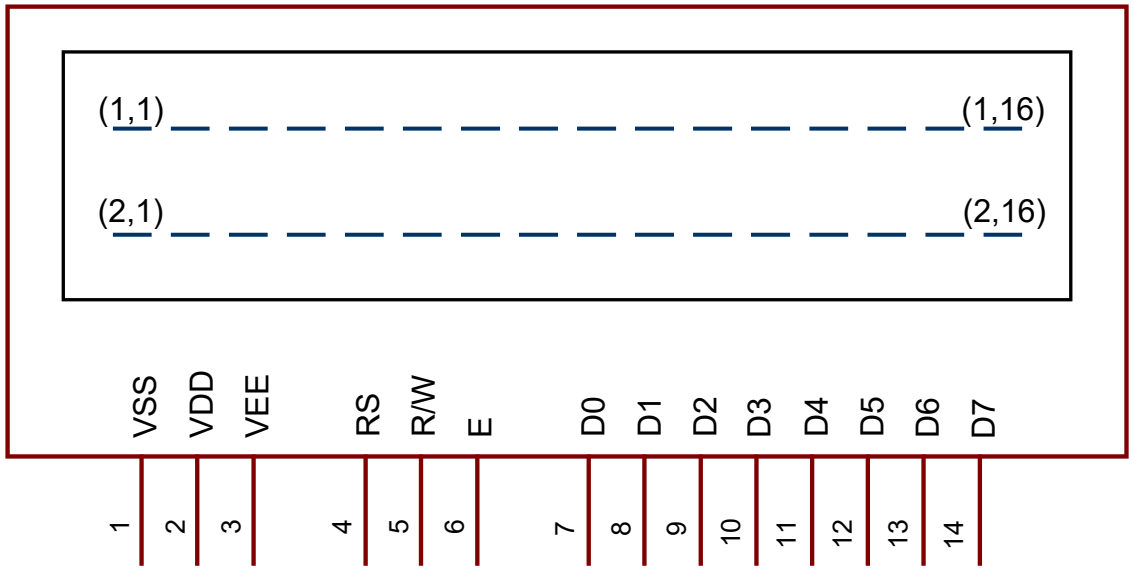
Pantalla de cristal líquido no gráfica (*Lcd: liquid crystal display*)

- En este curso se va a utilizar una pantalla de 32 caracteres, repartidos en 2 filas de 16 caracteres cada una (2 filas × 16 columnas).

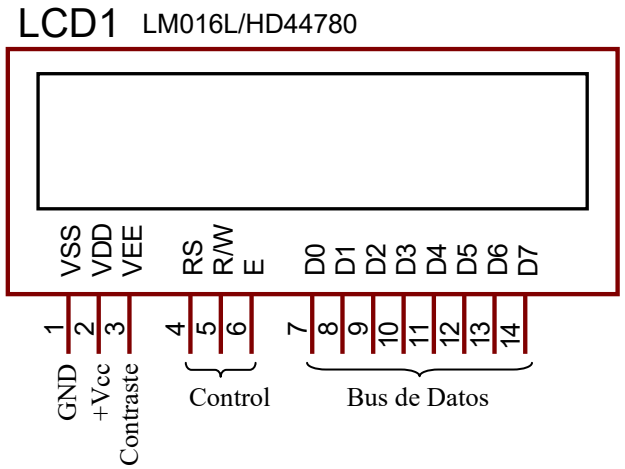
Componente ISIS: LM016L



· En la comunicación del *μC* con un Lcd LM016 se utilizan 2 tipos de señales digitales:

señales de control: RS, R/W, E

señales de datos: D7, D6, D5, D4, D3, D2, D1, D0



· Lo habitual es que no nos interese saber lo que está representando el *Lcd* (ya lo sabemos) y que sólo nos interese enviarle los caracteres a representar ⇒ R/W = 0

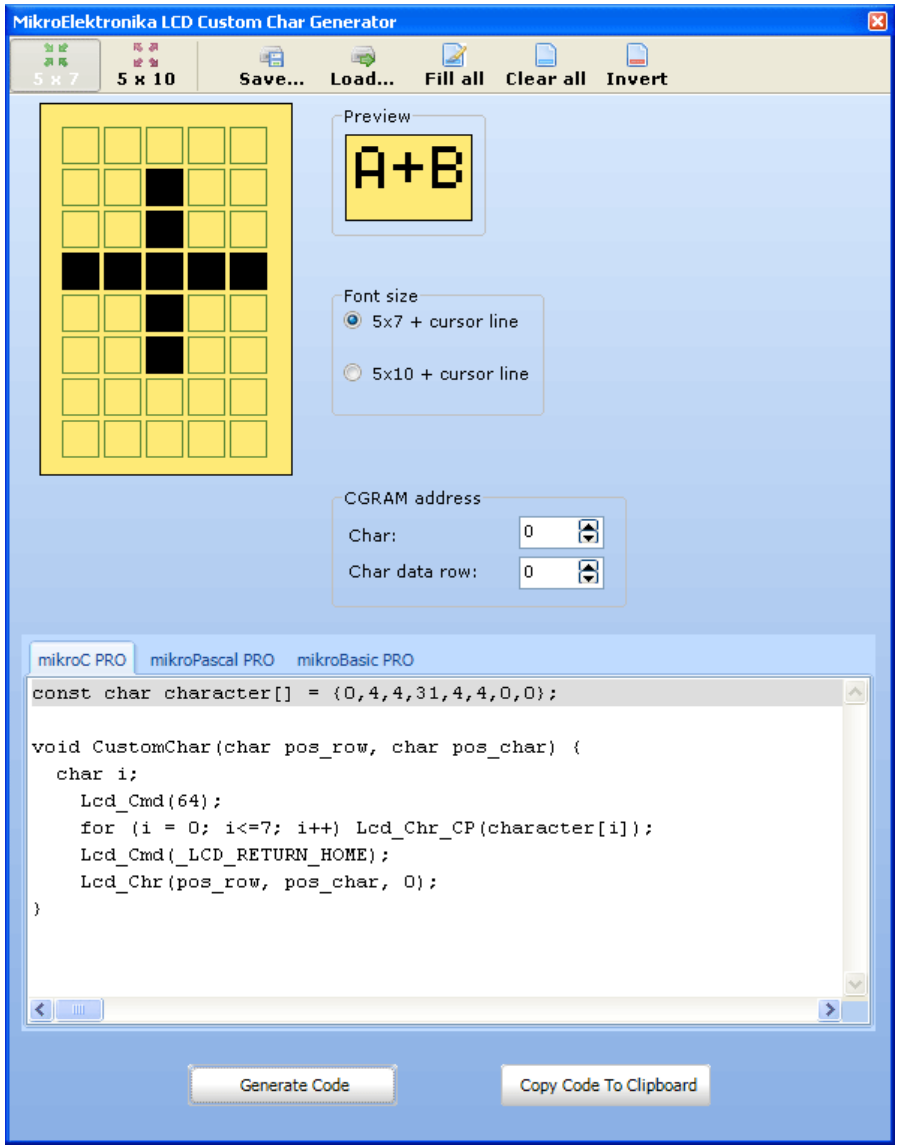
· Los datos a representar en el *Lcd* se pueden enviar utilizando un bus de datos de 4 bits (D7, D6, D5, D4) o bien utilizando un bus de datos de 8 bits (D7, D6, D5, D4, D3, D2, D1, D0). En esta asignatura vamos a utilizar siempre un bus de datos de 4 bits.

· El compilador *MikroC PRO* dispone de una herramienta que permite crear caracteres.

Se accede a dicha herramienta en:

Tools → LCD custom character

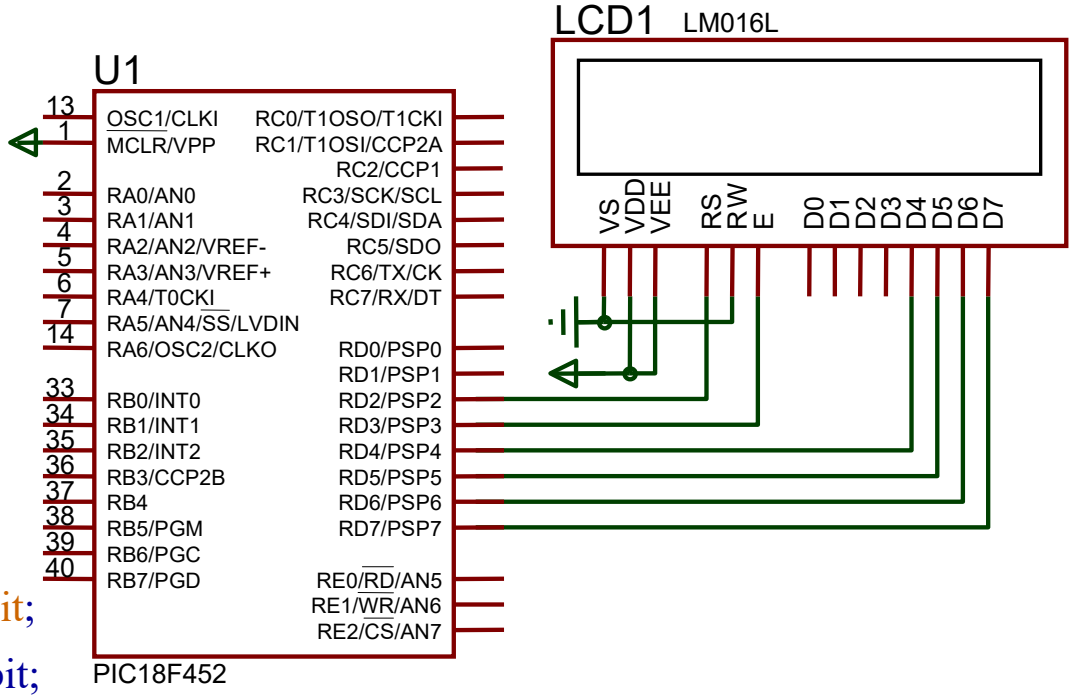
(no la vamos a utilizar en este curso)



▪ El compilador dispone de una serie de funciones que facilitan la programación de la comunicación del *µC* con el *Lcd*. Para poder utilizar dichas funciones es necesario indicar los terminales del *µC* que están conectados al *Lcd* mediante la declaración de una serie de *variables globales*. A continuación se muestra un ejemplo que corresponde a la conexión del *Lcd* a varios terminales del puerto D:

```
// Lcd pinout settings
sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D7 at RD7_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D4 at RD4_bit;

// Pin direction
sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
sbit LCD_D7_Direction at TRISD7_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D4_Direction at TRISD4_bit;
```



▪ Para inicializar el circuito controlador del *Lcd*, en la función *main*() hay que ejecutar la siguiente función: *Lcd_Init* ();

▪ Para representar caracteres individuales en el *Lcd* se dispone de las siguientes funciones:

» Representación de un carácter: *Lcd_Chr* (fila, columna, carácter);

Ejemplos:

Lcd_Chr (1, 5, 'r'); //se representa el carácter *r* en la fila 1, columna 5

char *alfa* = 48;
…
Lcd_Chr (2, 14, *alfa*); //se representa el carácter 0 en la fila 2, columna 14

Lcd_Chr_CP (71); //se representa el carácter *G* en la posición en la que se encuentre el cursor en el momento en el que se ejecute esta función.

Nota: las funciones para manejar el *Lcd* interpretan los valores a representar en ASCII. El compilador dispone de una tabla ASCII

» Representación de una cadena de caracteres: *Lcd_Out* (fila, columna, cadena de caracteres);

Ejemplos:

Lcd_Out (1, 1, “Hola”); se representa la cadena de caracteres *Hola* a partir de la fila 1, columna 1

unsigned char *beta* [] = {“adios”};

o bien

unsigned char *beta* [] = {‘a’, ‘d’, ‘i’, ‘o’, ‘s’, ‘\0’};

…

…

Lcd_Out (2, 1, *beta*); se representa la cadena *adios* en la fila 2 a partir de la columna 1

Lcd_Out_CP (“*bueno*”); se representa la cadena de caracteres *bueno* a partir de la posición actual en la que se encuentre el cursor en el momento en el que se ejecute esta función (*CP* ≡ *current position*).

» Representación de cantidades (números): para representar en el *Lcd* números con más de 1 dígito es necesario convertirlos en primer lugar a una cadena de caracteres. En la biblioteca de funciones del compilador hay varias funciones que, de acuerdo con el tipo y el tamaño del número a representar, permiten realizar esta tarea de forma sencilla:

Representación de un número entero de 8 bits sin signo (*unsigned char ó unsigned short*)$_{0\ -\ 255}$

Ejemplo:

```
unsigned short aux = 12;     // {0, · · ·, 255}
char txt [4];  // tamaño mínimo del array 4 para incluir el carácter null de fin de array.
 ...
ByteToStr (aux, txt);
...
Lcd_out(1,1, txt);  //en la pantalla aparece un espacio en blanco a la izquierda del 12
```

Representación de un número entero de 8 bits con signo (*signed char ó signed short*)$_{-128 - +127}$

Ejemplo:

```
signed short aux = –23;    // {–128, · · ·, +127}
char txt [5];
 ...
ShortToStr (aux, txt);
...
Lcd_out(1,1, txt); //en la pantalla aparece un espacio en blanco a la izquierda del –23
```

Representación de un número entero de 16 bits sin signo (*unsigned int*)$_{0-65535}$

Ejemplo:

```
unsigned int aux = 527;    //{0, · · ·, 65535}
char txt [6];
 ...
WordToStr (aux, txt);
...
Lcd_out(1,1, txt); //en la pantalla aparecen dos espacios en blanco a la izquierda de 527
```

Representación de un número entero de 16 bits con signo (*signed int*)$_{-32768 \text{ - } +32767}$

Ejemplo:

```
signed int aux = –1752;      // {–32768, · · ·, +32767}
char txt [7];
 ...
IntToStr (aux, txt);
...
Lcd_out(1,1, txt); //en la pantalla aparece un espacio en blanco a la izquierda del –1752
```

Representación de un número de 32 bits en coma flotante (*float*)

Ejemplo:

```
float aux = –783.45;
char txt [14];
 ...
FloatToStr (aux, txt);
...
Lcd_out(1,1, txt); //en la pantalla aparecen seis espacios en blanco a la izquierda de –783.45
```

Hay *comandos* que permiten:

_ mover el cursor, apagarlo, …

_ borrar la pantalla

_ desplazar el valor representado en la pantalla

_ etc. (consulta la ayuda del compilador para ver todos los comandos disponibles)

para enviar un comando se utiliza la función: *Lcd_Cmd*();

Ejemplos:

Lcd_Cmd(_LCD_CLEAR); //borra la pantalla

Lcd_Cmd(_LCD_CURSOR_OFF); //apaga el cursor

Para eliminar los espacios en blanco:

```
char i, j;
unsigned txt1[6], txt2[6];
…
for(i = 0, j = 0; i < 6; i++)
{
  if (txt1[i] != ' ')
  {
    txt2[j] = txt1[i];
    j++;
  }
}
```