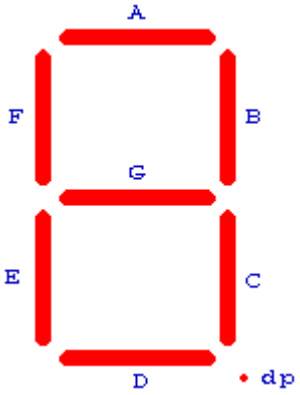
PIC
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA PER PRINCIPIANTI

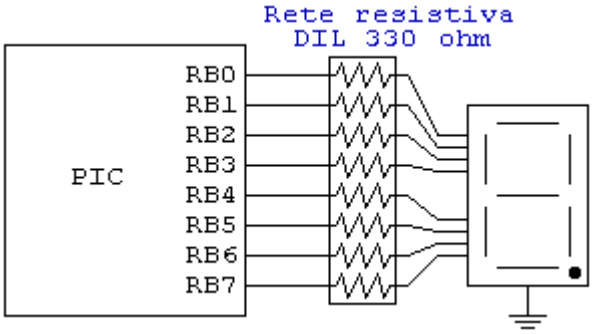
By Claudio Fin

[Precedente] [Indice principale]

CONTEGGIO DA 0 A 9 SU UN DISPLAY



Per avvicinarci ad una forma più «umana» di dialogo con la macchina vogliamo visualizzare su un comune display a 7 segmenti luminosi il valore contenuto in un registro (compreso tra 0 e 9). Questi display contengono 8 LED comandabili con i pin di uscita del PIC, e generalmente vengono identificati con i nomi visibili nella figura a fianco. Per far apparire una cifra occorre naturalmente accendere la giusta combinazione di diodi.



Usando un display a catodo comune lo schema è praticamente identico a quello del primo esperimento di visualizzazione, solo che i LED invece di essere separati sono uniti assieme nel contenitore del display, e tutti i catodi sono collegati assieme in un punto di massa comune (da qui il nome di catodo comune). Le 8 resistenze possono anche essere sostituite da una rete resistiva DIL da 330 ohm che ne contiene 8 racchiuse in un unico contenitore a forma di integrato. Possiamo collegare i pin di uscita ai segmenti nell'ordine che ci pare, ma per avere un certo criterio scegliamo di collegare il segmento A al pin RB0, B al pin RB1, per finire con il punto collegato a RB7. E' evidente che per far apparire la cifra 1 dovranno essere accesi i segmenti B e C, e quindi il valore binario scritto sulla porta dovrà essere 00000110. La seguente tabella riporta i codici da scrivere sulla PORTB per ogni valore che si vuole visualizzare.

Valore	Segmenti .GFEDCBA	Valore	Segmenti .GFEDCBA
0	00111111	5	01101101
1	00000110	6	01111101
2	01011011	7	00000111
3	01001111	8	01111111
4	01100110	9	01101111

Ora, per convertire un qualsiasi valore numerico binario (compreso tra 0 e 9) nel corrispondente codice display si possono seguire diversi procedimenti (in gergo un procedimento è detto anche algoritmo), si potrebbe per esempio usare una lunga serie di strutture condizionali IF THEN, ma questo è il tipico caso in cui invece l'uso delle tabelle dati trova l'impiego ideale semplificando il lavoro. Una tabella è un insieme di dati indirizzabili con un indice progressivo, nel nostro caso l'indice può essere proprio il valore numerico da 0 a 9, e i codici display possono essere contenuti nelle diverse posizioni della tabella. L'assembly dei PIC non permette di indirizzare direttamente aree di memoria programma contenenti dei dati, però è possibile scrivere una particolare subroutine (contenente una lista di istruzioni RETLW) che consente di ritornare con un valore caricato nel registro W dopo aver artificialmente alterato il valore del program counter per far saltare l'esecuzione alla RETLW desiderata:

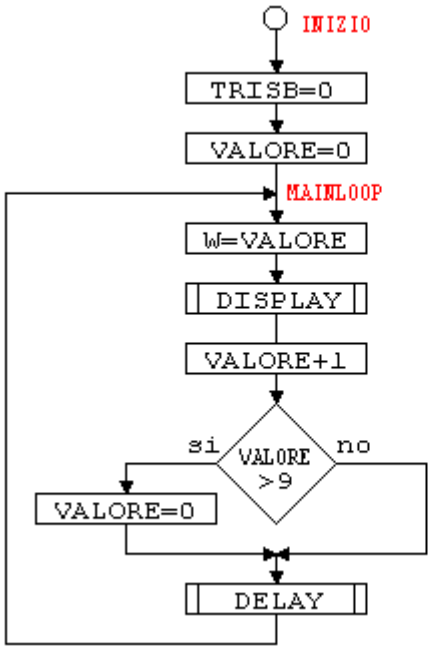
```
TABDISP     ADDWF       PCL,F          ;Conversione bcd->display
            RETLW       00111111B      ;0
            RETLW       00000110B      ;1
            RETLW       01011011B      ;2          -0-
            RETLW       01001111B      ;3        5|   |1
            RETLW       01100110B      ;4          -6-
            RETLW       01101101B      ;5        4|   |2
            RETLW       01111101B      ;6          -3- .7
            RETLW       00000111B      ;7
            RETLW       01111111B      ;8
            RETLW       01101111B      ;9
```

Per usare questa tabella occorre caricare in W l'indice dell'elemento desiderato (l'indice parte da 0) e chiamare la subroutine TABDISP. Il valore di W verrà sommato al program counter (registro PCL) ottenendo l'effetto di un GOTO ad una delle RETLW successive. Nel nostro caso in W va dapprima caricato il valore da visualizzare, poi va chiamata la subroutine di conversione, e al ritorno da essa W conterrà il codice display relativo. Lo schemino nei commenti sulla destra mostra un riepilogo di quali segmenti del display sono comandati dai vari bit.

C'è da ricordare che questo «trucco» di programmazione (l'alterazione del program counter) può portare a comportamenti anomali se usato in modo inadeguato. Per esempio è indispensabile che il valore di W sia compreso tra 0 e 9 quando la subroutine viene chiamata. Se valesse 10 o più verrebbe effettuato un salto oltre l'ultima RETLW ottenendo risultati imprevedibili, tipicamente un «crash» del programma o un reset. Inoltre, a causa del particolare funzionamento logico dei circuiti, si ottiene ugualmente un salto ad indirizzi errati se le istruzioni della tabella «sconfinano» dai bordi delle pagine di 256 indirizzi in cui si può pensare suddivisa l'intera memoria programma. Diciamo che se le istruzioni della tabella sono contenute entro i primi 256 indirizzi di memoria programma non si hanno problemi.

Vediamo quindi un programma completo che utilizza questa conversione tabellare (transcodifica) per visualizzare continuamente in sequenza le cifre da 0 a 9. Il main del programma è semplicemente un ciclo senza fine in cui un registro di nome VALORE viene continuamente incrementato, e quando diventa maggiore di 9 viene riazzerato. Ad ogni ciclo viene chiamata la subroutine DISPLAY che si incarica di tutto ciò che serve per la visualizzazione, e la subroutine DELAY per la solita necessità di rallentare la velocità di esecuzione. La subroutine DISPLAY si aspetta il valore da visualizzare nel registro W, e chiama a sua volta la subroutine TABDISP

per effettuare la conversione di codice, infine scrive il codice ottenuto sulla porta di uscita. In questo esempio si vede anche come realizzare in pratica un confronto tra due valori per capire se uno è maggiore dell'altro. Ad ogni ciclo viene eseguita la sottrazione 9-VALORE. Finchè VALORE è minore o uguale a 9 il flag C rimane a 1 e l'azzeramento di VALORE viene saltato. Invece quando VALORE diventa 10 il flag C viene resettato e viene eseguito subito l'azzeramento.




```
BSF     STATUS,RP0
CLRF    TRISB
BCF     STATUS,RP0
CLRF    VALORE

MOVF    VALORE,W

CALL    DISPLAY

INCF    VALORE,F

MOVF    VALORE,W
SUBLW   9
BTFSS   STATUS,C
CLRF    VALORE

CALL    DELAY
GOTO    MAINLOOP
```

```
;-----------------------------------------------------
; Programma di conteggio da 0 a 9 su display
;-----------------------------------------------------
            PROCESSOR  16F628
            RADIX      DEC
            INCLUDE    "P16F628.INC"
            __CONFIG   11110100010000B
;-----------------------------------------------------
            ORG        32
VALORE      RES        1
H_CONT      RES        1
L_CONT      RES        1
;-----------------------------------------------------
            ORG        0
            BSF        STATUS,RP0  ;Attiva banco 1
            CLRF       TRISB       ;Rende PORTB un'uscita
            BCF        STATUS,RP0  ;Ritorna al banco 0
            CLRF       VALORE      ;Azzera valore
MAINLOOP    MOVF       VALORE,W    ;W=valore
            CALL       DISPLAY     ;Chiama subroutine display
            INCF       VALORE,F    ;Incrementa valore di 1
            MOVF       VALORE,W    ;W=valore
            SUBLW      9           ;W=9-W
            BTFSS      STATUS,C    ;se valore <= 9 skip
            CLRF       VALORE      ;altrimenti valore=0
            CALL       DELAY       ;Richiama subroutine di ritardo
            GOTO       MAINLOOP    ;Nuovo ciclo del programma
;-----------------------------------------------------
DELAY       MOVLW      80
            MOVWF      H_CONT      ;Parte alta di CONT=80
            CLRF       L_CONT      ;Azzera parte bassa di CONT
DELAY2      DECF       L_CONT,F    ;Decrementa parte bassa di CONT
            COMF       L_CONT,W    ;Inverte i bit
```

```
            BTFSC       STATUS,Z    ;Se tutti zero c'è 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
;-----------------------------------------------------
DISPLAY     CALL        TABDISP     ;Chiama subroutine conversione
            MOVWF       PORTB       ;Scrive valore dei LED su PORTB
            RETURN
;-----------------------------------------------------
TABDISP     ADDWF       PCL,F       ;Conversione bcd->display
            RETLW       00111111B   ;0
            RETLW       00000110B   ;1
            RETLW       01011011B   ;2          -0-
            RETLW       01001111B   ;3        5|   |1
            RETLW       01100110B   ;4          -6-
            RETLW       01101101B   ;5        4|   |2
            RETLW       01111101B   ;6          -3- .7
            RETLW       00000111B   ;7
            RETLW       01111111B   ;8
            RETLW       01101111B   ;9
;                       .GFEDCBA    ;Segmenti
;-----------------------------------------------------
            END
```

Per tornare ancora un attimo sul discorso della programmazione strutturata e modulare, si può notare come la sezione main del programma funziona in un certo senso da «coordinatore», delegando a sottoprogrammi specializzati i dettagli del pilotaggio dell'hardware, le conversioni di codici e i ritardi. A loro volta i sottoprogrammi potrebbero appoggiarsi ad altri ancora più specializzati e di «basso livello». Da questo punto di vista i programmi possono essere pensati già dall'inizio come composti da diversi livelli o strati di astrazione via via crescenti mano a mano che ci si avvicina al main, e i dettagli dei livelli più bassi (ad esempio una subroutine di comunicazione) possono anche essere definiti (scritti o «rifiniti») in un secondo tempo quando, si è già progettata una struttura (o scheletro) funzionale ad alto livello. Questo modo di procedere si dice programmazione Top Down, dall'alto verso il basso, ed è particolarmente valida quando si lavora con linguaggi ad alto livello, o per scrivere le sezioni di «coordinamento» del programma. Personalmente in assembly trovo più comodo scrivere prima le subroutines di base necessarie per la gestione dell'hardware e per il funzionamento del programma, e poi rifinire la parte di controllo che le utilizza, questo modo di procedere si chiama invece programmazione Bottom Up, dal basso verso l'alto.

EFFETTI OTTICI COMPLESSI CON PROGRAMMI SEMPLICI

Il primo esempio di «animazione supercar» era realizzato con una soluzione algoritmica, era cioè ottenuto manipolando i bit grazie ad una serie di strutture decisionali IF THEN. Se si voleva ottenere un effetto più ricco si sarebbe dovuto rendere il programma molto più complesso. L'uso delle tabelle permette invece di ottenere tutti gli effetti che si vogliono semplicemente riscrivendo i dati in esse contenuti e leggendoli in sequenza. I due programmi seguenti creano il primo un effetto sul display, e il secondo è una variazione del tema supercar da guardarsi con i LED in fila come nel primissimo esperimento. Si può notare che i due programmi

sono identici, a parte la dimensione della tabella (e il valore limite del relativo indice) e la durata dei ritardi delle routines delay (determinati sperimentalmente per ottenere un effetto gradevole).

```
;-----------------------------------------------------
; Programma effetto su display
;-----------------------------------------------------
            PROCESSOR  16F628
            RADIX      DEC
            INCLUDE    "P16F628.INC"
            __CONFIG   11110100010000B
;-----------------------------------------------------
            ORG        32
INDICE      RES        1
H_CONT      RES        1
L_CONT      RES        1
;-----------------------------------------------------
            ORG        0
            BSF        STATUS,RP0
            CLRF       TRISB
            BCF        STATUS,RP0
            CLRF       INDICE      ;Azzera indice
MAINLOOP    MOVF       INDICE,W    ;W=indice
            CALL       DISPLAY     ;Chiama subroutine display
            INCF       INDICE,F    ;Incrementa indice di 1
            MOVF       INDICE,W    ;W=indice
            SUBLW      7           ;W=7-W
            BTFSS      STATUS,C    ;se indice <= 7 skip
            CLRF       INDICE      ;altrimenti indice=0
            CALL       DELAY       ;Richiama subroutine di ritardo
            GOTO       MAINLOOP
;-----------------------------------------------------
DELAY       MOVLW      40
            MOVWF      H_CONT
            CLRF       L_CONT
DELAY2      DECF       L_CONT,F
            COMF       L_CONT,W
            BTFSC      STATUS,Z
            DECF       H_CONT,F
            MOVF       L_CONT,W
            IORWF      H_CONT,W
            BTFSS      STATUS,Z
            GOTO       DELAY2
            RETURN
;-----------------------------------------------------
DISPLAY     CALL       TABDISP
            MOVWF      PORTB
            RETURN
;-----------------------------------------------------
TABDISP     ADDWF      PCL,F
            RETLW      00100001B
            RETLW      00000011B
            RETLW      01000010B
            RETLW      01010000B
            RETLW      00011000B
            RETLW      00001100B
            RETLW      01000100B
            RETLW      01100000B
;-----------------------------------------------------
            END



;-----------------------------------------------------
; Programma effetto supercar sofisticato
;-----------------------------------------------------
```

```
            PROCESSOR  16F628
            RADIX      DEC
            INCLUDE    "P16F628.INC"
            __CONFIG   11110100010000B
;-----------------------------------------------------
            ORG        32
INDICE      RES        1
H_CONT      RES        1
L_CONT      RES        1
;-----------------------------------------------------
            ORG        0
            BSF        STATUS,RP0
            CLRF       TRISB
            BCF        STATUS,RP0
            CLRF       INDICE        ;Azzera indice
MAINLOOP    MOVF       INDICE,W      ;W=indice
            CALL       VISUALIZZA    ;Chiama subroutine visualizza
            INCF       INDICE,F      ;Incrementa indice di 1
            MOVF       INDICE,W      ;W=indice
            SUBLW      23            ;W=23-W
            BTFSS      STATUS,C      ;se indice <= 23 skip
            CLRF       INDICE        ;altrimenti indice=0
            CALL       DELAY         ;Richiama subroutine di ritardo
            GOTO       MAINLOOP
;-----------------------------------------------------
DELAY       MOVLW      15
            MOVWF      H_CONT
            CLRF       L_CONT
DELAY2      DECF       L_CONT,F
            COMF       L_CONT,W
            BTFSC      STATUS,Z
            DECF       H_CONT,F
            MOVF       L_CONT,W
            IORWF      H_CONT,W
            BTFSS      STATUS,Z
            GOTO       DELAY2
            RETURN
;-----------------------------------------------------
VISUALIZZA  CALL       TABDISP
            MOVWF      PORTB
            RETURN
;-----------------------------------------------------
TABDISP     ADDWF      PCL,F
            RETLW      00000001B
            RETLW      00000001B
            RETLW      00000011B
            RETLW      00000111B
            RETLW      00001111B
            RETLW      00011111B
            RETLW      00111110B
            RETLW      01111100B
            RETLW      11111000B
            RETLW      11110000B
            RETLW      11100000B
            RETLW      11000000B
            RETLW      10000000B
            RETLW      10000000B
            RETLW      11000000B
            RETLW      11100000B
            RETLW      11110000B
            RETLW      11111000B
            RETLW      01111100B
            RETLW      00111110B
            RETLW      00011111B
            RETLW      00001111B
            RETLW      00000111B
            RETLW      00000011B
```

```
;-------------------------------------------------------
              END
```

[Segue]

Pagina creata nel febbraio 2004 - Ultimo aggiornamento 7-2-2005