Embedded System

L6

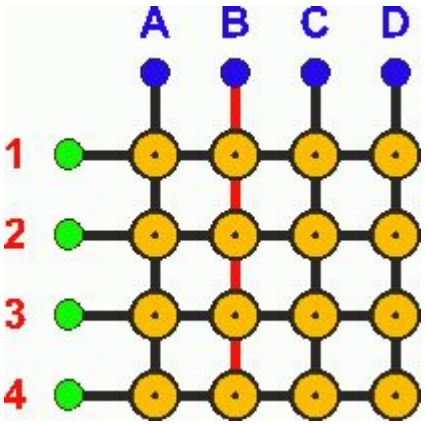
Keypad

Questi tastierini si trovano comunemente nei formati 3×4
(che è in pratica uguale a quello usato sui telefoni, ovvero numeri da 0 a 9 e
quindi asterisco e cancelletto)

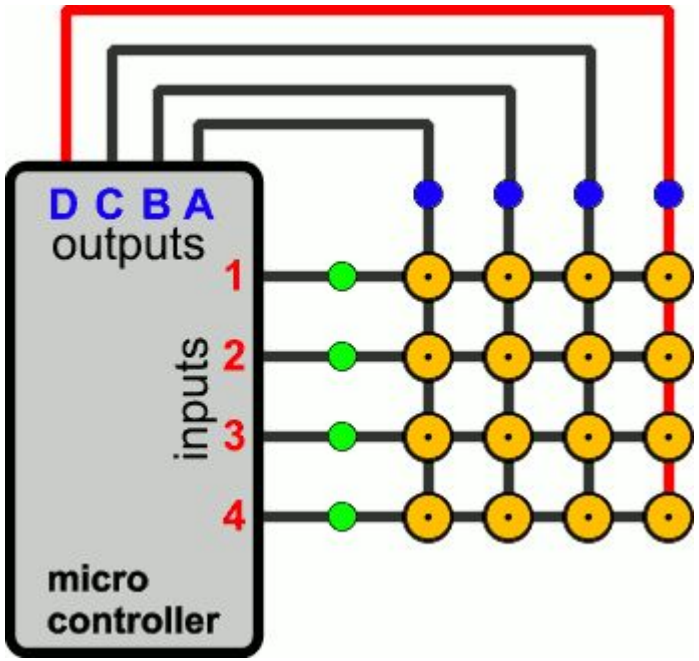
oppure come quello in foto da 4×4
(che ha in aggiunta le lettere ABCD)

Questi tastierini sono in realtà una semplice serie di pulsanti collegati in maniera un po' particolare in maniera da minimizzare i collegamenti necessari.

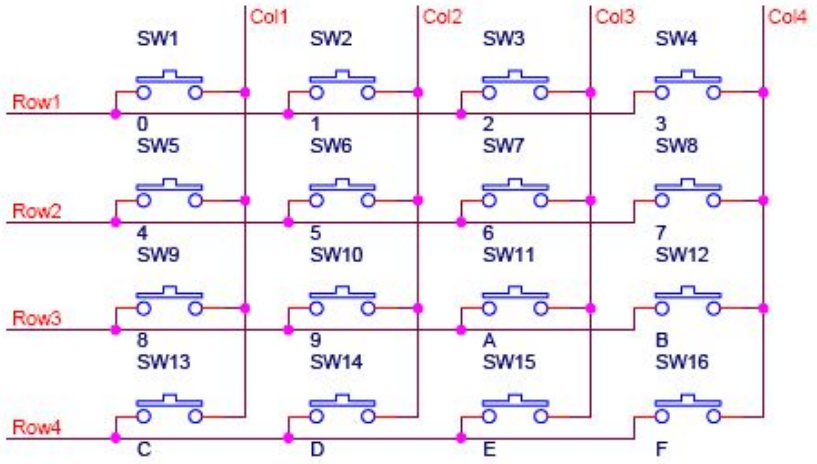
Il collegamento è a **matrice**, ovvero: sono organizzati in una serie di righe e colonne ed ogni pulsante mette in comunicazione una riga con una colonna.



Si tratta di controllare in sequenza ciascuna riga, in funzione di un particolare stato delle colonne



In pratica se premiamo il pulsante “1” (SW2)
metteremo in comunicazione
la riga 1 con la colonna 2,
premendo la “C” (SW13)
metteremo in comunicazione
la riga 4 con la colonna 1 e così via.



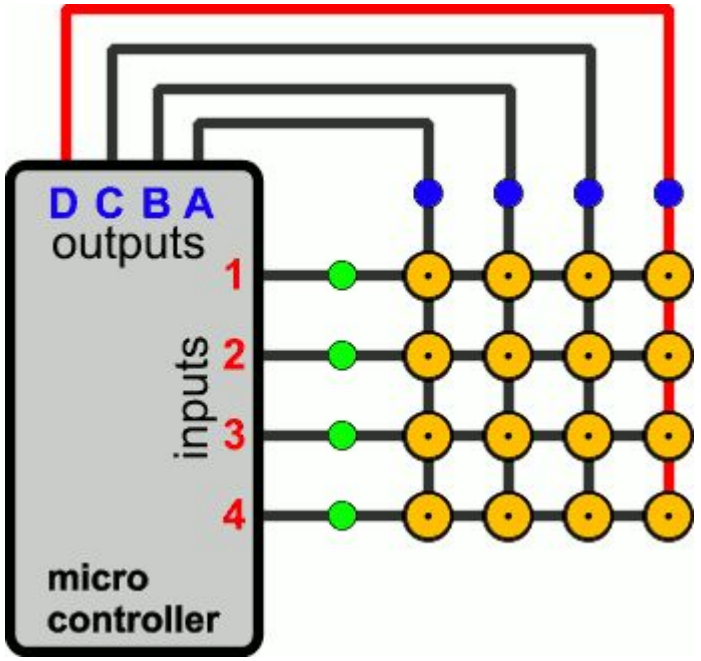
Esistono diversi modi per acquisire le informazioni da un tastierino di questo tipo, ci sono addirittura sistemi che permettono il riconoscimento del pulsante sfruttano un solo ingresso del micro

Noi invece vedremo un sistema più “digitale” che prevede di collegare le quattro righe e le quattro colonne direttamente al micro occupando quindi in totale 8 I/O.

Il concetto che sta alla base del funzionamento è molto semplice.
Basta seguire questo schema mentale:

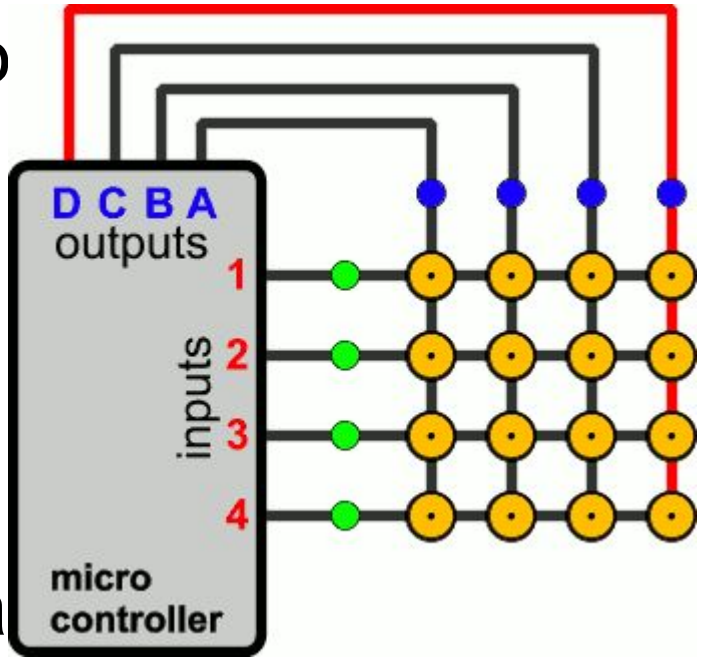
- Le 4 righe sono collegate a 4 pin predisposti come ingressi, questi ingressi hanno una resistenza di pullup (esterna o integrata) che permette loro di trovarsi a livello logico alto in condizioni di riposo.

- Le 4 colonne sono collegate a 4 pin predisposti come uscite.



- Partiamo da una situazione in cui le 4 uscite si trovano a livello lo 1.
- Poniamo quindi, una alla volta, colonne a livello logico basso.
- Le 4 colonne seguiranno in pra questa sequenza di stati logici:
- 1111 >0111 > 1011 > 1101 > 1110 per poi ricominciare da capo.

Lo Zero, è il filo che si illumina di rosso

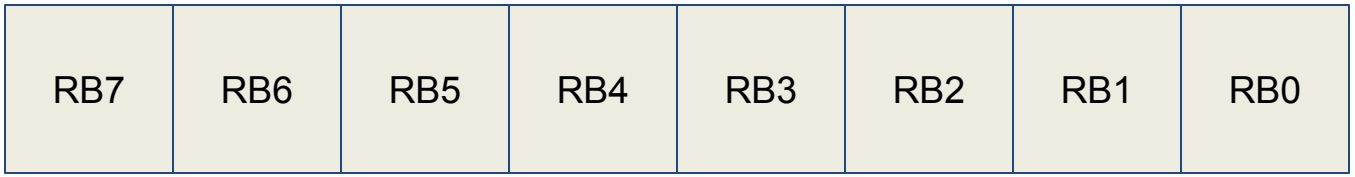


- Subito dopo aver posto una colonna a zero, scansioneremo una alla volta le 4 righe per controllare se una di loro si trova a livello logico basso indicando che un pulsante è stato premuto.

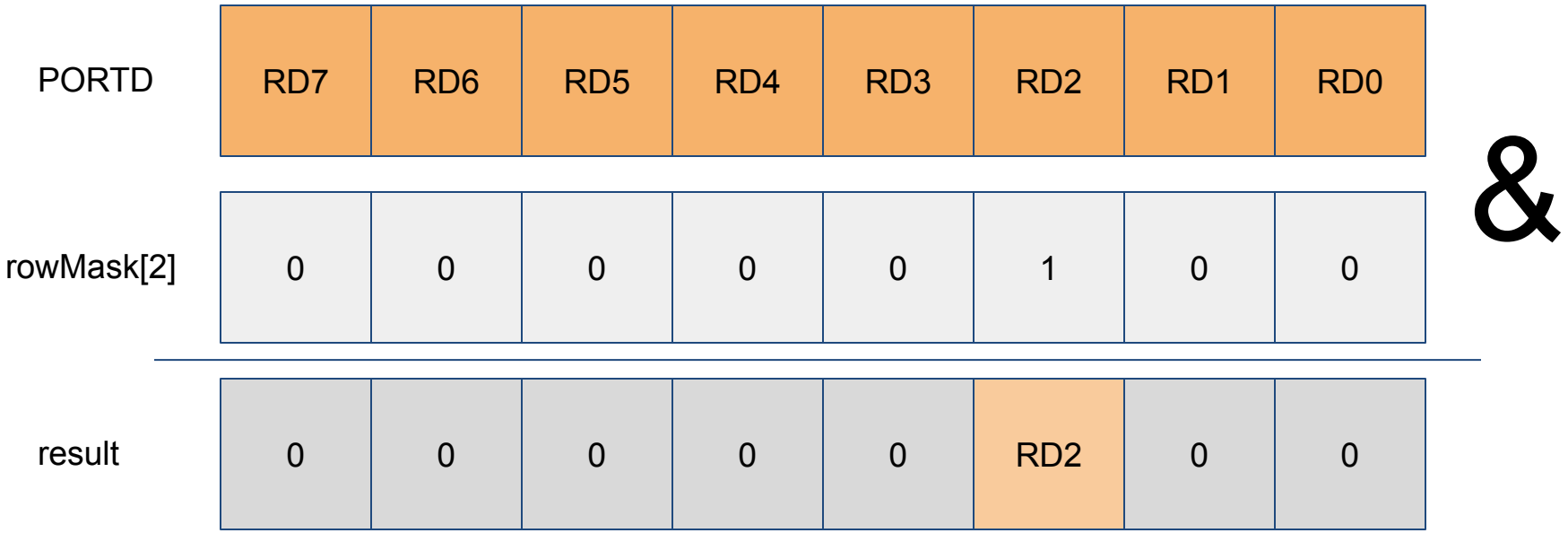
- Confrontando quale colonna e quale riga si trovano a livello logico basso, è possibile risalire al pulsante premuto.

Iniziamo con la preparazione delle maschere

PORTB




```
const unsigned char colMask[3]=
{
   //76543210 posizione del bit
   0b11111110, // Colonna 1 => RB0 a massa
   0b11111101, // Colonna 2 => RB1 a massa
   0b11111011  // Colonna 3 => RB2 a massa
};
```




```
const unsigned char rowMask[4]=
{
   0b00000001, // Riga 1
   0b00000010, // Riga 2
   0b00000100, // Riga 3
   0b00001000  // Riga 4
};
```

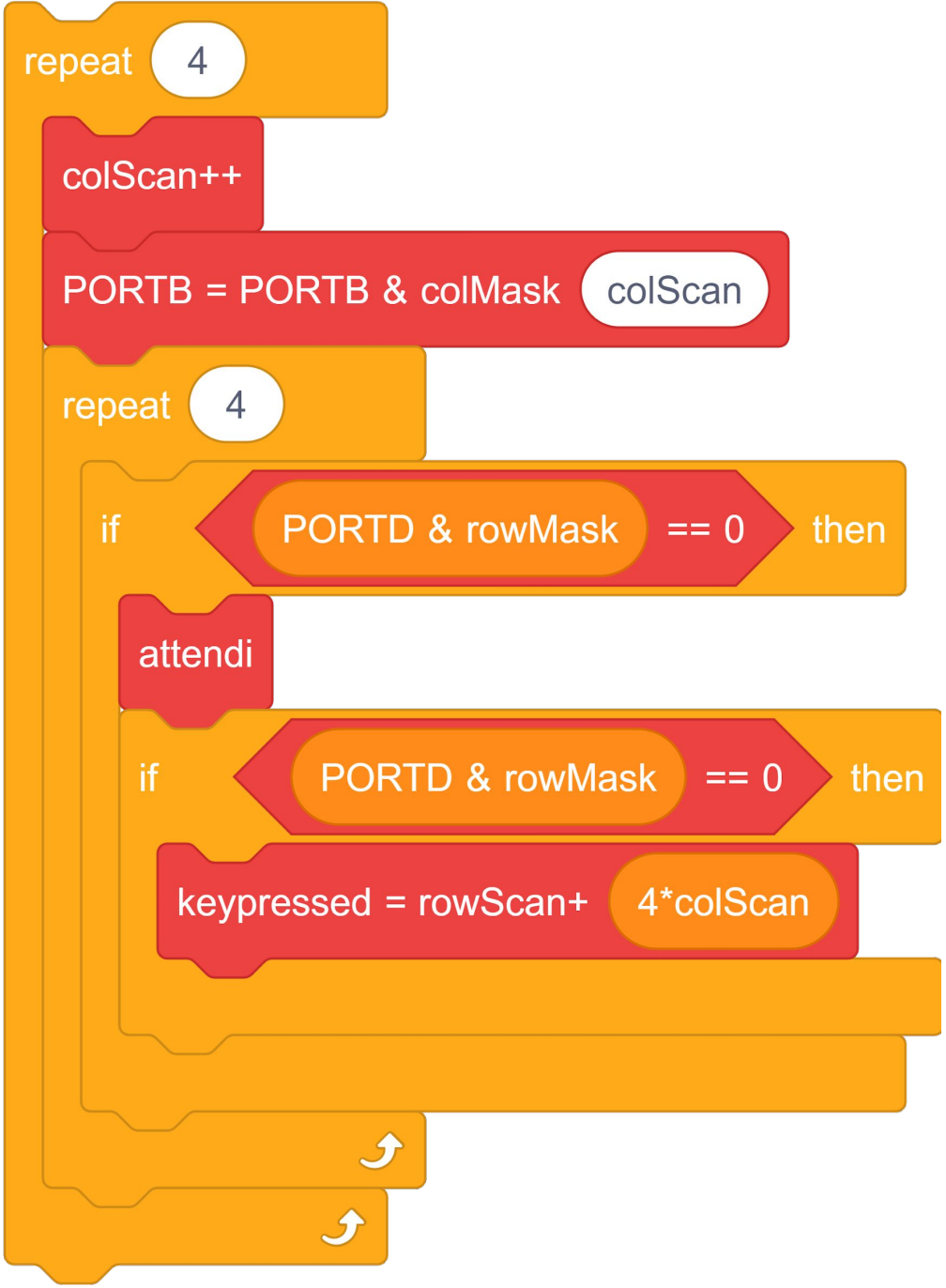
```
unsigned char colScan=0;

unsigned char rowScan=0;
```

```
const unsigned char keys[] =
       {/* inserire i valori corrispondenti */};
        { 1, 4, 7, '*',    };

// peso numerico del pulsante premuto
unsigned char keypressed=0;

char keyok; // flag di pulsante premuto
```


repeat
4
colScan++
PORTB = PORTB & colMask
colScan
repeat
4
if
PORTD & rowMask
== 0
then
attendi
if
PORTD & rowMask
== 0
then
keypressed = rowScan+
4*colScan

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
   // qui metto il codice che controlla PORTD









}
```

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
   PORTB = PORTB | 0x07; // porto tutte le colonne a 1
   PORTB = PORTB & colMask[colScan]; // porto a zero la colonna attuale

   // qui metto il codice che controlla PORTD









}
```

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
   PORTB = PORTB | 0x07; // porto tutte le colonne a 1
   PORTB &= colMask[colScan]; // porto a zero la colonna attuale

   for (rowScan=0; rowScan<4; rowScan++)
   {
      // controllo ogni singolo bit di PORTD









   }
}
```

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
   PORTB = PORTB | 0x07; // porto tutte le colonne a 1
   PORTB &= colMask[colScan]; // porto a zero la colonna attuale

   for (rowScan=0; rowScan<4; rowScan++)
   {
      if (!(PORTD & rowMask[rowScan])) // Riga rowScan trovata a massa
      {
         __delay_ms(5);
         // versione con delay, si può migliorare
         // rendendolo sensibile al fronte

         if (!(PORTD & rowMask[rowScan]))
         {
            // calcolo quale pulsante è stato premuto
         }
      }
   }
}
```

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
   PORTB = PORTB | 0x07; // porto tutte le colonne a 1
   PORTB &= colMask[colScan]; // porto a zero la colonna attuale

   for (rowScan=0; rowScan<4; rowScan++)
   {
      if (!(PORTD & rowMask[rowScan])) // Riga rowScan trovata a massa
      {
         __delay_ms(5);
         // versione con delay, si può migliorare
         // rendendolo sensibile al fronte

         if (!(PORTD & rowMask[rowScan]))
         {
            keypressed=rowScan+(4*colScan); // numero di pulsante premuto
            keyok=1; // E' stato premuto un pulsante
         }
      }
   }
}
```

```
// porto a massa una colonna alla volta
for (colScan = 0; colScan < 3; colScan++)
{
    PORTB = PORTB | 0x07; // porto tutte le colonne a 1
    PORTB &= colMask[colScan]; // porto a zero la colonna attuale

    for (rowScan=0; rowScan<4; rowScan++)
    {
        if (!(PORTD & rowMask[rowScan])) // Riga rowScan trovata a massa
        {
            __delay_ms(5);
            // versione con delay, si può migliorare
            //  rendendolo sensibile al fronte

            if (!(PORTD & rowMask[rowScan]))
            {
                keypressed=rowScan+(4*colScan);// numero di pulsante premuto
                keyok=1; // E' stato premuto un pulsante
            }
        }
    }

    if (keyok)
    {
        // se ho premuto un pulsante...
    }
}
```

```
if (keyok)
{
    // vorrei rappresentare il tasto premuto sui 4 bit
    // più significativi di PORTD

    PORTD = PORTD & 0x0F;
    PORTD = PORTD | (keys[keypressed] << 4);

    keyok=0; // resetto il flag di pulsante premuto

    // rimango in un ciclo continuo fino a che
    // il pulsante non viene rilasciato

    PORTB=PORTB | 0x07;
    while((PORTB & 0x07) != 0x07)
    {
        continue;
    }
}
```

Utilizzare la scheda numero 4, utilizzando un PIC16F877A.

La scheda così configurata, usa i tre bit meno significativi di PORTB e i quattro meno significativi di PORTD per il collegamento del Keypad.

Si Scriva un codice che legga il tastierino numerico e visualizzi il numero corrispondente sui quattro bit più significativi di PORTD.
Il numero verrà rappresentato in forma binaria.

Si Scriva un codice che legga il tastierino numerico e visualizzi il numero corrispondente sul Display a 7 segmenti (Disp4)

Si scriva un codice che simuli una semplice calcolatrice, nei limiti della scheda e delle cose viste.

La calcolatrice in questione deve permettere di effettuare solo le somme.

Il risultato deve essere rappresentato sfruttando, per le decine, la rappresentazione binaria sui 4 bit più significativi di PORTD, e per le unità, i 4 bit più significativi di PORTB. L'eventuale overflow verrà segnalato tramite l'unico led rimasto libero (RB3)

4 + 9 = 1 3