Esercitazioni C Enhanced Midrange

6 – Display a 7 segmenti

Vediamo ora come:

- utilizzare display a 7 segmenti
- utilizzare array e lookup table

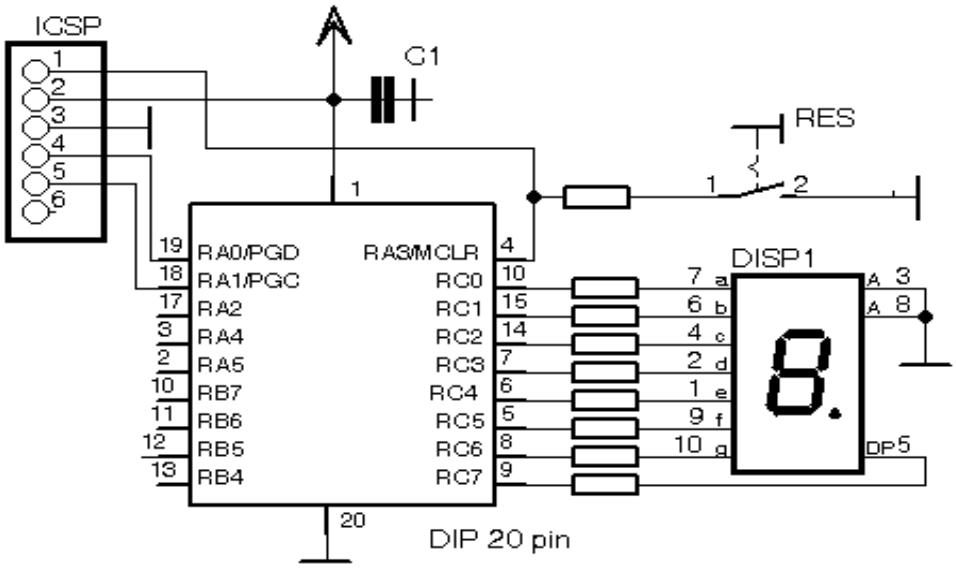
I LED possono servire a segnalare eventi e situazioni, ma per una indicazione più chiara possiamo utilizzare un display a 7 segmenti.
Attraverso questo display possiamo comunicare all'esterno del microrontroller con numeri e cifre.
Fisicamente i segmenti del display sono dei LED e come tali vanno utilizzati.

Possiamo implementare un semplice esercizio: all'accensione, il display indica la cifra 0, dopo di che le cifre crescono di 1 ogni secondo, comprendendo quelle esadecimali da A a F. Il pulsante di Reset azzera il conteggio e lo fa ripartire.

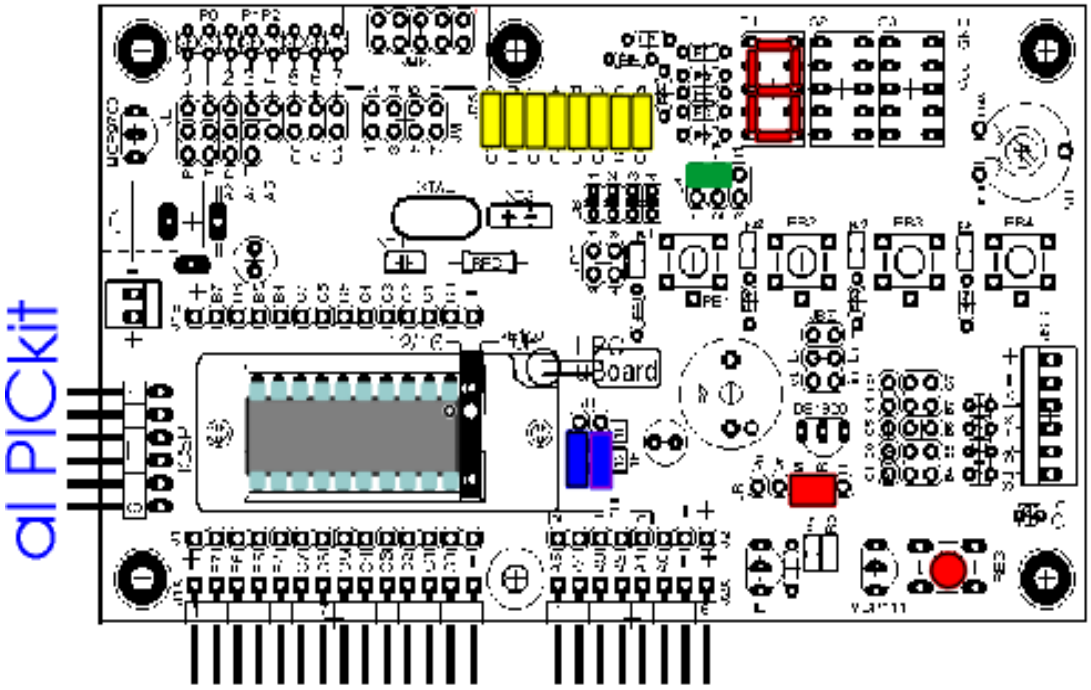
Poichè occorrono almeno 7 pin per comandare i segmenti (eventualmente più uno per il punto decimale) utilizziamo ancora un chip a 20pin **16F1619**.

Utilizziamo gli 8 pin del **PORTC** per comandare un display del tipo a catodo comune, che viene collegato alla Vss. Abbiamo, in questo modo, un comando analogo a quanto visto per i LED: il pin portato a livello 1 accenderà il relativo segmento.

Ovviamente, in serie ai segmenti sono interposte resistenze di limitazione della corrente.
Le resistenze potranno essere da 330 ohm a 1 kohm a seconda della sensibilità dei LED dei segmenti. La capacità di corrente di un port del PIC è, in genere, 25mA e non si deve superare questo limite. Per il calcolo della resistenza in serie ai LED, vedere qui.



La scheda **LPCuB** (www) offre una facile realizzazione, ma è possibile con facilità assemblare il circuito su una breadboard.



Il pulsante di **Reset** è collegato; il pull-up è integrato nel chip, quindi la resistenza esterna Rp può

essere omessa (ricordando che, per chip che non dispongono del wpu su MCLR, invece, dovrà essere presente).

6.1 – La gestione del display a 7 segmenti.

Abbiamo di fronte una esercitazione analoga all'ultima variazione della precedente, che abbiamo risolto con una semplice, ma inefficiente, lista di comandi.

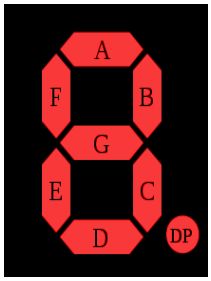
Ora vogliamo vedere come risolvere il problema in un modo più elegante ed efficace.

Operare con un display a segmenti coinvolge la necessità di accendere solo una parte di questi segmenti a seconda della cifra che vogliamo fare apparire.
Vediamo nei dettagli come operare.

Il display a 7 segmenti può rappresentare le cifre da 0 a 9 (**0 1 2 3 4 5 6 7 8 9**) e varie lettere e segni (**A b C d E F H I L o P S t u**, ecc.) . Le possibilità alfabetiche sono limitate e occorrerebbero display a 14 o 16 segmenti, non di uso comune perchè abbastanza costosi. Però, i 7 segmenti offrono comunque una gamma di indicazioni abbastanza ampia per molte applicazioni. Rispetto ai display LCD non richiedono retro illuminazione, dato che i segmenti sono di per se luminosi; per contro hanno un consumo di corrente maggiore degli LCD.

Per prima cosa, indichiamo in una tabella la corrispondenza tra le cifre esadecimali da 0 a F: **cifra rappresentata e segmenti accesi,** indicati con x:

	a	b	c	d	e	f	g		a	b	c	d	e	f	g	
0	x	x	x	x	x	x			x	x	x	x	x	x	x	8
1		x	x						x	x	x	x		x	x	9
2	x	x		x	x		x		x	x	x		x	x	x	A
3	x	x	x	x			x				x	x	x	x	x	b
4		x	x			x	x		x			x	x	x		C
5	x		x	x		x	x			x	x	x	x		x	d
6	x		x	x	x	x	x		x			x	x	x	x	E
7	x	x	x						x				x	x	x	F



Non c'è una relazione matematica tra segmenti e cifre; di conseguenza dobbiamo ricavare una ulteriore tabella.

Abbiamo collegato i segmenti ai pin in questa sequenza:

pin	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
segmento	dp	g	f	e	d	c	b	a

Il display usato è a catodo comune, ovvero il segmento è acceso se se il pin è a livello 1; in base a questo possiamo tracciare la **tabella di corrispondenza tra stato dei pin e indicazione del display**.

segmento		dp	g	f	e	d	c	b	a	
pin		RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	*hex*
0	0b	0	0	1	1	1	1	1	1	0x3F
1	0b	0	0	0	0	0	1	1	0	0x06
2	0b	0	1	0	1	1	0	1	1	0x5B
3	0b	0	1	0	0	1	1	1	1	0x4F
4	0b	0	1	1	0	0	1	1	0	0x66
5	0b	0	1	1	0	1	1	0	1	0x6D
6	0b	0	1	1	1	1	1	0	1	0x7D
7	0b	0	0	0	0	0	1	1	1	0x07
8	0b	0	1	1	1	1	1	1	1	0x7F
9	0b	0	1	1	1	1	1	1	1	0x6F
A	0b	0	1	1	1	0	1	1	1	0x77
b	0b	0	1	1	1	1	1	0	0	0x7C
C	0b	0	0	1	1	1	0	0	1	0x39
d	0b	0	1	0	1	1	1	1	0	0x5E
E	0b	0	1	1	1	1	0	0	1	0x79
F	0b	0	1	1	1	0	0	0	1	0x71

Il pin **RC7** è collegato al punto decimale (**dp**), che viene acceso, dove richiesto, portando il bit a 1.

Se dovessimo impiegare display ad **anodo comune** (collegato alla Vdd), i valori della tabella andrebbero invertiti in quanto il segmento sarebbe acceso dal pin a livello 0.

Data la corrispondenza tra pin e segmenti, i valori ricavati dalla tabella sono direttamente trasferibili al port dove colleghiamo **RC0** col **segmento a**, **RC1** col **segmento b** e così via.
La tabella indica sia in numeri binari, sia in esadecimali.

Però, dove necessario, possiamo avere una qualunque corrispondenza tra pin e segmenti: basterà ridisegnare la tabella.

Una volta ottenuta la lista delle condizioni dei pin per le varie cifre, la possiamo riportare in una forma adatta al compilatore: si tratta di definire un **array**. Si tratta in effetti di una lookup table come visto nelle esercitazioni Assembly.

Nel C una lookup table è definita solitamente attraverso un array inizializzato. La forma tipica compatta potrebbe essere questa:

```
unsigned char tab7seg[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
```

oppure anche in una forma che permette di commentare ogni valore:

```
unsigned char tab7seg[16] = {
      0x3F,   // 0
      0x06,   // 1
      0x5B,   // 2
      0x4F,   // 3
      0x66,   // 4
      0x6D,   // 5
      0x7D,   // 6
      0x07,   // 7
      0x7F,   // 8
      0x6F,   // 9
      0x77,   // A
      0x7C,   // b
      0x39,   // C
      0x5E,   // d
      0x79,   // E
      0x71    // F
};
```

Avendo a che fare con soli numeri a 8 bit, definiamo il tipo della variabile come **unsigned char**, il che permette valori da 0 a 255 (0x00 – 0xFF).

In questo array, il digit che deve essere rappresentato sul display è l'indice della tabella.

Per muovere il valore sul port basterà scrivere:

```
LATC = tab7seg[digit];
```

Altre informazioni sugli array.

6.2 - Il programma: accendiamo il display

Come prima cosa occorrerà gestire i pin del chip, come visto finora, quindi possiamo passare alla visualizzazione delle cifre.

Il **main** conterrà l'inizializzazione e il loop che coinvolge la funzione appena vista.

Il tempo tra una cifra e la successiva è posto a 800ms, ma cambiando il valore sarà possibile un'altra cadenza.

Il ciclo si ripete indefinatamente:.una volta raggiunta la cifra F il conteggio riprende da 0.
Premendo reset si interrompe il programma che riparte dall'inizio.

```
 void main (){

 unsigned char digit;            // cifra da inviare al display

    // Inizializza I/O
    ANSELA = 0;                  // disabilita analogiche su
    ANSELB = 0;                  // tutti port
    ANSELC = 0;

    // protezione pi non usati
    OPTION_REGbits,nWPUEN = 0;   // abilitazione generale wpu

    // inizio loop con display spento
    LATC = 0;                   // preset latch C a 0 (segmenti spenti)
    TRISC = 0;                  // tutti i pin di PORTC come uscite

  // loop di display delle cifre da 0 a F, ogni 800ms
   while (1){
      for (digit=0; digit<15; digit++){
      __delay_ms(800);
      LATC = Display[digit]
      }
   }
}
```

es6C

```
/*****************************************************************
*----------------------------------------------------------------
*
*    Titolo        :  C Enhanced - es6C
*                     Sequenza di 16 caratteri sul display a 7 segm.
*                     Cifre da 0 a F cadenzate di 0,8s.
*    Data          :  01-05-2011
*    Modificato il :
*    Versione      :  V0.0
*    Ref. Hardware :  PIC16F1619
*    Autore        :  afg
*----------------------------------------------------------------
*****************************************************************
*  Impiego pin :
*  ---------------
*       16F1619 @ 20 pin
*
*                   |__\/__|
*              Vdd -|1   20|- Vss
*       RA5/CLKIN -|2   19|- RA0/ICSPDAT
*      RA4/CLKOUT -|3   18|- RA1/ICSPCLK
*        RA3/MCLR -|4   17|- RA2
*              RC5 -|5   16|- RCO
*              RC4 -|6   15|- RC1
*              RC3 -|7   14|- RC2
*              RC6 -|8   13|- RB4
*              RC7 -|9   12|- RB5
*              RB7 -|10  11|- RB6
*                   |______|
*
* Impiego pin:
*    Vdd      1: ++
*    RA5      2: In
*    RA4      3: In
*    RA3      4: MCLR
*    RC5      5: Out segm. e
*    RC4      6: Out segm. d
*    RC3      7: Out segm. f
*    RC6      8: Out segm. g
*    RC7      9: Out segm. dp
*    RB7     10: In
*    RB6     11: In
*    RB5     12: In
*    RB4     13: In
*    RC2     14: Out segm. c
*    RC1     15: Out segm. b
*    RC0     16: Out segm. a
*    RA2     17: In
*    RA1     18: In
*    RA0     19: In
*    Vss             20: --
*
* Funzioni:
*    RA0/AN0/DAC1OUT1/C1IN+
*    RA1/AN1/Vref+/C1IN0-C2IN0-
*    RA2/AN2/[T0CKI/CWG1IN]/ZCD1IN/INT
```

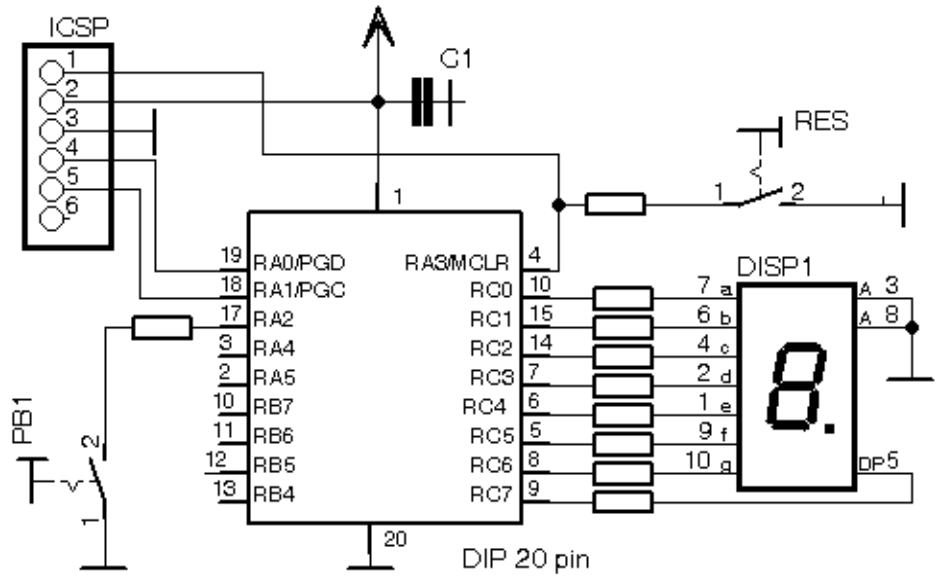
```
*    RA3/MCLR/[T6IN/SMTWIN2]
*    RA4/AN4/[T1G/SMTSIG1]
*    RA5/[T1CK/T2IN/SMTWIN1]/CLCIN3
*    RB4/AN10/[SDI]
*    RB5/AN11/[RX]
*    RB6/[SCK]
*    RB7/[CK]
*    RC0/AN4/C2IN+/[T5CKI]
*    RC1/AN5/C1IN1-/C2IN1-
*    RC2/AN6/C1IN2-/C2IN2-
*    RC3/AN7/C1IN3-/C2IN3-/[T5G/CCP2/CLCIN0/ATCC]
*    RC4/[T3G/CLCIN1]/HIC4
*    RC5/[T3CKI/CCP1/ATIN]/HIC5
*    RC6/AN8/[!SS]
*    RC7/AN9
*
* [] rilocabili con PPS
*
* Note: - INTOSC default @ 500kHz
*       - Tutti i pin con wpu e ioc
*****************************************************************/

#include "C:\Corso_C\MyIncludes\conf1619_0.h"

#include <xc.h>

#define _XTAL_FREQ  500000

// 7 segments lookup table per cifre da 0 a F
// Rx0=segm.a, Rx1=segm.b,...Rx7=segm.dp
  unsigned char tab7seg[16] = {
        0x3F,  // 0
        0x06,  // 1
        0x5B,  // 2
        0x4F,  // 3
        0x66,  // 4
        0x6D,  // 5
        0x7D,  // 6
        0x07,  // 7
        0x7F,  // 8
        0x6F,  // 9
        0x77,  // A
        0x7C,  // b
        0x39,  // C
        0x5E,  // d
        0x79,  // E
        0x71   // F
  };

/************************ MAIN PROGRAM ****************************/
void main(){

 unsigned char digit;            // cifra da inviare al display

    // Inizializza I/O
    ANSELA = 0;                  // disabilita analogiche su
    ANSELB = 0;                  // tutti port
    ANSELC = 0;
```

```
    // protezione pi non usati
    OPTION_REGbits,nWPUEN = 0;   // abilitazione generale wpu

    // inizio loop con display spento
    LATC = 0;                    // preset latch C a 0 (segmenti spenti)
    TRISC = 0;                   // tutti i pin di PORTC come uscite

  while(1){                      // loop principale
  //  display delle cifre da 0 a F, una al secondo
     for(digit=0; digit<16; digit++){
       __delay_ms(800);
       LATC = tab7seg[digit];
     }
  }
}
```

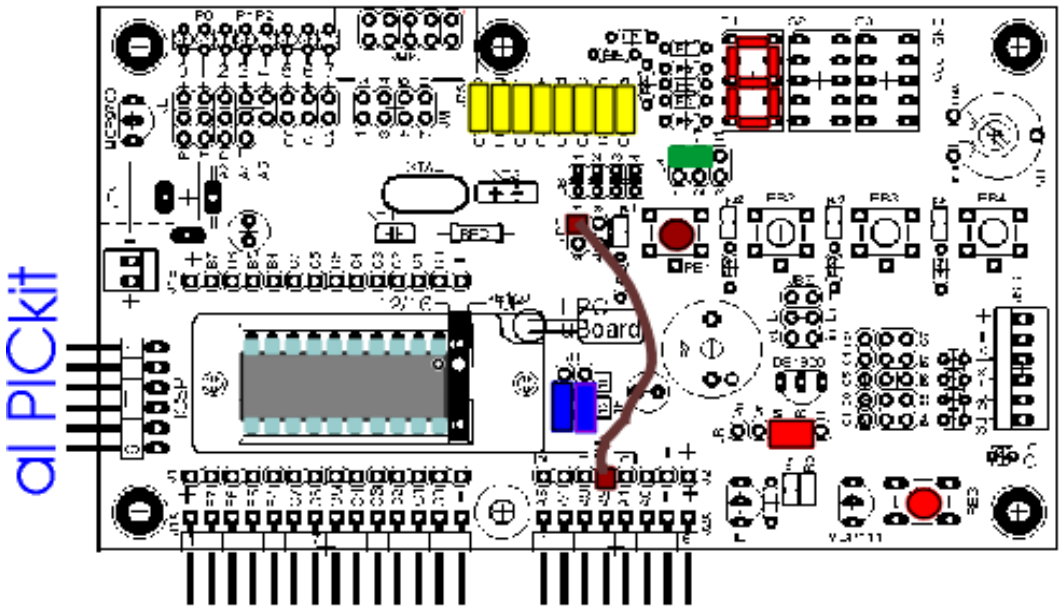
Questa versione è fornita come progetto completo (MPLABX v.4.13 o successivi, XC8 v.1.34 o successivi e PIC16F1619).
I file del progetto sono previsti per stare in una cartella *C:\Corso_C\es6C*.

6.3 - Variazione 1

Possiamo fare una variazione semplice: utilizziamo un pulsante come input del conteggio. Ad ogni pressione il display avanzerà di una cifra.



Sulla **LPCuB** (www) la configurazione è immediata, aggiungendo un ponticello volante (marrone).:



Notiamo che non sono necessari pull-up né su MCLR, né sul pulsante, dato che utilizziamo i weak pull-up integrati.

6.4 – Il sorgente

E' opportuno, prima del main, definire una label per il pulsante e per il punto decimale:

```
#define PB1 PORTAbits.RA2
#define dp  LATAbits.LATA7
```

cosa che ci consente di individuare gli I/O attraverso le label stesse ed alleggerisce il testo.

Definiamo il puntatore dell'array e lo inizializziamo, in modo da partire dalla cifra 0:

```
unsigned char digit = 0;
```

Il `main`: i commenti dovrebbero chiarire le operazioni svolte.

```
void main() {

 unsigned char digit=0;        // cifra da inviare al display

  // Inizializza I/O
  ANSELA = 0;                  // disabilita analogiche su
  ANSELB = 0;                  // tutti port
  ANSELC = 0;

  // abilitiamo wpu per protezione pin non usati
   OPTION_REGbits,nWPUEN = 0;  // abilitazione generale wpu

  // inizio loop con display spento
   LATC = 0;                   // preset latch C a 0 (segmenti spenti)
   TRISC = 0;                  // tutti i pin di PORTC come uscite
   LATC=tab7segm[digit]        // display = 0

   while(1){                   /* loop continuo */

     if (PB1==0){              // Solo se PB1=0 (pulsante premuto)
     __delay_ms (20);          // Ritardo di debounce

      if (PB1==0){             // Solo se PB2 è ancora chiuso
       if (digit==15){         // Se la cifra è F, fine conteggio
         dp=1;                 // accendi dp
         while(1);             // e blocca processo
         }else{
         digit++;              // aggiorna indice
         LATC=tab7segm[digit]; // Trasferisce dato da tabella a PORTC
          }
      }
     }
      while (PB1==0);          // Se PB2 è ancora chiuso non fare nulla
      __delay_ms(30);          // al rilascio, ritardo di debounce
    }
 }
```

In altre parole:

- iniziamo con il display che indica 0
- se il pulsante non è premuto, attendiamo
- quando il pulsante è premuto, se la cifra a display è diversa da F, avanziamo alla cifra successiva e ritorniamo nel ciclo di attesa del pulsante
- se è F, accendiamo il punto decimale e blocchiamo il programma

Utilizziamo un debounce di 30ms per sicurezza.
Se il conteggio avanza disordinatamente alla pressione o rilascio del pulsante, inserite un tempo maggiore.

L'aggiunta dei tempi di debouce rallenta la velocità di conteggio; in una applicazione pratica in cui si vuole ottenere la massima prestazione, occorrerà che il segnale di ingresso al contatore sia completamente privo di rimbalzi, cosa normale negli impianti industriali

Il punto decimale si accenderà una volta raggiunto il massimo del conteggio e resterà acceso per indicare la situazione, mentre il conteggio sarà bloccato.
Il reset cancella il display e riavvia il programma dall'inizio.

es6C_1

```
/*****************************************************************
*----------------------------------------------------------------
*
*    Titolo        :  C Enhanced - es6C_1
*                     Contatore su display a 7 segmenti.
*                     Il conteggio avanza di un passo ad ogni
*                     pressione del pulsante.
*                     Arrivati a F il programma si blocca, accendendo
*                     il punto decimale. Reset riavvia il ciclo.
*    Data          :  01-05-2011
*    Modificato il :
*    Versione      :  V0.0
*    Ref. Hardware :  PIC16F1619
*    Autore        :  afg
*----------------------------------------------------------------
*****************************************************************
*  Impiego pin :
*  ---------------
*       16F1619 @ 20 pin
*
*                    |‾‾\/‾‾|
*              Vdd -|1   20|- Vss
*       RA5/CLKIN -|2   19|- RA0/ICSPDAT
*      RA4/CLKOUT -|3   18|- RA1/ICSPCLK
*        RA3/MCLR -|4   17|- RA2
*              RC5 -|5   16|- RCO
*              RC4 -|6   15|- RC1
*              RC3 -|7   14|- RC2
*              RC6 -|8   13|- RB4
*              RC7 -|9   12|- RB5
*              RB7 -|10  11|- RB6
*                   |______|
*
* Impiego pin:
*    Vdd      1: ++
*    RA5      2: In
*    RA4      3: In
*    RA3      4: MCLR
*    RC5      5: Out segm. e
*    RC4      6: Out segm. d
*    RC3      7: Out segm. f
*    RC6      8: Out segm. g
*    RC7      9: Out segm. dp
*    RB7     10: In
*    RB6     11: In
*    RB5     12: In
*    RB4     13: In
*    RC2     14: Out segm. c
*    RC1     15: Out segm. b
*    RC0     16: Out segm. a
*    RA2     17: In  PB1
*    RA1     18: In
*    RA0     19: In
*    Vss              20: --
*
* Funzioni:
```

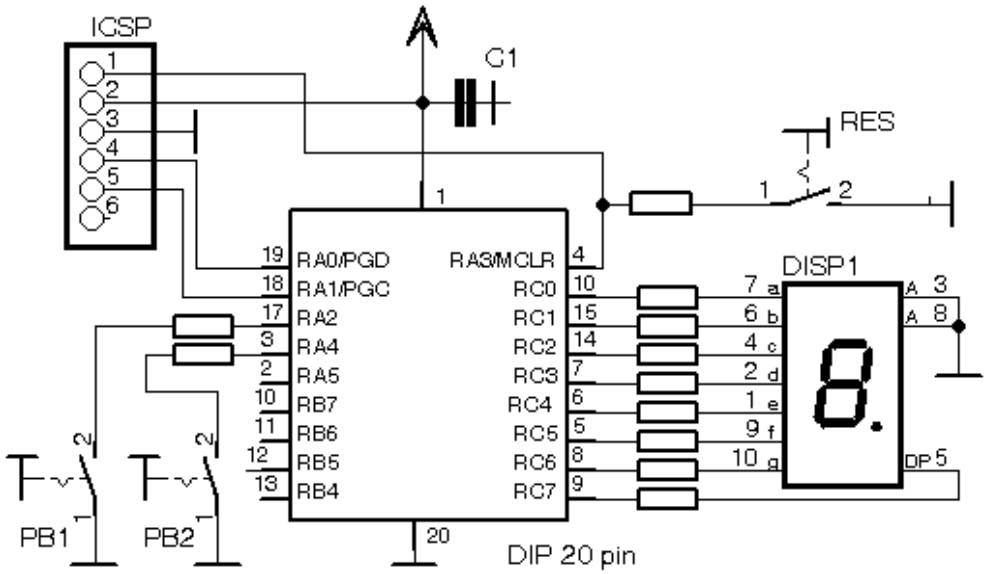
```
*    RA0/AN0/DAC1OUT1/C1IN+
*    RA1/AN1/Vref+/C1IN0-C2IN0-
*    RA2/AN2/[T0CKI/CWG1IN]/ZCD1IN/INT
*    RA3/MCLR/[T6IN/SMTWIN2]
*    RA4/AN4/[T1G/SMTSIG1]
*    RA5/[T1CK/T2IN/SMTWIN1]/CLCIN3
*    RB4/AN10/[SDI]
*    RB5/AN11/[RX]
*    RB6/[SCK]
*    RB7/[CK]
*    RC0/AN4/C2IN+/[T5CKI]
*    RC1/AN5/C1IN1-/C2IN1-
*    RC2/AN6/C1IN2-/C2IN2-
*    RC3/AN7/C1IN3-/C2IN3-/[T5G/CCP2/CLCIN0/ATCC]
*    RC4/[T3G/CLCIN1]/HIC4
*    RC5/[T3CKI/CCP1/ATIN]/HIC5
*    RC6/AN8/[!SS]
*    RC7/AN9
*
* [] rilocabili con PPS
*
* Note: - INTOSC default @ 500kHz
*       - Tutti i pin con wpu e ioc
*****************************************************************/

#include "C:\Corso_C\MyIncludes\conf1619_0.h"

#include <xc.h>

#define _XTAL_FREQ  500000

#define PB1  PORTAbits,RA2
#define dp   LATCbits.LATC7

// 7 segments lookup table per cifre da 0 a F
// Rx0=segm.a, Rx1= segm.b, ...Rx7= segm.dp
  unsigned char tab7segm[16] = {
        0x3F,  // 0
        0x06,  // 1
        0x5B,  // 2
        0x4F,  // 3
        0x66,  // 4
        0x6D,  // 5
        0x7D,  // 6
        0x07,  // 7
        0x7F,  // 8
        0x6F,  // 9
        0x77,  // A
        0x7C,  // b
        0x39,  // C
        0x5E,  // d
        0x79,  // E
        0x71   // F
  };

/************************ MAIN PROGRAM ***************************/
void main() {
```

```
 unsigned char digit=0;       // cifra da inviare al display

  // Inizializza I/O
  ANSELA = 0;                 // disabilita analogiche su
  ANSELB = 0;                 // tutti port
  ANSELC = 0;

  // abilitiamo wpu per protezione pin non usati
   OPTION_REGbits,nWPUEN = 0;  // abilitazione generale wpu

  // inizio loop con display spento
   LATC = 0;                  // preset latch C a 0 (segmenti spenti)
   TRISC = 0;                 // tutti i pin di PORTC come uscite

   LATC=tab7segm[digit];      // display iniziale 0

   while(1){                  /* loop continuo */

     if (PB1==0){             // Solo se PB1=0 (pulsante premuto)
     __delay_ms (20);         // Ritardo di debounce

      if(PB1==0){             // Solo se PB2 è ancora chiuso
       if (digit==15){        // Se indice = 15 , fine conteggio
        dp=1;                 // accendi dp
        while(1);             // e blocca processo
        }else{
        digit++;              // aggiorna indice
        LATC=tab7segm[digit]; // Trasferisce dato da tabella a PORTC
         }
      }
     }
      while (PB1==0);         // Se PB2 è ancora chiuso non fare nulla
      __delay_ms(30);         // al rilascio, ritardo di debounce
    }
}
```

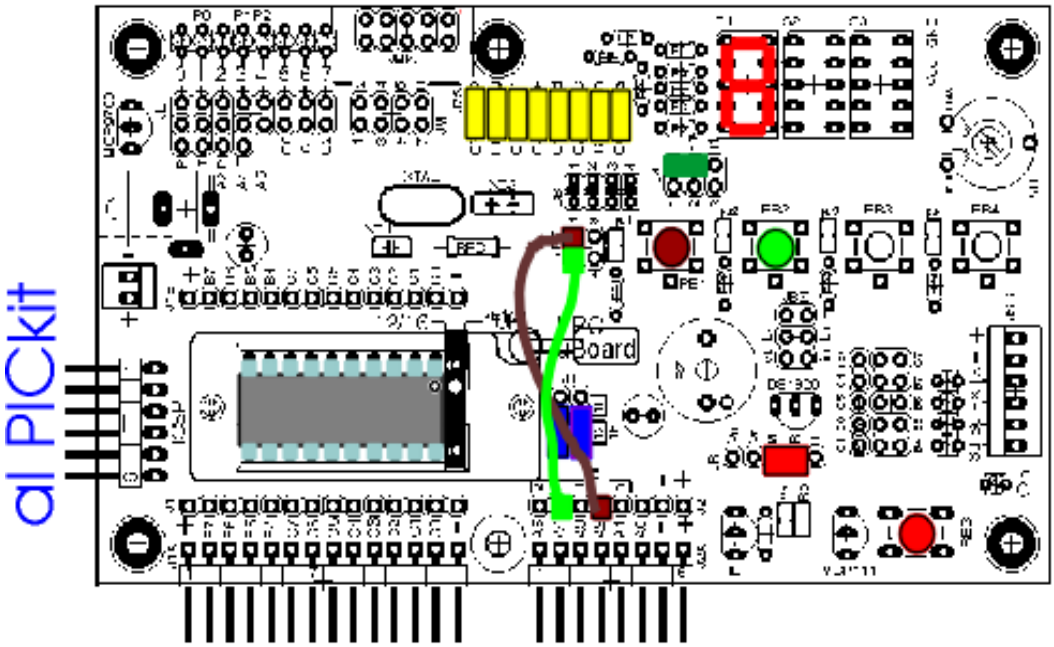
Questa versione è fornita come progetto completo (MPLABX v.4.13 o successivi, XC8 v.1.34 o successivi e PIC16F1619).
I file del progetto sono previsti per stare in una cartella ***C:\Corso_C\es6C\es6C_1***.

6. 5 - Variazione 2

Possiamo fare una ulteriore variazione: aggiungiamo un terzo pulsante che utilizzeremo come conteggio a scalare.



Sulla **LPCuB** (www) la configurazione è immediata, aggiungendo un ponticello volante (verde) alla precedente.



In sostanza abbiamo:

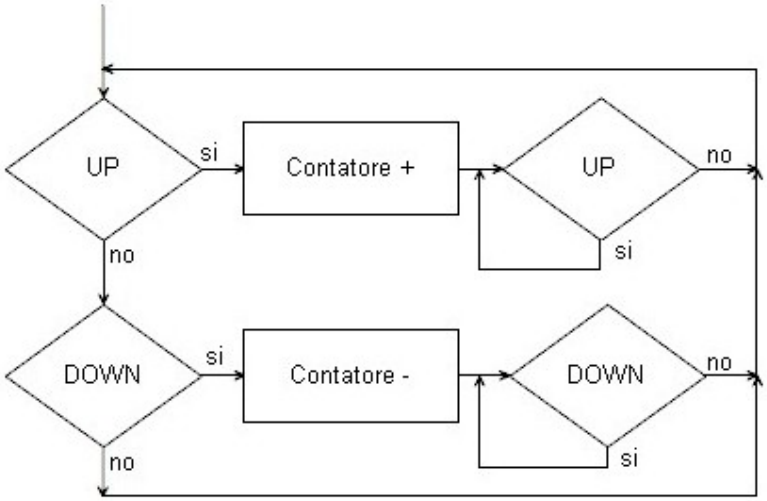
- **PB1** che verrà usato per il **conteggio UP**
- **PB2** che verrà usato per il **conteggio DOWN**
- **RES** che verrà usato per la **cancellazione**

Per semplicità definiamo i pulsanti Up e Down. e il punto decimale del display che ci servirà da indicazione ausiliaria.

```
#define Up     PORTAbits,RA2
#define Down   PORTAbits.RA4
#define dp     LATCbits.LATC7
```

Possiamo individuare la chiusura dei pulsanti semplicemente verificando il livello logico dei pin a cui sono collegati. Al pulsante **Up** corrisponde un incremento del contatore, al pulsante **Down** un decremento.

Fino a che un pulsante è premuto, l'altro pulsante non viene preso in considerazione.



Il diagramma di flusso ci permette di comprendere immediatamente la logica delle istruzioni da implementare. Dobbiamo però **aggiungere qualche particolare importante**.

- Innanzitutto un **debounce** in chiusura e apertura dei contatti.
- Poi un controllo del **pulsante rilasciato** prima di passare al test del successivo.
- Infine una **limitazione del conteggio** in su alla cifra F e in giù alla cifra 0.

Per identificare questi estremi, si è deciso di utilizzare il punto decimale: quando il conteggio arriva a F in salita o a 0 in discesa, si accende il punto decimale per identificare l'overflow/underflow del contatore e il pulsante Up o Down non è più attivo.

Per chiarire, supponiamo di stare usando il pulsante Up: il conteggio va da 0 a F, dopo di che si accende il punto decimale e il pulsante non ha più alcun effetto fino a che la cifra su display non viene scalata con il pulsante Down. Così pure, se contiamo in discesa con l'ingresso Down, arrivati a 0 si accende il punto decimale e il pulsante è inattivo. Occorrerà agire sul pulsante Up.

Il pulsante di reset, premuto, riavvia il programma, azzerando il contatore.

Il clock è sempre quello di default, giusto per ribadire che in applicazioni come questa un basso tempo di istruzione, ovvero un clock di frequenza elevata, non ha nessuna utilità. In effetti, si può anche utilizzare un un clock ancor più ridotto.

Il sorgente è sufficientemente commentato.

 Una nota: **questo contatore UP/DOWN è una esercitazione del corso e non è il progetto per una applicazione industriale.**
Lo scopo è quello di dimostrare le funzioni del C e del microcontroller e non realizzare applicazioni professionali.

es2C_2

```
/*****************************************************************
*----------------------------------------------------------------
*
*    Titolo        :  C Enhanced - es6C_2
*                     Contatore up/down su display a 7 segmenti.
*                   - Un pulsante Up fa avanzare il contatore fino
*                     alla cifra F, dopo di che si accende il punto
*                     decimale e il pulsante non ha più effetto.
*                     Occorrerà sbloccare la situazione con il
*                     il pulsante Down.
*                   - Un pulsante Down fa arretrare il conteggio fino
*                     alla cifra 0, dopo di che si accende il punto
*                     decimale e il pulsante non ha più effetto.
*                     Occorrerà sbloccare la situazione con il
*                     il pulsante Up.
*                     Reset è attivo e cancella il display.
*    Data          :  01-05-2011
*    Modificato il :
*    Versione      :  V0.0
*    Ref. Hardware :  PIC16F1619
*    Autore        :  afg
*----------------------------------------------------------------
*****************************************************************
*  Impiego pin :
*  ----------------
*       16F1619 @ 20 pin
*
*                    |__\/__|
*              Vdd -|1   20|- Vss
*       RA5/CLKIN -|2   19|- RA0/ICSPDAT
*      RA4/CLKOUT -|3   18|- RA1/ICSPCLK
*        RA3/MCLR -|4   17|- RA2
*              RC5 -|5   16|- RCO
*              RC4 -|6   15|- RC1
*              RC3 -|7   14|- RC2
*              RC6 -|8   13|- RB4
*              RC7 -|9   12|- RB5
*              RB7 -|10  11|- RB6
*                   |______|
*
* Impiego pin:
*    Vdd      1: ++
*    RA5      2: In
*    RA4      3: In  PB2
*    RA3      4: MCLR
*    RC5      5: Out segm. e
*    RC4      6: Out segm. d
*    RC3      7: Out segm. f
*    RC6      8: Out segm. g
*    RC7      9: Out segm. dp
*    RB7     10: In
```

```
*    RB6     11: In
*    RB5     12: In
*    RB4     13: In
*    RC2     14: Out segm. c
*    RC1     15: Out segm. b
*    RC0     16: Out segm. a
*    RA2     17: In  PB1
*    RA1     18: In
*    RA0     19: In
*    Vss              20: --
*
* Funzioni:
*    RA0/AN0/DAC1OUT1/C1IN+
*    RA1/AN1/Vref+/C1IN0-C2IN0-
*    RA2/AN2/[T0CKI/CWG1IN]/ZCD1IN/INT
*    RA3/MCLR/[T6IN/SMTWIN2]
*    RA4/AN4/[T1G/SMTSIG1]
*    RA5/[T1CK/T2IN/SMTWIN1]/CLCIN3
*    RB4/AN10/[SDI]
*    RB5/AN11/[RX]
*    RB6/[SCK]
*    RB7/[CK]
*    RC0/AN4/C2IN+/[T5CKI]
*    RC1/AN5/C1IN1-/C2IN1-
*    RC2/AN6/C1IN2-/C2IN2-
*    RC3/AN7/C1IN3-/C2IN3-/[T5G/CCP2/CLCIN0/ATCC]
*    RC4/[T3G/CLCIN1]/HIC4
*    RC5/[T3CKI/CCP1/ATIN]/HIC5
*    RC6/AN8/[!SS]
*    RC7/AN9
*
* [] rilocabili con PPS
*
* Note: - INTOSC default @ 500kHz
*       - Tutti i pin con wpu e ioc
*****************************************************************/

#include "C:\Corso_C\MyIncludes\conf1619_0.h"

#include <xc.h>

#define _XTAL_FREQ  500000
#define Up    PORTAbits,RA2
#define Down  PORTAbits.RA4
#define dp    LATCbits.LATC7

// 7 segments lookup table per cifre da 0 a F
// Rx0=segm.a, Rx1= segm.b, ...Rx7= segm.dp
  unsigned char tab7segm[16] = {
        0x3F,  // 0
        0x06,  // 1
        0x5B,  // 2
        0x4F,  // 3
        0x66,  // 4
```

```
        0x6D,  // 5
        0x7D,  // 6
        0x07,  // 7
        0x7F,  // 8
        0x6F,  // 9
        0x77,  // A
        0x7C,  // b
        0x39,  // C
        0x5E,  // d
        0x79,  // E
        0x71   // F
  };

/************************ MAIN PROGRAM ****************************/
void main(void){

 unsigned char digit=0;          // cifra da inviare al display

    // Inizializza I/O
    ANSELA = 0;                  // disabilita analogiche su
    ANSELB = 0;                  // tutti port
    ANSELC = 0;

    // protezione pin non usati
    OPTION_REGbits,nWPUEN = 0;   // abilitazione generale wpu

    // inizio loop con display spento
    LATC = 0;                    // preset latch C a 0 (segmenti spenti)
    TRISC = 0;                   // tutti i pin di PORTC come uscite

    LATC = tab7segm[digit];      // da lut a display = 0

  while(1){                      // loop principale
     if(Up==0){                  // se Up premuto, count up
      if(digit<15){              // se cifre non oltre F
       digit++;                  // incrementa digit
       LATC = tab7segm[digit];   // da lut a display
       dp=0;                     // no dp
       __delay_ms(30);           // debounce
      }else{                     // se F
        dp=1;                     // accendi dp
        }
     while(Up==0);               // attendi per Up rilasciato
        __delay_ms(30);          // debounce
     }

     if(Down==0){                // se Down premuto, count down
      if(digit>0){               // se cifre>0
       digit--;                  // decrementa digit
       LATC = tab7segm[digit];   // da lut a display
       dp=0;                     // no dp
       __delay_ms(30);           // debounce
      }else{                     // se = 0
        dp=1;                     // accendi dp
```

```
        }
      while(Down==0);              // attendi per Down rilasciato
         __delay_ms(30);           // debounce
      }
   }
}
```

Questa versione è fornita come progetto completo (MPLABX v.4.13 o successivi, XC8 v.1.34 o successivi e PIC16F1619).
I file del progetto sono previsti per stare in una cartella ***C:\Corso_C\esC\es6C_2***

Schede informative dettagliate. Esercitazione 6

Qui di seguito alcune pagine di informazioni più dettagliate sua argomenti trattati nella prima esercitazione. Che possono essere estratte e conservate separatamente.

- **Gli array**

Files allegati

Sono allegati i seguenti files/cartelle:

- **Cartella *es6C* : progetti e sorgenti**

Array e lookup table.

Gli array o matrici sono variabili che possono possono contenere un qualsiasi numero di elementi, che devono essere dello stesso tipo.
Il numero degli elementi (dimensione dell'array) deve essere specificato nella dichiarazione.
La sintassi è:

```
type arrayName[size]
```

`size` è il numero degli elementi, che deve essere una costante intera.
Il `type` si riferisce alla caratteristica degli elementi della matrice.
Gli elementi dell'array sono separati da virgole.
Ad esempio:

```
char   ciao[4];
int    xval[6];
```

Gli elementi dell'array possono essere inizializzati, assegnando loro dei valori.
Ad esempio, due precedenti array inizializzati:

```
char   ciao[4] = {'c', 'i', 'a', 'o'};
int    xval[6] = {10, 20, 30, 40, 50, 60};
```

Gli elementi dell'array sono accessibili come una variabili che sono puntate da un indice.
La sintassi è:

```
arrayName[index]
```

`index` può essere una variabile o una costante; il primo elemento dell'array, nell'ordine in cui sono scritti, ha indice 0.

Il C non fornisce alcun controllo dei limiti, quindi è cura del programmatore evitare di indirizzare elementi fuori dall'array.

```
int  i,  xval[10];  // array di dieci elementi

for(i = 0: i<10; i++) {
   xval[i] = 0;     //inizializza array
                       }
xval[5] = 66;       // imposta il sesto elemento
```

il limite `i<10` impedisce l'overflow dell'array.

E' possibile creare anche array multidimensionali. Ad esempio, una matrice bidimensionale, potrà essere:

```
int    xval[y][x];     // array a 2 dimensioni
```

dove **x** sono le colonne e **y** le righe della matrice. Possono essere inizializzate.

```
int    nval[3][3]{{0, 1, 2},
                  {3, 4, 5},
                  {6, 7 ,8}};
```

Un esempio di array:

```
/* stampa da 0 a 90 in incrementi di 10 */

int main(void){
      int i = 0;
      int a[10] = {0,1,2,3,4,5,6,7,8,9};

      while (i < 10)
      {
        a[i] *= 10;            //  a[i] = a[i] * 10
       printf("%d\n", a[i]);
         i++;
       }

     while (1);
}
```

Una **lookup table** (abbreviato in LUT) è una tabella di associazione, o tabella dati, che permette di associare ad ogni ammissibile combinazione di dati in ingresso una corrispondente configurazione di dati in uscita.
Il termine inglese (lookup) sottintende l'operazione di consultazione.

Una lookup table può contenere, ad esempio, i risultati una operazione su un dato di ingresso: immettendo il dato come puntatore, si ottiene in uscita il valore calcolato senza impegnare tempo e risorse per una operazione gravosa, come nel caso dei calcoli trigonometrici.
Oppure la tabella può contenere valori che non corrispondono ad un algoritmo calcolabile con il dato di ingresso; ad esempio, il dato di ingresso può essere una cifra e quello in uscita la maschera dei segmenti accesi/spenti da applicare ad un display per visualizzare la cifra.

Si fa ampio uso delle LUT nel campo della grafica e e dell'immagine per associare colori e tonalità ai pixel.

In C le tabelle di associazione sono degli array.