzo crescente che parte da 0. In particolare le prime locazioni della RAM prendono il nome di SFR (special function registers), e servono per controllare il funzionamento dell'hardware. In quest'area troviamo per esempio i registri che permettono l'accesso ai piedini (pin) del microcontrollore, che consentono cioè al programma di generare una tensione (livello logico) verso l'esterno per comandare ad esempio dei diodi LED , display, relè, transitor ecc..., oppure di leggerla, per esempio per determinare la posizione di un interruttore o di un un pulsante. Attraverso i registri della sezione SFR si possono anche attivare e usare le diverse periferiche interne, come i timers, il convertitore analogico/digitale (ADC), la memoria EEPROM ecc...

Le aree di memoria su cui si può agire da programma sono i registri della memoria dati e il registro accumulatore W, che non fa parte dell'area dati ma è un ulteriore registro hardware specializzato, usato nelle operazioni aritmetico logiche.

La RAM è inoltre suddivisa in due o più «banchi», un pò come se vi fossero più RAM attivabili una sola alla volta, questa selezione si ottiene impostando alcuni bit specifici (nel registro STATUS). Durante l'esecuzione di un programma è importante sapere sempre quale banco si sta utilizzando. Il registro STATUS è un registro molto importante dei PIC, perché permette di selezionare i banchi RAM e contiene anche i «flags», particolari bit di cui parleremo più avanti indispensabili per far funzionare programmi complessi.

A differenza di altri micro, i PIC sono microcontrollori di tipo RISC, dispongono cioè di un set ridotto di solamente 35 istruzioni elementari eseguite molto velocemente. Ogni istruzione occupa una sola locazione della memoria programma, e quasi tutte vengono eseguite in 4 cicli di clock. Se un PIC viene cloccato a 4Mhz è in grado di eseguire 1milione di istruzioni al secondo (1 mips) e ogni istruzione dura 1µS (1 microsecondo). Le istruzioni di branch (salto, ramificazione) possono richiedere 8 cicli di clock anzichè 4. Nella terminologia Microchip un gruppo di 4 cicli di clock è detto "ciclo macchina", per cui le istruzioni vengono eseguite in uno o due cicli macchina.

Ci sono molti modelli di pic all'interno di una famiglia (architettura), ciascuno con le proprie peculiarità, qualcuno ha più memoria programma, qualche altro ha più periferiche, qualcuno ha dimensioni ridotte (solo 8 pin), altri invece mettono a disposizione più di 30 pin di ingresso/uscita (I/O). Le istruzioni però sono le stesse e funzionano nello stesso modo. Per gli esempi che seguiranno verrà usato soprattutto il PIC16F628, che è molto diffuso ed economico, non ha bisogno di nessun componente esterno per funzionare (a parte un'alimentazione stabilizzata a 5V) e mette a disposizione fino a 15 pin di I/O singolarmente configurabili come ingressi o come uscite, e uno ulteriore utilizzabile solo come ingresso.

Lo scopo di queste pagine non è quello quello di descrivere nel dettaglio l'architettura interna di uno o più PIC, o delle periferiche in esso contenute, per queste cose si rimanda sicuramente ai relativi datasheets reperibili sul sito della casa costruttice www.microchip.com. Qui verranno date le spiegazioni strettamente indispensabili per la comprensione degli esempi pratici. Si da per scontato inoltre che si abbia già qualche conoscenza di elettronica, e che si abbia a disposizione un programmatore e i programmi necessari per l'assemblaggio (conversione in codice eseguibile) del programma e per la programmazione del chip.

IL LINGUAGGIO MACCHINA Istruzioni, programmi, dati

Il linguaggio macchina è quanto di più vicino ci sia all'hardware. Il comportamento di ogni istruzione elementare è realizzato in modo fisico dall'insieme dei circuiti logici del micro, ma si può anche dire che l'insieme delle funzioni logiche realizzabili dal circuito determina quelle che sono le istruzioni elementari utilizzabili. Dal punto di vista fisico ogni istruzione è una sequenza binaria di livelli elettrici in grado di attivare in modi differenti i circuiti interni. Le istruzioni assembler sono solo una forma mnemonica comoda per descrivere le sequenze binarie che danno luogo alle funzioni logiche che vogliamo far eseguire al micro, tra tutte (e solo) quelle che è fisicamente in grado di eseguire. Da questo punto di vista si può dire che

l'assembly è in rapporto 1:1 con il linguaggio macchina e con le funzionalità dell'hardware.

Un programma non è altro che un'insieme di istruzioni, che vengono eseguite una dopo l'altra producendo il risultato voluto, sia esso un gioco o un processo di controllo industriale. Ma cosa possono fare le istruzioni dei PIC e in generale di ogni altro microprocessore? Fondamentalmente solo poche cose:

- Assegnare un valore ad un registro
- Spostare un valore tra registri
- Effettuare somme e sottrazioni, incrementi e decrementi su un registro
- Effettuare operazioni logiche AND OR XOR NOT su un registro
- Effettuare scambi e rotazioni dei bit di un registro
- Settare o resettare singoli bit di un registro
- Effettuare dei salti condizionati da un punto all'altro del programma

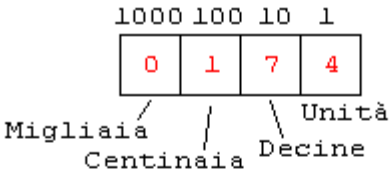
Come si vede la maggior parte dell'attività riguarda la manipolazione elementare dei valori o dei singoli bit dei registri. Come possono delle sequenze di queste poche cose costituire la base per il funzionamento di un robot o di uno strumento di misura? E'più facile vederlo in pratica che spiegarlo... ma si può già intuire che per compiere operazioni di una certa complessità sono richiesti moltissimi di questi "passi elementari".

Si è detto che i registri dei PIC possono contenere valori a 8 bit. Di fatto il considerarli valori (nel senso di numeri) è una nostra convenzione. Dal punto di vista fisico il contenuto dei registri non sono altro che sequenze binarie di bit. Questi bit possono rappresentare qualsiasi cosa o qualsisi tipo di informazione (un numero, un carattere, una parte di un numero più grande ecc...) ed è il programmatore che decide questo al momento in cui scrive il programma. Le istruzioni sono le operazioni elementari che può usare per manipolare i bit delle sue informazioni in modo da arrivare al risultato voluto. In particolare però le istruzioni aritmetiche considerano effettivamente i registri come byte (8 bit) che contengono un valore numerico da 0 a 255 codificato in forma binaria.

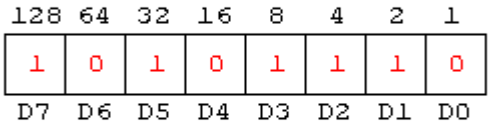
La figura seguente mostra le tre rappresentazioni numeriche comunemente usate in campo elettronico e "microinformatico".

Qui a fianco si vede la codifica del valore 174 nel nostro consueto sistema decimale posizionale che usiamo abitualmente. La cifra di destra è la meno significativa (leggera) ed è chiamata unità, mentre quella di sinistra è la più significativa (pesante) ed è chiamata migliaia. Il valore è dato dalla somma delle diverse cifre moltiplicate per il loro peso, quindi abbiamo una volta 100, 7 volte 10 e 4 volte 1, per un totale di 174. Il peso delle cifre è dato dalle potenze di 10, e quindi il sistema si dice decimale, o in base 10.

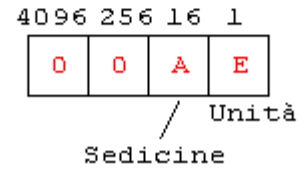
In binario è la stessa cosa, solo che invece di avere potenze di 10 (1, 10, 100, 1000 ecc) abbiamo potenze di 2 (1,2,4,8 ecc) e le cifre invece di poter assumere valori da 0 a 9 possono essere solo 0 e 1 (da qui il nome bit: binary digit). La cifra meno significativa è chiamata D0 o bit 0 (LSB) e ha peso 1, quella più significativa è chiamata D7 o bit 7 (MSB) ed ha peso 128. Si può calcolare che se tutti i bit fossero



1*100 + 7*10 + 4*1 = 174



1*128 + 1*32 + 1*8 + 1*4 + 1*2 = 174



10*16 + 14*1 = 174

a 1 la somma dei loro pesi darebbe esattamente 255, cioè il massimo valore codificabile con 8 bit.

La rappresentazione esadecimale è il terzo caso, ed è molto usata perché una cifra (digit) esadecimale rappresenta esattamente 4 bit binari (un nibble). Con due cifre esa da 00 a FF si rappresenta l'intero range di valori codificabili con 8 bit (1 byte). Anche per l'esadecimale abbiamo la cifra meno significativa a destra e quella più significativa a sinistra. Il «peso» delle diverse cifre questa volta è dato dalle potenze di 16 (sistema in base 16). le cifre possono assumere tutti i valori compresi tra 0 e 15, e i valori tra 10 e 15 vengono indicati con le lettere dalla A alla F. L'esadecimale è molto usato per rappresentare in modo compatto i valori binari contenuti in memoria e il valore degli indirizzi di memoria, e anche perché permette di passare rapidamente alla notazione binaria. Nell'esempio della figura infatti le due cifre A ed E corrispondono alle due sequenze binarie del 10 e del 14, cioè ai due nibbles 1010 e 1110. La cifra nella posizione delle «sedicine» rappresenta valori 16 volte più grandi di quelli della cifra delle unità, pertanto 10*16 + 14 = 174.

In ogni caso va ricordato che tutte queste rappresentazioni sono solo convenzioni create per facilitarci la lettura dei valori e la scrittura del programma, i circuiti del micro a livello fisico lavorano solo e sempre con livelli binari, cioè assenza o presenza di tensione.

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
CLRW            Z=1  W = 0
```

Si è visto dal prospetto sui tipi di istruzioni che la manipolazione dei byte (o dei singoli bit) è l'attività principale svolta dai circuiti del micro. La prima importante categoria di istruzioni è composta perciò da quelle che permettono di assegnare valori ben precisi ai registri, e di spostare questi valori da un registro all'altro. Nella tabella qui sopra sono riportate tutte le istruzioni di assegnazione e spostamento.

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

```
        MOVLW     174
```

fa assumere all'accumulatore W il valore 174 (binario 10101110). Siccome l'assemblatore accetta anche numeri scritti direttamente in binario o in esadecimale è possibile scrivere anche:

```
        MOVLW     10101110B
        MOVLW     0xAE
        MOVLW     0AEh
```

La seconda istruzione della tabella trasferisce il contenuto dell'accumulatore in una locazione di memoria RAM di indirizzo «reg». Il PIC 16F628 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:

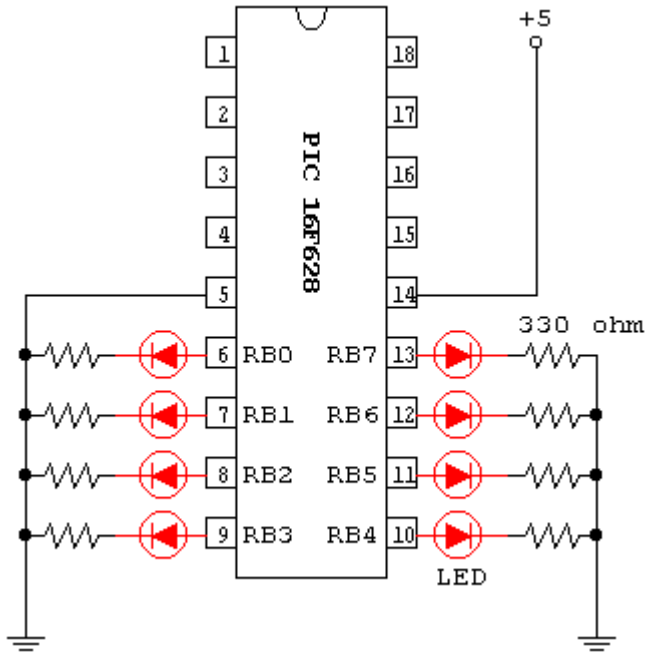
```
PIPPO   EQU       32

        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 usbile 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.

Sotto c'è il programma completo con evidenziate le istruzioni viste finora. Il risultato è che 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       10101110B     ;Carica 174 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, rammentando 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 apparentemente strana 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 da 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                ;Azzera 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       ;Azzera 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 stao 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.

ISTRUZIONI ARITMETICHE

```
ADDLW   n       C Z  W = W + n
ADDWF   reg,d   C Z  d = W + (reg)
SUBLW   n       C Z  W = n - W
SUBWF   reg,d   C Z  d = (reg) - W
INCF    reg,d     Z  d = (reg) + 1
DECF    reg,d     Z  d = (reg) - 1
```

La seconda grande categoria di istruzioni è quella aritmetica, grazie ad esse il micro ha la possibilità di effettuare dei semplici calcoli o di confrontare dei valori. I PIC delle famiglie 12F e 16F sono in grado di sommare, sottrarre, incrementare e decrementare valori a 8 bit (compresi tra 0 e 255). Come si può vedere dalla tabella tutte le istruzioni di questo gruppo alterano il flag Z, che viene posto a 1 se il risultato dell'operazione è 0. Le operazioni di somma e sottrazione invece modificano anche il flag C (carry, detto anche flag di prestito/riporto) che è il bit 0 del registro STATUS.

Durante una somma il flag C è normalmente a 0, e viene posto a 1 nel caso in cui si verifichi un overflow, nel caso cioè in cui il risultato ecceda il valore 255. Durante la sottrazione invece il flag C viene sempre tenuto a 1, e viene messo a 0 solo se la sottrazione causa un prestito, cioè se il risultato dell'operazione è negativo. Va detto che il valore di un registro non può diventare negativo (come non può aumentare oltre il 255), in questi casi si ha il «rollover», un registro arrivato al limite torna cioè all'inizio come se i valori da 0 a 255 fossero disposti in un circolo in cui 0 e 255 sono vicini. Aggiungendo 1 al 255 ritorniamo infatti zero, sottraendo 1 allo 0 otteniamo 255. Il flag C indica l'avvenuto rollover in un senso o nell'altro.

Avendo già dimestichezza con le istruzioni di caricamento è semplice capire come funzionano quelle di questo gruppo. La prima somma semplicemente un valore "n" (compreso tra 0 e 255) all'accumulatore W. La seconda invece somma l'accumulatore con un registro, e il risultato viene come sempre posto dove specificato con il parametro «d».

Le istruzioni di sottrazione sono un pò diverse da quelle di altri tipi di assembly, infatti qui è sempre l'accumulatore ad essere sottratto:

```
SUBLW 15          significa: W = 15 - W
SUBWF PIPPO,W     significa: W = PIPPO - W
SUBWF PIPPO,F     significa: PIPPO = PIPPO - W.
```

Le ultime due istruzioni (INCF e DECF) incrementano o decrementano di 1 il valore contenuto nel registro specificato, il risultato viene posto dove specificato con «d». Queste istruzioni settano flag Z se il risultato dell'operazione è 0.

Se si hanno dei dubbi sui valori che assumono i registri durante queste operazioni è sempre possibile visualizzarli con i LED. Per esempio con il seguente esempio dovremmo ottenere un valore binario di 250+170=420. Per determinare il valore assunto da un registro a causa del rollover è sufficiente sottrarre 256 ai valori

che superano il 255 (o aggiungerlo a quelli che scendono sotto lo zero). Nel nostro caso 420-256=164 (10100100):

```
MOVLW   250       ;Carica 250 nell'accumulatore
MOVWF   PIPPO     ;Lo mette nel registro PIPPO
MOVLW   170       ;Carica 170 nell'accumulatore
ADDWF   PIPPO,W   ;Lo somma con il valore di PIPPO
MOVWF   PORTB     ;Lo manda sui pin di uscita
```

Inoltre l'operazione setta sicuramente il flag C, per cui visualizzando il registro status (swappato) sui LED si deve trovare il led corrispondente al pin RB4 acceso:

```
MOVLW   250       ;Carica 250 nell'accumulatore
MOVWF   PIPPO     ;Lo mette nel registro PIPPO
MOVLW   170       ;Carica 170 nell'accumulatore
ADDWF   PIPPO,W   ;Lo somma con il valore di PIPPO
SWAPF   STATUS,W  ;Carica STATUS swappato su W
MOVWF   PORTB     ;Lo manda sui pin di uscita
```

ISTRUZIONI LOGICHE

```
ANDLW   n       Z  W = W AND n
ANDWF   reg,d   Z  d = W AND (reg)
IORLW   n       Z  W = W OR n
IORWF   reg,d   Z  d = W OR (reg)
XORLW   n       Z  W = W XOR n
XORWF   reg,d   Z  d = W XOR (reg)
COMF    reg,d   Z  d = NOT (reg)
```

Queste istruzioni sono quelle che forse più assomigliano alle funzioni svolte dai comuni circuiti logici, ed in effetti a livello hardware si comportano proprio come delle semplici porte logiche che operano sui bit dei registri o dell'accumulatore.

Tutte le istruzioni a parte l'ultima richiedono due «operandi» su cui effettuare l'operazione logica. Gli operandi possono essere l'accumulatore e un valore numerico diretto "n", oppure l'accumulatore e un registro, in questo caso naturalmente va specificata la destinazione con il parametro «d».

Le funzioni logiche vengono applicate tra ogni bit corrispondente dei due operandi, cioè ad esempio tra il bit 0 dell'accumulatore e il bit 0 del registro, tra l'1 dell'accumulatore e l'1 del registro e così via:

```
100111010 AND          00101100  OR         00010001  XOR
000111000 =            10000010  =          10000001  =
-------------          ------------         -------------
000111000              10101110             10010000
```

Come si può vedere dalla tabella tutte le istruzioni logiche settano il flag Z nel caso in cui il loro risultato sia 0.

Queste istruzioni permettono in modo semplice di settare, resettare o far cambiare di stato uno o più bit di un registro. Nel primo esempio dopo aver caricato 00110011 in PIPPO tramite un'operazione di OR vengono settati i suoi bit 3 e 2 che inizialmente erano a 0:

```
MOVLW   00110011B  ;Carica 51 nell'accumulatore
MOVWF   PIPPO      ;Lo mette nel registro PIPPO
MOVLW   00001100B  ;Carica 12 nell'accumulatore
IORWF   PIPPO,F    ;PIPPO=00111111
```

Nell'esempio seguente si parte sempre con PIPPO caricato nello stesso modo, ma si effettua poi una AND con 11110000. Il risultato è che tutti i bit corrispondenti agli 0 della AND vengono messi a 0, gli altri rimangono indisturbati. Questa operazione si chiama anche «maschera AND», in quanto «lascia passare» i bit corrispondenti agli 1, mentre azzera tutti gli altri.

```
MOVLW   00110011B  ;Carica 51 nell'accumulatore
MOVWF   PIPPO      ;Lo mette nel registro PIPPO
MOVLW   11110000B  ;Carica 240 nell'accumulatore
ANDWF   PIPPO,F    ;PIPPO=00110000
```

Nell' ultimo esempio si usa la funzione logica XOR per scambiare lo stato dei bit corrispondenti ai suoi 1 e lasciare indisturbati gli altri:

```
MOVLW   00110011B  ;Carica 51 nell'accumulatore
MOVWF   PIPPO      ;Lo mette nel registro PIPPO
MOVLW   10000001B  ;Carica 129 nell'accumulatore
XORWF   PIPPO,F    ;PIPPO=10110010
```

L'ultima istruzione del gruppo,la (COMF) effettua semplicemente una negazione dei livelli logici (NOT), il valore 00110011 diventerebbe 11001100. E'da notare che una COMF è identica ad uno XOR con tutti i bit a 1, ma richiede una sola istruzione mentre un'operazione di XOR ne richiederebbe 2.

E'utile spendere qualche altra parola sullo XOR, in quanto permette dei trucchetti non immediatamente evidenti. Questi si basano sul fatto che un doppio XOR con lo stesso valore riporta al valore iniziale. è abbastanza semplice infatti comprendere che se il primo XOR inverte alcuni bit, il secondo li riinverte riportandoli al loro valore originale. Questa caratteristica permette anche di «mescolare» i bit di due registri e di ricostruirli in posizioni diverse della memoria. Come esempio Immaginiamo di voler scambiare tra di loro il contenuto dell'accumulatore e quello del registro PIPPO. A prima vista servono almeno altri due registri temporanei (chiamaiamoli TEMP1 e TEMP2) in cui depositare i valori iniziali dell'accumulatore e di PIPPO:

```
MOVWF   TEMP1    ;Salva accumulatore in TEMP1
MOVF    PIPPO,W  ;W=PIPPO
MOVWF   TEMP2    ;Salva PIPPO in TEMP2
MOVF    TEMP1,W  ;Recupera valore originale di W
MOVWF   PIPPO    ;Lo mette in PIPPO
MOVF    TEMP2,W  ;W = Vecchio valore di PIPPO
```

Sfruttando le caratteristiche dell'operazione logica XOR è possibile evitare l'uso di registri temporanei e ridurre le istruzioni necessarie solamente a 3:

```
XORWF   PIPPO,F
XORWF   PIPPO,W
XORWF   PIPPO,F
```

La prima non altera il valore di W, ma in PIPPO si viene a trovare il risultato di PIPPO XOR W. La seconda effettua di nuovo uno XOR tra W e PIPPO, il risultato è perciò complessivamente PIPPO XOR W XOR W, cioè il valore inizialmente

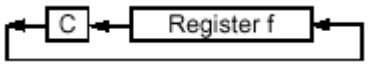
contenuto in PIPPO, che viene tenuto in W. Infine si effettua un terzo XOR tra PIPPO (che contiene sempre l'iniziale PIPPO XOR W) e W che contiene il valore iniziale di PIPPO, il risultato dell'operazione è complessivamente PIPPO XOR W XOR PIPPO, che è perciò il valore iniziale di W che viene salvato in PIPPO... i due valori hanno così cambiato di posto. Sfruttando lo stesso principio è possibile anche scambiare tra di loro il valore di due registri (chiamiamoli REG1 e REG2) senza usarne altri «di appoggio»:

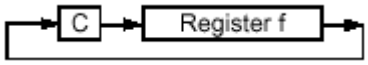
```
MOVF    REG1,W
XORWF   REG2,F
XORWF   REG2,W
XORWF   REG2,F
MOVWF   REG1
```

ROTAZIONI E SET/RESET DI SINGOLI BIT

```
RLF   reg,d   C   d = rlf (reg)
RRF   reg,d   C   d = rrf (reg)
BCF   reg,b       Bit b di (reg) = 0
BSF   reg,b       Bit b di (reg) = 1
```

Le istruzioni RLF e RRF ruotano rispettivamente a sinistra o a destra i bit contenuti in un registro. Il risultato è depositato nell'accumulatore o nel registro stesso, la rotazione avviene sempre attraverso il flag C come mostrato nelle due figure seguenti:

RLF: 

RRF: 

Nella RLF il bit 7 del registro specificato viene spostato nel flag C, mentre il vecchio valore di C rientra nel bit 0 del registro dopo che tutti i bit sono stati spostati di una posizione a sinistra. La RRF funziona nello stesso modo, solo che la rotazione avviene nell'altro senso. L'utilità può non essere evidente, ma queste sono in realtà istruzioni molto potenti, che permettono di risolvere e semplificare numerosi problemi, per esempio legati al controllo di ogni singolo bit di un registro, alla serializzazione dei bit durante una trasmissione o al loro «riassemblaggio» in ricezione. Inoltre va ricordato che spostare a sinistra o a destra di una posizione i bit di un registro equivale rispettivamente a moltiplicare o dividere per 2 il suo valore numerico.

Le ultime due istruzioni permettono di resettare (BCF) o settare (BSF) un qualsiasi bit di un qualsiasi registro lasciando invariati gli altri. Questo permette per esempio di usare i singoli bit di un registro come 8 semplici memorie a due stati, ottenendo così un grande risparmio nell'utilizzo di registri. Un esempio di registro usato in «bit mode» è lo STATUS, che non viene mai considerato come contenente un valore numerico, ma ogni suo bit ha invece un significato e utilizzo diverso e ben preciso, i flags C e Z sono due di questi bit. Per completezza è perfettamente lecito usare una BCF o BSF sui flag. Queste istruzioni funzionano naturalmente anche sui registri che comandano i pin configurati come uscite, e permettono di «alzare» o «abbassare» il livello della tensione in uscita anche di uno solo di essi in modo semplice (è così possibile generare dei segnali, anche delle frequenze audio se si

vuole). Il parametro «b» delle istruzioni BCF e BSF indica il bit su cui agire, il suo valore va da 0 (bit meno significativo) a 7 (bit più significativo).

L'esempio seguente genera un impulso positivo della durata di un ciclo macchina dal pin RB0 (con clock di 4MHz l'impulso dura 1µS):

```
BSF     PORTB,0
BCF     PORTB,0
```

Importante! Bisogna sempre fare attenzione a non confondersi quando si indica il numero del bit all'interno di un byte. Siccome i bit sono numerati da 0 a 7, il bit 3 non è il terzo, ma il quarto!

[Segue]

Pagina creata nel gennaio 2004 - Ultimo aggiornamento 25-3-2006