

## Tarefa 13 - Uso do Display LCD padrão Hiatchi HD44780

Monte o circuito e escreva um programa para que o ATMEGA328P escreva a frase:

"**UTFPR - DAELN**" na primeira linha e

"**uC2 - ATMEGA328P**" na segunda linha

- Deve ser entregue o diagrama esquemático do circuito, feito em software apropriado, e os códigos fonte do programa.
- Deve ser usado barramento de 4 bits para interfacear com o LCD.

Use como referência o livro "AVR e Arduino - Técnicas de Projeto", páginas 132 a 145 (link no item [Planejamento](#) no Moodle)

## Tarefa 14 - Medidor de temperatura

Monte um circuito com o ATMEGA328P para medir a temperatura com o sensor LM35. O circuito deve conter um display padrão HITACHI HD44780 e três botões para ajuste de alarme de temperatura.

- O valor da temperatura deve ser apresentado em °C na primeira linha do display.
- O valor da temperatura de alarme deve ser apresentado na segunda linha do display, através dos três botões (um para decrementar o valor do alarme, outro para incrementar o valor do alarme e o terceiro para setar o valor escolhido).
- O sensor LM35 deve ser ligado a uma das entradas analógicas do ATMEGA328P.

Deve ser entregue:

- o circuito desenhado em software apropriado.
- os códigos fonte do programa.
- a equação de conversão do valor da temperatura para °C.

## 5.5 ACIONANDO LCDs 16 x 2

Os módulos LCDs são interfaces de saída muito úteis em sistemas microcontrolados. Estes módulos podem ser gráficos ou a caractere (alfanuméricos). Os LCDs comuns, tipo caractere, são especificados em número de linhas por colunas, sendo mais usuais as apresentações 16×2, 16×1, 20×2, 20×4, 8×2. Além disso, os módulos podem ser encontrados com *backlight* (LEDs para iluminação de fundo), facilitando a leitura em ambientes escuros. Os LCDs mais comuns empregam o CI controlador HD44780 da Hitachi com interface paralela. Há no mercado também LCDs com controle serial, sendo que novos LCDs são constantemente criados.

A seguir, será descrito o trabalho com o LCD de 16 caracteres por 2 linhas (ver o apêndice B); o uso de qualquer outro baseado no controlador HD44780 é similar. Na fig. 5.16, é apresentado o circuito microcontrolado com um *display* de 16×2.

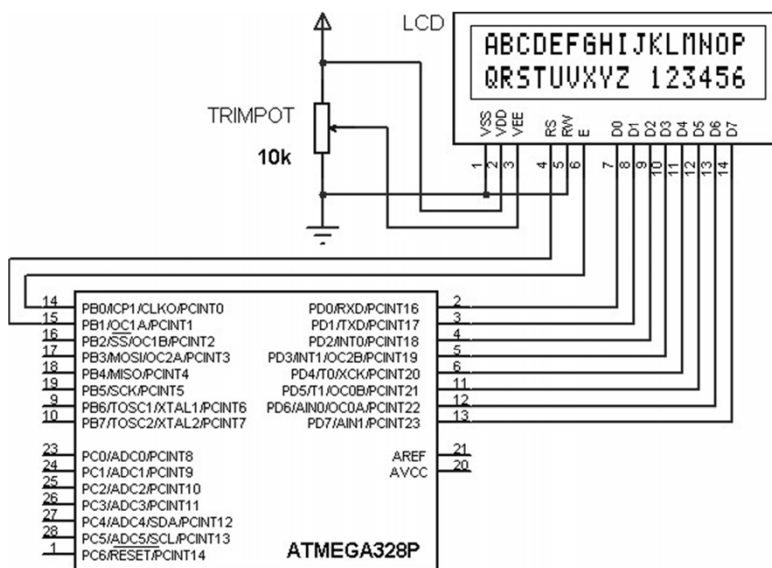


Fig. 5.16 – Circuito para acionamento de um LCD 16×2 usando 8 vias de dados.

Existem duas possibilidades de comunicação com o *display* da fig. 5.16. Uma é empregando 8 via de dados para a comunicação (D0-D7) e a outra, 4 vias de dados (D4-D7). Nesta última, o dado é enviado separadamente em duas partes (2 *nibbles*).

### 5.5.1 INTERFACE DE DADOS DE 8 BITS

Para ler ou escrever no *display* LCD com uma via de dados de 8 bits, é necessário a seguinte sequência de comandos:

1. Levar o pino R/W (*Read/Write*) para 0 lógico se a operação for de escrita e 1 lógico se for de leitura. Aterra-se esse pino se não há necessidade de monitorar a resposta do LCD (forma mais usual de trabalho).
2. Levar o pino RS (*Register Select*) para o nível lógico 0 ou 1 (instrução ou caractere).
3. Transferir os dados para a via de dados (8 bits).
4. Gerar um pulso de habilitação. Ou seja, levar o pino E (*Enable*) para 1 lógico e, após um pequeno tempo, para 0 lógico.
5. Empregar uma rotina de atraso entre as instruções ou fazer a leitura do *busy flag* (o bit 7 da linha de dados que indica que o *display* está ocupado) antes do envio da instrução, enviando-a somente quando esse *flag* for 0 lógico.

Os passos 1, 2 e 3 podem ser efetuados em qualquer sequência, pois o pulso de habilitação é que faz o controlador do LCD ler os dados dos seus pinos. É importante respeitar os tempos de resposta do LCD à transição dos sinais enviados ao mesmo<sup>26</sup>.

Toda vez que a alimentação do LCD é ligada, deve ser executada uma rotina de inicialização. O LCD começa a responder aproximadamente 15 ms após a tensão de alimentação atingir 4,5 V. Como não se conhece o

---

<sup>26</sup> Para uma melhor compreensão sobre o assunto, consultar o manual do fabricante do LCD empregado.

tempo necessário para que ocorra a estabilização da tensão no circuito onde está colocado o LCD, pode ser necessário frações bem maiores de tempo para que o LCD possa responder a comandos. Muitas vezes, esse detalhe é esquecido e o LCD não funciona adequadamente. Para corrigir esse problema, basta colocar uma rotina de atraso suficientemente grande na inicialização do LCD. Na fig. 5.17, é apresentado o fluxograma de inicialização do LCD conforme especificação da Hitachi. Se desejado, o *busy flag* pode ser lido após o ajuste do modo de utilização do *display*. Os comandos para o LCD são detalhado no apêndice B.

A seguir são apresentadas as rotinas de escrita no LCD em conjunto com um programa exemplo que escreve na linha superior do LCD “ABCDEFGHJKLMNOP” e “QRSTUVWXYZ 123456” na linha inferior (circuito da fig. 5.16). Muitos programadores não utilizam a rotina de inicialização completa do diagrama da fig. 5.17, respeitando apenas o tempo de resposta do *display*, seguido do modo de utilização e dos outros controles, prática que geralmente funciona.

Para a visualização dos caracteres é imprescindível o emprego do *trimpot* (fig. 5.16) para ajuste do contraste do *display* de cristal líquido.

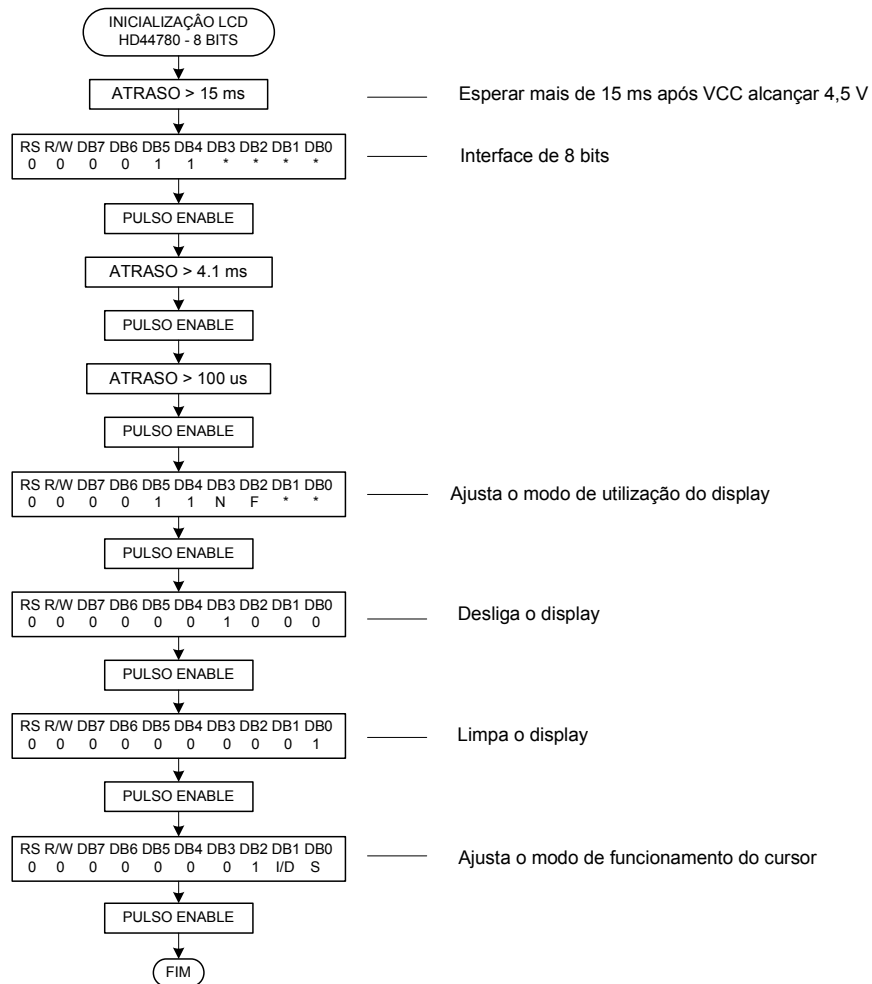


Fig. 5.17 – Rotina de inicialização de 8 bits para um LCD com base no CI HD44780.

## LCD\_8bits.c

```
//===== //
//      ACIONANDO UM DISPLAY DE CRISTAL LIQUIDO DE 16x2      //
//      //      //
//      Interface de dados de 8 bits      //
//===== //
#define F_CPU 16000000UL //define a frequência do microcontrolador - 16MHz

#include <avr/io.h> //definições do componente especificado
#include <util/delay.h> //biblioteca para o uso das rotinas de delay
#include <avr/pgmspace.h> //uso de funções para salvar dados na memória de programa

//Definições de macros - empregadas para o trabalho com o bits
#define set_bit(Y,bit_x) (Y|=(1<<bit_x)) //ativa o bit x da variável Y
#define clr_bit(Y,bit_x) (Y&=~(1<<bit_x)) //limpa o bit x da variável Y
#define tst_bit(Y,bit_x) (Y&(1<<bit_x)) //testa o bit x da variável Y
#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x)) //troca o estado do bit x da variável Y

//para uso no LCD (deve estar na mesma linha)
#define pulso_enable() _delay_us(1); set_bit(CONTR_LCD,E); _delay_us(1);
                                     clr_bit(CONTR_LCD,E); _delay_us(45)

#define DADOS_LCD PORTD //8 bits de dados do LCD na porta D
#define CONTR_LCD PORTB //os pinos de controle estão no PORTB
#define RS PB1 //pino de instrução ou dado para o LCD
#define E PB0 //pino de enable do LCD

//mensagem armazenada na memória flash
const unsigned char msg1[] PROGMEM = "ABCDEFGHIIJKLMNOP";

//-----
//Sub-rotina para enviar caracteres e comandos ao LCD
//-----
void cmd_LCD(unsigned char c, char cd)//c é o dado e cd indica se é instrução ou caractere
{
    DADOS_LCD = c;

    if(cd==0)
        clr_bit(CONTR_LCD,RS); //RS = 0
    else
        set_bit(CONTR_LCD,RS); //RS = 1

    pulso_enable();

    //se for instrução de limpeza ou retorno de cursor espera o tempo necessário
    if((cd==0) && (c<4))
        _delay_ms(2);
}

//-----
//Sub-rotina de inicialização do LCD
//-----
void inic_LCD_8bits()//sequência ditada pelo fabricante do circuito de controle do LCD
{
    clr_bit(CONTR_LCD,RS);//o LCD será só escrito então R/W é sempre zero

    _delay_ms(15); /*tempo para estabilizar a tensão do LCD, após VCC ultrapassar
                                     4.5 V (pode ser bem maior na prática)*/

    DADOS_LCD = 0x38; //interface 8 bits, 2 linhas, matriz 7x5 pontos

    pulso_enable(); //enable respeitando os tempos de resposta do LCD
    _delay_ms(5);
    pulso_enable();
    _delay_us(200);
}
```

```

    pulso_enable();
    pulso_enable();

    cmd_LCD(0x08,0);    //desliga LCD
    cmd_LCD(0x01,0);    //limpa todo o display
    cmd_LCD(0x0C,0);    //mensagem aparente cursor inativo não piscando
    cmd_LCD(0x80,0);    //escreve na primeira posição a esquerda - 1ª linha
}
//-----
//Sub-rotina de escrita no LCD
//-----
void escreve_LCD(char *c)
{
    for (; *c!=0;c++) cmd_LCD(*c,1);
}
//-----
int main()
{
    unsigned char i;

    DDRB = 0xFF;    //PORTB como saída
    DDRD = 0xFF;    //PORTD como saída
    UCSRB = 0x00; //habilita os pinos PD0 e PD1 como I/O para uso no Arduino

    inic_LCD_8bits();    //inicializa o LCD

    for(i=0;i<16;i++)    //enviando caractere por caractere
        cmd_LCD(pgm_read_byte(&msg1[i]),1); //lê na memória flash e usa cmd_LCD

    cmd_LCD(0xC0,0);    //desloca o cursor para a segunda linha do LCD
    escreve_LCD("QRSTUVWXYZ 123456");//a cadeia de caracteres é criada na RAM

    for(;;);            //laço infinito
}
//=====

```

## 5.5.2 INTERFACE DE DADOS DE 4 BITS

Nesta seção, o trabalho com o LCD e suas rotinas serão melhores detalhados, visto que utilizar uma interface de dados de 8 bits para um LCD de  $16 \times 2$  não é recomendado. Isso se deve ao uso excessivo de vias para o acionamento do *display* (10 ou 11). O emprego de 4 bits de dados libera 4 pinos de I/O do microcontrolador para outras atividades, além de diminuir o número de trilhas necessárias na placa de circuito impresso. O custo é um pequeno aumento na complexidade do programa de controle do LCD, o que consome alguns bytes a mais de programação.

Para ler ou escrever no *display* LCD com uma via de dados de 4 bits, é necessário a seguinte sequência de comandos:

1. Levar o pino R/W (*Read/Write*) para 0 lógico se a operação for de escrita e 1 lógico se for de leitura. Aterra-se esse pino se não há necessidade de monitorar a resposta do LCD (forma mais usual de trabalho).
2. Levar o pino RS (*Register Select*) para o nível lógico 0 ou 1 (instrução ou caractere).
3. Transferir a parte mais significativa dos dados para a via de dados (4 bits mais significativos (MSB) – *nibble* maior).
4. Gerar um pulso de habilitação. Ou seja, levar o pino E (*Enable*) para 1 lógico e após um pequeno tempo de espera para 0 lógico.
5. Transferir a parte menos significativa dos dados para a via de dados (4 bits menos significativos (LSB) – *nibble* menor).
6. Gerar outro pulso de habilitação.
7. Empregar uma rotina de atraso entre as instruções ou fazer a leitura do *busy flag* (o bit 7 da linha de dados que indica que o *display* está ocupado) antes do envio da instrução, enviando-a somente quando esse *flag* for 0 lógico.

Os passos 1, 2 e 3 podem ser efetuados em qualquer sequência, pois o pulso de habilitação é que faz o controlador do LCD ler os dados dos seus pinos.

Na fig. 5.18, é apresentado o circuito microcontrolado com o LCD 16×2 com via de dados de 4 bits (conexão igual a do módulo LCD *shield* da Ekitszone<sup>27</sup>). Na sequência, mostram-se o fluxograma de inicialização do *display*, de acordo com a Hitachi, e o programa de controle do LCD; os arquivos com os programas para o trabalho com o LCD foram organizados de forma estruturada (ver a seção 4.5.11). O resultado prático é visto na fig. 5.20.

---

<sup>27</sup> [www.ekitszone.com](http://www.ekitszone.com). Existem outros módulos LCD disponíveis no mercado, depende somente da escolha do usuário. Se desejado o circuito pode ser montado em uma matriz de contatos.





### **def\_principais.h** (arquivo de cabeçalho do programa principal)

```
#ifndef _DEF_PRINCIPAIS_H
#define _DEF_PRINCIPAIS_H

#define F_CPU 16000000UL //define a frequência do microcontrolador - 16MHz

#include <avr/io.h> //definições do componente especificado
#include <util/delay.h> //biblioteca para o uso das rotinas de _delay_ms e _delay_us()
#include <avr/pgmspace.h> //para a gravação de dados na memória flash

//Definições de macros para o trabalho com bits
#define set_bit(y,bit) (y|=(1<<bit)) //coloca em 1 o bit x da variável Y
#define clr_bit(y,bit) (y&=~(1<<bit)) //coloca em 0 o bit x da variável Y
#define cpl_bit(y,bit) (y^=(1<<bit)) //troca o estado lógico do bit x da variável Y
#define tst_bit(y,bit) (y&(1<<bit)) //retorna 0 ou 1 conforme leitura do bit

#endif
```

### **LCD\_4bits.c** (programa principal)

```
//===== //
//          ACIONANDO UM DISPLAY DE CRISTAL LIQUIDO DE 16x2          //
//          Interface de dados de 4 bits                             //
//===== //

#include "def_principais.h" //inclusão do arquivo com as principais definições
#include "LCD.h"

//definição para acessar a memória flash
prog_char mensagem[] = " DADOS DE 4BITS!\0"; //mensagem armazenada na memória flash

//-----
int main()
{
    DDRD = 0xFF; //PORTD como saída
    DDRB = 0xFF; //PORTB como saída

    inic_LCD_4bits(); //inicializa o LCD
    escreve_LCD(" INTERFACE DE"); //string armazenada na RAM
    cmd_LCD(0xC0,0); //desloca cursor para a segunda linha
    escreve_LCD_Flash(mensagem); //string armazenada na flash

    for(;;){} //laço infinito, aqui vai o código principal
}
//=====
```

## LCD.h (arquivo de cabeçalho do LCD.c)

```
#ifndef _LCD_H
#define _LCD_H

#include "def_principais.h"

#define DADOS_LCD PORTD//4 bits de dados do LCD no PORTD
#define nibble_dados 1 /*0 para via de dados do LCD nos 4 LSBs do PORT
                        empregado (Px0-D4, Px1-D5, Px2-D6, Px3-D7), 1 para via de
                        dados do LCD nos 4 MSBs do PORT empregado (Px4-D4, Px5-D5,
                        Px6-D6, Px7-D7) */
#define CONTR_LCD PORTB//PORT com os pinos de controle do LCD (pino R/W em 0).
#define E PB1 //pino de habilitação do LCD (enable)
#define RS PB0 //pino para informar se o dado é uma instrução ou caractere

#define tam_vetor 5 //número de dígitos individuais para a conversão por ident_num()
#define conv_ascii 48 /*48 se ident_num() deve retornar um número no formato ASCII (0 para
                        formato normal)*/

//sinal de habilitação para o LCD
#define pulso_enable() _delay_us(1); set_bit(CONTR_LCD,E); _delay_us(1);
                        clr_bit(CONTR_LCD,E); _delay_us(45)

//protótipo das funções
void cmd_LCD(unsigned char c, char cd);
void inic_LCD_4bits();
void escreve_LCD(char *c);
void escreve_LCD_Flash(const char *c);

void ident_num(unsigned int valor, unsigned char *disp);

#endif
```

## LCD.c (funções para o LCD)

```
//===== //
// Sub-rotinas para o trabalho com um LCD 16x2 com via de dados de 4 bits //
// Controlador HD44780 - Pino R/W aterrado //
// A via de dados do LCD deve ser ligado aos 4 bits mais significativos ou //
// aos 4 bits menos significativos do PORT do uC //
//===== //

#include "LCD.h"

//-----
// Sub-rotina para enviar caracteres e comandos ao LCD com via de dados de 4 bits
//-----
//c é o dado e cd indica se é instrução ou caractere (0 ou 1)
void cmd_LCD(unsigned char c, char cd)
{
    if(cd==0) //instrução
        clr_bit(CONTR_LCD,RS);
    else //caractere
        set_bit(CONTR_LCD,RS);

        //primeiro nibble de dados - 4 MSB
    #if (nibble_dados)//compila o código para os pinos de dados do LCD nos 4 MSB do PORT
        DADOS_LCD = (DADOS_LCD & 0xF0)|(0xF0 & c);
    #else //compila o código para os pinos de dados do LCD nos 4 LSB do PORT
        DADOS_LCD = (DADOS_LCD & 0xF0)|(c<>4);
    #endif

    pulso_enable();
}
```

```

//segundo nibble de dados - 4 LSB
#if (nibble_dados) //compila o código para os pinos de dados do LCD nos 4 MSB do PORT
    DADOS_LCD = (DADOS_LCD & 0x0F) | (0xF0 & (c<<4));
#else //compila o código para os pinos de dados do LCD nos 4 LSB do PORT
    DADOS_LCD = (DADOS_LCD & 0xF0) | (0x0F & c);
#endif

pulso_enable();

if((cd==0) && (c<4)) //se for instrução de retorno ou limpeza espera LCD estar pronto
    _delay_ms(2);
}
//-----
//Sub-rotina para inicialização do LCD com via de dados de 4 bits
//-----
void inic_LCD_4bits()//sequência ditada pelo fabricante do circuito integrado HD44780
{
    //o LCD será só escrito. Então, R/W é sempre zero.

    clr_bit(CONTR_LCD,RS);//RS em zero indicando que o dado para o LCD será uma instrução
    clr_bit(CONTR_LCD,E);//pino de habilitação em zero

    _delay_ms(20); //tempo para estabilizar a tensão do LCD, após VCC
                    //ultrapassar 4.5 V (na prática pode ser maior).*/

    //interface de 8 bits
    #if (nibble_dados)
        DADOS_LCD = (DADOS_LCD & 0x0F) | 0x30;
    #else
        DADOS_LCD = (DADOS_LCD & 0xF0) | 0x03;
    #endif

    pulso_enable(); //habilitação respeitando os tempos de resposta do LCD
    _delay_ms(5);
    pulso_enable();
    _delay_us(200);
    pulso_enable(); //até aqui ainda é uma interface de 8 bits.

    //interface de 4 bits, deve ser enviado duas vezes (a outra está abaixo)
    #if (nibble_dados)
        DADOS_LCD = (DADOS_LCD & 0x0F) | 0x20;
    #else
        DADOS_LCD = (DADOS_LCD & 0xF0) | 0x02;
    #endif

    pulso_enable();
    cmd_LCD(0x28,0); //interface de 4 bits 2 linhas (aqui se habilita as 2 linhas)
                    //são enviados os 2 nibbles (0x2 e 0x8)
    cmd_LCD(0x08,0); //desliga o display
    cmd_LCD(0x01,0); //limpa todo o display
    cmd_LCD(0x0C,0); //mensagem aparente cursor inativo não piscando
    cmd_LCD(0x80,0); //inicializa cursor na primeira posição a esquerda - 1a linha
}
//-----
//Sub-rotina de escrita no LCD - dados armazenados na RAM
//-----
void escreve_LCD(char *c)
{
    for (; *c!=0;c++) cmd_LCD(*c,1);
}
//-----
//Sub-rotina de escrita no LCD - dados armazenados na FLASH
//-----
void escreve_LCD_Flash(const char *c)
{
    for (;pgm_read_byte(&(*c))!=0;c++) cmd_LCD(pgm_read_byte(&(*c)),1);
}

```

```

//-----
//Conversão de um número em seus dígitos individuais - função auxiliar
//-----
void ident_num(unsigned int valor, unsigned char *disp)
{
    unsigned char n;

    for(n=0; n<tam_vetor; n++)
        disp[n] = 0 + conv_ascii;    //limpa vetor para armazenagem dos dígitos

    do
    {
        *disp = (valor%10) + conv_ascii; //pega o resto da divisão por 10
        valor /=10;                      //pega o inteiro da divisão por 10
        disp++;
    }while (valor!=0);
}
//-----

```

É fundamental compreender as funções apresentadas para a programação do LCD. As funções foram desenvolvidas para serem flexíveis quanto à conexão dos pinos do LCD ao microcontrolador, a única ressalva é quanto a disposição dos 4 pinos de dados do LCD (D4-D7), os quais devem ser conectados em um *nibble* alto ou em um *nibble* baixo do PORT utilizado. As definições dos pinos é feita no arquivo LCD.h.



Fig. 5.20 – Resultado do programa para controle de um LCD 16 × 2 com interface de dados de 4 bits (módulo LCD – *shield*, da Ekitszone).

A principal função para o controle do LCD é a **cmd\_LCD(dado, 0 ou 1)**, que recebe dois parâmetros: o dado que se deseja enviar ao LCD e o

número 0 ou 1; onde 0 indica que o dado é uma instrução e 1 indica que o dado é um caractere.

A função **inic\_LCD\_4bits( )** deve ser utilizada no início do programa principal para a correta inicialização do LCD. Existe uma sequência de comandos que deve ser seguida para que o LCD possa funcionar corretamente.

A função **escreve\_LCD("frase")** recebe uma *string*, ou seja um conjunto de caracteres. Como na programação em C toda *string* é finalizada com o caractere nulo (0), essa função se vale desse artifício para verificar o final da *string*. Deve-se ter cuidado ao se utilizar essa função, porque a *string* é armazenada na memória RAM do microcontrolador, o que pode limitar a memória disponível para o programa. Para evitar esse problema, pode-se utilizar a função **escreve\_LCD\_Flash(frase)**, onde a frase, previamente declarada no programa, é armazenada na memória *flash*.

Uma vez inicializado o LCD, se escolhe em qual posição dele se deseja escrever. Cada vez que um caractere é escrito, o cursor é automaticamente deslocado para a próxima posição de escrita, à direita ou à esquerda conforme inicialização (ver a tabela B.3 do apêndice B). Assim, é importante entender como mudar a posição do cursor antes de escrever o caractere. Na fig. 5.21, são apresentados os endereços correspondentes a cada caractere do LCD 16 × 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Linha 1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Linha 2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Fig. 5.21 – Endereços para escrita num LCD 16 × 2.

Por exemplo, quando se deseja escrever o caractere A na 6ª posição da linha superior (linha 1), deve ser empregado o seguinte código:

```
cmd_LCD(0x85,0);    //desloca cursor para o endereço 0x86
cmd_LCD('A',1);     //escrita do caractere
```

Para escrever o conjunto de caracteres “Alo mundo!” na linha inferior começando na terceira posição, deve-se empregar o código:

```
cmd_LCD(0xC2,0);           //desloca cursor para o endereço 0xC2
escreve_LCD("Alo mundo!"); //escrita da string
```

A mensagem não aparecerá caso se escreva em uma posição que não exista na tela do LCD. Se outro LCD for empregado, como por exemplo um  $20 \times 4$  (20 caracteres por 4 linhas) o endereçamento será diferente. Na fig. 5.22, são apresentados os endereços dos caracteres para um LCD  $20 \times 4$ . A linha 3 é continuação da linha 1 e a linha 4, continuação da linha 2. Na dúvida, o manual do fabricante deve ser consultado.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Linha 1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
Linha 2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
Linha 3	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7
Linha 4	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7

Fig. 5.22 – Endereços para a escrita num LCD  $20 \times 4$ .

### **Exercícios:**

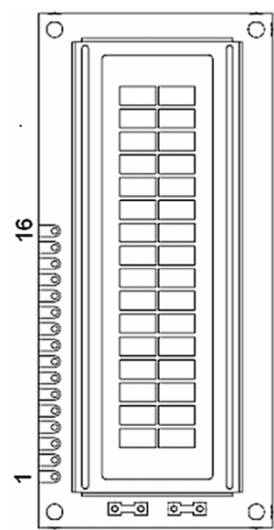
- 5.15** – Elaborar um programa para deslocar um caractere ‘\*’ (asterisco) no LCD da fig. 5.18, da esquerda para a direita, ao chegar ao final da linha o caractere deve retornar (vai e vem).
- 5.16** – Repetir o exercício 5.15 empregando as duas linhas do LCD. Ao chegar ao final da linha superior, o asterisco começa na linha inferior (endereço 0xD3). Dessa forma, na linha superior o asterisco se desloca da esquerda para a direita e na linha inferior, da direita para a esquerda.
- 5.17** – Elaborar um programa para realizar o movimento de um cursor num LCD  $16 \times 2$  com o uso de 4 botões, conforme fig. 5.23.

## B. DISPLAY DE CRISTAL LÍQUIDO 16 x 2 - CONTROLADOR HD44780

### B.1 PINAGEM

A pinagem do LCD 16×2 geralmente segue o padrão abaixo. Entretanto, alguns fabricantes podem inverter a ordem dos pinos (recomenda-se a consulta ao manual do fabricante).

Tab. B1: Pinagem de um LCD 16×2.

	Pino	Função	Descrição
	1	Alimentação	VSS (GND)
	2	Alimentação	VCC
	3	VEE	Tensão para ajuste do contraste do LCD
	4	RS	Register Select: 1 = dado, 0 = instrução
	5	R/W	Read/Write: 1 = leitura, 0 = escrita
	6	E	Enable: 1 = habilita, 0 = desabilita
	7	DB0	Barramento de dados
	8	DB1	
	9	DB2	
	10	DB3	
	11	DB4	
	12	DB5	
	13	DB6	
	14	DB7	
	15	LED+ (A)	Anodo do LED de iluminação de fundo
	16	LED - (K)	Catodo do LED de iluminação de fundo



## B.2 CÓDIGOS DE INSTRUÇÕES

Tab. B2: Detalhamento do códigos de instruções.

INSTRUÇÃO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Descrição	Execução
Limpa Display	0	0	0	0	0	0	0	0	0	1	Limpa todo o display e retorna o cursor para a primeira posição da primeira linha.	1,6 ms
Retorno do cursor	0	0	0	0	0	0	0	0	1	-	Retorna o cursor para a 1ª coluna da 1ª linha. Retorna a mensagem previamente deslocada a sua posição original.	1,6 ms
Fixa o modo de Funcionamento	0	0	0	0	0	0	0	1	X	S	Ajusta o sentido de deslocamento do cursor (X=0 p/ a esquerda, X=1 p/ a direita). Determina se a mensagem deve ou não ser deslocada com a entrada de um novo caractere (S = 1, SIM). Esta instrução tem efeito somente durante a leitura e escrita de dados.	40 µs
Controle do Display	0	0	0	0	0	0	1	D	C	B	Liga (D=1) ou desliga display (D=0). Liga (C=1) ou desliga cursor (C=0). Cursor piscante (B=1) se C=1.	40 µs
Desloca cursor ou mensagem	0	0	0	0	0	1	C	R	-	-	Desloca o cursor (C=0) ou a mensagem (C=1) para a direita se R=1 ou esquerda se R=0. Desloca sem alterar o conteúdo da DDRAM	40 µs
Fixa modo de utilização do módulo LCD	0	0	0	0	1	Y	N	F	-	-	Comunicação do módulo com 8 bits (Y=1) ou 4 bits (Y=0). Número de linhas: 1 (N=0) e 2 ou mais (N=1). Matriz do caractere: 5×7 (F=0) ou 5×10 (F=1). Esta instrução deve ser empregada na inicialização.	40 µs
Endereço da CGRAM	0	0	0	1	Endereço da CGRAM						Fixa o endereço da CGRAM para posterior envio ou leitura de um dado (byte).	40 µs
Endereço da DDRAM	0	0	1	Endereço da DDRAM						Fixa o endereço da DDRAM para posterior envio ou leitura de um dado (byte).	40 µs	
Leitura do bit de ocupado e do conteúdo de endereços	0	1	B F	AC						Lê o conteúdo do contador de endereços AC e o BF. O bit 7 do BF indica se a última operação foi concluída (BF=0 concluída, BF=1 em execução).		-
Escreve dado na CGRAM/ DDRAM	1	0	Dado a ser gravado no LCD								Grava o byte presente nos pinos de dados no local apontado pelo contador de endereços (posição do cursor).	40 µs
Lê dado da CGRAM/ DDRAM	1	1	Dado lido do módulo								Lê o byte do local apontado pelo contador de endereços (posição do cursor).	40 µs

Tab. B3: Resumo dos códigos de instruções.

Descrição	Modo	Código Hexa
Controle do display	Liga (sem cursor)	0x0C
	Desliga	0x0A/0x08
Limpa display com retorno do cursor		0x01
Controle do cursor	Liga	0x0E
	Desliga	0x0C
	Desloca p/ a esquerda	0x10
	Desloca p/ a direita	0x14
	Retorno	0x02
	Cursor piscante	0x0D
	Cursor com alternância	0x0F
Sentido de deslocamento do cursor na entrada de um caractere	Para a esquerda	0x04
	Para a direita	0x06
Deslocamento da mensagem na entrada de um caractere	Para a esquerda	0x07
	Para a direita	0x05
Deslocamento da mensagem sem a entrada de caractere	Para a esquerda	0x18
	Para a direita	0x1C
Endereço da primeira posição do cursor	Primeira linha	0x80
	Segunda linha	0xC0

### B.3 ENDEREÇO DOS SEGMENTOS (DDRAM)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Linha 1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Linha 2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

### B.4 CONJUNTO E CÓDIGO DOS CARACTERES

Os principais caracteres reconhecidos pelo módulo LCD seguem o código ASCII. Todavia, existem outros caracteres. Na tabela B.4, encontra-se o conjunto total de caracteres que podem ser apresentados. O código a ser enviado ao LCD é obtido concatenando-se o valor horizontal da parte superior da tabela com o valor vertical do lado esquerdo (o *nibble* alto com

o *nibble* baixo). Por exemplo, para o caractere **L**, o *nibble* alto é **4\_** e o *nibble* baixo é **\_C**, o que resulta no valor hexadecimal 0x4C.

Na tabela B.4, pode-se, ainda, observar os endereços dos 8 caracteres que podem ser criados na CGRAM (0x00 até 0x07).

Tab. B4: Conjunto dos caracteres para um LCD 16 × 2.

NIBBLE		NIBBLE ALTO															
	NIBBLE BAIXO	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
_0	xxxx 0000	CG RAM (1)				0	0	B	P	¿	P					—	9
_1	xxxx 0001	CG RAM (2)				!	1	A	Q	a	q					•	7
_2	xxxx 0010	CG RAM (3)				"	2	B	R	b	r					「	イ
_3	xxxx 0011	CG RAM (4)				#	3	C	S	c	s					」	ウ
_4	xxxx 0100	CG RAM (5)				\$	4	D	T	d	t					、	イ
_5	xxxx 0101	CG RAM (6)				%	5	E	U	e	u					•	オ
_6	xxxx 0110	CG RAM (7)				&	6	F	U	f	v					ヲ	カ
_7	xxxx 0111	CG RAM (8)				'	7	G	U	g	w					フ	チ
_8	xxxx 1000	(1)				(	8	H	X	h	x					イ	ウ
_9	xxxx 1001	(2)				)	9	I	Y	i	y					ウ	ツ
_A	xxxx 1010	(3)				*	#	J	Z	j	z					エ	コ
_B	xxxx 1011	(4)				+	;	K	L	k	{					★	ウ
_C	xxxx 1100	(5)				,	<	L	¥	l	l					ト	ウ
_D	xxxx 1101	(6)				—	=	M	J	m	}					ユ	ズ
_E	xxxx 1110	(7)				.	>	N	^	n	~					ヨ	エ
_F	xxxx 1111	(8)				/	?	O	_	o	+					ッ	ウ