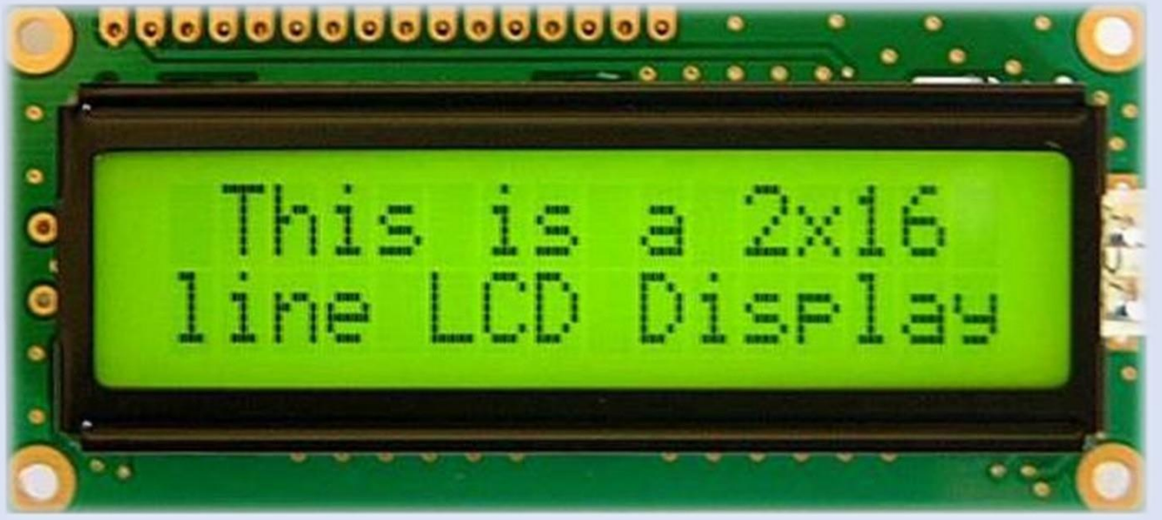
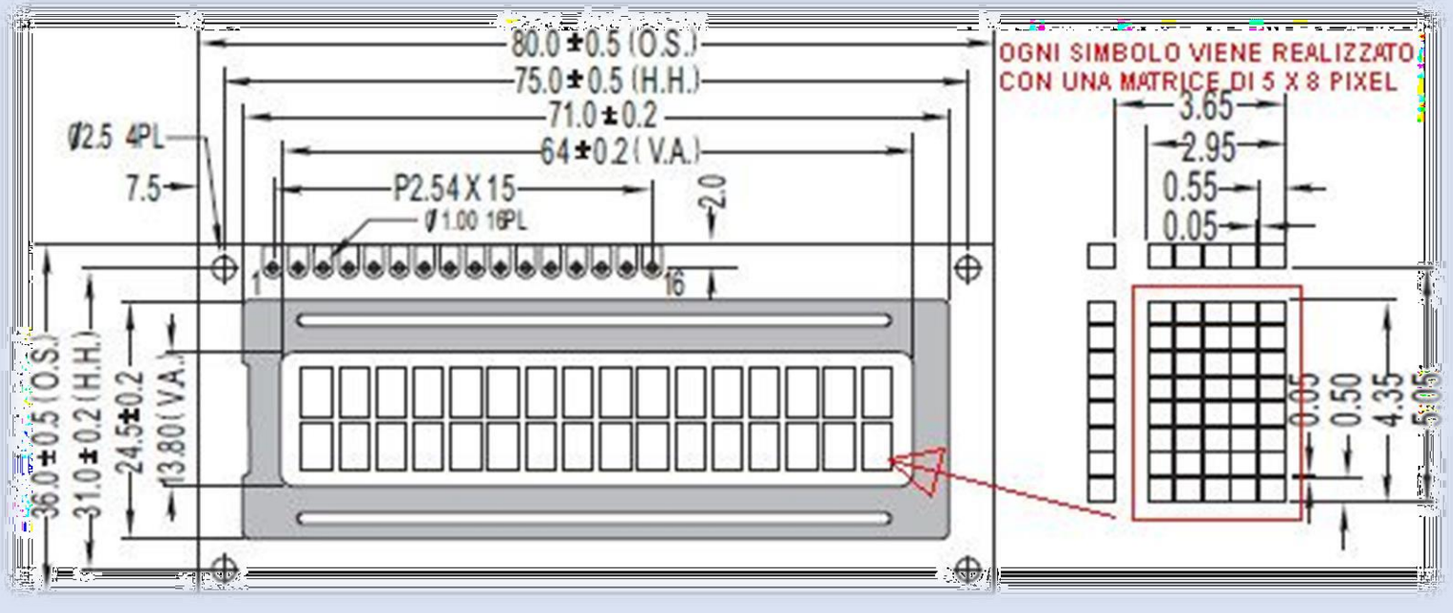
# LCD ALfanumerico

- *Il display **LCD alfanumerico (Dot matrix)** è un dispositivo che consente di visualizzare su una matrice di righe e colonne uno o più caratteri contenenti lettere, numeri e simboli.*



# LCD ALfanumerico

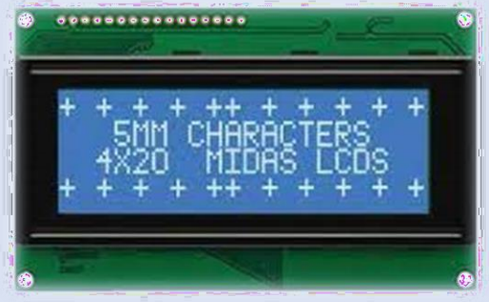
- *Ogni carattere è composto da una matrice di punti (dot matrix) nei tipi più comuni di display ogni carattere è compost da una matrice di **5x8 pixel**. Esistono all'interno dei vari ambienti di programmazione  delle funzioni per creare e personalizzare simboli diversi da quelli standard.*

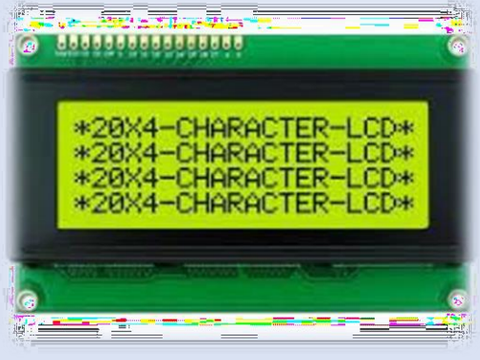


# LCD ALfanumerico

- *In commercio troviamo display con massimo **4 righe** ed al massimo **20 colonne**, anche con caratteri di diverse dimensioni. Quasi tutti i display sono retroilluminati, in questo modo è possible anche ottenere colorazioni differenti.*



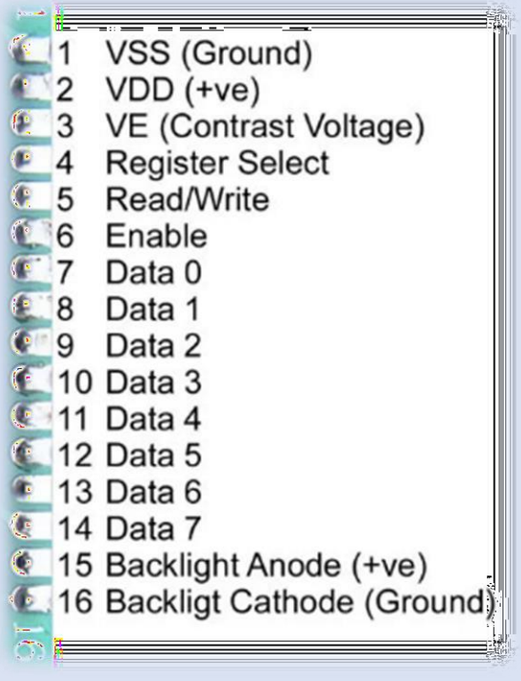


# LCD ALfanumerico

- *Al bordo di questi display è presente un **CONTROLLER** cioè un circuito integrato che gestisce la parte fisica del display cioé l'**accensione** e spegnimento dei vari pixel, con la gestione di un'**interfaccia** parallela per il collegamento con microprocessori e microcontrollori. Il controllore più diffuso ormai divenuto uno standard, è il circuito Asic della **HITACHI HD44780** che prevede un bus di collegamento parallelo compost dai seguenti pin.*

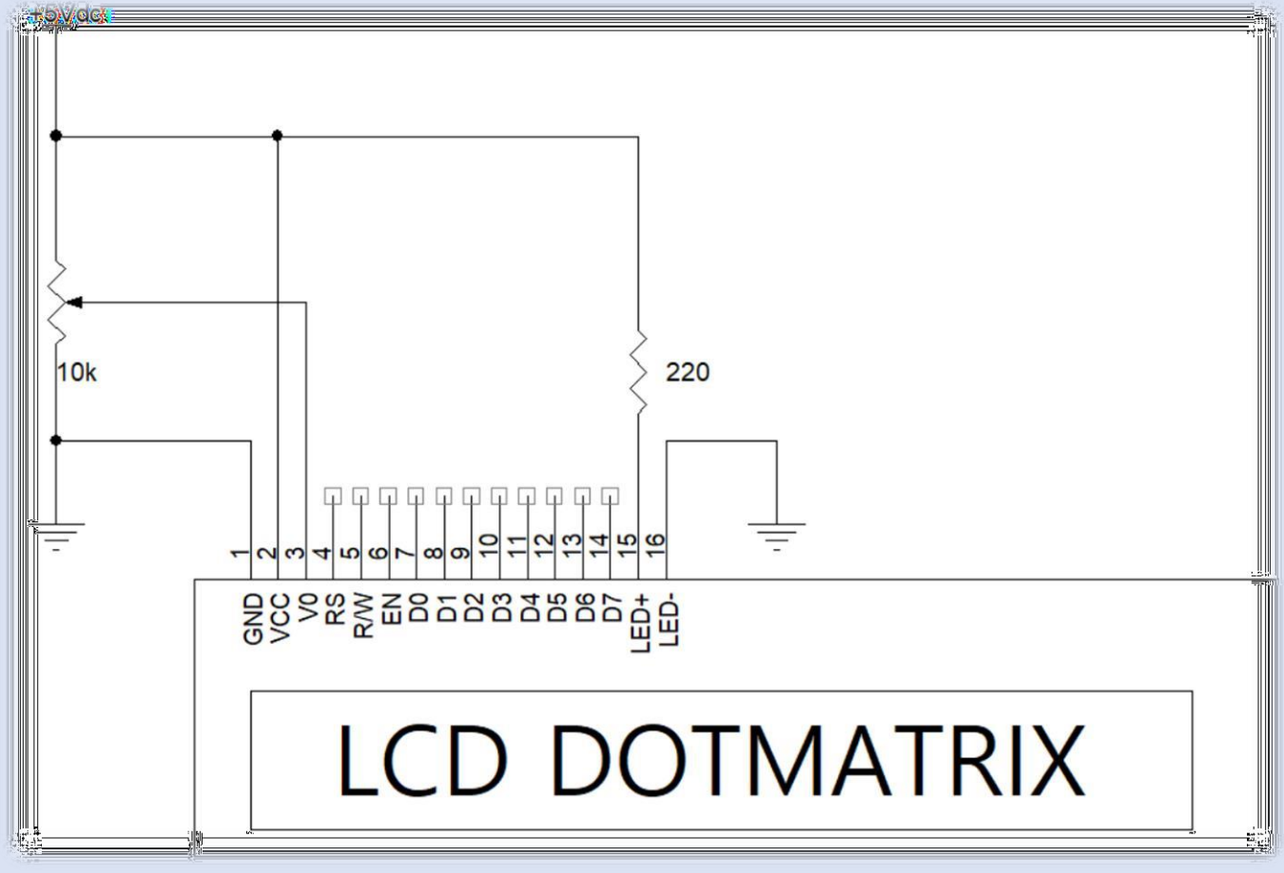
| PIEDINO | NOME | DESCRIZIONE |
|---|---|---|
| 1 | VSS | Riferimento 0Volt della tensione di alimentazione, GND |
| 2 | VDD | Tensione di alimentazione positiva +5Vdc |
| 3 | VO | Tensione variabile 0-5Vdc per regolazione contrast |
| 4 | RS | Segnale di controllo per selezione registro (Register Select) |
| 5 | R/W | Segnale di controllo per scelta lettura (Read) o scrittura (Write) |
| 6 | EN | Segnale di controllo per abilitazione display (ENable) |
| 7 | D0 | Segnale bus dati D0 (LSB bit meno significativo) |
| 8 | D1 | Segnale bus dati D1 |
| 9 | D2 | Segnale bus dati D2 |
| 10 | D3 | Segnale bus dati D3 |
| 11 | D4 | Segnale bus dati D4 |
| 12 | D5 | Segnale bus dati D5 |
| 13 | D6 | Segnale bus dati D6 |
| 14 | D7 | Segnale bus dati D7 |
| 15 | A | Anodo del led per retroilluminazione, va collegata una tensione positiva +5V |
| 16 | K | Catodo del led per retroilluminazione, va collegato il riferimento GND |



# LCD ALfanumerico

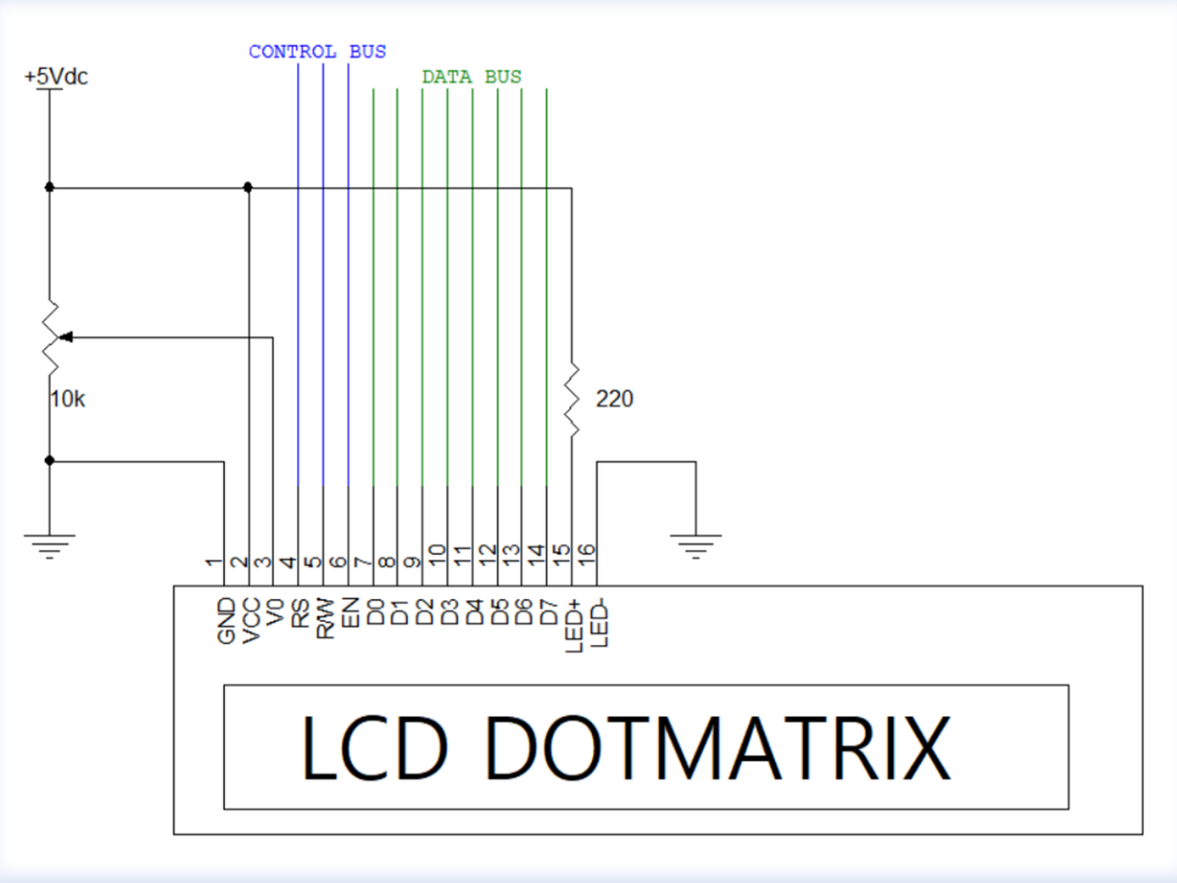
- *Il display va alimentato tra il pin 1 e 2 e sul pin 3 (V0) va collegata una tensione variabile per regolare il contrasto.*

- *Sui pin 15 e 16 va inserita una resistenza per controllare la corrente del LED della retroilluminazione.*



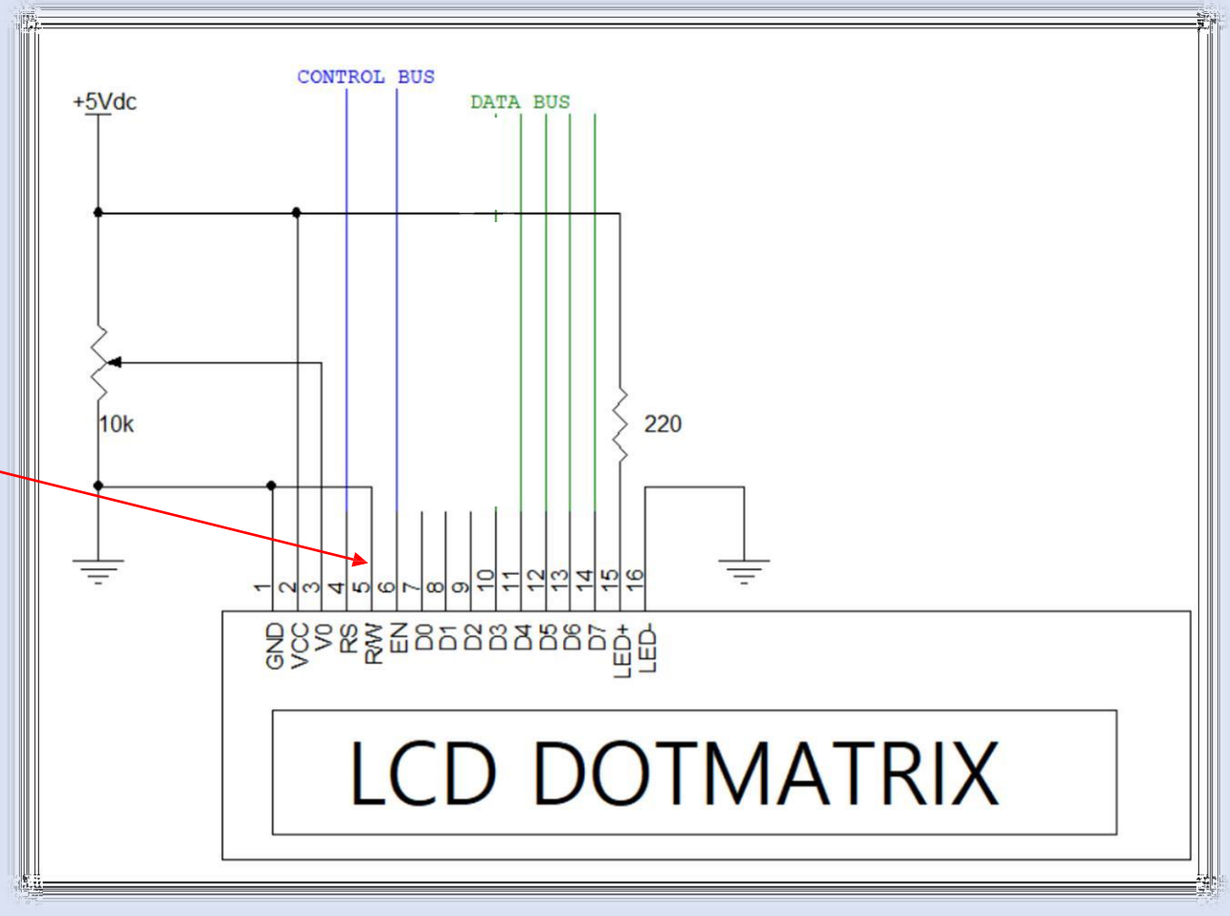
# LCD ALfanumerico

- *IL BUS parallelo è composto da un BUS DATI (D0…D7) ad 8 bit e da un BUS di CONTROLLO a 3 bit.*
- *Sul BUS di controllo troviamo 3 segnali RS, R/W e EN.*
  - *RS definisce l'invio di un dato o di un comando.*
  - *R/W definisce se dobbiamo scrivere o leggere sulla memoria del display.*
  - *EN abilita il display.*



# LCD ALfanumerico

- *Questi segnali vengono gestiti dalle librerie dei vari programmi, e nella maggior parte dei casi queste librerie utilizzano il minor numero di collegamenti possibili, escludendo la gestione del segnale R/W (lasciato fisso a 0Volt cioè scrittura sul display) ed escludendo la gestione dei segnali D0,D1,D2 e D3, in quanto il controller consente l'invio dei dati anche sui restanti 4 bit.* ***Pertanto sono sufficienti le 6 linee indicate: RS, EN, D4, D5, D6 e D7.***



# LCD ALfanumerico

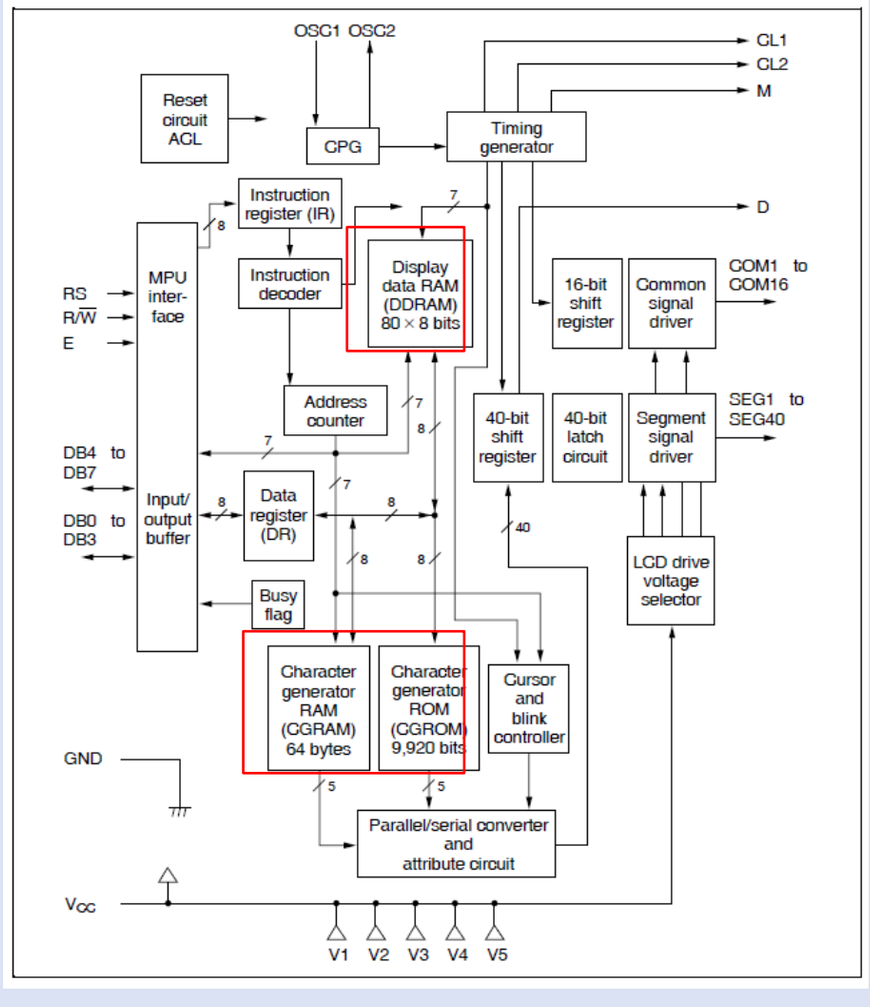
*Nell'immagine a destra troviamo lo schema a blocchi interno del controller HD44780, per scendere nel dettaglio occorre leggere Il manuale completo al seguente link: [Datasheet HD44780]*

*Cerchiamo comunque di comprendere sinteticamente il funzionamento interno delle tre memorie principali la CGROM, la CGRAM e la DDRAM.*

*La CGROM è una memoria a sola lettura e di fatto contiene tutti i caratteri visualizzabili sul display.*

*La CGRAM contiene gli stessi caratteri della CGROM con l'aggiunta dei caratteri personalizzabili.*

*La DDRAM invece è una memoria da 80 byte che corrisponde alle 80 posizioni possibili sul display (dimensioni massime display 4x20 o 2x40 perciò 80 caratteri).*



# LCD ALfanumerico

*Nell'immagine a destra invece possiamo vedere il codice da inviare sui due bus, per realizzare la relativa funzione descritta.*

*Ad esempio per scrivere nella DDRAM si utilizza il bit D7 ad 1, ed i restanti 7 bit identificano l'indirizzo della memoria. Questa memoria rappresenta la posizione del carattere che scriveremo sul display, secondo la seguente struttura.*

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 |
| 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |

*Dovendo mettere il bit 7 ad 1, l'indirizzo da scrivere sul databus in realtà è il seguente.*

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 |
| C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 | D2 | D3 |
| 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 |
| D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 |

| Instruction | Code RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Description | Execution Time (max) (when $f_{cp}$ or $f_{OSC}$ is 270 kHz) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clear display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Clears entire display and sets DDRAM address 0 in address counter. | |
| Return home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | — | Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged. | 1.52 ms |
| Entry mode set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | Sets cursor move direction and specifies display shift. These operations are performed during data write and read. | 37 µs |
| Display on/off control | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B). | 37 µs |
| Cursor or display shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | — | — | Moves cursor and shifts display without changing DDRAM contents. | 37 µs |
| Function set | 0 | 0 | 0 | 0 | 1 | DL | N | F | — | — | Sets interface data length (DL), number of display lines (N), and character font (F). | 37 µs |
| Set CGRAM address | 0 | 0 | 0 | 1 | ACG | ACG | ACG | ACG | ACG | ACG | Sets CGRAM address. CGRAM data is sent and received after this setting. | 37 µs |
| Set DDRAM address | 0 | 0 | 1 | ADD | ADD | ADD | ADD | ADD | ADD | ADD | Sets DDRAM address. DDRAM data is sent and received after this setting. | 37 µs |
| Read busy flag & address | 0 | 1 | BF | AC | AC | AC | AC | AC | AC | AC | Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents. | 0 µs |
| Write data to CG or DDRAM | 1 | 0 | Write data | | | | | | | | Writes data into DDRAM or CGRAM. | 37 µs<br>$t_{ADD}$ = 4 µs* |
| Read data from CG or DDRAM | 1 | 1 | Read data | | | | | | | | Reads data from DDRAM or CGRAM. | 37 µs<br>$t_{ADD}$ = 4 µs* |

# LCD ALfanumerico

*Il valore scritto nella memoria DDRAM, rappresenta il codice ASCII del simbolo da visualizzare che però è anche l'indirizzo della CGRAM che conterrà una sequenza di 40 bit (i caratteri sono 5x8) che consente la visualizzazione del carattere scelto.*

*A destra vediamo ad esempio che in corrispondenza dell'indirizzo CGRAM pari a 30 in esadecimale, sono contenuti 40 bit che rappresentano il simbolo 0.*

*Ci sono inoltre 8 caratteri (5x8 pixel) personalizzabili nella CGRAM.*

*A meno che non si voglia gestire il display in assembly o senza librerie, non è indispensabile conoscere altre informazioni, in quanto il lavoro verrà svolto dalle librerie che vedremo di seguito.*

*Comunque sia nel datasheet menzionato prima, si trovano tutte le informazioni particolareggiate.*

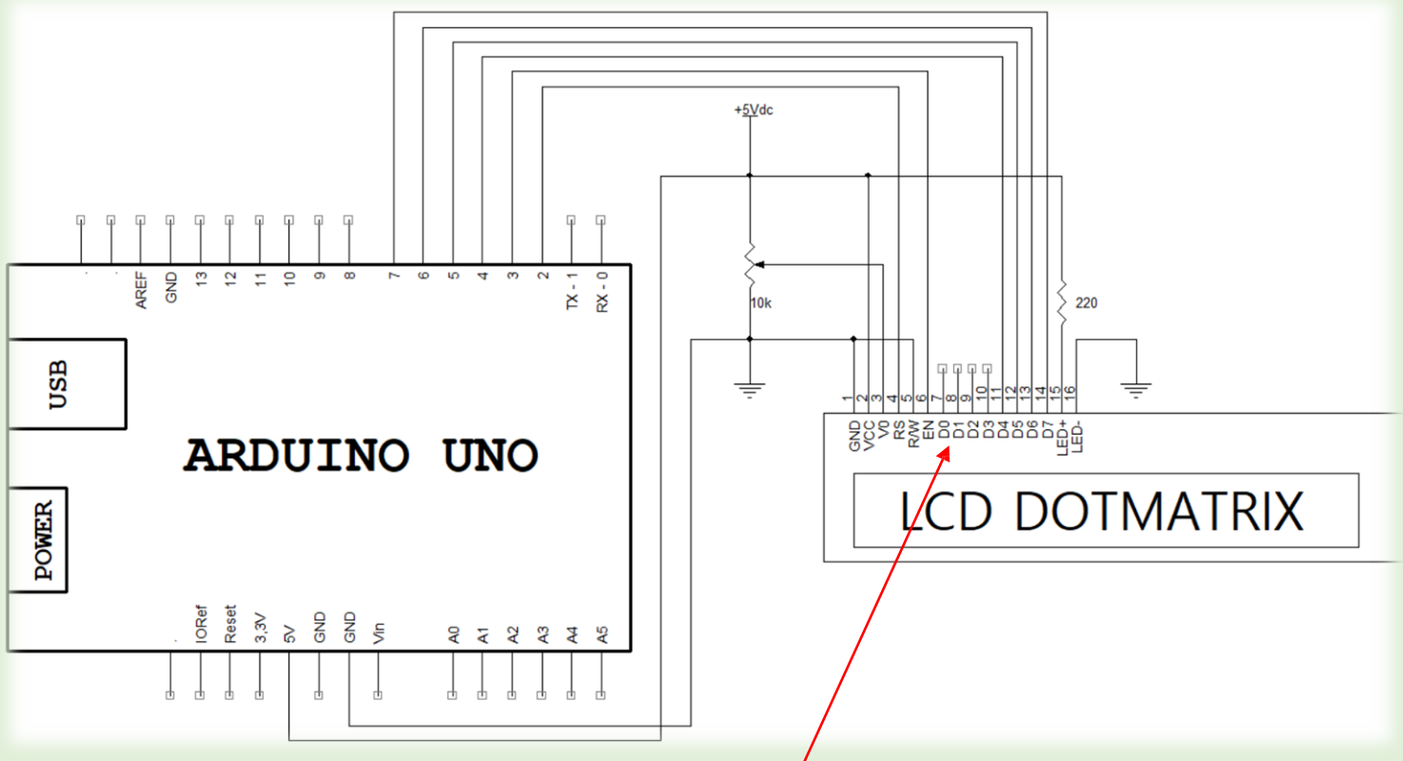
# LCD + ARDUINO

*Vediamo ora come controllare un display LCD con la scheda Arduino Uno.*

*Le librerie sono le stesse anche per le altre schede del progetto Arduino ed in ogni caso è previsto il collegamento ridotto con 6 terminali come indicato precedentemente.*

*Nello schema abbiamo il seguente collegamento:*

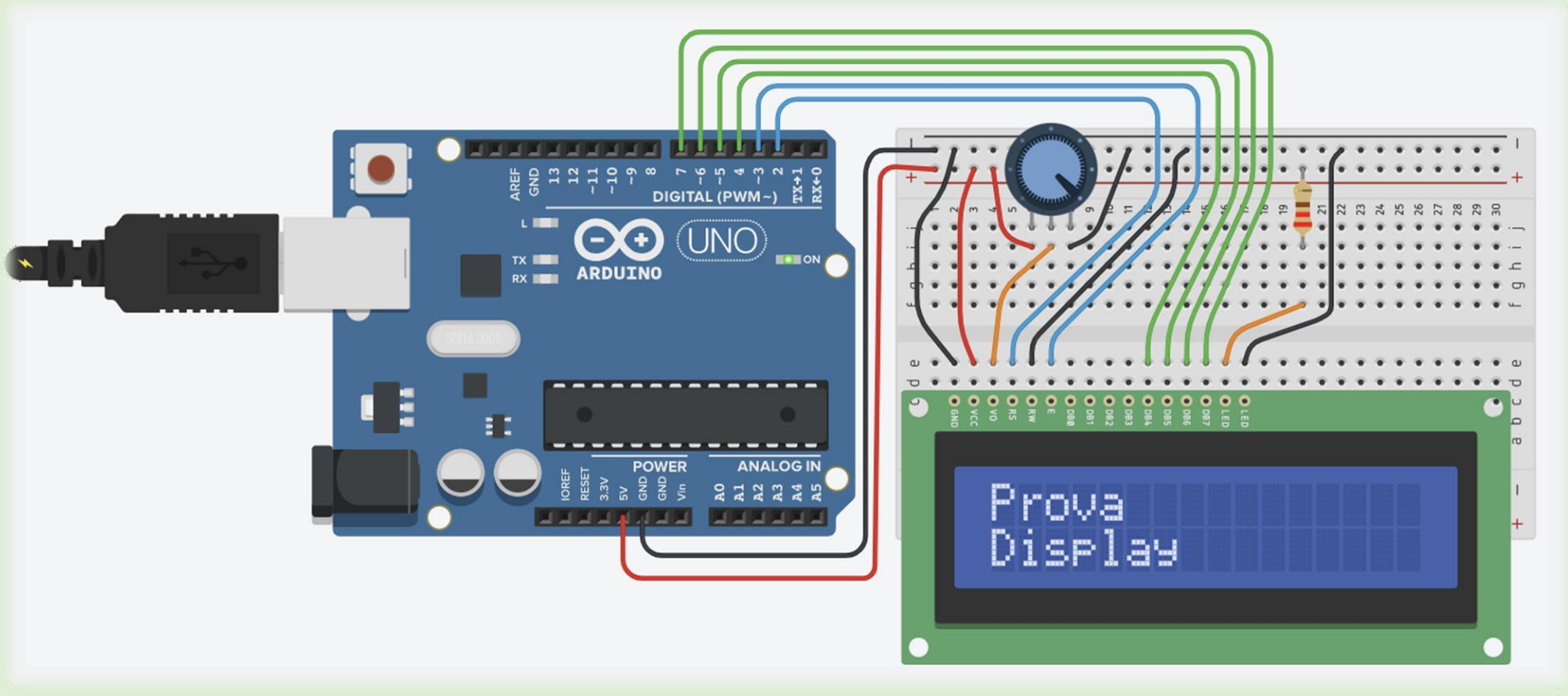
| | | |
|---|---|---|
| *RS* | - | *pin 2* |
| *EN* | - | *pin 3* |
| *D4* | - | *pin 4* |
| *D5* | - | *pin 5* |
| *D6* | - | *pin 6* |
| *D7* | - | *pin 7* |




***I pin del display D0,D1,D2 e D3 possono essere lasciati scollegati o collegati a 0Volt.***

# LCD + ARDUINO

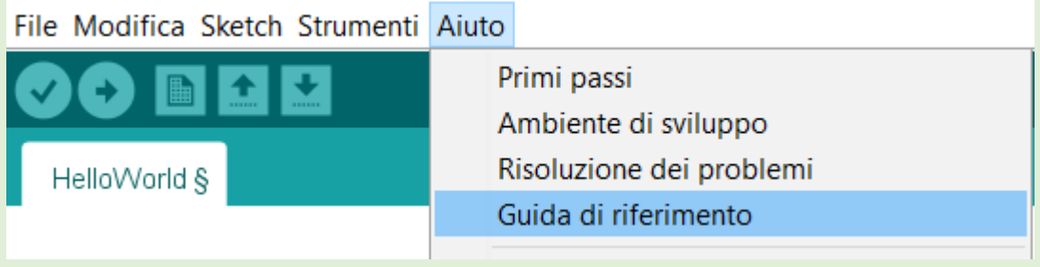
*Il collegamento precedente può essere effettuato nel seguente modo.*

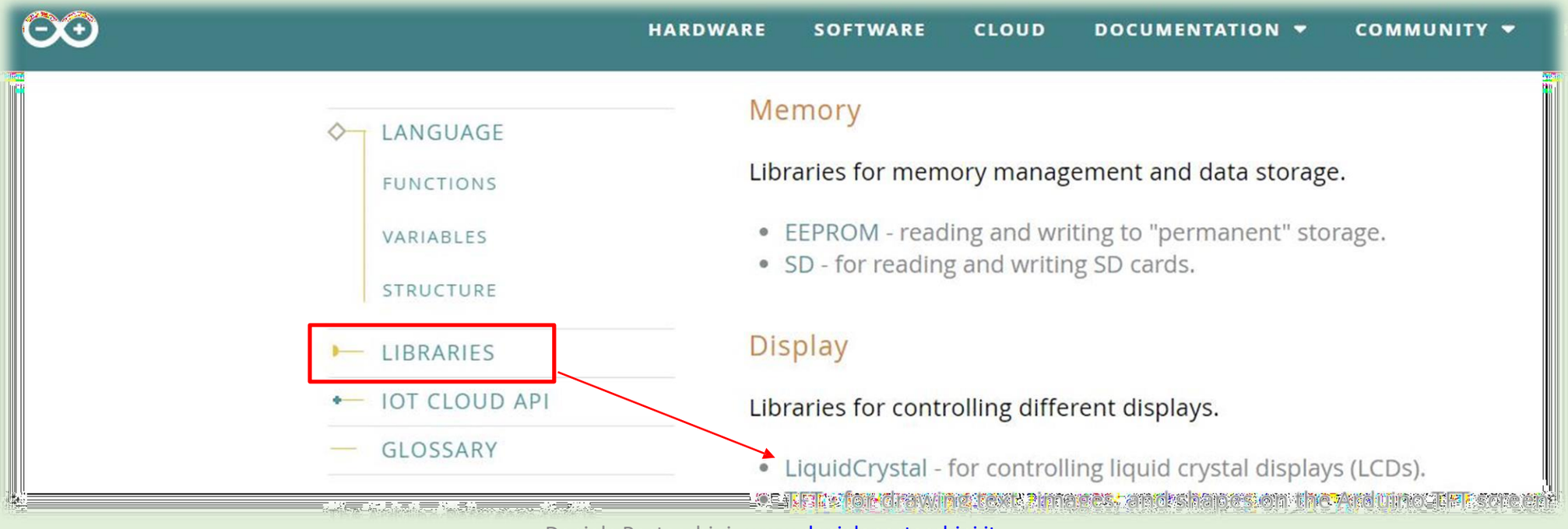


# LCD + ARDUINO

*Andiamo ora ad analizzare il funzionamento della libreria disponibile sull'I.D.E. di Arduino.*

*Andando sulla guida di riferimento, sotto la sezione LIBRARIES, troviamo la libreria LIQUIDCRYSTAL.*



# LCD + ARDUINO

*La libreria è la **LyquidCrystal.h**, e le funzioni sono quelle descritte di fianco, sempre nella guida, cliccando sulla funzione, vengono date le informazioni essenziali per il suo utilizzo.*


***LiquidCrystal()*** *crea la variabile appartenente alla classe **LiquidCrystal** definendo i pin a cui è connesso il display, si possono definire i pin per il collegamento a 4 o 8 bit del databus, ed a 2 o 3 bit del controlbus, pertanto le opzioni possibili sono:*

**LiquidCrystal nome(RS,EN,D4,D5,D6,D7);**

LiquidCrystal nome(RS,WR,EN,D4,D5,D6,D7);

LiquidCrystal nome(RS,EN,D0,D1,D2,D3,D4,D5,D6,D7);

LiquidCrystal nome(RS,WR,EN,D0,D1,D2,D3,D4,D5,D6,D7);

*Volendo dichiarare la variabile di tipo LiquidCrystal per il display collegato con 6 linee, dovremo scrivere quanto segue all'inizio del programma.*

```
#include <LiquidCrystal.h>
LiquidCrystal miolcd(2,3,4,5,6,7);
```

*Il nostro oggetto in questo modo si chiamerà **miolcd**. E' possibile dichiarare un'altra variabile per la gestione di un secondo display collegato ad altri pin, ma con un nome differente.*

# LCD + ARDUINO

*Dopo aver dichiarato la variabile all'interno del Setup, va inizializzato il display con il metodo begin().*

***begin()*** *inizializza e definisce le righe e le colonne del display, all'interno delle parentesi vanno indicate prima le colonne e poi le righe. Ad esempio scrivendo* `miolcd.begin(16,2);` *si definisce un display di 16 colonne e 2 righe.*

***clear()*** *cancella il contenuto del display, es.* `miolcd.clear();`

***cursor() e noCursor()*** *visualizza il cursore o non lo visualizza, es.* `miolcd.cursor();`

***blink() e noblink()*** *visualizza il cursore lampeggiante o no, es.* `miolcd.blink();`

***display() e noDisplay()*** *accende e spegne il display, es.* `miolcd.noDisplay();`

***home()*** *posiziona il cursore alla prima riga e prima colonna, es.* `miolcd.home();`

# LCD + ARDUINO

***scrollDisplayLeft() e scrollDisplayRight()*** *sposta di una posizione a sinistra o a destra il contenuto dell'intero display insieme al cursore.*

*Es.*

```
miolcd.scrollDisplayLeft();
```

***leftToRight() e rightToLeft()*** *di default è attiva la funzione **leftToRight(),** pertanto ad ogni scrittura, il cursore si sposta a destra, attivando **rightToLeft()** invece la scrittura avverrà da destra a sinistra.*

***autoscroll() e noAutoscroll()*** *con **autoscroll()** il display scorre automaticamente a destra o a sinistra dopo la scrittura, in base all'impostazione precedente. **noAutoscroll()** disattiva lo scorrimento automatico del display.*

```
#include <LiquidCrystal.h>

Functions
  LiquidCrystal()
  begin()
  clear()
  home()
  setCursor()
  write()
  print()
  cursor()
  noCursor()
  blink()
  noBlink()
  display()
  noDisplay()
  scrollDisplayLeft()
  scrollDisplayRight()
  autoscroll()
  noAutoscroll()
  leftToRight()
  rightToLeft()
  createChar()
```

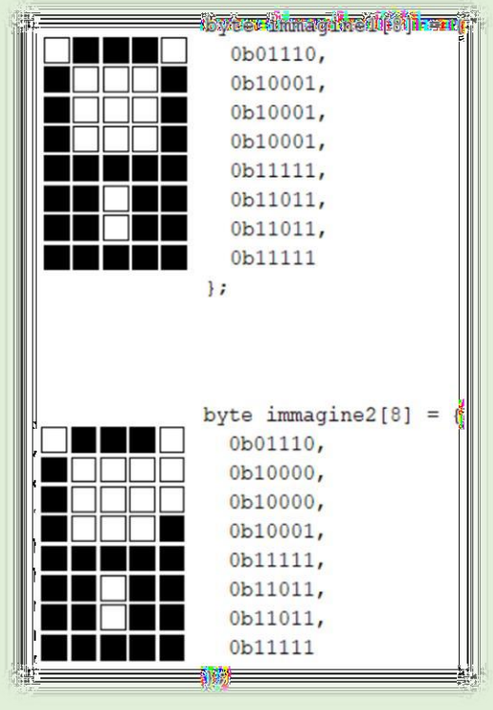
# LCD + ARDUINO

***createChar()*** *con questa funzione si può personalizzare il carattere da visualizzare.*

*Si possono creare fino a 7 immagini definite con degli array di 8 righe e 5 colonne.*

*Ad ogni array va associato un numero da 0 a 7.*

*Con la funzione **write()** si richiama l'array scelto.*




*A destra un programma di esempio con la dichiarazione di due array che rappresentano i caratteri identificati con 0 ed 1.*

```
byte immagine1[8] = {
  0b01110,
  0b10001,
  0b10001,
  0b10001,
  0b10001,
  0b11111,
  0b11011,
  0b11011,
  0b11111
};

byte immagine2[8] = {
  0b01110,
  0b10000,
  0b10000,
  0b10000,
  0b11111,
  0b11011,
  0b11011,
  0b11111
};

void setup(){
  miolcd.createChar(0,immagine1);
  miolcd.createChar(1,immagine2);
  miolcd.begin(16, 2);
  miolcd.write(byte(0));
}
```

*Ovviamente tutte le funzioni viste, possono essere utilizzate solo dopo aver creato la variabile di tipo LiquidCrystal come visto inizialmente.*

# LCD + ARDUINO

***setCursor()*** *posiziona il cursore alla colonna e riga indicata, le righe e le colonne partono dal numero 0 ed all'interno delle parentesi vanno indicate prima le colonne e poi le righe. Perciò per posizionare la prima colonna e la seconda riga dovremo scrivere:*

```
miolcd.setCursor(0,1);
```

***print()*** *scrive nella posizione corrente una stringa tra doppi apici, o un carattere tra singoli o doppi apici, o il contenuto di una variabile.*

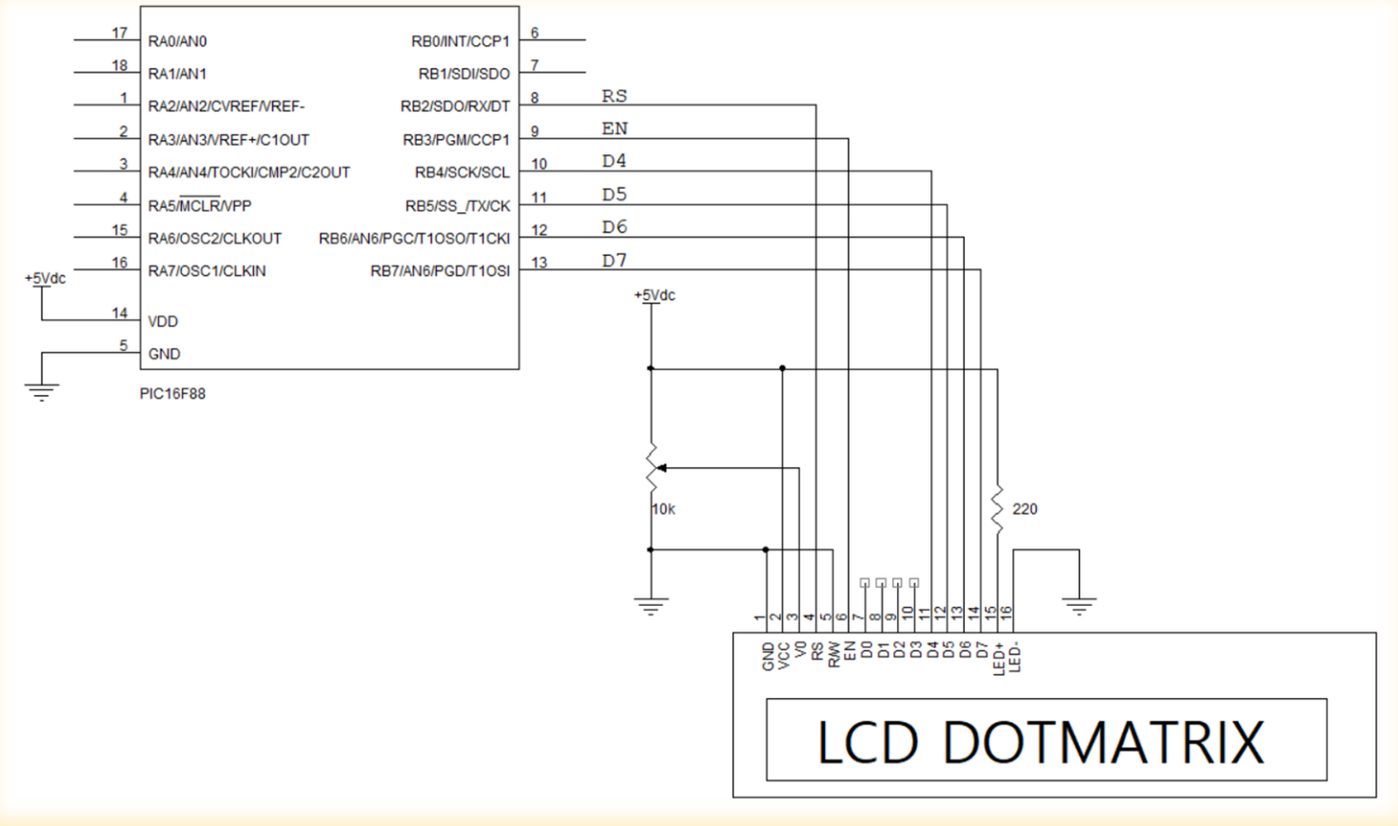
***write()*** *invia al display il valore numerico di un byte.*

```
miolcd.print("Ciao"); //scrive la stringa Ciao
miolcd.print("A");     //scrive il carattere A
miolcd.print('A');     //scrive il carattere A

int cont=10;
miolcd.print(cont);    //scrive il valore 10

cont=65;               //codice ASCII del carattere A
miolcd.write(cont);    //scrive il carattere A
```

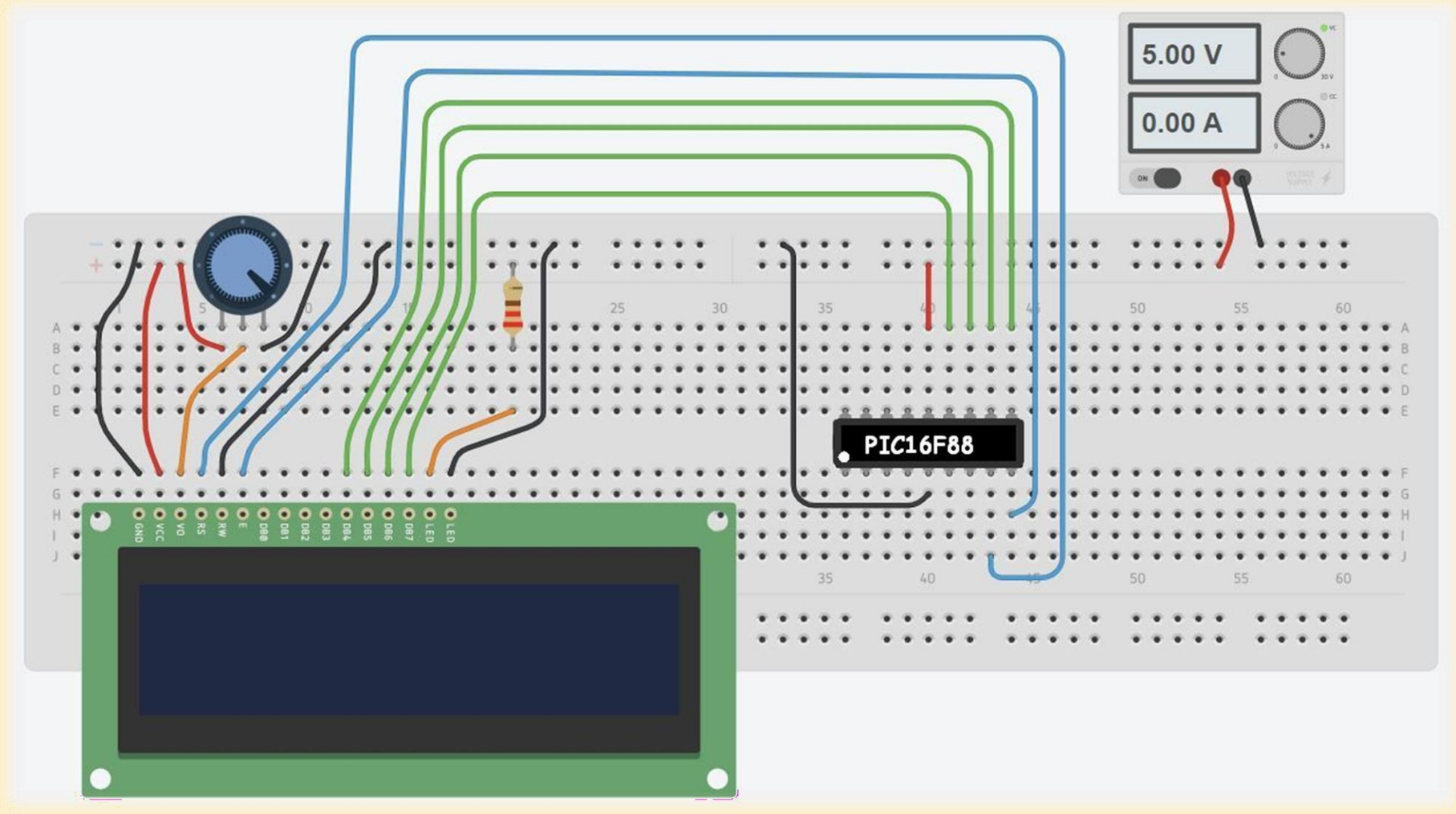
```
#include <LiquidCrystal.h>

Functions
• LiquidCrystal()
• begin()
• clear()
• home()
• setCursor()
• write()
• print()
• cursor()
• noCursor()
• blink()
• noBlink()
• display()
• noDisplay()
• scrollDisplayLeft()
• scrollDisplayRight()
• autoscroll()
• noAutoscroll()
• leftToRight()
• rightToLeft()
• createChar()
```

# PIC+LCD

*Il collegamento con un microcontrollore è identico a quello visto nel caso precedente, pertanto si possono utilizzare nella soluzione più ridotta le 6 linee viste prima. Consideriamo ad esempio il seguente schema con un PIC16F88.*

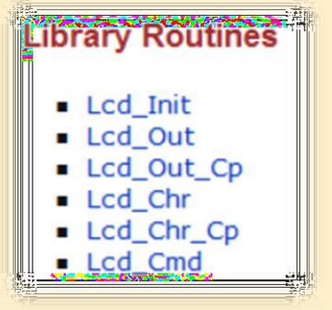


# PIC+LCD


5.00 V
0.00 A
ON
PIC16F88
GND
VCC
VO
RS
RW
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
LED
LED

# PIC+LCD

*Vediamo ora le funzioni presenti nella libreria messa a disposizione dall'ambiente di sviluppo della Mikroelektronika, per la gestione di un LCD alfanumerico.*
*Sul compilatore MikroC troviamo le 6 funzioni indicate a destra.*

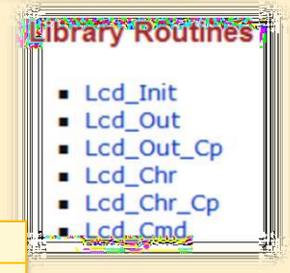



***Lcd_Init()*** *con questa funzione si inizializza il display secondo la dichiarazione dei pin effettuata in precedenza.*

*Bisogna fare attenzione a rispettare i nomi assegnati (LCD_RS … LCD_RS_Direction…) ed ad indicare in entrambe le dichiarazioni i terminali utilizzati secondo lo schema di collegamento.*

```
// definizione dei pin collegati al display
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;


void main(){
  Lcd_Init();   //inizializzazione del display


}
```

# PIC+LCD




***Lcd_Cmd()*** *con questa funzione si invia al display un comando, i comandi possibili sono indicati di seguito.*

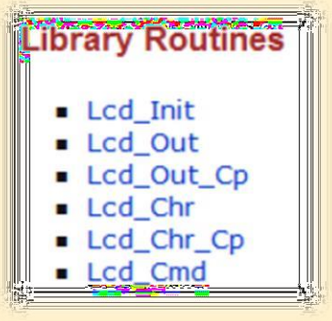
*Es.*

```
//Cancella il display
Lcd_Cmd(_LCD_CLEAR);
```

```
//Cursore lampeggiante
Lcd_Cmd(_LCD_BLINK_CURSOR_ON);
```

| COMANDO | DESCRIZIONE |
|---|---|
| _LCD_FIRST_ROW | Sposta il cursore sulla prima riga |
| _LCD_SECOND_ROW | Sposta il cursore sulla seconda riga |
| _LCD_THIRD_ROW | Sposta il cursore sulla terza riga |
| _LCD_FOURTH_ROW | Sposta il cursore sulla quarta riga |
| _LCD_CLEAR | Cancella il display |
| _LCD_RETURN_HOME | Sposta il cursore alla posizione di home (prima riga prima colonna) senza modificare il contenuto del display. |
| _LCD_CURSOR_OFF | Spegni il cursore |
| _LCD_UNDERLINE_ON | Visualizza il cursore con una sottolineatura |
| _LCD_BLINK_CURSOR_ON | Visualizza il cursore lampeggiante |
| _LCD_MOVE_CURSOR_LEFT | Sposta il cursore a sinistra senza modificare il contenuto del display |
| _LCD_MOVE_CURSOR_RIGHT | Sposta il cursore a destra senza modificare il contenuto del display |
| _LCD_TURN_ON | Accendi il display |
| _LCD_TURN_OFF | Spegni il display |
| _LCD_SHIFT_LEFT | Shifta a sinistra il contenuto del display. |
| _LCD_SHIFT_RIGHT | Shifta a destra il contenuto del display. |

# PIC+LCD




***Lcd_Chr()*** *Scrive un carattere nella posizione indicata secondo l'ordine riga-colonna, il carattere va indicato tra apici.*

*Es.*

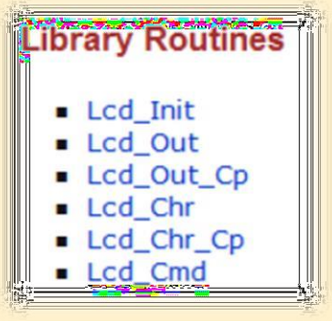
```
//Scrivo il carattere A
//sulla prima riga seconda colonna
Lcd_Chr(1,2,'A');
```

***Lcd_Chr_Cp()*** *Scrive un carattere nella posizione attuale del cursore (CP sta per Current Position) pertanto non va indicata riga e colonna. Il carattere va indicato tra apici.*

*Es.*

```
//Scrivo il carattere A
//sulla posizione attuale del cursore
Lcd_Chr_Cp('A');
```

# PIC+LCD




***Lcd_Out()*** *Scrive una stringa nella posizione indicata secondo l'ordine riga-colonna, la stringa va indicata tra doppi apici.*

*Es.*

```
//scrive la stringa "Ciao"
//sulla prima riga seconda colonna
Lcd_Out(1,2,"Ciao");
```

***Lcd_Out_Cp()*** *Scrive una stringa nella posizione attuale del cursore (CP sta per Current Position) pertanto non va indicata riga e colonna. La stringa va indicata tra doppi apici.*

*Es.*

```
//scrive la stringa "Ciao"
//sulla posizione attuale del cursore
Lcd_Out_Cp("Ciao");
```

# PIC+LCD

*Le due precedenti funzioni **Lcd_Out() e Lcd_Out_Cp()** possono essere utilizzate per visualizzare il contenuto di una variabile di qualsiasi tipo dopo averla convertita in stringa.*

*L'importante è utilizzare un array di dimensione adatta per il tipo di variabile.*
*Ad esempio per una variabile intera occorre un vettore di 7 spazi, in quanto un numero intero va da -32768 a +32767 perciò in tutto utilizza un segno e 5 cifre, cioè*

*6 caratteri, a cui si aggiunge il carattere vuoto di fine stringa.*

*Per convertire il valore in stringa si utilizzano le funzioni presenti nella libreria **Conversion**. Le funzioni utilizzate nell'esempio sono:*

- *ByteToStr* — *variabile 8 bit in stringa*
- *IntToStr* — *variabile intera 16 bit con segno in stringa*
- *LongToStr* — *variabile intera 32 bit con segno*
- *FloatToStr* — *variabile con virgola in stringa.*

```
unsigned short valore1;
int valore2;
long valore3;
float valore4;

char testo1[4];
char testo2[7];
char testo3[12];
char testo4[15];

void main(){
  //inizializzazione del display
  Lcd_Init();

  //visualizzo il contenuto di valore1
  ByteToStr(valore1,testo1);
  Lcd_Out(1,1,testo1);

  //visualizzo il contenuto di valore2
  IntToStr(valore2,testo2);
  Lcd_Out(1,1,testo2);

  //visualizzo il contenuto di valore3
  LongToStr(valore3,testo3);
  Lcd_Out(1,1,testo3);

  //visualizzo il contenuto di valore4
  FloatToStr(valore4,testo4);
  Lcd_Out(1,1,testo4);
}
```

# PIC+LCD

*Un'altra modalità di visualizzazione di una stringa, è quella mostrata in figura.*

*Nel primo caso il vettore **testo1**, si dimensiona automaticamente ed è composto da 5 caratteri (4 caratteri e il carattere vuoto di fine stringa).*

*Nel secondo caso invece **testo2** è il puntatore della stringa memorizzata ad un determinato indirizzo.*

*Per meglio comprendere di seguito è riportato il contenuto delle variabili e della RAM*

```
char testo1[] = "Ciao";
char *testo2="Ciao";

void main(){
  //inizializzazione del display
  Lcd_Init();

  //visualizzo la scritta Ciao
  Lcd_Out(1,1,testo1);

  //visualizzo la scritta Ciao
  Lcd_Out(1,1,testo2);
}
```

| Name | Value | Address |
|---|---|---|
| ⊟ testo2 | 0x26 | 0x0025 |
| [value] | 0x43 | 0x0026 |
| ⊟ testo1 | {...} | 0x0020 |
| [0] | 0x43 | 0x0020 |
| [1] | 0x69 | 0x0021 |
| [2] | 0x61 | 0x0022 |
| [3] | 0x6F | 0x0023 |
| [4] | 0x00 | 0x0024 |

RAM

RAM

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 00 | 00 | 19 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0010 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0020 | 43 | 69 | 61 | 6F | 00 | 26 | 43 | 69 | 61 | 6F |

```
0x43 = C
0x69 = i
0x61 = a
0x6F = o
```

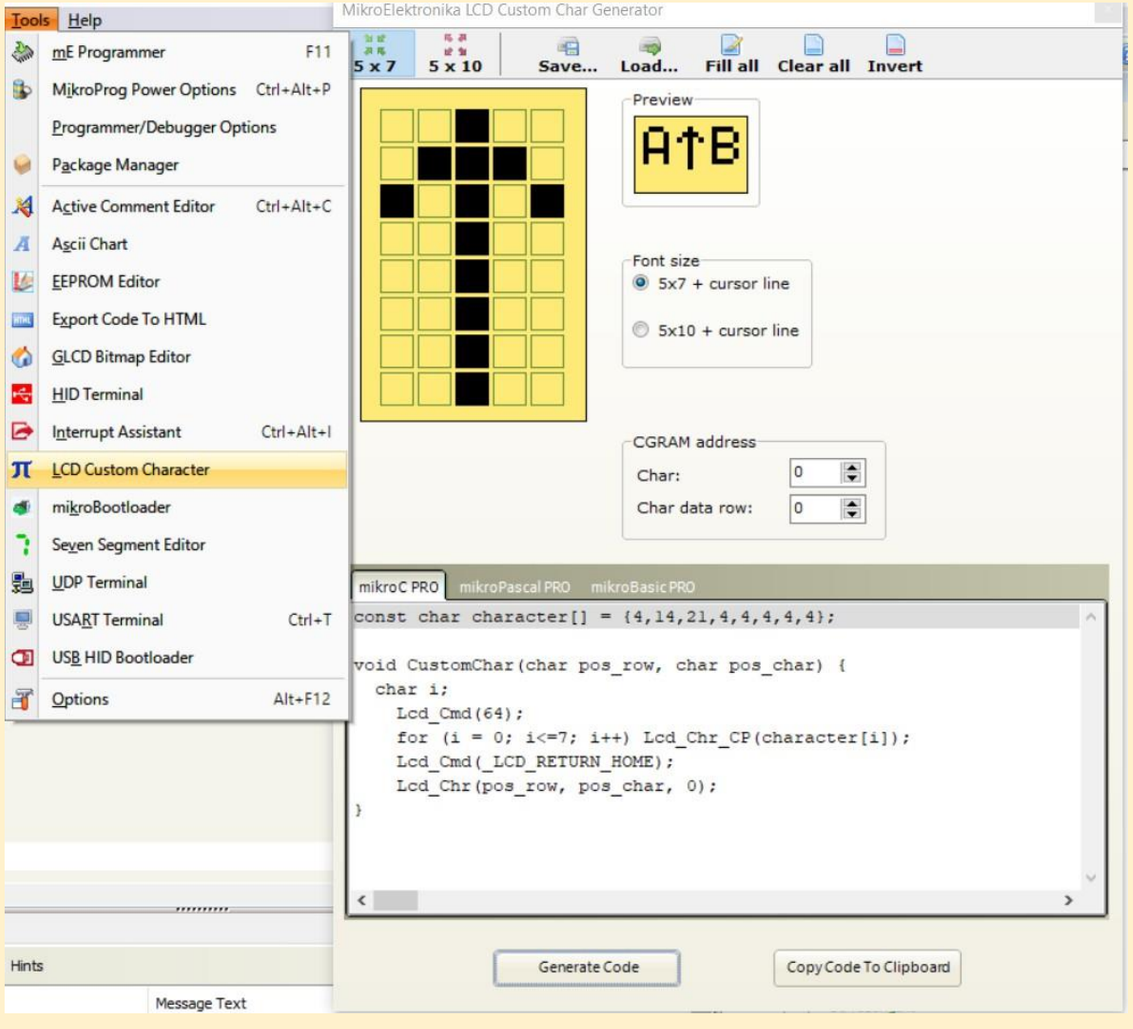
*In entrambi i casi il risultato è la visualizzazione della scritta «**Ciao**» sulla prima riga e prima colonna.*

# PIC+LCD

*Anche con MikroC è possibile personalizzare i caratteri.*

*Per farlo è sufficiente lanciare il tools LCD_Custom_Character.*

*Si aprirà una finestra dove creare il carattere accendendo e spegnendo i pixel, e con Generate Code, si otterrà il codice da inserire nel programma sottoforma di funzione.*



# LCD ALfanumerico

*Pur se piccolo, un display LCD alfanumerico, consente di ottenere ad un basso costo, uno strumento di debug per i nostri progetti, o una piccola e complete interfaccia uomo-macchina.*

*I moderni display grafici stanno soppiantando gli alfanumerici, ma per molte applicazioni questi ultimi presentano i notevoli vantaggi derivati dai costi contenuti e dalla facilità di utilizzo.*

*Per concludere segnalo la seguente dispensa, dove si può vedere come realizzare un programma per gestire con un semplice display da 4 righe, un menu a scorrimento anche con decine di comandi:*

*Gestione di un menu con LCD DotMatrix*

*A questo link, invece una descrizione del funzionamento di un display LCD seriale su bus I2C.*

*LCD Dot Matrix su I2C*