LE TABELLE DI LOOKUP
azioniamo un motore

Nella schedina sperimentale che abbiamo costruito c'è una morsettiera a sette poli facente capo all'integrato ULN2004.
Questo altro non è che un driver contenente 7 darlington NPN open collector in grado di sopportare fino a 500mA ciascuno. Il pin di uscita di ciascuno di questi transistor verrà chiuso a massa quando il corrispondente pin di pilotaggio verrà attivato applicandogli +5v ovvero un uno logico.
Un carico deve quindi essere collegato fra il positivo dell'alimentazione e il morsetto di uscita.
Si ribadisce che l'uscita è open collector, vale a dire che ponendo un uno sul comando si ottiene uno zero in uscita ma non è vero il contrario !
Ponendo uno zero logico sul pin di comando l'uscita viene semplicemente posta in stato di alta impedenza e non và automaticamente a livello alto.
Collegando un carico fra morsetto di uscita e massa non faremo quindi funzionare alcunchè...

La corrispondenza fra pin di ingresso e pin di uscita dell'ULN, guardando lo schema elettrico, è in senso orizzontale ovvero il pin 1 comanderà il pin 16, il pin 2 comanderà il pin 15 e così via.
Sui pin di ingresso è riportato anche il collegamento alle porte del pic, quindi tirando le somme si deduce che ponendo un uno logico sul pin RA0 del pic noi chiudiamo a massa il morsetto uno di uscita della schedina.

All'interno dell'integrato ULN ci sono dei diodi di protezione dalle extracorrenti. Ce ne è uno per ogni uscita con tutti i catodi collegati insieme al pin 9 che guarda caso noi abbiamo cablato al +12 dell'alimentazione. Se un carico di tipo induttivo dovesse generare una extracorrente quando diseccitato, questo diodo proteggerà il transistor d'uscita scaricando questa corrente inversa sull'alimentazione che si suppone avere una impedenza sufficientemente bassa.
Questo però limita anche il valore massimo di alimentazione da fornire al carico, 12 v. nel nostro caso ovvero l'alimentazione in ingresso alla schedina non stabilizzata.

Cosa possiamo collegare alle uscite di questo ULN ?
Un display gigante, un relè di potenza, un'elettrocalamita, un piccolo motorino in CC o.....
gli avvolgimenti di un motore a passo.

I motori a passo o stepper sono un tipo di motore un po particolare.
Possiedono **sempre** 4 avvolgimenti che collegati e pilotati in vario modo ci consentono di posizionare con precisione l'albero di uscita.
Potete immaginare uno stepper nella sua forma piu semplice come una serie di 4 elettrocalamite che eccitate in sequenza attirano i poli di una calamita permanente montata solidale con l'albero.
Ogni volta che azioniamo una elettrocalamita differente e facciamo avanzare l'albero compiamo **un passo**.
Se la sequenza è giusta potremo far ruotare in maniera continuativa l'albero motore.
Il vantaggio di uno stepper rispetto ad altri tipi di motore è che contando il numero di passi e sapendone le caratteristiche meccaniche noi possiamo posizionare l'albero con precisione, ed ottenere anche una discreta coppia di stallo mantenendo alimentate le fasi una volta terminato il posizionamento.

Per un sistema a microprocessore è molto semplice comandare in sequenza 4 uscite ad intervalli regolari di tempo. Contandone il numero di commutazioni, risulterà altrettanto semplice muovere in qualunque posizione il nostro motore !

Scanner e stampanti utilizzano questo sistema a basso costo per muovere i propri carrelli, nei sistemi industriali invece si usa oggigiorno ben altro... ma a noi non interessa !

Gli avvolgimenti all'interno di un motore sono sempre 4, ma purtroppo il modo di collegarli ha diverse varianti.
Noi utilizzeremo per questo esperimento un motore unipolare. Lo riconoscete perché ha 5 o 6 fili di collegamento. Se ne avete un modello che ha solo 4 connessioni significa che è bipolare e non va bene per la nostra applicazione.
Il vantaggio nell'uso della configurazione unipolare è nella grande semplicità di pilotaggio, bastano 4 delle nostre uscite di potenza dall'integrato ULN.
IL sistema bipolare prevederebbe ponti di transistor configurati ad H ... troppo complicato per un esercizio didattico come il nostro.

NON ci concentreremo sulla teoria di funzionamento di uno stepper, ci fidiamo di un paio di tabelle precompilate su come pilotare in sequenza i 4 avvolgimenti del motore detti anche **fasi**.

quella che segue è la prima delle due tabelle che vedremo:

Step	*Q1 P1.0, Black*	*Q2 P1.1, Yellow*	*Q3 P1.2, Orange*	*Q4 P1.3, Brown*	*Value*
1	ON	OFF	ON	OFF	1010 = 10
2	ON	OFF	OFF	ON	1001 = 9
3	OFF	ON	OFF	ON	0101 = 5
4	OFF	ON	ON	OFF	0110 = 6
1	ON	OFF	ON	OFF	1010 = 10

A sinistra vedete il numero sequenziale di passo di rotazione che compie il motore, guarda caso il numero di passi al giro che i motorini hanno è sempre un multiplo di 4...
In una rotazione continua si comincia con l'1 e si finisce con il 4, la sequenza si ripete poi dall'inizio con il numero 1, 2 e così via. Se il vostro motore dichiara sui dati di targa 200 passi/giro significa che ripetendo 50 volte la sequenza di cui sopra farete compiere all'albero un giro intero ritornando **esattamente** al punto di partenza.

ON OFF sono gli stati che dovremo far assumere alle 4 fasi, i colori dei fili sono assolutamente indicativi dal momento che non esiste unificazione fra costruttori. Difficile sarà per un motore ignoto determinare la sequenza corretta degli avvolgimenti, l'unica cosa da fare è provare !

All'accensione del nostro sistema lo stato di posizione del motore non è noto, l'alberino potrebbe essere al passo n.1 così come al n 3, ma noi ignoriamo questo problema. Al massimo i primi 4 passi di rotazione dell'albero saranno scoordinati dato che la sequenza potrebbe non essere quella corretta ma a noi non interessa.

Ci prefiggiamo inizialmente di creare un programma che ci permetta di ruotare il motore in una sola direzione a velocità variabile. Per quest'ultimo compito utilizzeremo la routine di ritardo variabile creata in precedenza, questo è un'ottimo esempio di come la struttura a subroutine del programma ci

permetta di riutilizzarne alcune parti !

Stepper

Il listato per pilotare il motore è il seguente:

```
        list P=16f84A , R=DEC
        #INCLUDE P16F84A.inc

        __FUSES  _CP_OFF&_WDT_OFF&_XT_OSC

        CBLOCK 0ch
        Timer1                  ;Subcontatore per timer
        Timer2                  ;Subcontatore per timer
        Timer3                  ;Contatore per timer
        Time_set                ;valore programmabile di conteggio tempo
        StepNo                  ;numero di step attuale
        ENDC

Timer2Preset=5                  ;valore presettato di subtimer2
Timer3Preset=40                 ;valore iniziale per velocita' rotazione motore

;inizio programma dopo un reset

        org 0

;inizializzazione sistema e variabili

        bsf STATUS,5
        movlw 01110001b                 ;0=uscita 1=ingresso
        movwf TRISB
        movlw 11110000b
        movwf TRISA
        bcf STATUS,5

        movlw Timer3Preset              ;valore iniziale di Time_set
        movwf Time_set

        movlw 1                                 ;inizializza posizione stepper a 1
        movwf StepNo

;loop principale programma

Loop
        incf StepNo,f  ;incrementa contatore posizione stepper
        movlw 5                         ;il range deve essere 1-4
        subwf StepNo,w          ;sottrae w a StepNo ma NON altera StepNo stesso !!!
```

```
        btfss STATUS,Z          ;se e' diventato zero, allora StepNo e' diventato 4
        goto StepNo4
        movlw 1                         ;quindi lo dobbiamo reinizializzare ad 1
        movwf StepNo
StepNo4

        movlw 1                         ;guarda se siamo al passo n.1
        subwf StepNo,w
        btfsc STATUS,Z
        call Step1              ;se affermativo allora setta le fasi del motore in maniera
corrispondente

        movlw 2
        subwf StepNo,w
        btfsc STATUS,Z
        call Step2

        movlw 3
        subwf StepNo,w
        btfsc STATUS,Z
        call Step3

        movlw 4
        subwf StepNo,w
        btfsc STATUS,Z
        call Step4

        call Delay              ;ritardo programmabile
        call Button             ;aggiorna il valore del timer in base allo stato dei pulsanti

        goto Loop               ;ricomincia da capo



;subroutine Step1: aziona motorino stepper in posizione 1

Step1
        bcf PORTA,0
        bsf PORTA,1
        bcf PORTA,2
        bsf PORTA,3
        return


;subroutine Step2: aziona motorino stepper in posizione 2

Step2
        bsf PORTA,0
```

```
        bcf PORTA,1
        bcf PORTA,2
        bsf PORTA,3
        return

;subroutine Step3: aziona motorino stepper in posizione 3

Step3
        bsf PORTA,0
        bcf PORTA,1
        bsf PORTA,2
        bcf PORTA,3
        return

;subroutine Step4: aziona motorino stepper in posizione 4

Step4
        bcf PORTA,0
        bsf PORTA,1
        bsf PORTA,2
        bcf PORTA,3
        return


;subroutine Delay. Ritarda l'esecuzione di un valore programmabile da Time_set

Delay   clrf Timer1                 ;azzera i timer
        movlw Timer2Preset          ;presetta timer2... vedi inizio programma...
        movwf Timer2
        movf Time_set,w             ;preleva il settaggio di tempo
        movwf Timer3

D1      nop
        nop
        decfsz Timer1,f
        goto D1
        decfsz Timer2,f
        goto D1
        movlw Timer2Preset  ;dato che non ricomincia da zero, timer2 va' ogni volta ricaricato
        movwf Timer2
        decfsz Timer3,f
        goto D1
        return


;subroutine Button , legge lo stato dei pulsanti e modifica il valore di Time_set

Button
```

```
        ;pulsante 1: incrementa timer
        btfsc PORTB,4           ;leggi lo stato del pulsante 1
        incf Time_set,f         ;se premuto incrementa il valore del settaggio tempo

        ;pulsante 2: decrementa timer
        btfss PORTB,5           ;controlla ora il pulsante di decremento
        goto Butt_end           ;esce se NON premuto
        decf Time_set,f         ;decrementa il timer
        btfss STATUS,Z          ;controlla se diventato zero
        goto Butt_end           ;esce se non diventato zero
        movlw 1                         ;ricarica il timer con il valore minimo
        movwf Time_set
Butt_end

        ;pulsante 3: azzera timer al minimo
        movlw Timer3Preset              ;carica il valore minimo del timer
        btfsc PORTB,6           ;controlla il terzo pulsante
        movwf Time_set          ;se premuto carica il valore minimo timer nel timer stesso

        return                  ;esce dalla subroutine

        end
```

è un po lungo quindi non vi biasimo se lo pescherete già fatto dalla cartella *esempi*.

Si comincia con le solite direttive, e riserviamo poi memoria ram per le variabili utilizzate. Oltre ai già visti contatori di tempo creiamo anche una variabile chiamata *StepNo* . In questa variabile memorizzeremo il numero di passo attuale dell'albero motore. Questa variabile dovrà essere incrementata continuamente e ciclicamente da 1 a 4 per decidere in quale sequenza pilotare poi le fasi ed avere così una rotazione lineare e continuativa.
Avremo così due compiti:

- incrementare un contatore riposizionandolo ad 1 dopo il numero 4
- controllare la posizione attuale e pilotare di conseguenza le fasi del motore

Proseguendo nel listato troviamo due attribuzioni di valore:

```
Timer2Preset=5          ;valore presettato di subtimer2
Timer3Preset=40         ;valore iniziale per velocita' rotazione motore
```

Con queste due righe indichiamo all'assemblatore che dovrà sostituire le etichette *Timer2Preset* e *Timer3Preset* con i rispettivi valori 5 e 40 ogni qual volta che le incontrerà lungo il listato.
In questo modo quando successivamente incontreremo ad esempio l'istruzione:

movlw Timer3Preset

questa verrà tradotta in

```
movlw 40
```

dall'assemblatore e in modo assolutamente automatico.
A cosa serve ?
Se il timer3 deve essere caricato con questo valore di preset in più di un punto del programma, qualora noi vorremo variare il valore 40 ad esempio in 50, ci basterà farlo solo all'inizio del listato senza dover scorrere e modificare tutto il programma compiendo magari anche errori o dimenticanze.
Inoltre mantenere tutti i valori che pensiamo essere modificabili all'inizio del listato ci permetterà la messa a punto del sistema in un tempo inferiore, non dovremo cercare qua e là un timer da aumentare o diminuire!

Appena poche istruzioni piu sotto infatti carichiamo i contatori *Timer2* e *Timer3* con i valori poco prima attribuiti.
In questo listato non facciamo compiere al timer 2 il massimo conteggio di 256 dato che vogliamo ottenere dei tempi relativamente corti per poter vedere ruotare il motorino ad una velocità significativamente alta.

Inizializzati i due timer passiamo al contatore di posizione corrente dello stepper che mettiamo ad uno ovvero il primo passo. All'inizio del ciclo qualunque valore fra 1 e 4 andrebbe bene dato che non sappiamo lo stato del motore, ma un valore maggiore di 4 non va bene quindi il contatore dobbiamo comunque inizializzarlo. Il contenuto della ram è imprevedibile all'accensione !

Poi si comincia con il loop principale di programma.
La prima cosa che facciamo è incrementare il contatore di posizione corrente:

```
incf StepNo,f  ;incrementa contatore posizione stepper
```

notate che con l'appendice *,f* riportiamo nella locazione *StepNo* il valore incrementato di uno.

Al primo loop di programma il contatore diventa 2, poi 3, 4 e poi ?
5 !
Quando il contatore diventa 5 allora subito noi dobbiamo reinizializzarlo ad uno, la posizione 5 non esiste !
Questo compito viene assolto dal codice seguente:

```
        movlw 5                 ;il range deve essere 1-4
        subwf StepNo,w          ;sottrae w a StepNo ma NON altera StepNo stesso !!!
        btfss STATUS,Z          ;se il risultato è zero, allora StepNo e' diventato 4
        goto StepNo4
        movlw 1                 ;quindi lo dobbiamo reinizializzare ad 1
        movwf StepNo
StepNo4
```

che significa letteralmente:

- prendi il valore 5
- sottrailo alla locazione di memoria StepNo lasciando il risultato in W e NON in *StepNo* per

evitare di alterarlo inutilmente
- controlla le flag del registro STATUS settate della precedente istruzione matematica. Se la flag di Zero è settata allora vuol dire che il risultato della sottrazione è zero ovvero *W* e *StepNo* erano uguali cioè *StepNo* vale 5. Se questa condizione è vera salta l'istruzione che segue.
- *Goto StepNo4* verrà quindi eseguita solo se *W* NON è 5.
- Carichiamo il valore 1 nella locazione di memoria *StepNo* con le due istruzioni rimanenti.

Il risultato sarà che il contatore *StepNo* verrà incrementato ogni volta che questa parte di programma verrà eseguita, ma se il valore diverrà 5 allora lo reinizializziamo immediatamente ad 1, con il risultato che *StepNo* seguirà sempre la sequenza 1,2,3,4,1,2,3,4,1,2...
Ci sarà un'istante in cui *StepNo* varrà 5, ma sarà solo per pochi microsecondi e comunque prima che altre parti di programma lo possano leggere.

Abbiamo introdotto qui un nuovo algoritmo. Utilizziamo l'operazione matematica di sottrazione per vedere se due valori sono uguali fra loro.

Nei pic l'istruzione *subwf* è sempre fonte di problemi. Il problema è che si fa spesso confusione fra chi sottrae cosa. Nel caso di una sottrazione è importante capire bene chi è il minuendo ed il sottraendo altrimenti il risultato cambia !
Si può però essere aiutati dalla sintassi dell'istruzione stessa:

```
subwf StepNo,f
```

leggendola in sequenza: sottrai Work a StepNo cioè

```
risultato = StepNo - W
```

Un esempio aiuterà forse un poco a capire:

```
StepNo=5 W=4 risultato -> StepNo=1
StepNo=4 W=5 risultato -> StepNo= -1 (oppure 255 a seconda del sistema usato)
```

In altri microprocessori ci sono comodissime istruzioni per fare un confronto diretto fra valori, nel pic occorre ingegnarsi un poco, ma è l'unico modo disponibile.

Lo stesso sistema lo utilizziamo appena piu avanti per vedere a che passo siamo e saltare a 4 routine diverse che provvedono a settare-resettare i pin di comando delle fasi motore:

```
movlw 1                 ;guarda se siamo al passo n.1
subwf StepNo,w
btfsc STATUS,Z
call Step1              ;se affermativo allora setta le fasi del motore pos. 1
```

Call Step1 verrà eseguita solo se la flag *STATUS,Z* è settata, quindi se il risultato dell'operazione di sottrazione fra *StepNo* e W caricato con 1 è zero, cioè se *StepNo* è uguale ad 1.

I tre confronti seguenti sono assolutamente identici tranne che per i valori 2,3 e 4 e naturalmente gli

indirizzi di chiamata delle subroutine *Step2 Step3* e *Step4.*

Il resto:

```
        call Delay          ;ritardo programmabile
        call Button         ;aggiorna il valore del timer in base allo stato dei pulsanti
        goto Loop           ;ricomincia da capo
```

è preso pari pari dall'esempio precedente e non necessita di ulteriori spiegazioni.

Le quattro subroutine *Step1...4* sono dal canto loro molto elementari. Settano o resettano i pin di comando fasi motore secondo la tabellina che abbiamo visto prima. Per farlo usiamo semplicemente le istruzioni *bsf* e *bcf* nella maniera opportuna. L'unica cosa da notare è che rispetto alla tabellina ho attribuito :

- Q4 e PORTA,0 cioè morsetto 1
- Q3 a PORTA,1 -> morsetto 2
- Q2 a PORTA,2 -> morsetto 3
- Q1 a PORTA,3 -> morsetto 4

ho quindi invertito l'ordine Q1/4 rispetto alle uscite 4/1.

Nella subroutine Delay ricarichiamo ogni volta Timer2 al valore di preset definito con Timer2Preset definito all'inizio del programma. Se il motore ruoterà troppo velocemente o troppo lentamente... basterà modificare l'inizio del listato come anzidetto senza preoccuparsi che il valore venga ricaricato in ben tre punti diversi del listato.

Button è la stessa dell'esercizio precedente.

Vi do un consiglio:
Le routine che considerate ben fatte e potenzialmente utili in altre applicazioni salvatele in un file unico chiamato ROUTINE.ASM (che fantasia eh ???).
Io l'ho fatto per calcoli di conversione ascii-esadecimale o ricetrasmissione seriale quando non dialogo con display lcd.
Le avrete già fatte e pronte da implementare con poche modifiche all'occorrenza !

Ora assemblate il tutto, caricatelo e divertitevi a vedere il motorino girare, potete variare la velocità con i soliti 3 pulsanti. Magari mettete un pezzo di nastro adesivo sull'alberino per vedere meglio la rotazione.

Stepper migliorato
le tabelle di lookup

Il listato appena visto, pur essendo funzionante, può essere notevolmente migliorato.
Una prima innovazione consiste nell'estensione del numero di passi.
Un qualunque motore stepper puo funzionare in modo passo intero come appena visto e provato, ma può anche funzionare in modo **mezzo passo** pilotando le fasi come indica la seguente tabella:

Step	*Q1 P1.0, Black*	*Q2 P1.1, Yellow*	*Q3 P1.2, Orange*	*Q4 P1.3, Brown*	*Value*
1	ON	OFF	ON	OFF	1010 = 10
2	ON	OFF	OFF	OFF	1000 = 8
3	ON	OFF	OFF	ON	1001 = 9
4	OFF	OFF	OFF	ON	0001 = 1
5	OFF	ON	OFF	ON	0101 = 5
6	OFF	ON	OFF	OFF	0100 = 4
7	OFF	ON	ON	OFF	0110 = 6
8	OFF	OFF	ON	OFF	0010 = 2
1	ON	OFF	ON	OFF	1010 = 10

In questa modalità l'alberino si sposta ad ogni passo della metà rispetto a quanto indicato nei dati di targa. Avendo un motorino da 200 step/giro potremo quindi ottenere un raddoppio del numero di passi ovvero 400 step/giro. Potremo in taluni casi evitare un riduttore meccanico oppure ottenere un miglioramento nella precisione del sistema semplicemente con una modifica software.
Il rovescio della medaglia è una riduzione nella coppia fornibile sull'albero. Guardando bene la tabella si vede che il mezzo passo è ottenuto lasciando eccitata una sola fase fra due passi della tabella precedente. Una fase di meno significa anche meno forza... quindi meno coppia e di conseguenza riduzione della velocità massima di rotazione.

Un ciclo completo significa ora compiere 8 passi...
Conviene forse rivedere il nostro programma o ci rassegnamo a duplicare tutte le funzioni di posizionamento per portarle da 4 ad 8 passi?

Nel pic esiste un modo per costruire una tabella di dati che potrà essere consultata tramite un puntatore. Costruiremo quindi una cosiddetta tabella di lookup.
Se noi mettiamo i codici della colonna *value* che vedete qui spora in una tabella e li estraiamo sequenzialmente ecco che possiamo pilotare tutte le 4 fasi del motore con una unica routine cui dovremo fornire solo il numero di passo desiderato fra 1 e 8.

Il nuovo listato modificato segue:

```
list P=16f84A , R=DEC
#INCLUDE P16F84A.inc

__FUSES  _CP_OFF&_WDT_OFF&_XT_OSC

CBLOCK 0ch
Timer1              ;Subcontatore per timer
Timer2              ;Subcontatore per timer
```

```
        Timer3                  ;Contatore per timer
        Time_set                ;valore programmabile di conteggio tempo
        StepNo                  ;numero di step attuale
        ENDC

Timer2Preset=5                  ;valore presettato di subtimer2
Timer3Preset=40                 ;vvalore iniziale per velocita' ritazione motore

;inizio programma dopo un reset

        org 0

;inizializzazione sistema e variabili

        bsf STATUS,5
        movlw 01110001b         ;0=uscita 1=ingresso
        movwf TRISB
        movlw 11110000b
        movwf TRISA
        bcf STATUS,5

        movlw Timer3Preset      ;valore iniziale di Time_set
        movwf Time_set

        movlw 1                 ;inizializza posizione stepper a 1
        movwf StepNo

;loop principale programma

Loop
        bcf PORTB,1             ;spegne il led

        incf StepNo,f           ;incrementa contatore posizione stepper
        movlw 9                 ;il range deve essere 1-8
        subwf StepNo,w          ;sottrae w a StepNo ma NON altera StepNo stesso !!!
        btfss STATUS,Z          ;se e' diventato zero, allora StepNo e' diventato 9
        goto StepNo9
        movlw 1                 ;quindi lo dobbiamo reinizializzare ad 1
        movwf StepNo
        bsf PORTB,1             ;accendiamo anche il led
StepNo9

        movf StepNo,w           ;preleva step attuale
        call Tabella            ;converti il numero di passo in valore per PORTA
        movwf PORTA             ;emetti dato

        call Delay              ;ritardo programmabile
        call Button             ;aggiorna il valore del timer in base allo stato dei pulsanti
```

```
        goto Loop                   ;ricomincia da capo



;tabella posizioni stepper

Tabella
        addwf PCL                   ;W deve contenere il puntatore alla tabella
        nop                         ;il valore zero non lo usiamo
        retlw 10
        retlw 8
        retlw 9
        retlw 1
        retlw 5
        retlw 4
        retlw 6
        retlw 2


;subroutine Delay. Ritarda l'esecuzione di un valore programmabile da Time_set

Delay   clrf Timer1                 ;azzera i timer
        movlw Timer2Preset          ;presetta invece timer2 ad un valore prefissato
        movwf Timer2
        movf Time_set,w             ;preleva il settaggio di tempo
        movwf Timer3

D1      nop
        nop
        decfsz Timer1,f
        goto D1
        decfsz Timer2,f
        goto D1
        movlw Timer2Preset          ;dato che non ricomincia da zero, timer2 va' ricaricato
        movwf Timer2
        decfsz Timer3,f
        goto D1
        return


;subroutine Button , legge lo stato dei pulsanti e modifica il valore di Time_set

Button

        ;pulsante 1: incrementa timer
        btfsc PORTB,4       ;leggi lo stato del pulsante 1
```

```
        incf Time_set,f               ;se premuto incrementa il valore del settaggio tempo

        ;pulsante 2: decrementa timer
        btfss PORTB,5                 ;controlla ora il pulsante di decremento
        goto Butt_end                 ;esce se NON premuto
        decf Time_set,f               ;decrementa il timer
        btfss STATUS,Z                ;controlla se diventato zero
        goto Butt_end                 ;esce se non diventato zero
        movlw 1                       ;ricarica il timer con il valore minimo
        movwf Time_set
Butt_end

        ;pulsante 3: azzera timer al minimo
        movlw Timer3Preset            ;carica il valore minimo del timer
        btfsc PORTB,6                 ;controlla il terzo pulsante
        movwf Time_set                ;se premuto carica il valore minimo timer nel timer stesso

        return                        ;esce dalla subroutine


        end
```

Notate che è quasi piu corto del precedente nonostante esegua il doppio di lavoro.

Fino alla etichetta *Loop* il istato è sempre lo stesso (ormai comincia ad annoiarmi !).

Accendiamo e spegniamo anche un led fra i tre disponibili, ci aiuterà nel debug. Iniziamo a led spento. Cosa che fa l'istruzione:

```
        bcf PORTB,1                   ;spegne il led
```

Segue lo stesso algoritmo precedente di incremento del contatore *StepNo* , l'unica aggiunta è l'accensione del led ogni volta che il contatore viene reinizializzato. Dato che il led viene spento all'inizio del loop principale, ne otterremo l'accensione solo durante l'esecuzione del passo n.1 .

Subito dopo cominciano le novità:

```
        movf StepNo,w                 ;preleva step attuale
        call Tabella                  ;converti il numero di passo in valore per PORTA
        movwf PORTA                   ;emetti dato
```

preleviamo il valore di StepNo in W. Per farlo usiamo l'istruzione *movf StepNo,w* la sintassi è abbastanza intuitiva,

MOVe File *indirizzo sorgente,destinazione*

dove *destinazione* può essere W oppure F che l'assemblatore in realtà tradurrà in 0 o 1. Se mettiamo

W il dato viene mosso dalla locazione specificata verso il registro W altrimenti viene preso, e subito riposto nuovamente nella locazione di origine.
Potreste lecitamente chiedervi a cosa serve prelevare un dato e subito riporlo !
Se consultate il datasheet scoprirete però che questa operazione coinvolge la flag Z del registro STATUS . Questo significa che con una istruzione:

movf *indirizzo*,f

possiamo testare, guardando subito dopo STATUS,Z , se la locazione *indirizzo* contiene un valore zero, senza dover scomodare W o fare ulteriori sottrazioni.

Nel nostro caso usiamo l'istruzione semplicemente per prelevare il valore di StepNo nel registro W.

Segue una call ad una routine un po speciale:

```
Tabella
        addwf PCL               ;W deve contenere il puntatore alla tabella
        nop                     ;il valore zero non lo usiamo
        retlw 10
        retlw 8
        retlw 9
        retlw 1
        retlw 5
        retlw 4
        retlw 6
        retlw 2
```

Per capire come funziona occorre prima comprendere come il pic calcola l'indirizzo delle istruzioni da eseguire dalla memoria programmi.

Nei pic di fascia media l'indirizzo della prossima istruzione da eseguire proviene da un registro a 13 bit. Questo significa anche che la massima estensione della memoria programmi è limitata a 8192 locazioni ma nei micro che noi usiamo (16x84 e 16f628) il problema non ci riguarda dato che solo le prime 1024 / 2048 locazioni sono fisicamente implementate.

In realtà questo registro da 13 bit fisicamente non esiste. Ne esistono invece due da 8 bit che vengono combinati insieme dal processore per avere i 13 bit necessari. Uno fornisce la parte bassa dell'indirizzo ed da un'altro vengono prelevati i restanti 5 bit necessari.
Questi due registri sono chiamati PCL e PCH cioè Program Counter Low e Program Counter High.

Il byte basso cioè PCL è un registro che possiamo leggere e scrivere ed è all'indirizzo 2 della memoria ram. PCH che è la parte alta dell'indirizzo invece non è a noi accessibile direttamente. Abbiamo accesso solo ad un'altro registro chiamato PCLATH (Program Counter LaTch High) e che funge da memoria intermedia fra noi e PCH.
Quando noi scriviamo un valore su PCL modifichiamo immediatamente il puntatore alla prossima istruzione da eseguire, mentre se carichiamo un valore in PCLATH questo verrà effettivamente utilizzato solo all'esecuzione di una istruzione GOTO, CALL o alla scrittura di PCL. Durante l'esecuzione di tutte le altre istruzioni PCLATH resterà inutilizzato.

Se scriviamo un valore in PCL il programma devia immediatamente il suo corso sequenziale. Possiamo utilizzare questa caratteristica per creare dei GOTO calcolati in base ad un valore. Il limite è dato dal valore massimo di PCL a 256 , valore che ci limita la lunghezza massima di salto. Immaginiamo ora di suddividere la nostra memoria programmi in blocchi da 256 byte. Nel 16x84 ne avremo 4. Cosa succede se tentiamo di scrivere su PCL con una istruzione troppo vicino al limite superiore del blocco attuale di 256 byte ?
Che il programma NON devierà verso la posizione richiesta con conseguenze tutt'altro che prevedibili.
Fortunatamente è molto facile evitare questo. Nei programmi molto corti semplicemente basta verificare con l'aiuto di MPLAB che non abbiamo superato la lunghezza di 256 byte. Nei programmi un po piu lunghini basta usare una direttiva ORG per posizionarsi all'inizio di uno dei quattro blocchi possibili prima di scrivere su PCL.

Il disegno che segue illustra il funzionamento di PCL-PCH.

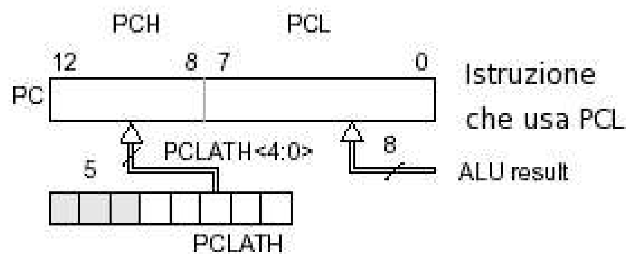

fig. 1

PCLATH serve invece per superare un limite del set di istruzioni del pic. Infatti qualunque istruzione di salto del pic CALL o GOTO ha a disposizione solo un indirizzo di 11 bit che limita la massima estensione del salto ai primi 2k. Siccome esistono processori che hanno fino a 8 k di memoria, scrivendo un valore nei bit 3 e 4 di PCLATH questo verrà utilizzato per esprimere l'indirizzo totale a cui saltare.

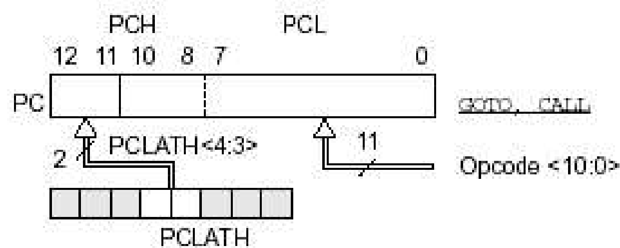

fig. 2

Dato che operiamo su processori che hanno solo 1-2 k di memoria noi vedremo solamente l'uso di PCL. PCLATH viene resettato dal pic all'accensione e per quanto ci riguarda verrà sempre lasciato a zero. I salti GOTO o CALL ricadono d'altra parte nel caso di fig.2 che pure non ci richiede intervento diretto.

Tornando al nostro esempio ricordiamo che prima di chiamare la routine *Tabella* abbiamo depositato in W il valore di passo con cui eccitare le fasi del motore.

Saltiamo a Tabella e la prima istruzione che incontriamo è

```
    addwf PCL
```

addwf significa somma il contenuto di W alla locazione file specificata che nel nostro caso è PCL.

Sommiamo quindi W, che è il numero di passo desiderato, al contenuto attuale di PCL. Dato che abbiamo omesso suffissi *,f* o *,w* il processore ripone il valore appena calcolato di nuovo in PCL.

Se non avessimo alterato PCL questo sarebbe stato normalmente incrementato di uno e l'istruzione successiva sarebbe stata prelevata ed eseguita. Noi però vi sommiamo un valore quindi la prossima istruzione eseguita sarà quella successiva la *addwf* più il valore di W.
Se W vale zero non cambierà nulla, se W varrà 1 l'istruzione eseguita sarà *retlw 10* , se W varrà 2 l'istruzione eseguita sarà *retlw 8* e così via.
Abbiamo creato un salto calcolato sul valore di W che, lo ricordiamo, conteneva un valore 1-8 pari al passo da eseguire.

retlw d'altro canto è simile alla return che già conosciamo, ma prima di uscire dalla subroutine e tornare al codice chiamante, carica W con il valore indicato nell'istruzione stessa. *retlw* significa infatti *RETurn Literal to Work.*

Eseguire una *retlw 10* comporta caricare il valore 10 in W e poi uscire dalla subroutine.
Usando 8 istruzioni *retlw* abbiamo costruito una vera e propria tabella con i valori necessari a pilotare il nostro motore.
Dato che il valore zero non è contemplato abbiamo per chiarezza inserito una *nop* alla locazione zero della tabella.

Una volta tornati al programma principale l'istruzione

```
    movwf PORTA             ;emetti dato
```

depone il valore appena preso dalla tabella direttamente sulla PORTA settando-resettando contemporaneamente tutti i 4 bit necessari. A dire il vero resettiamo sempre anche il PORTA,4 ma tanto nella nostra applicazione non lo usiamo.....

Il programma risulta piu elegante, facilmente modificabile e flessibile.
Se vogliamo modificare la sequenza della fasi ci basta ricalcolare i valori da applicare a PORTA e metterli in una unica tabella.

Assemblatelo e caricatelo nel micro. Provatelo e fantasticateci un po su....

> Provate ad immaginare...
>
> due motorini montati sull'accordatore che avete sul tetto... li muovete in remoto di un passo alla volta fino a trovare il punto di accordo... memorizzate la posizione in passi con un'altro pulsante nella memoria eerom che piu avanti impareremo ad utilizzare... potrete richiamare quello o altri accordi in un baleno !

Modificate il programma in modo che all'accensione esegua solo un numero limitato di passi, ad esempio 30, e poi il sistema si fermi (basta una goto che salti sempre su se stessa)

vi servirà un'altro contatore...