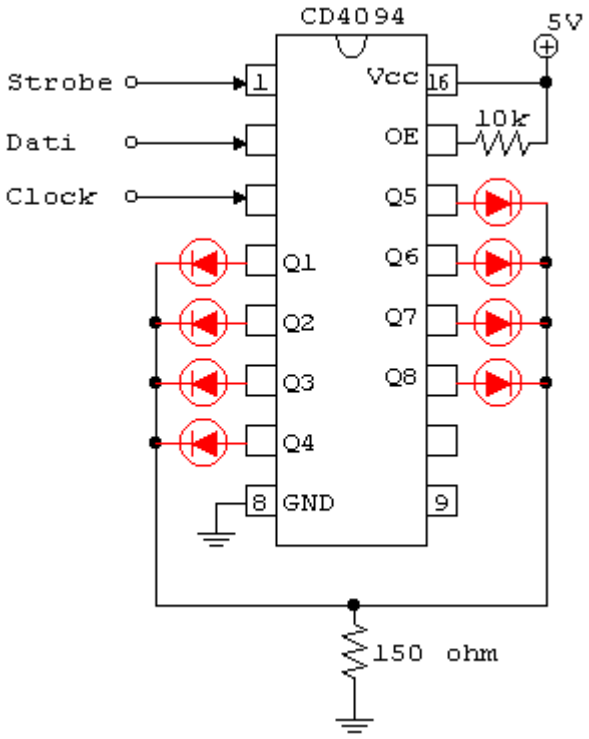
PIC
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA PER PRINCIPIANTI

By Claudio Fin

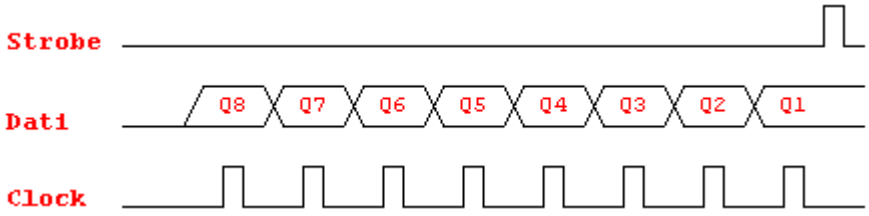
[Precedente] [Indice principale]

COMANDO DI UNO SHIFT REGISTER CD4094

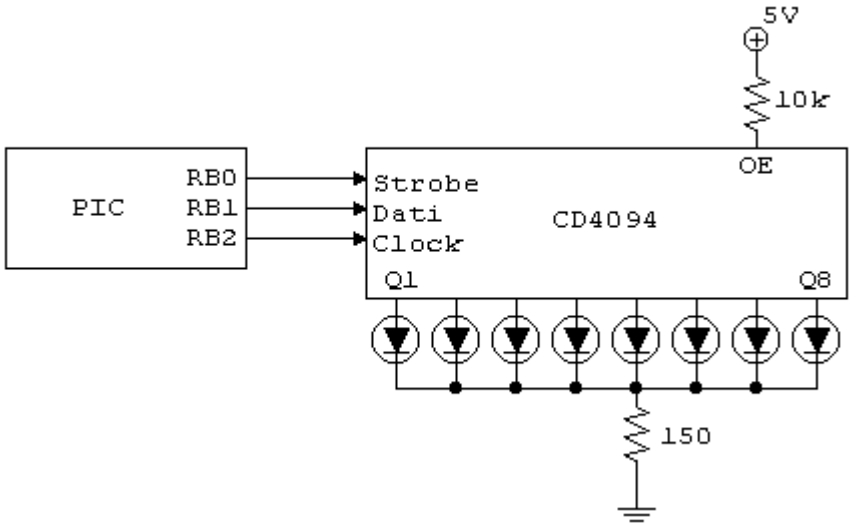
Fino ad ora abbiamo visto come emettere un byte in parallelo sugli 8 pin RB0..RB7, e come creare un'animazione con scritture ripetute. Lo scopo di questo capitolo è quello di mostrare come si possono usare i pin di I/O per generare sequenze arbitrarie di segnali per controllare circuiti esterni. Useremo uno shift register di tipo CD4094 che dispone di 8 uscite per ricreare la stessa «animazione supercar». Il PIC dovrà generare gli opportuni segnali per trasferire il dato (registro LUCE) serialmente un bit alla volta verso il 4094, ed infine convalidarlo con un impulso di conferma (strobe).



Questo integrato è uno shift register di tipo SIPO (serial input parallel output). Può anche essere usato per ampliare le uscite digitali nel caso in cui quelle del PIC siano insufficienti. Richiede solo 3 segnali di controllo: Dati e Clock per caricare serialmente un byte al suo interno, e Strobe per portarlo sulle sue 8 uscite (Q1..Q8). Come si può vedere dal diagramma temporale più sotto il primo bit che entra finisce (scorre) sull'uscita Q8, l'ultimo sull'uscita Q1. Durante il trasferimento le uscite rimangono stabili, il loro valore viene aggiornato in un colpo solo dando l'impulso di Strobe. Le uscite del 4094 non possono fornire tanta corrente come le uscite di un PIC, i LED risultano perciò meno luminosi. Volendo riprodurre il solito effetto supercar (in cui è acceso un solo LED alla volta) èsufficiente mettere una sola resistenza comune verso massa. Va tenuto conto che al momento dell'accensione i flip flop interni del 4094 contengono valori casuali, per cui i LED per un attimo possono essere accesi casualmente fino a quando non viene effettuata la prima scrittura nel registro.



Per questo esperimento si può usare il seguente collegamento tra un generico PIC (di cui sono indicati i soli pin di I/O usati) e il 4094 (le alimentazioni sono naturalmente sottointese).

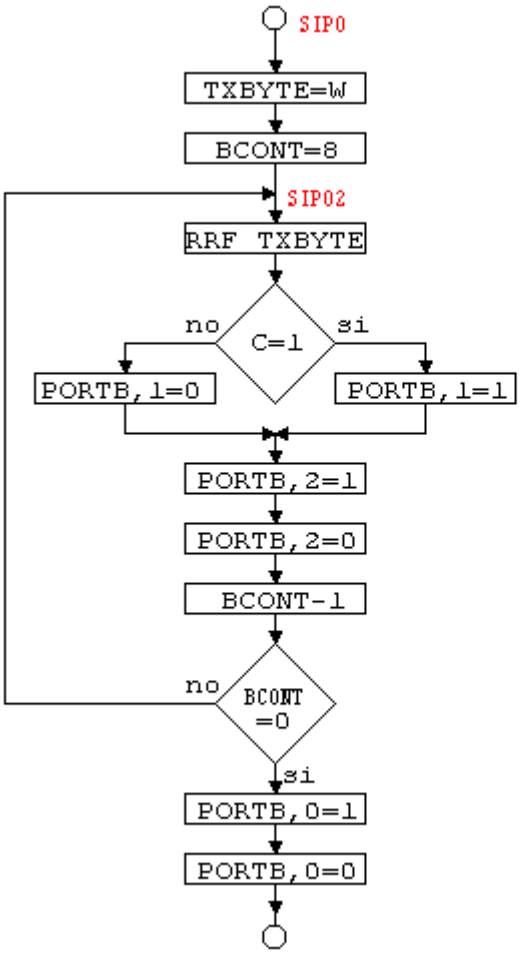


Si vuole usare lo stesso programma dell'effetto supercar, però scrivendo il valore del registro LUCE sul registro esterno invece che direttamente sui pin della porta B. Questi pin verranno usati invece per generare i segnali di controllo per il 4094. Ci interessa inoltre fare in modo che il bit0 del registro LUCE finisca sull'uscita Q8, e il bit7 sull'uscita Q1. Per fare questo si dovranno inviare in sequenza i bit del registro LUCE dal pin RB1 (ciascuno seguito da un impulso di clock emesso dal pin RB2) partendo dal bit0. Alla fine, dopo 8 impulsi di clock, si conferma il dato con un impulso di strobe emesso dal pin RB0.

Si potrebbe pensare di dover riprogettare da capo tutto il programma. Invece, ragionando in modo modulare, basta aggiungere una sola subroutine con funzione di «driver» e poche altre cose. In particolare occorre definire i nomi di altri due registri dati che servono alla subroutine di scrittura sul 4094, bisogna azzerare le uscite della porta B dopo averla configurata come uscita (per impostare il «livello di riposo» delle uscite, nel nostro caso vanno messe a 0), e al posto della scrittura diretta sulla porta si deve chiamare la subroutine di trasferimento seriale.

```
;-------------------------------------------------------
; Programma effetto supercar attraverso shift register
;-------------------------------------------------------
          PROCESSOR 16F628
          RADIX     DEC
          INCLUDE   "P16F628.INC"
          __CONFIG  11110100010000B
;-------------------------------------------------------
LUCE      EQU       32          ;Valore da scrivere sui LED
H_CONT    EQU       33          ;Parte alta contatore ritardo
L_CONT    EQU       34          ;Parte bassa contatore ritardo
DIREZ     EQU       35          ;Direzione 0=sinistra 1=destra
TX_BYTE   EQU       36          ;Valore da trasferire serialmente
B_CONT    EQU       37          ;Contatore dei bit da trasmettere
;-------------------------------------------------------
          ORG       0
          BSF       STATUS,RP0  ;Attiva banco 1
          CLRF      TRISB       ;Rende PORTB un'uscita
          BCF       STATUS,RP0  ;Ritorna al banco 0
          CLRF      PORTB       ;Azzera uscite PORTB
          MOVLW     00000001B
```

```
            MOVWF     LUCE          ;LUCE=00000001
            CLRF      DIREZ         ;DIREZ=0
MAINLOOP    MOVF      LUCE,W        ;W=LUCE
            CALL      SIPO          ;Richiama subroutine SIPO
            BCF       STATUS,C      ;Azzera flag C
            BTFSC     DIREZ,0       ;Se DIREZ=0 (sinistra) skip
            RRF       LUCE,F        ;Ruota LUCE verso destra
            BTFSS     DIREZ,0       ;Se DIREZ=1 (destra) skip
            RLF       LUCE,F        ;Ruota LUCE verso sinistra
            BTFSC     LUCE,7        ;Se bit 7 di LUCE ? 0 skip
            INCF      DIREZ,F       ;altrimenti direzione=destra
            BTFSC     LUCE,0        ;Se bit 0 di luce ? 0 skip
            CLRF      DIREZ         ;altrimenti direzione=sinistra
            CALL      DELAY         ;Richiama subroutine di ritardo
            GOTO      MAINLOOP      ;Nuovo ciclo del programma
;-----------------------------------------------------
DELAY       MOVLW     20            ;Carica 5320 nei 16 bit
            MOVWF     H_CONT        ;formati dai due byte
            MOVLW     200           ;H_CONT e L_CONT
            MOVWF     L_CONT
DELAY2      DECF      L_CONT,F      ;Decrementa parte bassa del contatore
            COMF      L_CONT,W      ;Inverte i bit
            BTFSC     STATUS,Z      ;Se tutti zero c'? stato rollover
            DECF      H_CONT,F      ;allora decrementa parte alta
            MOVF      L_CONT,W      ;Carica in W la parte bassa
            IORWF     H_CONT,W      ;Mettila in OR con la parte alta
            BTFSS     STATUS,Z      ;Se tutto zero skip (fine ciclo)
            GOTO      DELAY2        ;Altrimenti ritorna a DELAY2
            RETURN
;-----------------------------------------------------
SIPO        MOVWF     TX_BYTE       ;TX_BYTE=W
            MOVLW     8             ;Carica 8 nel contatore
            MOVWF     B_CONT        ;dei bit da trasmettere
SIPO2       RRF       TX_BYTE,F     ;Ruota TX_BYTE a destra nel flag C
            BTFSC     STATUS,C      ;Se flag C=0 skip
            BSF       PORTB,1       ;altrimenti dato out=1
            BTFSS     STATUS,C      ;Se flag C=1 skip
            BCF       PORTB,1       ;Altrimenti dato out=0
            BSF       PORTB,2       ;Clock=1
            BCF       PORTB,2       ;Clock=0
            DECFSZ    B_CONT,F      ;Decr.contat.bit skip se 0
            GOTO      SIPO2         ;Prossimo bit da trasmettere
            BSF       PORTB,0       ;Strobe=1
            BCF       PORTB,0       ;Strobe=0
            RETURN
;-----------------------------------------------------
            END
```



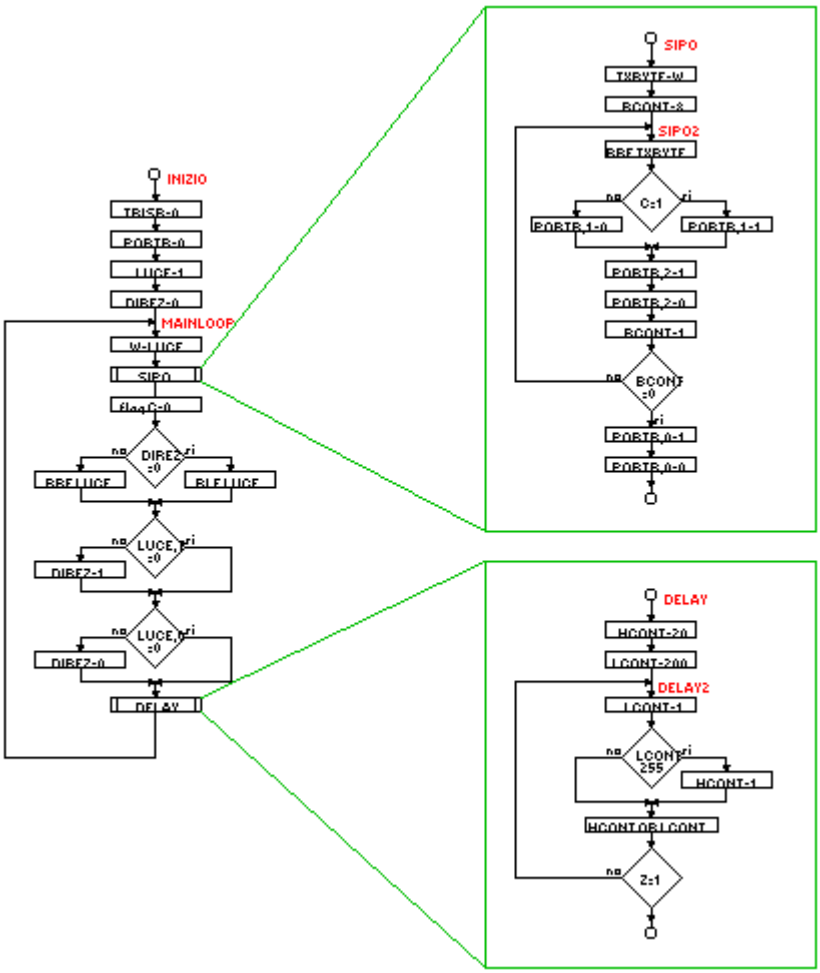

```
MOVWF TX_BYTE

MOVLW 8
MOVWF B_CONT

RRF   TX_BYTE,F

BTFSC STATUS,C
BSF   PORTB,1
BTFSS STATUS,C
BCF   PORTB,1

BSF   PORTB,2

BCF   PORTB,2


DECFSZ B_CONT,F
GOTO   SIPO2


BSF   PORTB,0

BCF   PORTB,0

RETURN
```

Si è accennato al fatto che il sottoprogramma SIPO può essere considerato l'equivalente di un piccolo driver software, infatti sarebbe possibile usare anche altri tipi di shift register senza alterare minimamente la sezione principale del programma, ma apportando le modifiche necessarie solo al driver. L'unica cosa che deve rimanere inalterata è «l'interfaccia», cioè il modo in cui il programma principale «comunica» il dato da trasmettere al sottoprogramma. Nel nostro caso l'interfaccia è semplicemente il registro accumulatore W, il main deve caricare in W il dato da trasmettere, il sottoprogramma «preleva» da W il dato.

Diventa così ancora più chiaro il concetto di «modulo» software, inteso come blocco funzionale autonomo che svolge al meglio un certo compito e «comunica» con il resto del programma attraverso un'interfaccia ben precisa, senza interferire con nessun altro registro o funzione. Un modulo concepito in questo modo può facilmente essere trasportato in un altro programma.

Guardando la figura seguente, che è il nostro programma completo, diventa anche più chiaro il concetto di «struttura». Si vede come è realizzato con le strutture fondamentali sequenziale, condizionale e iterativa, e si vede inoltre come ogni rettangolo può in realtà anche rappresentare intere sezioni di programma ad un livello di maggiore astrazione. Questo è il caso delle due chiamate a sottoprogramma CALL SIPO e CALL DELAY, che virtualmente racchiudono in due rettangolini i due flowcharts completi visibili sulla loro destra. A loro volta i rettangoli di un sottoprogramma potrebbero rappresentare altri blocchi di istruzioni o sottoprogrammi. Ciò che in ogni caso viene mantenuto è il fatto che ogni struttura (o flowchart) ha un solo punto di inizio e un solo punto di termine, questo facilita la

scrittura e la comprensione del programma, porta a scrivere programmi modulari in modo naturale, e prende il nome di «programmazione strutturata».



Se non ci fossero le istruzioni CALL e RETURN la cosa sarebbe molto meno «pulita», si dovrebbero mettere molti GOTO, soprattutto nel caso in cui il sottoprogramma venisse chiamato da più punti differenti del programma principale. In questo caso infatti si dovrebbe usare un ulteriore registro per «dire» al sottoprogramma chi è il chiamante, in modo da poter effettuare il corretto GOTO di ritorno... Grazie allo stack fortunatamente tutto ciò non serve.

In base a quanto detto finora risulta chiaro che l'uso del GOTO, particolarmente libero (e obbligatorio) in assembly, andrebbe limitato alla sola «creazione di strutture», e non usato a casaccio, altrimenti si ottengono i cosiddetti «programmi spaghetti», in cui è difficile seguire/comprendere/modificare il flusso logico dell'esecuzione (benchè il programma possa funzionare ugualmente).

#DEFINE

Sempre nell'ottica di rendere il tutto il più pulito e trasportabile possiamo usare la potente direttiva di assemblaggio «#define» per dare dei nomi simbolici ad istruzioni o parti di esse. Per esempio possiamo definire i pin utilizzati per comandare il 4094 nel seguente modo:

```
#DEFINE    PIN_TX     PORTB,1
#DEFINE    PIN_CLK    PORTB,2
```

```
#DEFINE    PIN_STRB   PORTB,0
```

E nel programma possiamo usare i nomi simbolici:

```
SIPO       MOVWF      TX_BYTE      ;TX_BYTE=W
           MOVLW      8            ;Carica 8 nel contatore
           MOVWF      B_CONT       ;dei bit da trasmettere
SIPO2      RRF        TX_BYTE,F    ;Ruota TX_BYTE a destra nel flag C
           BTFSC      STATUS,C     ;Se flag C=0 skip
           BSF        PIN_TX       ;altrimenti dato out=1
           BTFSS      STATUS,C     ;Se flag C=1 skip
           BCF        PIN_TX       ;Altrimenti dato out=0
           BSF        PIN_CLK      ;Clock=1
           BCF        PIN_CLK      ;Clock=0
           DECFSZ     B_CONT,F     ;Decr.contat.bit skip se 0
           GOTO       SIPO2        ;Prossimo bit da trasmettere
           BSF        PIN_STRB     ;Strobe=1
           BCF        PIN_STRB     ;Strobe=0
           RETURN
```

A questo punto il sottoprogramma è «indipendente dall'hardware», può usare qualsiasi pin di ingresso/uscita semplicemente modificando la sezione di definizione, e può essere facilmente usato su altri modelli di PIC, o usare anche pin di porte differenti. L'unica cosa a cui bisogna fare attenzione è di non dare nomi già usati nel file include specifico per ogni modello di PIC (in questo caso P16F628.INC). La sezione delle #define può essere messa subito dopo la definizione dei registri dati utilizzati dal programma.

ORG e RES

La direttiva di assemblaggio ORG l'abbiamo già vista messa prima della prima istruzione del programma, ed indica l'indirizzo a cui va caricato fisicamente il programma nella memoria flash (tipicamente 0). Ha però anche un secondo utilizzo nel caso venga usata per definire degli indirizzi di memoria dati tramite la direttiva RES. Finora abbiamo dato un nome ai registri di lavoro scrivendone esplicitamente l'indirizzo tramite la direttiva EQU. Nel caso in cui si voglia trasportare un programma su un PIC diverso occorre naturalmente cambiare tutti gli indirizzi dell'area dati e questo è un pòscomodo oltre che fonte di possibili errori. Tramite ORG e RES possiamo invece dichiarare solo l'indirizzo di inizio dell'area, e dire all'assemblatore quanti byte riservare per ogni variabile di lavoro, sarà lui a stabilire l'indirizzo di ogni registro partendo dal valore specificato con ORG:

```
;-----------------------------------------------------
           PROCESSOR 16F628
           RADIX     DEC
           INCLUDE   "P16F628.INC"
           __CONFIG  11110100010000B
;-----------------------------------------------------
           ORG       32          ;Indirizzo inizio RAM dati
LUCE       RES       1           ;Valore da scrivere sui LED
H_CONT     RES       1           ;Parte alta contatore ritardo
L_CONT     RES       1           ;Parte bassa contatore ritardo
DIREZ      RES       1           ;Direzione 0=sinistra 1=destra
TX_BYTE    RES       1           ;Valore da trasferire serialmente
B_CONT     RES       1           ;Contatore dei bit da trasmettere
#DEFINE    PIN_TX    PORTB,1     ;Pin di trasmissione dato
```

```
#DEFINE    PIN_CLK    PORTB,2     ;Pin di trasmissione clock
#DEFINE    PIN_STRB   PORTB,0     ;Pin di trasmissione strobe
;-----------------------------------------------------
```

A questo punto se volessimo trasferire il programma su un PIC16F84 sarebbe sufficiente effettuare le seguenti modifiche:

```
 ;-----------------------------------------------------
           PROCESSOR 16F84
           RADIX     DEC
           INCLUDE   "P16F84.INC"
           __CONFIG  11111111110001B
;-----------------------------------------------------
           ORG       12          ;Indirizzo inizio RAM dati
LUCE       RES       1           ;Valore da scrivere sui LED
H_CONT     RES       1           ;Parte alta contatore ritardo
L_CONT     RES       1           ;Parte bassa contatore ritardo
DIREZ      RES       1           ;Direzione 0=sinistra 1=destra
TX_BYTE    RES       1           ;Valore da trasferire serialmente
B_CONT     RES       1           ;Contatore dei bit da trasmettere
#DEFINE    PIN_TX    PORTB,1     ;Pin di trasmissione dato
#DEFINE    PIN_CLK   PORTB,2     ;Pin di trasmissione clock
#DEFINE    PIN_STRB  PORTB,0     ;Pin di trasmissione strobe
;-----------------------------------------------------
```

[Segue]

Pagina creata nel febbraio 2004 - Ultimo aggiornamento 11-5-2008