Esercitazioni PIC Baseline

4A - Assembly - Sfruttiamo le possibilità degli I/O

Abbiamo visto nelle esercitazioni precedenti come sia possibile comandare un pin del microcontroller come uscita digitale e con questo accendere e spegnere un LED.

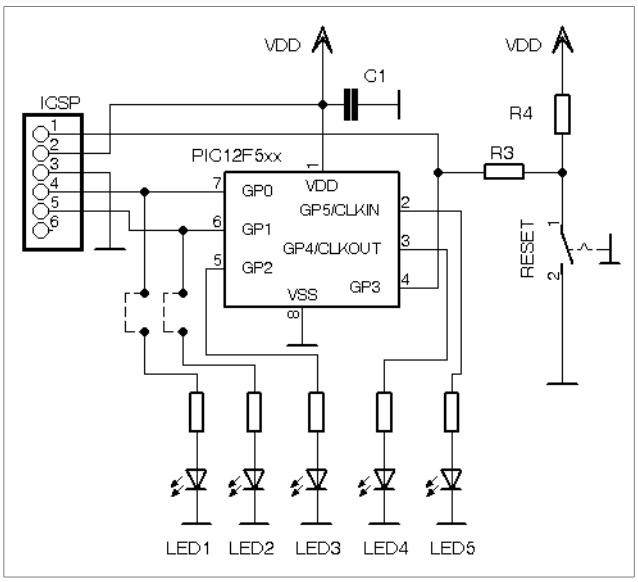
Dovrebbero essere evidenti le possibilità che si offrono all' utilizzatore; qui ne vediamo una: quella di comandare in sequenza più LED.

E' la base di un gran numero di applicazioni pratiche. Al posto dei LED, usando un **buffer opportuno**, si potranno collegare relais ed altri attuatori che permettono di accendere e spegnere carichi in sequenze temporizzate (sequenziatori).

Cosa vogliamo ottenere

Lo scopo dell' esercitazione è quello di comandare in sequenza alcuni LED attraverso il microcontroller.

Ci divertiamo un attimo con qualche gioco di luci, utilizzando i soliti LED e verificando come sia possibile con facilità ottenere interessanti risultati.
Utilizziamo sempre il **12F519** (ma va bene qualsiasi PIC, con gli opportuni adattamenti al sorgente)
Inoltre saranno proposte le variazioni per 12F508/509 e 16F505/526.
Per questa esercitazione non consideriamo i micro PIC 10F2xx che, avendo solamente 3 pin in uscita permettono poche variazioni (ma possono essere sufficienti per particolari impieghi e, come negli esercizi precedenti, il sorgente può essere adattato senza fatica).



Lo schema elettrico è quello dell' esercitazione 1, a cui aggiungiamo:

- un LED con relativa resistenza sui tutti i pin configurabili come uscite digitali (`GP0/1/2/4/5`)
- il pulsante del reset con le resistenze R4 (10k) e R3 (1k).

Il pulsante sarà utilizzato per evidenziare la funzione del pin MCLR.

R4 è il pull-up e R3 serve ad isolare il pulsante dalla connessione ICSP.

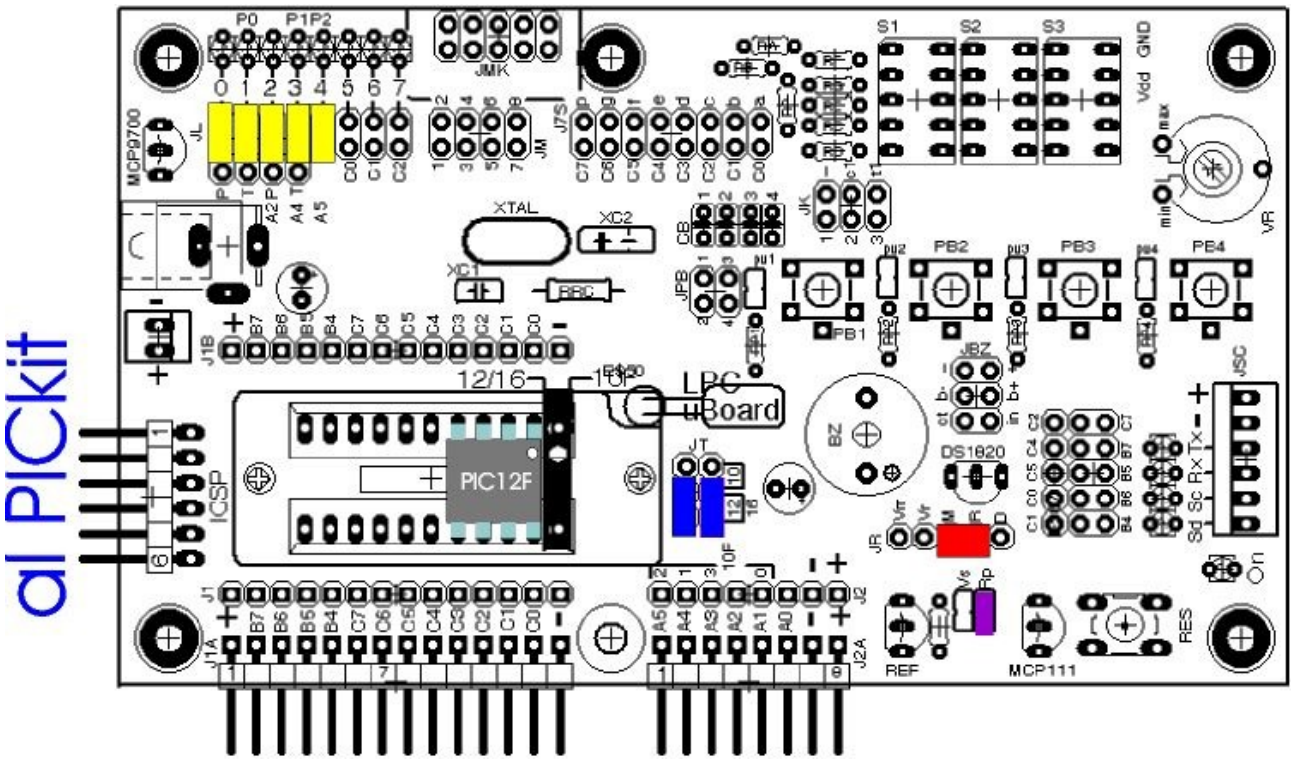
Osserviamo che i LED 1 e 2, che dipendono da `GP0`/`GP1` sono isolabili con ponticelli. Questo è necessario dato che i due pin sono utilizzati dalla connessione **ICSP** per il clock e i dati durante la

scrittura della memoria programma.
I segnali di queste due linee devono essere fondamentalmente privi di elementi di disturbo.

Se, come in questo caso, servono anche a comandare un carico, occorre che questo sia trascurabile come assorbimento e capacità aggiunta, onde non deformare i segnali della connessione **ICSP**. Quindi è opportuno prevedere jumper per scollegare questo carico durante la programmazione del chip. Nel caso della **LPCuB** i LED sono del tipo a bassissima corrente (2mA circa), il che permette di mantenerli inseriti anche durante la programmazione; in questa fase si potrà osservarne un rapido lampeggio. Se usate circuiti diversi, con LED che assorbono correnti maggiori, per precauzione meglio inserire i jumper come dallo schema precedente.

Questi collegamenti sono già presenti sulla demo board, mentre se usiamo una breadboard vanno eseguiti una volta per tutte, dato che questo schema di base sarà utilizzato in gran parte delle altre esercitazioni.

Per la **LPCuB**:



Come detto all'inizio, per questa esperienza si possono usare senza problemi anche i PIC10F, ma, dato che hanno solo 3 pin utili come uscite digitali, il risultato sarebbe meno interessante. Questo però non impedisce di usare questi chip per applicazioni dove siano necessarie al massimo tre uscite.

Sarà invece proposta una versione per Baseline a 14 pin, come PIC16F526, in grado di comandare 8 LED.

Il sorgente

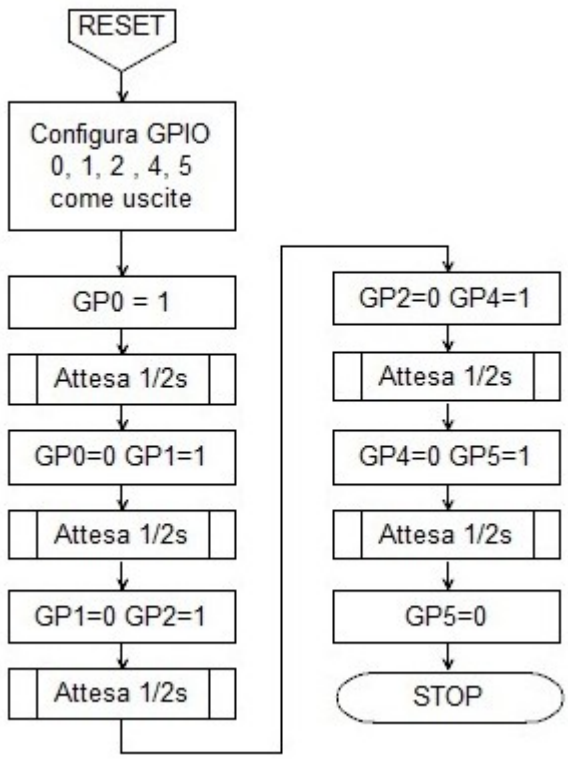
Come sempre, la prima cosa è quella di avere ben chiaro cosa si vuole ottenere :

- **accendere in successione 1 LED lungo la linea dei 5 LED collegati alle uscite del microcontroller**

e come:

- **programmando i pin GP0/1/2/4/5 come uscite digitali e portandoli uno dopo l' altro a livello alto a distanza di 1/2 secondo, spegnendo il precedente.**

Quindi tracciamo il diagramma di flusso che mostra in forma grafica quanto dovrà essere eseguito dal programma.



Al reset, **`GP0/1/2/4/5`** vengono programmati come uscite digitali.
`GP3` è escluso in quanto, ricordiamolo, il pin **`GP3`**/**`MCLR`** può essere utilizzato solamente come ingresso digitale.

`GP0` viene portato a livello 1 (LED1 acceso); dopo 1/2 secondo viene spento e viene acceso il LED collegato a **`GP1`** e così per gli altri.

Se i LED sono posti uno accanto all' altro su una fila, si otterrà l' effetto visivo di un punto luminoso che si muove lungo la linea.

Terminata la sequenza, il programma si arresta dopo aver spento il LED collegato al **`GP5`**.

La routine di tempo è la stessa utilizzata nell' esercitazioni precedenti, **`Delay10msW.asm`**.

Osserviamo come, attraverso il digramma a blocchi, possiamo avere una chiara visone di quello che desideriamo ottenere e questo rende semplice la successiva la trasformazione in istruzioni**.**
Anche in questo esempio il sorgente è reso disponibile già completamente scritto, nel file **4A_519.asm** e **4A_5089.asm** per il PIC12F508/509.
E' sempre necessario che non lo accettiamo così come sta, ma ne facciamo una analisi dettagliata, in modo da vere ben chiare tutte le sue componenti ed il perchè delle varie scelte, anche se continuiamo ad utilizzare le strutture già viste, sia per il tempo che per la gestione della shadow del port di I/O.
Il sorgente, quindi, non presenta "novità" nelle sue parti generali, se non la diversa sequenza delle istruzioni necessarie a svolge il lavoro desiderato.

Sequenza

Per effettuare la sequenza delle accensioni e spegnimenti si possono utilizzare vari metodi. Qui vediamo come sia possibile introducendo una istruzione non ancora utilizzata, la rotazione.
Nel set di istruzione dei Baseline sono presenti due istruzioni di rotazione che fanno scorre i bit di una locazione di memoria a destra (**`rrf`**) o sinistra (**`rlf`**).

ROTATE

Le istruzioni di rotazione sono anche dette di shift (scorrimento) perché operano spostando a destra o a sinistra di una posizione il contenuto di un file.
La sintassi delle istruzioni di rotazione è:

[label]	sp	rrf/rlf	, f/w	sp	oggetto	sp	[; commento]

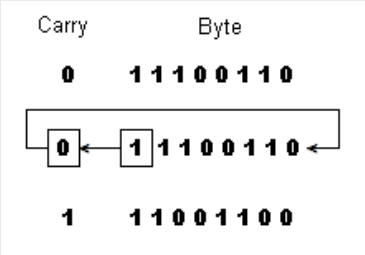
Si tratta di uno di quegli opcode già segnalati che necessitano di una "coda". Rammentiamo che la coda indica dove viene immagazzinato il risultato dell' operazione.

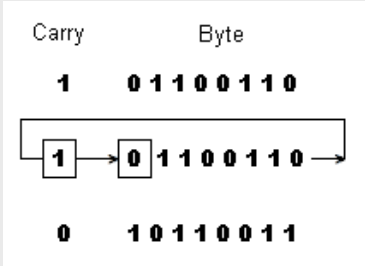
- Indicando **,f** si avrà il risultato della rotazione nel file stesso che è stato ruotato.
- Indicando **,w** si avrà il risultato in W, senza modificare il file in oggetto. Questo rende molto potente l' istruzione.

Come già detto, se omettiamo la "coda", **MPASM** segnalerà il problema intervenendo con il default **,f** , senza interrompere la compilazione e producendo il file ***.hex***. Ovviamente, se intendevamo questa scelta, non ci sono problemi, ma se la logica del programma richiedeva l' altra, il file eseguibile ottenuto non funzionerà come ci aspettiamo. Quindi, ripetiamo, è consigliato vivamente:

- completare sempre la "coda", in qualsiasi caso
- dare uno sguardo alla finestra di ***Output*** al termine della compilazione e verificare se sono presenti messaggi o warning, anche se è apparsa la linea **`BUILD SUCCEED`**

Questa istruzione, apparentemente strana, è in realtà indispensabile nelle operazioni matematiche.
Da un punto di vista pratico, le istruzioni di rotazione operano così:

RLF		***Rotate left file through Carry***, ruota il file ***f*** a sinistra attraverso il ***Carry***. Il risultato può andare in f o in W. Modifica il flag C dello STATUS.
		
		Esempio:
		`movlw b'11100110' ; target= 11100110`
		`movwf target ;`
		`bcf STATUS, C ; flag C = 0`
		`rlf target ; target = b'11001100'`
		`C = 0`

RRF		***Rotate right file through Carry***, ruota il file ***f*** a destra attraverso il ***Carry***. Il risultato può andare in f o in W. Modifica il flag C dello STATUS.
		
		Esempio:
		`movlw b'01100110' ; target= 01100110`
		`movwf target ;`
		`bsf STATUS, C ; flag C = 1`
		`rrf target, f ; target = b'10110011"`
		`C = 0`

Queste operazioni utilizzano come appoggio uno speciale bit del registro SFR chiamato **STATUS**: il bit di **C**, ovvero il ***Carry*** (solitamente corrispondente al bit 0 del registro).

Questo bit, come indica il nome, ha funzioni di "trasporto" del risultato di una operazione: il suo valore cambia a seconda del risultato di molte istruzioni. In particolare, per la rotazione, ha funzione di "accogliere" il bit espulso dalla rotazione stessa e di passare il suo valore al bit che si è svuotato nel registro che viene ruotato.

In particolare notiamo che la rotazione, di un bit per volta, viene effettuata come se il registro fosse "arrotolato" comprendendo il bit ***Carry*** dello STATUS.

In sostanza, ad ogni esecuzione della istruzione **RLF** tutti i bit si spostano di una posizione verso sinistra, includendo il contenuto del Carry nel bit 0 e passando al Carry il bit 7.
All' opposto opera la **RRF** , che ruota a destra. In sostanza, è come se il registro da ruotare e il Carry fossero avvolti su un cilindro.

Lo STATUS

Tra i vari **SFR**, il registro **STATUS** è sempre presente in tutti i microcontroller e svolge una funzione importante: i suoi bit sono essenzialmente delle **segnalazioni del risultato di istruzioni**.
Ecco, ad esempio, la mappa per il 12F519:

REGISTER 4-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
RBWUF	CWUF	PA0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

In particolare, nello **STATUS** dei Baseline si trovano bit con varie funzioni:

- **C** (Carry), **DC** (Digit Carry) e **Z** (Zero) sono bit di segnalazione (flag) e sono relativi al risultato di istruzioni.
- **T0** e **PD** sono bit di segnalazione (flag), relativi alle modalità con cui avviene un reset.
- **GPWUF** è un bit di segnalazione e indica un reset dovuto al cambio di livello dei pin, di cui vedremo più avanti la funzione
- **PA0** è un bit di controllo e serve come switch per il passaggio tra le pagine di memoria.

Nell' esempio qui sopra (12F508-509), il bit 6 non è usato, ma in altri chip può assumere una funzione. Ad esempio, in 16F526 la situazione è la seguente:

REGISTER 4-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
RBWUF	CWUF	PA0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

Qui i bit 7 e 6 sono segnalazioni, rispettivamente di reset dovuto al cambio di stato di un pin del **PORTB** (**RBWUF**) e alla commutazione del comparatore (**CWUF**) .
I rimanenti sono identici, in quanto la tendenza del costruttore è quella di mantenere quanto possibile la stessa struttura tra i vari chip, per facilitare l' utilizzatore. Però, nonostante questa tendenza, è opportuno verificare sul foglio dati del componente la reale situazione dei bit dello **STATUS** per il chip che si sta utilizzando.

I bit del registro sono scrivibili e leggibili, ma **quelli di segnalazione vengono aggiornati automaticamente** in seguito alle numerose operazioni dove è fondamentale che venga segnalata una condizione specifica del risultato:

- **C - Carry** ha importanza per indicare un riporto in somme o in sottrazioni (dove prende il nome di ***Borrow***).

- **Z - Zero** indica che il risultato di una operazione è zero
- **DC - Digit Carry** indica se si è verificato un carry nei 4 bit bassi ed è usato per operazioni di conversione

Al POR, questi bit, non essendo ancora stata effettuata alcuna istruzione, hanno valore casuale.

- **TO** e **PD** sono relativi alle condizioni da cui dipende il Reset. Al POR assumono entrambi il valore di 1. La loro funzione è quella di permettere di distinguere la causa che ha prodotto il reset.

PA0 al POR viene azzerato, rendendo operativa la pagina 0 di memoria).
GPWUF al POR è a zero, dato che la funzione di cambio livello sui pin non è attivata.

Per ora non ci interessa saper altro dello **STATUS** perchè vedremo nel corso dei successivi esercizi di approfondire l' argomento, dove necessario.

E, in effetti, non è neppure quale posizione che occupa il bit **C**, dato che uno degli elementi fondamentali della programmazione, come abbiamo ripetutamente detto, è la sostituzione di assoluti con label.
Anche in questo caso, il file ***nomeprocessore.inc*** che contiene le definizioni del processore usato e che inseriamo nel sorgente con un `#include`, contiene anche gli elementi simbolici per la gestione dello STATUS e dei suoi bit.

Quindi, **per azzerare il bit Carry, non useremo mai una indicazione di indirizzi assoluti** come questa:

```
bcf  3, 0          ; azzera il Carry
```

Questa scrittura è funzionante, ma completamente contraria alle regole di una buona programmazione Assembly, percè:

- in primo luogo, sarà utilizzabile solo ed esclusivamente con un processore che ha lo STATUS all' indirizzo di memoria 3 e il Carry al bit 0, cosa che può valere per un chip, ma non per altri e quindi risulta non portabile.
- inoltre non chiarisce minimamente cosa l' istruzione faccia, a meno di un commento.
  E', quindi, un modo di scrivere linee di Assembly del tutto condannabile e che va evitato assolutamente.

Una corretta scrittura sarà:

```
bcf  STATUS, C
```

che è di per se chiaro e non necessita commento, oltre ad adattarsi a qualsiasi processore, dato che con questa scrittura lasciamo agli automatismi del compilatore il compito di assegnare i giusti valori numerici alle label **STATUS** e **C** , traendoli dal file ***nomeprocessore.inc*** .
E, se ci secca ripetere questa linea, possiamo sostituirla usando lo pseudo opcode **clrc** offerto da MPASM, oppure, dove non supportato, realizzando una Macro:

```
; Macro per clear Carry
clrc    MACRO
            bcf     STATUS,C
        ENDM
```

Applichiamo la rotazione dei bit

Vista la funzione dell' istruzione di rotazione e del bit Carry, le applichiamo al nostro caso, dove utilizziamo la **RLF** con lo scopo di spostare il bit che mantiene acceso il LED lungo il registro **GPIO** degli I/O, che abbiamo provveduto a configurare come uscite.
Abbiamo però il problema della non continuità dei GP, dato che il **GP3** non può essere usato come uscita, ma solo come ingresso (e, inoltre, in questo caso ha funzione di **MCLR**). Resta pertanto escluso dalla rotazione, che, a quel punto, deve passare da **GP2** a **GP4**.
Nel caso dei **PIC12F5xx** il **GPIO** ha questo aspetto:

GPIO	Bit	7	6	5	4	3	2	1	0
	label	-	-	GP5	GP4	GP3	GP2	GP1	GP0
	pin	-	-	2	3	4	5	6	7

Essendoci solamente 6 pin utilizzati per l' I/0, i bit 6 e 7 del registro non hanno alcuna funzione: leggendoli, si leggerà uno 0 e scrivendoli non si avrà alcun effetto.

Se portiamo a 1 il bit 0 (**GP0**), ci troviamo in questa situazione:

GPIO	Bit	7	6	5	4	3	2	1	0
	label	-	-	GP5	GP4	GP3	GP2	GP1	GP0
	valore	-	-	0	0	0	0	0	1

Il LED collegato a **GP0** sarà acceso, gli altri saranno spenti. Facciamo ora ruotare (shiftare) il bit 0 verso sinistra, iniettando al suo posto un ***Carry=0***:

GPIO	Bit	7	6	5	4	3	2	1	0
	label	-	-	GP5	GP4	GP3	GP2	GP1	GP0
	valore	-	-	0	0	0	0	1	0

Ora il LED acceso sarà quello collegato al bit 1 (**GP1**). Ripetendo l' operazione con una cadenza tale da permettere la visione dei LED accesi e spenti, otterremo l' effetto voluto.
Arrivati al bit 3, come detto, dovremo saltarlo, passando al bit 4. Con il bit 5, abbiamo esaurito i

pin di I/O e possiamo chiudere il ciclo.
Rendiamo questo in istruzioni:

```
mainloop:
        bsf     GPIO,GP0        ; LED1 acceso
        movf    GPIO,w          ; copia stato I/O in shadow
        movwf   sGPIO
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; LED2
        clrc
        rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; LED3
        clrc
        rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; LED4 su GP4 - salta GP3
        clrc
        rlf     sGPIO, f        ; ruota shadow per saltare GP3
         rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; LED5
        clrc
        rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; fine sequenza
        clrf    sGPIO           ; tutti i LED spenti
        movf    sGPIO, w
        movlw   .100            ; wait per 1s
        call    Delay10msW
        goto    $               ; stop
```

Comprimiamo…

Possiamo osservare che la sequenza di istruzioni di rotazione si ripete identica per ogni LED.
Potrebbe essere il caso di trasformarla in una subroutine:

```
;                       SUBROUTINES
; ruota shadow e I/O
Rotate  clrc
```

```
        rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w        ; trasferisci all' I/O
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
        retlw   0
```

Per cui il main diventa:

```
mainloop:
        bsf     GPIO,GP0        ; LED0 acceso
        movf    GPIO,w          ; copia stato I/O in shadow
        movwf   sGPIO
        movlw   .50             ; attesa 1/2 s
        call    Delay10msW
; LED1
        call    Rotate          ; ruota shadow e I/O
; LED2
        call    Rotate          ; ruota shadow e I/O
; LED3 su GP4 - salta GP3
         clrc
         rlf     sGPIO, f        ; ruota shadow per saltare GP3
        call    Rotate          ; ruota shadow e I/O

; LED4
        call    Rotate          ; ruota shadow e I/O
; fine sequenza
        clrf    sGPIO           ; tutti i LED spenti
        movf    sGPIO, w
        movlw   .100            ; wait per 1s
        call    Delay10msW
        goto    $               ; stop
```

Decisamente più stringato.

La soluzione ha i seguenti vantaggi:

- si riduce l' impiego di memoria programma. Questo è indispensabile per programmi consistenti, date le piccole dimensioni della Flash del chip
- si rende più leggibile il sorgente

Per contro, va ricordato che i Baseline:

- hanno seri limiti sia come posizionamento delle subroutines (entro le prime 256 di ogni pagina)
- hanno uno stack a due soli livelli che mal si prestano all'uso intensivo di subroutines

Va notato, inoltre, che il tempo di esecuzione si appesantisce della chiamata e del rientro dalla subroutine, ma questo solitamente non riveste grande importanza, a meno di avere a che fare con una applicazione molto critica per quello che riguarda i tempi di esecuzione.

Va tenuta presente una cosa importante:

STOP Nella subroutine **Rotate** viene richiamata una ulteriore subroutine, quella per il ritardo **Delay10msW**. Per la citata limitazione dello stack a due soli livelli, non è possibile che la **DelayW10ms** richiami a sua volta un' altra subroutine, pena l' overflow dello stack e il fallimento dell' esecuzione del programma.

Per inciso, possiamo osservare che la situazione dello stack durante l' esecuzione della sarà quella di aver caricato prima l' indirizzo di rientro all' istruzione successiva a **call Rotate** e subito sopra l' indirizzo di rientro dalla **call DelayW10ms**. Con questo i due livelli dello stack minimale dei Baseline sono impegnati completamente.
Non è possibile una ulteriore chiamata all'interno ("annidata") della **call Rotate**.

… e qualche trucco…

Però possiamo limitare l' uso dello stack utilizzando un trucco per effettuare un rientro dalla catena di subroutines. Se osserviamo il sorgente, notiamo che la sequenza:

```
        call    Delay10msW
        retlw   0
```

Questo comporta che:

- lo stack già contiene l' indirizzo di rientro dalla chiamata della routine **Rotate**
- viene chiamata la subroutine **Delay10msW**
- nello stack viene caricato anche l' indirizzo della linea successiva. Entrambi i livelli sono occupati
- al rientro dalla subroutine, lo stack viene scaricato di una posizione e ora contiene l' indirizzo di rientro per la **Rotate**
- la linea **retlw** è un rientro da subroutine che preleva l' indirizzo dallo stack

Si può utilizzare, allora, una forma che in inglese è definita ***dirty return*** (ritorno "sporco"), ovvero chiamare la **Delay10msW** non con **call**, ma con **goto** !
goto trasferisce il Progam Counter all' indirizzo indicato, ma non tocca lo stack, che contiene le coordinate di ritorno dell' istruzione seguente alla **call Rotate** . La sequenza è questa:

- lo stack contiene l' indirizzo di rientro dalla chiamata della routine **Rotate**
- viene ordinato un salto all' indirizzo della label **Delay10msW**
- al termine dell' esecuzione della routine di tempo il programma incontra un **retlw** la cui funzione è quella di recuperare dallo stack l' indirizzo di rientro
- ma lo stack contiene solamente l' indirizzo di rientro dalla chiamata della routine **Rotate**

Quando l' esecuzione raggiunge il **retlw 0** della **Delay10msW**, il Program Counter preleva dallo stack un indirizzo per il ritorno e questo è quello relativo alla chiamata della **Rotate** .
Si risparmia una istruzione in coda alla subroutine, un livello di stack e un microsecondo nell' esecuzione.

```
;                          SUBROUTINES
; ruota shadow e I/O
Rotate  clrc
        rlf     sGPIO, f        ; ruota shadow
        movf    sGPIO, w        ; trasferisci all' I/O
        movwf   GPIO            ; LED acceso
        movlw   .50             ; attesa 1/2 s
        goto    Delay10msW
```

Questa tecnica fa parte dei tanti trucchi che si possono mettere in atto per migliorare l'efficienza di un programma. Va però detto che essa, come le altre che vedremo, va impiegata solamente quando è ben chiaro quale sia il suo scopo e le sue implicazioni.

… oltre ad una ottimizzazione

Una seconda variazione al sorgente la possiamo fare considerando che il tempo di lampeggio potrebbe essere inadatto all' applicazione; modificarlo vorrebbe dire sostituire il valore passato da W alla sub di attesa. Molto meglio dichiarare a priori questo valore sotto forma simbolica :

```
;                          COSTANTI
LEDtime  equ  .50       ; tempo accensione LED 50 x 10ms = 500ms
```

E, di conseguenza, sostituendo il valore con il simbolo:

```
        movlw   LEDtime             ; tempo accensione LED
        goto    Delay10msW
```

Questo consente di modificare facilmente il tempo agendo in un solo punto del sorgente.

Attenzione al percorso

Un' ultima nota riguarda il posizionamento della subroutine che viene inclusa nel sorgente:

```
; subroutine di ritardo 10ms x W
 #include  C:\PIC\Library\Baseline\Delay10msW.asm
```

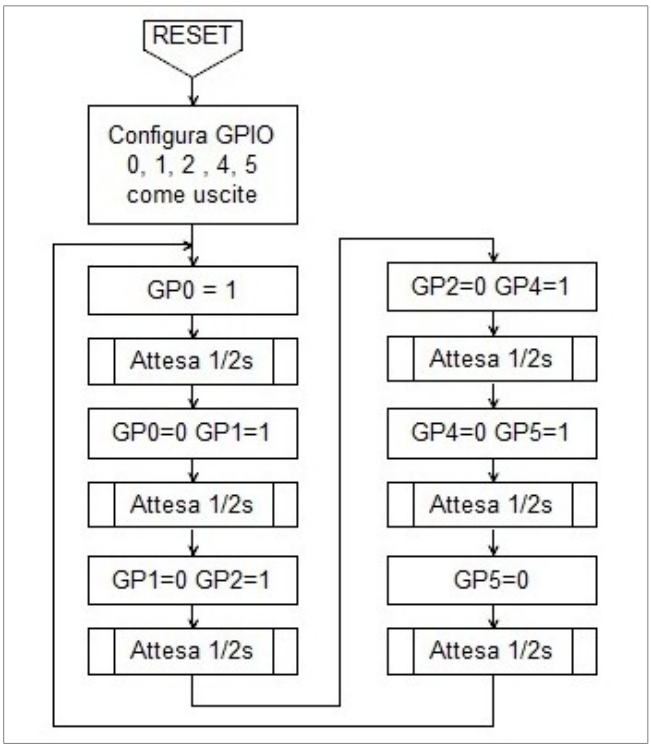
Questo vuol dire che la sub deve trovarsi nella cartella indica, ovvero **`C:\PIC\Library\Baseline\`**.
Se così non fosse, dovete modificare il path (= percorso, pista, traccia) completo in modo che punti alla posizione in cui si trova il file, altrimenti il compilatore indicherà un errore relativo all' impossibilità dell' inclusione, oltre ad un elevato numero di altri errori ogni volta che incontra un elemento definito nel file mancante.

Una variazione semplice.

Così come scritto, il programma esegue lo scorrimento dei LED e quindi si arresta. Per riavviare la sequenza dobbiamo agire sul RESET, che riporta il Program Counter all' inizio. Se, però, sostituiamo l' istruzione **goto $** con un **goto mainloop**, determiniamo un loop continuo

```
; fine conteggio
countend    goto    $           ; fine conteggio e blocco
```

creiamo un loop che ripete indefinitamente la sequenza del LED in movimento.



La stesura del flow chart permette immediatamente di afferrare la differenza: non chiudiamo il programma con un loop su se stesso, ma portiamo il loop all' inizio della sequenza, in modo che essa si ripeta indefinitamente.

Viene introdotta una pausa prima del riavvio della sequenza per migliorare l' effetto.

Ovviamente, per una diversa velocità di esecuzione , basterà variare la routine di tempo.

Anche qui potremo verificare la funzione del pin di reset che riporta l' esecuzione all' inizio del programma.

Possiamo anche inserire, al termine di una sequenza, una attesa maggiore, ad esempio 1 secondo. Così pure potremo utilizzare tempi diversi per ogni LED.

Più LED - PIC16F526

Nella famiglia Baseline troviamo non solo processori a 6 e 8 pin, ma anche con un numero maggiore di I/O, con 14, 18, 20 e anche 40 pin.

Se non abbiamo considerato i chip a 6 pin in questa esercitazione, dato che l' esiguo numero di I/O utili come uscite digitali (solo 3) avrebbe reso la cosa poco interessante, possiamo vedere invece un chip a 14 pin, il **PIC16F526**, di cui troviamo qui una descrizione abbastanza dettagliata.

Questo chip integra due port da 6 bit ciascuno (PORTB e PORTC).

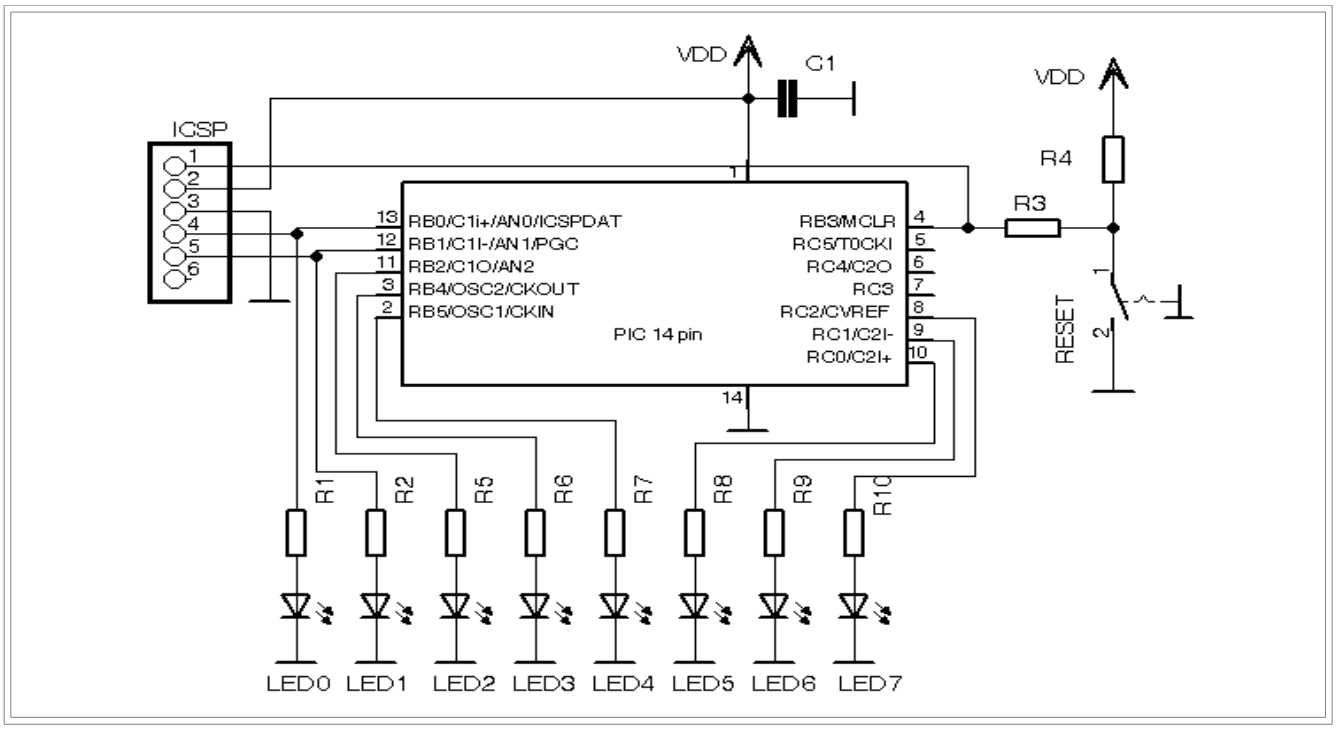
PORTB	Bit	7	6	5	4	3	2	1	0
	label	-	-	RB5	RB4	RB3	RB2	RB1	RB0
	pin	-	-						
PORTC	Bit	7	6	5	4	3	2	1	0
	label	-	-	RC5	RC4	RC3	RC2	RC1	RC0
	pin	-	-						

Essendoci solamente 6 pin validi, i bit 6 e 7 dei registri non hanno alcuna funzione: leggendoli, si otterrà 0 e scrivendoli non ci sarà alcun effetto.

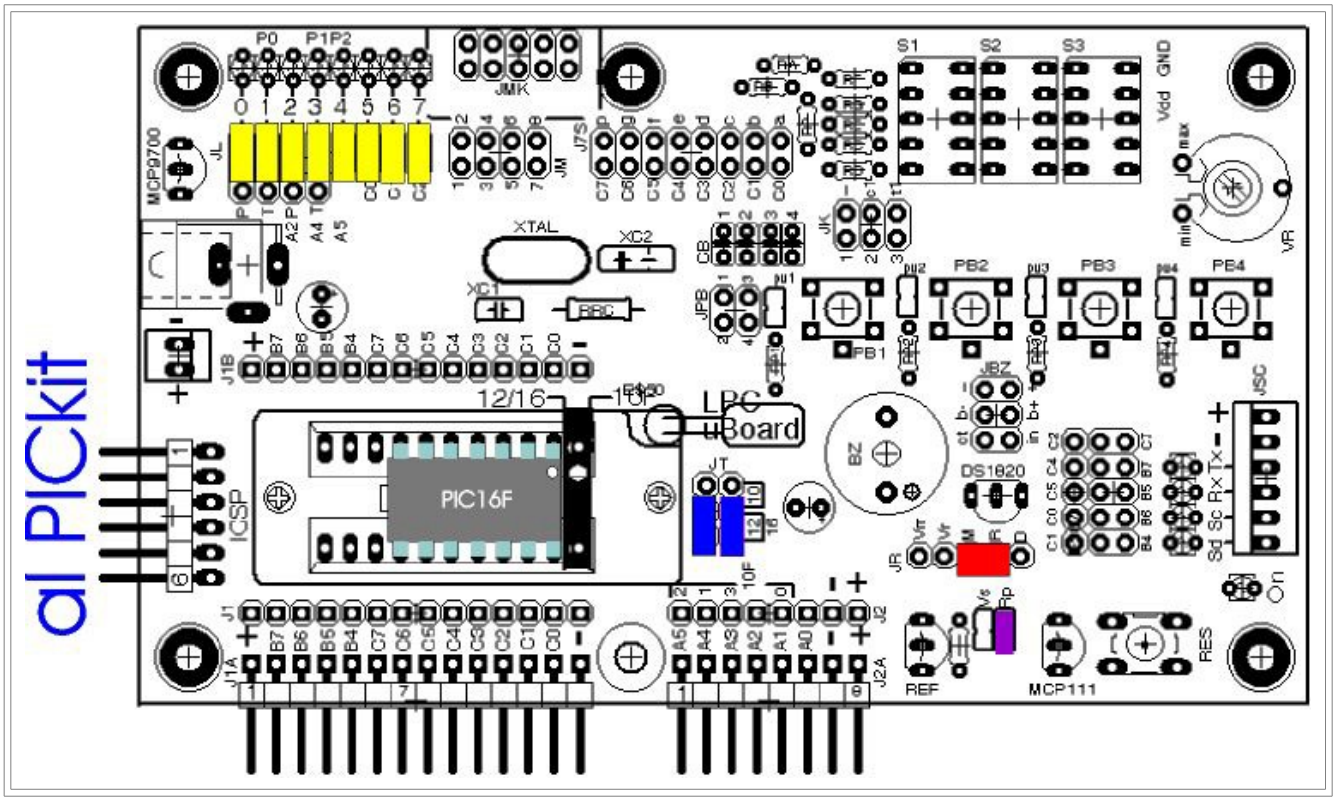
Questo ci permette di comandare facilmente 11 carichi in uscita e quindi gli 8 LED che la mette a disposizione.
Abbiamo detto 11 e non 12, dato che **RB3** ha la solita funzione di **MCLR** e in modo digitale può essere usato solo come ingresso.

Da notare che Microchip, per i processori con più di 8 pin identifica i blocchi di I/O non come **GPIO**, ma come **PORTx.** Quindi, ad esempio, non si farà riferimento a **GP0**, ma a **Px0** o **Rx0.**

La struttura di questo chip è del tutto analoga a quanto visto per i 12F508/9/10/19 già visti; le differenze consistono essenzialmente nel fatto che è disponibile una maggiore quantità di risorse e sarà necessaria, di conseguenza, una maggior quantità di registri interni. Avremo quindi, ad esempio, registri **TRISB** e **TRISC** e registri **PORTB** e **PORTC** per i due gruppi di I/O.
Ci sono altre differenze, come la quantità di memoria disponibile, RAM, moduli funzione integrati e relativi SFR, che, per ora, non consideriamo nel funzionamento specifico, perchè, dove necessario, richiedono una propria gestione, ma solo dal punto di vista dell' impiego degli I/O digitali.
Lo schema elettrico dell' applicazione è il seguente:



La scheda è facilmente configurabile con i soliti jumper:



Qui possiamo collegare tutti gli 8 LED, dipendenti dai jumper "gialli" JL.

Anche qui, quanto collegato sulle line **ICSP** dati e clock (**`RB0`** e **`RB1`**) vanno isolate con jumper durante la programmazione se costituiscono un carico eccessivo per il programmatore. Nel caso della scheda che utilizziamo, questo problema non si pone, dato il consumo minimo del LED.

Va sempre osservata la posizione dei jumper "blu", per i chip 16F e l' inserzione del chip sullo zoccolo, che salta la prima posizione, riservata ai 10F.

Il sorgente per il PIC16F526

Consultando il foglio dati del componente osserviamo che esso, oltre i due port, dispone di alcuni **moduli integrati analogici:**

- **doppio comparatore**
- **ADC**

Per i chip in cui sono integrati moduli funzione di tipo analogico, occorre una particolare cautela da parte del programmatore:

1. Questi moduli hanno ingressi e uscite condivise su alcuni pin, assieme alle funzioni di I/O digitale.
2. Non è possibile abilitare contemporaneamente più funzioni sullo stesso pin.
3. In base alla già citata filosofia di Microchip, al default sono attive le configurazioni a minior consumo, ovvero, se disponibili, quelle analogiche.

Ne consegue che si richiede di disabilitarle se si vuole accedere all' uso dei pin come semplici I/O digitali.

Si potrà ottenere questo agendo sui registri **ADCON0**, **CM1CON0** e **CM2CON0**, che controllano rispettivamente il funzionamento del modulo ADC e dei comparatori.

```
; disabilita gi ingressi analogici
      clrf    ADCON0

; disabilita comparatori per liberare la funzione digitale
      bcf     CM1CON0, C1ON
      bcf     CM2CON0, C2ON
```

Come solito per i Baseline, la funzione **T0CKI**, se non utilizzata, va disabilitata. Qui non sarebbe necessario, dato che non utilizziamo il pin **RC5**, ma è utile per tenere a mente questa particolarità.

```
; disabilita T0CKI da RB5
; OPTION       11111111
       movlw b'11011111'
       OPTION
```

Tra le risorse, il chip dispone di ***Wake Up per Pin Change*** e dei relativi pull-up, che comunque sono disabilitati portando a 1 i bit 6 e 7 dell' Option_Reg.
Esistono poi altri moduli, come ***CVref***, la tensione di riferimento per i comparatori, ma è disabilitato per default e non ha alcuna influenza sugli I/O.
Il chip ha SFR e RAM su **4 banchi**, ma, dato il minimo consumo richiesto dal programma, non ne teniamo conto, visto che non si superano le disponibilità di quanto accessibile dal banco 0. Il chip

dispone anche di **EEPROM**, ma anche di questa non facciamo uso e quindi non occorre tenerne conto.

Occorre invece tenere conto di un particolare che ad una indagine superficiale sul foglio dati, potrebbe sfuggire; si tratta del fatto che:

- **l' oscillatore interno ha la possibilità di essere settato su due frequenze**, ovvero 4 o 8MHz
- e che il pin **RB4** può essere utilizzato come uscita del clock F0sc/4.

Queste opzioni devono essere disposte nel config iniziale:

```
; Oscillatore interno, 4MHz, no WDT, no CP, RB3=MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF
& _MCLRE_ON
```

Osserviamo che la selezione dell' oscillatore interno, senza uscita su RB4, avviene con il comando **_IntRC_OSC_RB4** , mentre quella dei 4MHz con **_IOSCFS_4MHz**.
Se volessimo un clock a 8MHz, dovremmo impostare **_IOSCFS_8MHz** . Queste opzioni saranno trattate a fondo in una esercitazione successiva, dedicata ai sistemi di clock dei PIC.

Come negli esempi precedenti, curiamo di escludere in configurazione il Watchdog (**_WDTE_OFF**), che, altrimenti, è attivo per default ed andrebbe gestito opportunamente.

Curiamo anche di disabilitare le protezioni alla rilettura della memoria del chip, per evitare problemi nell' uso sperimentale; questo non è indispensabile, ma vivamente consigliato. Le protezioni andranno attivate solo quando, nella versione finale per la produzione, non si desidera che il firmware possa venire copiato.
Inoltre, dato che utilizziamo il pin **RB3** come **MCRL**, attiviamo la funzione relativa.

Come sempre la riga di configurazione, con questa modalità di scrittura, assume una lunghezza spropositata, ma è utile che tutte le opzioni siano configurate. Per default è possibile che alcune opzioni che ci interessano siano già nel giusto valore (come ad esempio le protezioni), ma è sempre opportuno effettuare **una configurazione completa di tutti i parametri disponibili** del chip, in modo da avere ben chiara la situazione base dell' hardware.
Tralasciando questo è molto facile incorrere in errori che impediranno il funzionamento corretto del programma e che sono poi molto difficili da individuare.

Ricordiamo che, come sempre, queste informazioni relative alla configurazione sono reperibili consultando il file ***16F526.inc***.

Altre **pagine di dettaglio sul 16F526** le trovate qui.

Subroutines per il 16F526

Volendo usare 8 LED dobbiamo impiegare i bit di **PORTB** e parte dei bit **PORTC**, dato che ogni port è composto da soli 6 bit.
La sequenza dei LED, in base ai collegamenti hardware eseguiti, è la seguente, saltando **RB3** :

```
1. RB0      LPCuB LED0
2. RB1      LPCuB LED1
3. RB2      LPCuB LED2
4. RB4      LPCuB LED3
5. RB5      LPCuB LED4
6. RC0      LPCuB LED5
7. RC1      LPCuB LED6
8. RC2      LPCuB LED7
```

Leggendo il sorgente possiamo osservare che sono state realizzate due subroutine di rotazione, una che interessa **PORTB** e una che interessa **PORTC**. Tra le tante possibilità, potrebbero avere questa forma:

```
; Subroutine di rotazione e attesa
RotateB:
    clrc                    ; azzera Carry
    rlf     sPORTB, f       ; ruota shadow
    movf    sPORTB, w
    movwf   PORTB           ; LED acceso
    movlw   .50             ; 50*10ms = 500ms
    call    Delay10msW      ; tempo
    retlw   0

RotateC:
    clrc                    ; azzera Carry
    rlf     sPORTC, f       ; ruota shadow
    movf    sPORTC, w
    movwf   PORTC           ; LED acceso
    movlw   .50             ; 50*10ms = 500ms
    call    Delay10msW      ; tempo
    retlw   0
```

Le due sequenze sono del tutto analoghe; la differenza sta nel PORT su cui vanno ad agire.
La sequenza delle istruzioni è abbastanza semplice:

- viene azzerato il carry per iniettare uno 0 nella rotazione del registro
- viene ruotata la shadow del port, con il risultato della rotazione immagazzinato nello stesso file
- la shadow viene copiata in **WREG** e quindi sul port
- viene aggiunto il tempo di ritardo voluto

Ancora trucchi

Anche qui possiamo introdurre il ritorno “sporco” (*dirty return*), salvando spazio e stack.
Quindi il sorgente diventa:

```
; Subroutine di rotazione e attesa
RotateB:
    clrc                ; azzera Carry
    rlf   sPORTB, f     ; ruota shadow
    movf  sPORTB, w
    movwf PORTB         ; LED acceso
    movlw LEDtime       ; 50*10ms = 500ms
    goto  DelayW10ms    ; dirty return
RotateC:
    clrc                ; azzera Carry
    rlf   sPORTC, f     ; ruota shadow
    movf  sPORTC, w
    movwf PORTC         ; LED acceso
    movlw LEDtime       ; 50*10ms = 500ms
    goto  DelayW10ms    ; dirty return
```

Altri trucchi !

L' analogia tra le due subroutines fa si che esistano punti in comune che permettono una riduzione del numero di righe scritte nel sorgente:

```
; Subroutine di rotazione e attesa
RotateB:
    clrc                ; azzera Carry
    rlf   sPORTB, f     ; ruota shadow
    movf  sPORTB, w
    movwf PORTB         ; LED acceso
    goto  rcomm         ; uscita comune

RotateC:
    clrc                ; azzera Carry
    rlf   sPORTC, f     ; ruota shadow
    movf  sPORTC, w
    movwf PORTC         ; LED acceso
rcomm
    movlw LEDtime       ; 50*10ms = 500ms
    goto  DelayW10ms    ; dirty return
```

Attraverso un salto facciamo terminare la prima subroutine nella "coda" della seconda.
Questa tecnica permette di risparmiare spazio nella memoria programma in quanto al posto delle due linee

```
    movlw LEDtime
    goto  DelayW10ms
```

viene scritta una linea sola

```
goto  rcomm
```

con un risparmio di due bytes di memoria programma. Questo modo di operare non è obbligatorio, ma può essere indispensabile quando si ha a che fare con programmi di una certa grandezza e disponibilità di memoria programma limitata, come nei Baseline.
Attenzione perchè questo ha un risvolto negativo nell' esecuzione, in quanto vengono aggiunti i due cicli necessari all' esecuzione del **goto**. Quindi occorre, anche in questo caso, valutare vantaggi e svantaggi del "trucco". Nell' esempio in corso non esistono problemi di spazio in memoria programma e neppure problemi di criticità nell' esecuzione; quindi l' applicazione della soluzione descritta è del tutto facoltativa.

```
mainloop:
     bsf   PORTB,RB0       ; LED 0 acceso
     movf  PORTB,sPORTB    ; copia stato I/O in shadow

; attesa 1/2 s e rotazione
     movlw LEDtime
     call  Delay10msW

; inizio rotazione
     call  RotateB          ; LED0 acceso
     call  RotateB          ; LED1 acceso

; salta PB3
     bcf   STATUS,C
     rlf   sPORTB, f        ; ruota shadow

; continua rotazione
     call  RotateB          ; LED2 acceso
     call  RotateB          ; LED3 acceso
     call  RotateB          ; LED4 acceso
```

e il passaggio da **PORTB** a **PORTC**:

```
; spegni LED di PORTB
     bcf   PORTB

; accende LED su PORTC
     bsf   PORTC,RC0        ; LED5 acceso
     movf  PORTC,sPORTC     ; copia stato I/O in shadow

; attesa 1/2 s e rotazione
     movlw LEDtime
     call  DelayW10ms

; prima rotazione
     call  RotateC          ; LED6 acceso
     call  RotateC          ; LED7 acceso

; spegni LED di PORTC
     bcf  PORTC
```

```
    goto $                  ; loop bloccato
```

il **goto** finale chiude il loop indefinito di arresto del programma. La pressione del tasto Reset riavvia il ciclo.

Dove abbiamo precedentemente dichiarato:

```
;#################################################################
;                          COSTANTI
;
LEDtime  EQU  .50            ; tempo accensione LED 50 x 10ms = 500ms
ENDtime  equ  (LEDtime * 2) ; tempo di fine sequenza - 1s
```

Notiamo che parametro per il tempo di 1s (100 x 10ms) è fatto calcolare dall'Assembler con l'espressione **(LEDtime * 2)**.

Sull' argomento potete anche consultare le **pagine qui linkate**.

Finale alternativo!

Come indicato prima, se vogliamo un loop continuo, basta puntare il salto finale non su se stesso, ma sull' inizio della sequenza di rotazioni, in modo da ripetere indefinitamente.

```
; spegni LED di PORTC
    bcf     PORTC

; attesa 1s e nuovo ciclo
    movlw   ENDtime
    call    Delay10msW

    ;goto    $               ; stop
    goto    mainloop        ; finale alternativo: ripeti sequenza
```

Nei sorgenti forniti, basterà commetare (iniziarle con ;) o decommetare (eliminare il semicolon iniziale) le linee finali per passare da una versione all' altra. Ovviamente per ogni modifica occorrerà ricompilare. Inoltre abbiamo dichiarato due costanti con label, in modo da averne il controllo in un solo punto del sorgente:

```
;#################################################################
;                          COSTANTI
LEDtime  EQU  .50            ; tempo accensione LED 50 x 10ms = 500ms
ENDtime  equ  (LEDtime * 2) ; tempo di fine sequenza - 1s
```

Notiamo che parametro per il tempo di 1s (100 x 10ms) è fatto calcolare dall'Assembler con l'espressione **(LEDtime * 2)**.

Sull' argomento potete anche consultare le **pagine qui linkate**.

Altre variazioni

Dovrebbe essere chiaro il numero indeterminato di possibilità che si aprono con una semplice gestione degli I/O in uscita. Vediamo alcune semplici variazioni.

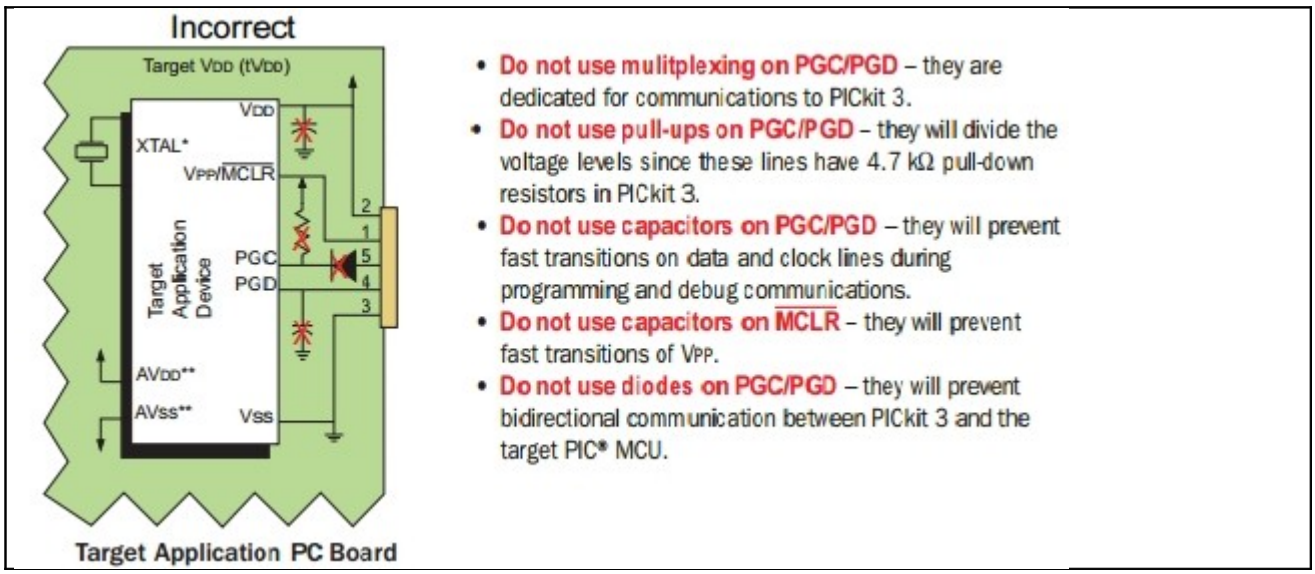
Potete sperimentare autonomamente sequenze continue o meno, inventando qualsiasi altra variazione sia sul numero di LED coinvolti, sia con la variazione dei tempi di intervento.

Queste strutture sono la base dei circuiti detti sequenziatori (*seuqencer*), che hanno notevole importanza in molti settori, dall' automazione alla musica, dalla domotica alle applicazioni civili.

Nota sulla programmazione

Dallo schema elettrico si nota che vengono utilizzati i pin **GP0**/**GP1** (**RB0**/**RB1**) per comandare i **LED 0/1** della scheda **LPCuB**.

Questi pin sono anche quelli utilizzati per la programmazione del chip, ovvero **PGC** e **PGD**, segnali che arrivano dal **Pickit**.
La programmazione del chip, come già detto, avviene con una comunicazione seriale sincrona che si stabilisce tra il chip e il tools di programmazione. Questa comunicazione avviene a frequenza abbastanza elevata e richiede che i segnali mantengano tempi di commutazione e di durata ben definiti. Applicare un carico a questi pin potrebbe voler dire danneggiare questa comunicazione, con l' impossibilità di programmare il componente.
La documentazione di Microchip è molto rigorosa a questo riguardo:



Quando sulla scheda **LPCuB** inseriamo i jumper di **LED0** e **LED1**, mettiamo questi LED in connessione con le linee di comunicazione. Ma questo non è un problema in quanto si tratta di carichi con capacità trascurabile e che quindi non deformano i fronti di commutazione e non introducono errori nelle temporizzazioni. Inoltre, la corrente assorbita da questi LED è minima,

inferiore a 2mA, cosa che non disturba la comunicazione, che è pilotata dai pin dei microcontroller (quello in programmazione e quello interno al tool di programmazione) che possono fornire 25mA.

Di conseguenza, **durante la programmazione, si vedranno lampeggiare rapidamente i due LED** e questo non creerà alcun problema all'operazione.

Se non state utilizzando la scheda di sviluppo e state usando LED a corrente più elevata, è possibile che ci siano problemi di comunicazione tra tool e chip. In tal caso inserite, come consigliato all' inizio, due jumper per isolare i LED durante la programmazione.

12F519 - 4A_519.ASM

```
;*******************************************************************
;-------------------------------------------------------------------
;
;    Titolo        : Corso Assembly & C - Esercitazione 4A_519
;                    Scorrimento LED con le istruzioni di rotazione
;                    Cadenza di 1/2 secondo. Ciclo continuo.
;
;    PIC           : 12F519
;    Supporto      : MPASM
;    Versione      : V.519-1.0
;    Data          : 01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-------------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;       12F519 @ 8 pin
;
;                  |__\/__|
;             Vdd -|1    8|- Vss
;             GP5 -|2    7|- GP0
;             GP4 -|3    6|- GP1
;        GP3/MCLR -|4    5|- GP2
;                  |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out  LED alla Vss
;    GP4/OSC2            3: Out  LED alla Vss
;    GP3/!MCLR/VPP       4: MCLR
;    GP2/T0CKI           5: Out  LED alla Vss
;    GP1/ICSPCLK         6: Out  LED alla Vss
;    GP0/ICSPDAT         7: Out  LED alla Vss
;    Vss                 8: --
;
;*******************************************************************
;===================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|LED5 |LED4 | MCLR|LED2 |LED1 |LED0 |
;
#define     LED0 GPIO,GP0    ; LED tra pin e Vss
#define     LED1 GPIO,GP1    ; LED tra pin e Vss
#define     LED2 GPIO,GP2    ; LED tra pin e Vss
;#define    GPIO,GP3         ; MCLR
#define     LED3 GPIO,GP4    ; LED tra pin e Vss
#define     LED4 GPIO,GP5    ; LED tra pin e Vss
;
;*******************************************************************
;
```

```
;    Note :
;
;#####################################################################

        LIST      p=12F519         ; Definizione del processore
        #include  <p12F519.inc>
        radix     dec

;#####################################################################
;=====================================================================
;                           CONFIGURAZIONE
;
; Oscillatore interno, no WDT, no CP, pin4=MCLR
;
 __config  _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON


;#####################################################################
;=====================================================================
;                         MEMORIA RAM
;
; general purpose RAM
        CBLOCK 0x07         ; inizio area RAM
      sGPIO                 ; shadow di GPIO
      d1,d2,d3              ; contatori per ritardo
        ENDC


;#####################################################################
;=====================================================================
;                          RESET ENTRY
; Reset Vector
        ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL


;#####################################################################
;=====================================================================
;                         MAIN PROGRAM
MAIN:
; inizializzazioni al reset
        clrf    GPIO            ; preset GPIO latch a 0

; disabilita T0CKI per avere GP2 come I/O digitale
         ;      b'11010111'
         ;        1-------     GPWU disabilitato
         ;        -1------     GPPU disabilitato
         ;        --0-----     clock interno
         ;        ---1----     falling
         ;        ----1---     prescaler al WDT
         ;        -----111     1:256
        movlw   b'11011111'
        OPTION
```

```
; tutti i GP come out
        movlw   0
        tris    GPIO

mainloop:
         bsf     GPIO,GP0        ; LED0 acceso
         movf    GPIO,w
         movwf   sGPIO           ; copia stato I/O in shadow

         movlw   .50             ; attesa 1/2 s
         call    Delay10msW

; LED1
         call    Rotate          ; ruota shadow e I/O

; LED2
         call    Rotate          ; ruota shadow e I/O

; LED3 su GP4 - salta GP3
         clrc
         rlf     sGPIO, f        ; ruota shadow per saltare GP3
         call    Rotate          ; ruota shadow e I/O

; LED4
         call    Rotate          ; ruota shadow e I/O

; fine sequenza
         clrf    sGPIO           ; tutti i LED spenti
         movf    sGPIO, w

         goto    $               ; stop
        ;goto    mainloop        ; finale alternativo: ripeti sequenza

;#####################################################################
;                        SUBROUTINES
; rotazione
Rotate   clrc
Rot1     rlf     sGPIO, f        ; ruota shadow
         movf    sGPIO, w
         movwf   GPIO            ; LED acceso
         movlw   .50             ; attesa 1/2 s
         goto    Delay10msW

; subroutine di ritardo 10ms x W
  #include  C:\PIC\Library\Baseline\Delay10msW.asm

;*********************************************************************
;=====================================================================
;                              THE END
         END
```

12F508/509 - 4A_5089.ASM

```
;*******************************************************************
;-------------------------------------------------------------------
;
;    Titolo        :  Corso Assembly & C - Esercitazione 4A_5089
;                     Scorrimento LED con le istruzioni di rotazione
;                     Cadenza di 0.5s
;
;    PIC           : 12F508/509
;    Supporto      : MPASM
;    Versione      : V.519-1.0
;    Data          : 01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-------------------------------------------------------------------
;
;  Impiego pin :
;  ---------------
;       12F508/9 @ 8 pin
;
;                  |__\/__|
;             Vdd -|1    8|- Vss
;             GP5 -|2    7|- GP0
;             GP4 -|3    6|- GP1
;        GP3/MCLR -|4    5|- GP2
;                  |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out  LED4 alla Vss
;    GP4/OSC2            3: Out  LED3 alla Vss
;    GP3/!MCLR/VPP       4: MCLR
;    GP2/T0CKI           5: Out  LED2 alla Vss
;    GP1/ICSPCLK         6: Out  LED1 alla Vss
;    GP0/ICSPDAT         7: Out  LED0 alla Vss
;    Vss                 8: --
;
;*******************************************************************
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|LED4 |LED3 | MCLR|LED2 |LED1 |LED0 |
;
;#define     LED0 GPIO,GP0    ; LED tra pin e Vss
;#define     LED1 GPIO,GP1    ; LED tra pin e Vss
;#define     LED2 GPIO,GP2    ; LED tra pin e Vss
;#define     GPIO,GP3         ; MCLR
;#define     LED3 GPIO,GP4    ; LED tra pin e Vss
;#define     LED4 GPIO,GP5    ; LED tra pin e Vss
;
;*******************************************************************
;###################################################################
```

```
; scelta del processore
 #ifdef  __12F509
        LIST      p=12F509          ; Definizione del processore
        #include <p12F509.inc>
 #endif
 #ifdef  __12F508
       LIST      p=12F508           ; Definizione del processore
        #include <p12F508.inc>
 #endif

        radix     dec

;#################################################################
;                        CONFIGURAZIONE
;
; Oscillatore interno, no WDT, no CP, MCLR
 __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_ON


;#################################################################
;                        MEMORIA RAM
;
; general purpose RAM
        CBLOCK  0x07
   sGPIO                     ; shadow di GPIO
   d1, d2, d3                ; contatori per ritardo
        ENDC

;#################################################################
;                        MEMORIA RAM
;
; general purpose RAM
        CBLOCK  0x07
   sGPIO                     ; shadow di GPIO
        ENDC

;#################################################################
;                         COSTANTI
;
LEDtime  EQU   .50   ; tempo accensione LED 50 x 10ms = 500ms

;#################################################################
;                        MAIN
;
; inizializzazioni al reset
        clrf    GPIO            ; preset GPIO latch a 0

; tutti i GP come out
        movlw   0
        tris    GPIO

mainloop:
        bsf     GPIO,GP0        ; LED0 acceso
        movf    GPIO,w          ; copia stato I/O in shadow
        movwf   sGPIO
```

```
        movlw   LEDtime         ; attesa 1/2 s
        call    Delay10msW

; LED1
        call    Rotate          ; ruota shadow e I/O

; LED2
        call    Rotate          ; ruota shadow e I/O

; LED3 su GP4 - salta GP3
        clrc
        rlf     sGPIO, f        ; ruota shadow per saltare GP3
        call    Rotate          ; ruota shadow e I/OO

; LED4
        call    Rotate          ; ruota shadow e I/O

; fine sequenza
        clrf    sGPIO           ; tutti i LED spenti
        movf    sGPIO, w

         goto    $               ; stop
        ;goto    mainloop        ; finale alternativo: ripeti sequenza

;#################################################################
;                         SUBROUTINES
; ruota shadow e I/O
Rotate   clrc
         rlf     sGPIO, f        ; ruota shadow
         movf    sGPIO, w
         movwf   GPIO            ; LED acceso
         movlw   LEDtime         ; attesa 1/2 s
         call    Delay10msW
         retlw   0

; subroutine di ritardo 10ms x W
 #include  C:\PIC\Library\Baseline\Delay10msW.asm

;*****************************************************************
;=================================================================
;                             THE END

     END
```

16F526/505 - 4A_526.asm

```
;*******************************************************************
;-------------------------------------------------------------------
;
;    Titolo        :  Corso Assembly & C - Esercitazione 4A_526
;                     Scorrimento LED con le istruzioni di rotazione
;                     Cadenza di 1/2 secondo.
;
;    PIC           : 16F526
;    Supporto      : MPASM
;    Versione      : V.519-1.0
;    Data          : 01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-------------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;       16F505 - 16F526 @ 14 pin
;
;                  |__\/__|
;             Vdd -|1   14|- Vss
;             RB5 -|2   13|- RB0
;             RB4 -|3   12|- RB1
;        RB3/MCLR -|4   11|- RB22
;             RC5 -|5   10|- RC0
;             RC4 -|6    9|- RC1
;             RC3 -|7    8|- RC2
;                  |______|
;
;    Vdd                       1: ++
;    RB5/OSC1/CLKIN            2: Out  LED4 alla Vss
;    RB4/OSC2/CLKOUT           3: Out  LED3 alla Vss
;    RB3/!MCLR/VPP             4: MCLR
;    RC5/T0CKI                 5:
;    RC4/[C2OUT]               6:
;    RC3                       7:
;    RC2/[Cvref]               8: Out  LED7 alla Vss
;    RC1/[C2IN-]               9: Out  LED6 alla Vss
;    RC0/[C2IN+]              10: Out  LED5 alla Vss
;    RB2/[C1OUT/AN2]          11: Out  LED2 alla Vss
;    RB1/[C1IN-/AN1/]ICSPC    12: Out  LED1 alla Vss
;    RB0/[C1IN+/AN0/]ICSPD    13: Out  LED0 alla Vss
;    Vss                      14: --
;
;    [ ] solo 16F526
;
;###################################################################

        LIST      p=16F526         ; Definizione del processore
        #include <p16F526.inc>
        radix     dec

;###################################################################
```

```
;                           CONFIGURAZIONE
;
; Oscillatore interno, 4MHz, no WDT, no CP, RB3=MCLR
 __config  _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_ON


;#################################################################
;                            MEMORIA RAM
; general purpose RAM
        CBLOCK 0x10          ; inizio area RAM
      sPORTB                 ; shadow di PORTB
      sPORTC                 ; shadow di PORTC
      d1,d2,d3               ; contatori per ritardo
        ENDC

 ;################################################################
 ;                           COSTANTI
 ;
LEDtime  EQU  .50            ; tempo accensione LED 50 x 10ms = 500ms
ENDtime  equ  (LEDtime * 2)  ; tempo di fine sequenza - 1s

;#################################################################
;                           RESET ENTRY
;
; Reset Vector
        ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL

;#################################################################
;                         MAIN PROGRAM
Main:
; inizializzazioni al reset
        clrf    PORTB            ; preset GPIO latch a 0
        clrf    PORTC

; disabilita gi ingressi analogici
        clrf    ADCON0

; disabilita comparatori per liberare la funzione digitale
        bcf     CM1CON0, C1ON
        bcf     CM2CON0, C2ON

; disabilita T0CKI da RB5
         ;      b'11010111'
         ;        1-------     GPWU disabilitato
         ;        -1------     GPPU disabilitato
         ;        --0-----     clock interno
         ;        ---1----     falling
         ;        ----1---     prescaler al WDT
         ;        -----111     1:256
        movlw   b'11011111'
        OPTION

; Tutti i port utili come out
```

```
        movlw   0
        tris    PORTB           ; al registro direzione
        tris    PORTC

mainloop:
        bsf     PORTB,RB0       ; LED0 acceso
        movf    PORTB,w
        movwf   sPORTB          ; copia stato I/O in shadow

; attesa 1/2 s e rotazione
        movlw   LEDtime
        call    Delay10msW

; LED1
        call    RotateB
; LED2
        call    RotateB

; LED3 su PB4 - salta PB3
        clrc
        rlf     sPORTB, f       ; ruota shadow
        call    RotateB
; LED4
        call    RotateB

; spegni LED di PORTB
        clrf    PORTB

; LED5 su RC0
        bsf     PORTC,RC0
        movf    PORTC,w
        movwf   sPORTC          ; copia stato I/O in shadow

; attesa 1/2 s e rotazione
        movlw   LEDtime
        call    Delay10msW

; LED6
        call    RotateC
; LED7
        call    RotateC

; spegni LED di PORTC
        clrf    PORTC

; attesa 1s e nuovo ciclo
        movlw   ENDtime
        call    Delay10msW

         goto    $               ; stop
        ;goto    mainloop        ; finale alternativo: ripeti sequenza

;#################################################################
;                        SUBROUTINES
;
; Subroutine di rotazione e attesa
RotateB:
```

```
        clrc                    ; carry = 0
        rlf     sPORTB, f       ; ruota shadow
        movf    sPORTB, w       ; copia in W
        movwf   PORTB           ; copia sul port
        goto    rcom            ; rientro comune

RotateC:
        clrc                    ; carry = 0
        rlf     sPORTC, f       ; ruota shadow
        movf    sPORTC, w       ; copia in W
        movwf   PORTC           ; copia sul port
rcom    movlw   LEDtime         ; attesa 1/2 s e rotazione
        goto    Delay10msW      ; dirty return

; subroutine di ritardo 10ms x W
 #include  C:\PIC\Library\Baseline\Delay10msW.asm

;*********************************************************************
;                            THE END
        END
```