1602 字符液晶使用说明

2007.11

目　　录

1 模块简介

1.1 特点

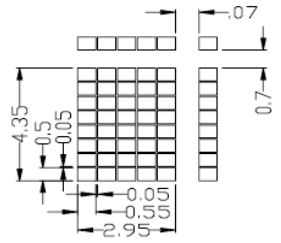
1602 字符型 LCD 模块的应用非常广泛，而各种液晶厂家均有提供几乎都是同样规格的 1602 模块或兼容模块，尽管各厂家的对其各自的产品命名不尽相同；1602 字符型 LCD 模块最初采用的 LCD 控制器采用的是 HD44780，在各厂家生产的 1602 模块当中，基本上也都采用了与之兼容的控制 IC，所以从特性上基本上是一样的；当然，很多厂商提供了不同的字符颜色、背光色之类的显示模块。

通常所见到的 1602 模块的规格基本如下：

显示容量:	16X2 个字符
芯片工作电压:	4.5～5.5V
工作电流:	2.0mA(5.0V)
模块最佳工作电压:	5.0V
字符尺寸:	2.95X4.35(WXH)mm

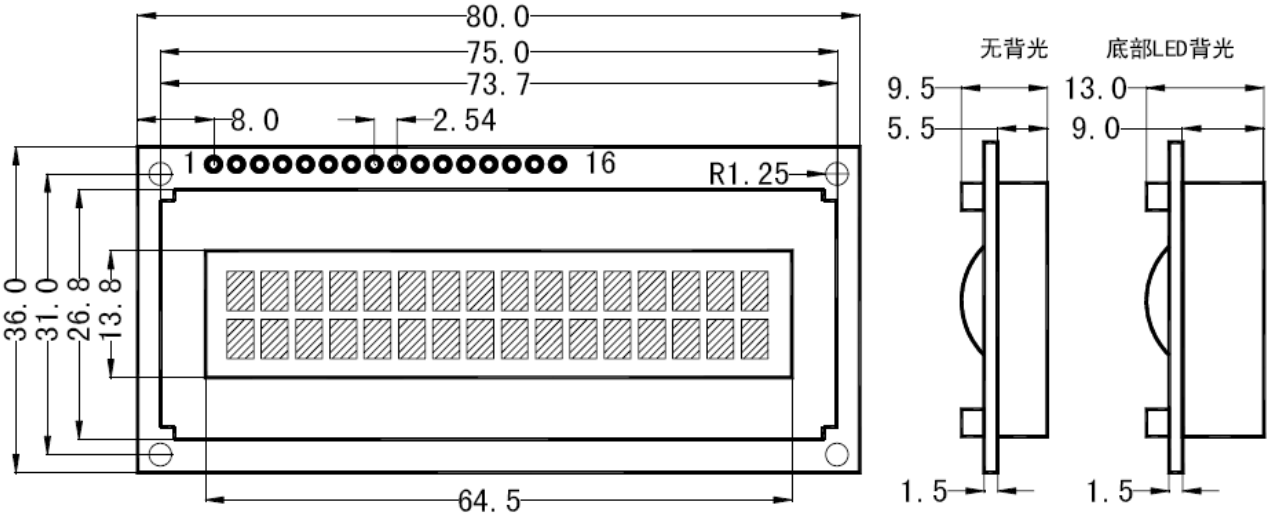
模块内置字符类型为西文字符，具体请参考文档后面附带的字符表。

如 JHD162A 的模块当中，每个字符的点阵情况如下图：



1.2 结构及引脚示意

下图为一般的 1602 LCD 模块的结构尺寸示意图。



模块的引脚说明如下表：

编号	符号	引脚说明	编号	符号	引脚说明
1	VSS	电源地	9	D2	Data I/O
2	VDD	电源正极	10	D3	Data I/O
3	VL	液晶显示偏压信号	11	D4	Data I/O
4	RS	数据/命令选择端（H/L）	12	D5	Data I/O
5	R/W	读/写选择端（H/L）	13	D6	Data I/O
6	E	使能信号	14	D7	Data I/O
7	D0	Data I/O	15	BLA	背光源正极
8	D1	Data I/O	16	BLK	背光源负极

当然，有的模块是不带背光的，这时候 15 和 16 脚是没有意义的。

2 快速使用指南

1602 液晶的功能相对较简单，而控制器的设置也没有几个，下面就最常见的使用来介绍 1602 LCD 的控制驱动特性。

2.1 基本操作时序

1602 液晶的基本的操作分为以下四种：

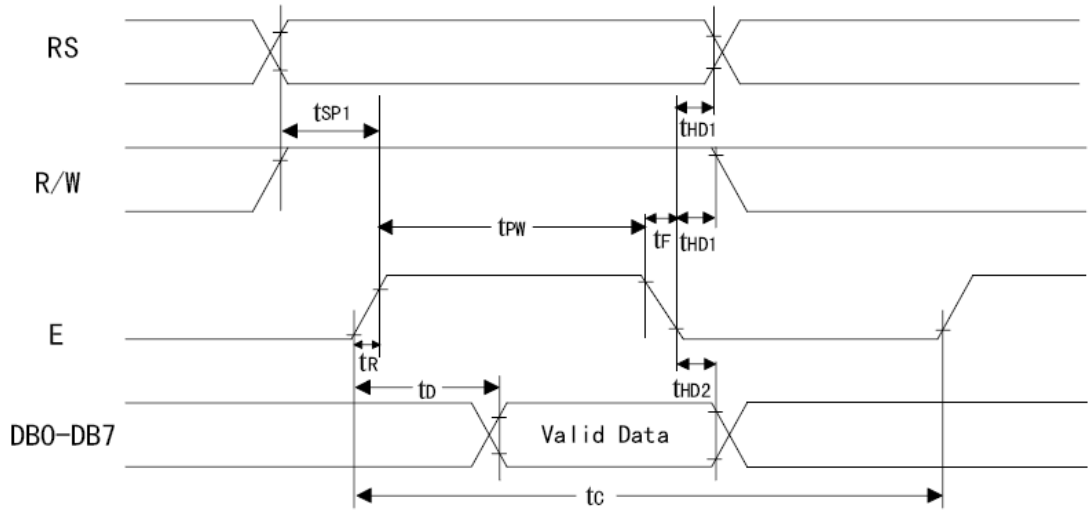
状态字读操作：输入　RS=低、RW=高、EP=高；　输出：DB0~7 读出为状态字；

数据读出操作：输入　RS=高、RW=高、EP=高；　输出：DB0~7 读出为数据；

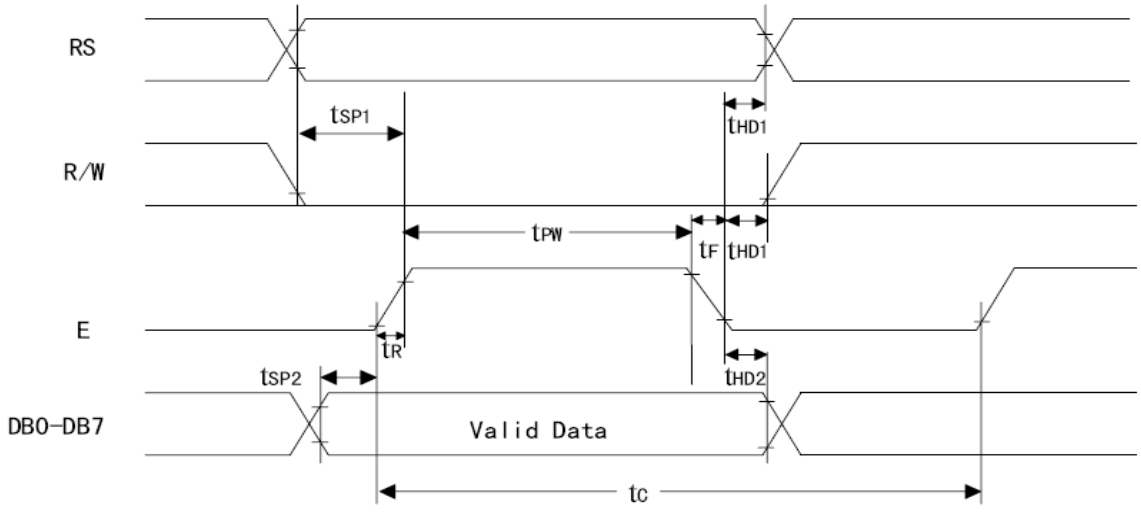
指令写入操作：输入　RS=低、RW=低、EP=上升沿；　输出：无；

数据写入操作：输入　RS=高、RW=低、EP=上升沿；　输出：无。

读操作的时序图如下：



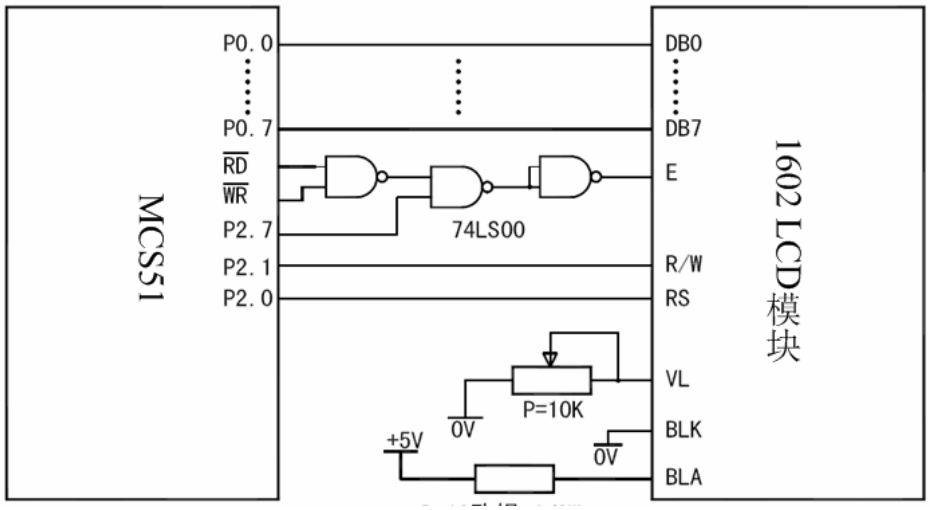
而写操作的时序图如下：



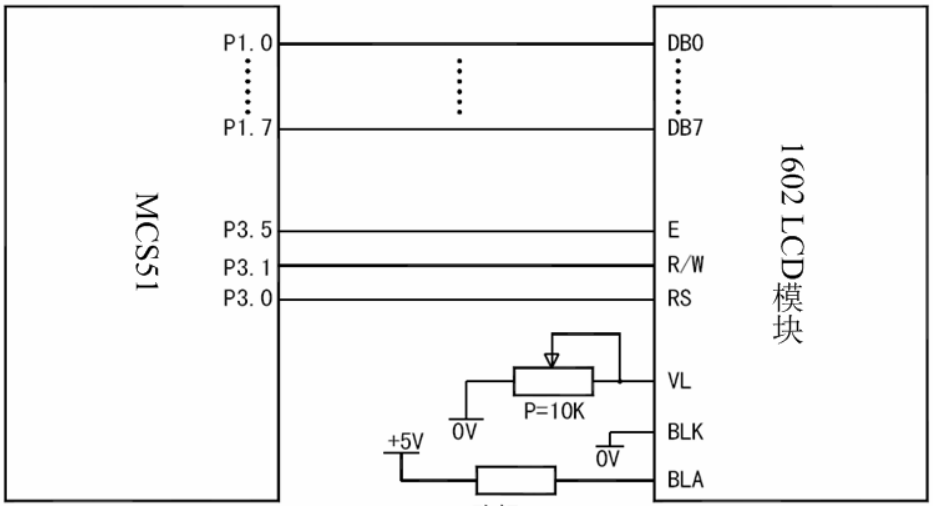
时序时间参数如下表：

时序参数	符号	极限值			单位	测试条件
		最小值	典型值	最大值		
E 信号周期	t_C	400	–	–	ns	引脚 E
E 脉冲宽度	t_{PW}	150	–	–	ns	
E 上升沿/下降沿时间	t_R, t_F	–	–	25	ns	
地址建立时间	t_{SP1}	30	–	–	ns	引脚 E、RS、R/W
地址保持时间	t_{HD1}	10	–	–	ns	
数据建立时间(读操作)	t_D	–	–	100	ns	引脚 DB0～DB7
数据保持时间(读操作)	t_{HD2}	20	–	–	ns	
数据建立时间(写操作)	t_{SP2}	40	–	–	ns	
数据保持时间(写操作)	t_{HD2}	10	–	–	ns	

MCS51 系列 MCU 与 1602 LCD 模块的连接方式一般来说分两中，一是利用总形，将 LCD 模块挂在 MCU 的外部总线上，当然 MCS51 的总线是 8080 的格式，而 1602 LCD 的接口为 6800 总线，两者有区别，是需要经过一些门电路改造一下接口，如下图：

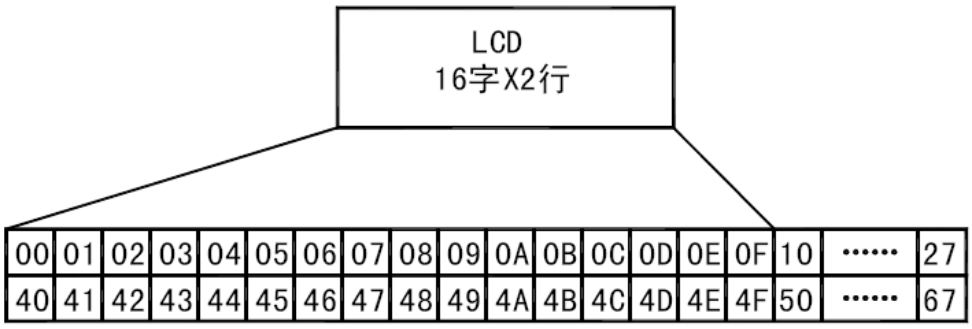


不过也有很多 MCU 与 1602 的接口连接时采用的是端口模拟时序的方法，如下图：



2.2 1602 LCD 的指令

1602 LCD 的控制器内置有 80 个 byte 的显存，而 1602 LCD 只有两行 X 16 个字符的显示区域，所以显存中有些地址是无法对应上 LCD 屏的，下图为显存地址对应图：



状态字：

1602 LCD 读回的状态字中，最高位为读/写允许位，低 7 位为当前数据指针的地址值（即 AC 值）；如下表：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	BF	AC6~0						

BF 位为读/写允许位，在 MCU 对 LCD 进行读写操作前，都要确认该位值为 0 才可进行操作。

AC6~0 为 7 位的 AC 值，表示当前数据指针的地址值。

工作方式设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	DL	N	F	0	0

DL：设置控制器与 MCU 的接口形式，一般设置为 1；

DL=1：数据总线宽度为 8 位，即 DB7~DB0 有效；

DL=0：数据总线宽度为 4 位，即 DB7~DB4 有效；

N：设置显示字符的行数，1602 LCD 一般都设置为 1；

N=0：为一行字符；

N=1：为两行字符；

F：设置显示字符的字体，一般设置为 0；

F=0：为 5X7 点阵字符体；

F=1：为 5X11 点阵字符体；

显示状态设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	0	1	D	C	B

该指令控制着画面、光标和闪烁的开与关。

D：画面显示状态位；D=1 为显示开，D=0 为显示关；该指令仅影响显示屏的开关，并不影响显存中的数据。

C：光标显示状态位；C=1 光标显示；C=0 光标不显示。

B：闪烁显示状态位；当 B=1 时闪烁启动；B=0 时闪烁关闭；闪烁只对于当前地址指针指和的字符位有效。

光标或画面滚动设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	1	S/C	R/L	0	0

该指令设置光标和画面的特性。

S/C：滚动对像的选择

S/C=1　画面滚动

S/C=0　光标滚动

R/L：滚动方向的选择

R/C=1　向右滚动

R/C=0　向左滚动

输入方式设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	0	0	1	I/D	S

该指令的功能在于设置显示字符的输入方式，即在操作数据写入/读出后，AC 数据地址指针的修改方式。

I/D：I/D=0　AC 为自动减 1 的计数器，操作数据后 AC 自动减一；ID=1　AC 为自动加 1 计数器，操作数据后 AC 自动加 1；

S：设置写入字符数据时是否允许画面滚动/光标移动（AC 自动变化）

S=0　禁止

S=1　允许

清屏指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	0	0	0	0	1

该指令将空格码（0x20）写入显存中；达到清屏显示的功能。

归位指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	0	0	0	1	0

该指令完成 AC 清零的功能。

CGRAM 地址设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	1	A5	A4	A3	A2	A1	A0

该指令将 6 位的 CGRAM 地址写入地址指针计数器 AC 内，随后的对数据的操作是对 CGRAM 的读/读操作。

注：CGRAM 为用户自定义字符字符的空间，并非对应 LCD 屏上字符位置的显存（DDRAM）。

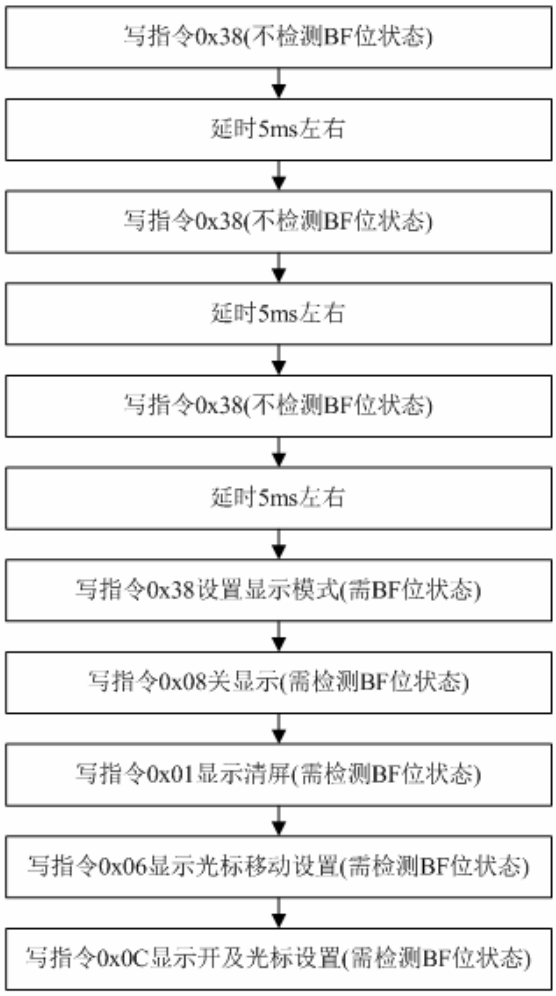
DDRAM 地址设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	1	A6	A5	A4	A3	A2	A1	A0

该指令将 7 位的 DDRAM 地址写入地址指针计数器 AC 当中，随后的数据操作是对 DDRAM（显存）的读/写操作。

2.3 1602 LCD 的初始化

如下流程图所示：



2.4 参考代码

以下代码的 MCU 为 89S51，编译环境为 Keil C51 uV3。

基本时序接口程序：

```
#include "REG51.h"
#include "intrins.h"                //包含此头文件可直接操作内核的寄存器以及一些定义好的宏
// this file for MCU I/O port or the orther`s hardware config
// for LCD Display
// Define for the port use by LCD Driver
sbit LCD_EP=P2^7;
sbit LCD_RW=P2^6;
sbit LCD_RS=P2^5;

#define LCD_Data_BUS_Out        P0
#define LCD_Data_BUS_In         P0
```

```
code unsigned char LCD_InitialCode[]={0x30,0x30,0x30,0x38,0x01,0x06,0x0c};
//=========================================================================
// 函数: void LCD_DataWrite(unsigned char Data)
// 描述: 写一个字节的显示数据至 LCD 中的显示缓冲 RAM 当中
// 参数: Data 写入的数据
// 返回: 无
//=========================================================================
void LCD_DataWrite(unsigned char Data)
{
        unsigned int Read_Dat=0;
        LCD_EP = 0;                     //EP、RS 端口为低，RW 为高
        LCD_RS = 0;
        LCD_RW = 1;
        do{
            LCD_Data_BUS_In = 0xff;
            LCD_EP = 1;
            Read_Dat = LCD_Data_BUS_In&0x80;
            LCD_EP = 0;
        }while(Read_Dat!=0);        //读状态字并判断是否可进行读写操作
        LCD_RW = 0;                     //EP RW to Low
        LCD_RS = 1;                 //RS Hight
        LCD_Data_BUS_Out = Data;
        LCD_EP = 1;                 //EP to Hight
        LCD_EP = 0;                     //EP to low
}
//=========================================================================
// 函数: void LCD_RegWrite(unsigned char Command)
// 描述: 写一个字节的数据至 LCD 中的控制寄存器当中
// 参数: Command       写入的数据（byte）
// 返回: 无
//=========================================================================
void LCD_RegWrite(unsigned char Command)
{
        unsigned int Read_Dat=0;
        LCD_EP = 0;                     //EP、RS 置低，RW 置高，表为读状态字
        LCD_RS = 0;
        LCD_RW = 1;
        do{
```

```
        LCD_Data_BUS_In = 0xff;
        LCD_EP = 1;
        Read_Dat = LCD_Data_BUS_In&0x80;
        LCD_EP = 0;
    }while(Read_Dat!=0);          //读状态字并判断是否可进行读写操作
    LCD_RW = 0;                   //RW to Low，表为写指令
    LCD_Data_BUS_Out = Command;
    LCD_EP = 1;                   //EP to Hight
    LCD_EP = 0;
}
//=============================================================================
// 函数: unsigned char LCD_DataRead(void)
// 描述: 从 LCD 中的显示缓冲 RAM 当中读一个字节的显示数据
// 参数: 无
// 返回: 读出的数据，低八位有效（byte）
//=============================================================================
unsigned char LCD_DataRead(void)
{
    unsigned char Read_Dat=0;
    LCD_EP = 0;                   //EP、RS 置低，RW 置高，表为读状态字
    LCD_RS = 0;
    LCD_RW = 1;
    do{
        LCD_Data_BUS_In = 0xff;
        LCD_EP = 1;
        Read_Dat = LCD_Data_BUS_In&0x80;
        LCD_EP = 0;
    }while(Read_Dat!=0);          //读状态字并判断是否可进行读写操作
    LCD_RS = 1;                   //RS 置高，表为读数据
    LCD_EP = 1;                   //EP to Hight
    Read_Dat = LCD_Data_BUS_In;   //读出数据
    LCD_EP = 0;
    return Read_Dat;
}
//=============================================================================
// 函数: unsigned char LCD_StatusRead(void)
// 描述: 从 LCD 中的显示缓冲 RAM 当中读一个字节的显示数据
// 参数: 无
// 返回: 读出的数据，低八位有效（byte）
```

```
//=========================================================================
unsigned char LCD_StatusRead(void)
{
        unsigned char Read_Dat=0;
        LCD_EP = 0;                         //EP、RS 置低，RW 置高，表为读状态字
        LCD_RS = 0;
        LCD_RW = 1;
        LCD_Data_BUS_In = 0xff;
        LCD_EP = 1;
        Read_Dat = LCD_Data_BUS_In;     //读状态字
        LCD_EP = 0;
        return Read_Dat;
}
```

初始化程序：

```
//=========================================================================
// 函数: void LCD_Init(void)
// 描述: LCD 初始化程序，在里面会完成 LCD 初始所需要设置的许多寄存器，具体如果
//          用户想了解，建议查看 DataSheet 当中各个寄存器的意义
// 参数: 无
// 返回: 无
// 备注:
// 版本:
//        2007/11/14        First version
//=========================================================================
//延时程序
void TimeDelay(int Time)
{
        int i;
        while(Time > 0)
        {
            for(i = 0;i < 800;i++)
            {
                _nop_();
            }
            Time --;
        }
}
```

```
void LCD_Init(void)
{
        unsigned char uiTemp=0,i;
        unsigned char * Point;
        //LCD 驱动所使用到的端口的初始化
        Point = (unsigned char *)LCD_InitialCode;       //获取初始化序列数据的首地址
        LCD_EP = 0;
        LCD_RS = 0;
        LCD_RW = 0;
        for(i=0;i<4;i++)
        {
            uiTemp = *Point++;
            LCD_Data_BUS_Out = uiTemp;
            LCD_EP = 1;                 //EP to Hight
            LCD_EP = 0;                 //EP to Hight
            TimeDelay(4);  //延一定的时间，一般要求 4.5ms 以上就可以，没有那么严格的了
        }
        LCD_RegWrite(*Point++);
        LCD_RegWrite(*Point++);
        LCD_RegWrite(*Point++);
}
```

应用范例代码：

```
void main()
{
        unsigned char uiTemp=0;
        unsigned char * String_s;
        LCD_Init();
        uiTemp = LCD_StatusRead();          //无意义，只是测试读状态字的子程序
        String_s = "LCD1602 Demo";
        LCD_RegWrite(0x80);                     //设置地址为第一行第一个字符的位置
        while(*String_s!=0)                 //显示字符串
        {
            LCD_DataWrite(*String_s);
            String_s++;
        }
        String_s = "     Mzdesign";
        LCD_RegWrite(0xc0);                     //设置地址为第二行第一个字符的位置
        while(*String_s!=0)
        {
```

```
            LCD_DataWrite(*String_s);
            String_s++;
        }
        //以下仅为测试使用，测试读数据程序的功能
        uiTemp = LCD_DataRead();          //读数据
        LCD_RegWrite(0x80);                    //设置地址后再读数据
        uiTemp = LCD_DataRead();
        while(1)
        {
            ;
        }
    }
```

MCU 与 LCD 的连接参考如下：

```
//          LCD DB0~DB7   ---P0.0~P0.7
//          LCD EP        ---P2.7
//          LCD RW        ---P2.6
//          LCD RS        ---P2.5
//          LCD 5V 供电~~~
```

3 1602 LCD 的另外一些特性

3.1 AC 地址指针计数器

地址指针计数器 AC 是可读可写的，它是 DDRAM（显存）和 CGRAM 区作的地址指针计数器，指示当前 DDRAM 或 CGRAM 的地址。而指示着哪种存储器的地址是由 MCU 对 1602 最近写入的地址设置指令的标识码决定的。地址指针计数器 AC 可以设置成自动加一计数器或自动减一计数器。地址指针计数器 AC 有两个作用，一是指示当前的 DDRAM 或 CGRAM 地址；二是为光标和闪烁的位置地址指针，指示当前光标和闪烁的位置地址。

3.2 光标或画面滚动设置

在前面的内容中，简单介绍过该指令，这里主要以图形示意，再进一步介绍该指令的作用。

光标或画面滚动设置指令：

Bit	D7	D6	D5	D4	D3	D2	D1	D0
功能	0	0	0	1	S/C	R/L	0	0

该指令设置光标和画面的特性。

S/C：滚动对像的选择

S/C=1　画面滚动

S/C=0　光标滚动

R/L：滚动方向的选择

R/C=1　向右滚动

R/C=0　向左滚动

画面滚动是在一行连续循环进行的，也就是说一行的第一个单元和最后一个单元连接起来，形成闭环式的滚动；滚动时两行同时滚动。画面滚动的显示效果如下图所示：

显示位置	1	2	3	4	……	15	16			
第一行DDRAM	00	01	02	03	……	0E	0F	……	26	27
第二行DDRAM	40	41	42	43	……	4E	4F	……	66	67

（a）两行 DDRAM 单元与显示字符在画面滚动前的位置关系

显示位置	1	2	3	4	……	15	16			
第一行DDRAM	27	00	01	02	……	0D	0E	……	25	26
第二行DDRAM	67	40	41	42	……	4D	4E	……	65	66

（b）画面向右滚动时 DDRAM 单元与显示字符位置关系

而光标的滚动就很好理解了，只是相当于 AC 指针变化，加一或减一。不过光标的滚动是在全部的 DDRAM 范围内连接滚动的，而不像画面滚动只在一行内闭环滚动。

这条指令与前面介绍的输入方式设置指令是不一样的，这条指令在执行后就进行滚动了，执行一次滚动一次，而不像输入方式的设置指令，它是不执行滚动的，而是在往后的数据操作时才会滚动。

3.3 显示存储器 DDRAM

1602 LCD 模块的控制器里拥有 80 个字节的显示存储器 DDRAM。DDRAM 用于存储当前所要显示的字符的字符代码。DDRAM 的地址由地址指针计数器 AC 提供，MCU 可以对 DDRAM 进行读写操作。DDRAM 各单元对应着显示屏上的各字符位。

3.4 字符发生器与 CGRAM

1602 LCD 模块的控制器内部有两种字符发生器，一种是 CGROM 即已固化好的字模库，见后面附上的字符表。MCU 控制 1602 时，只需要写入某个字符的字符代码，1602 将以其作为字模库的地址将该字符输出给驱动器显示。

另一种为 CGRAM，即可随时定义的字符字模库，1602 提供了 64 个字节的 CGRAM，地址为 0x00~0x3F。它可以生成 8 个 5X8 点阵的自定义字符或 4 个 5X11 点阵的字模库。由于 1602 仅使用一行 5 位数据作为字符点阵，所以作主 CGRAM 字模库仅使用存储单元字节的低 5 位，而高 3 位虽然存在但并不作为字模数据使用。

1602 的控制 IC 提供给 CGRAM 的字符码为 0x00~0x07 或 0x08~0x0F。作为 5X8 点阵字符的字模库时，CGRAM 每 8 个字节为一个字符的字模数据，字模数据的顺序是从上至下排列。每个字符代码都对应着 CGRAM 的 8 个字节单元；作为 5X11 点阵的字符的字模库，CGRAM 每 16 个字节为一个字符的字模数据，其中前 11 个字节为字模数据存储单元，后 5 个字节与字模无关。当然，一般 1602 LCD 模块的字符点阵都为 5X8 的为多，可以在指令中设置，但如 LCD 屏上的字符点阵本来就为 5X8 而无 5X11 时，选用 5X11 是没有意义的或者会出现意想不到的现像。

字符代码与 CGRAM 地址的对应关系为：

5X8 点阵字符

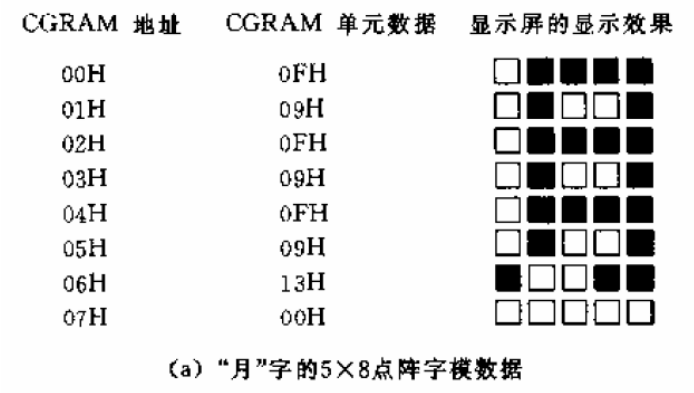
字符代码	CGRAM 地址
0x00(0x08)	0x00~0x07
0x01(0x09)	0x08~0x0F
0x02(0x0A)	0x10~0x17

0x03(0x0B)	0x18~0x1F
0x04(0x0C)	0x20~0x27
0x05(0x0D)	0x28~0x2F
0x06(0x0E)	0x30~0x37
0x07(0x0F)	0x38~0x3F

5X11 点阵字符：

字符代码	CGRAM 地址
0x00(0x08)	0x00~0x0F
0x01(0x09)	0x10~0x1F
0x02(0x0A)	0x20~0x2F
0x03(0x0B)	0x30~0x3F

下图给出了字符代码为 0x00 的 CGRAM 地址以及字模数据与显示效果的对应关系：



(a) “月”字的5×8点阵字模数据

4 1602 LCD 字符表

在 1602 LCD 的控制 IC 当中，字符库中 0x00~0x0F 未定义,留给使用者的自定义字符使用。但只能使用 0x00~0x07 或者 0x08~0x0F 之一。

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															