Capítulo 8

Displays de 7 segmentos e displays de cristal líquido

Um display de sete segmentos é um tipo de display (mostrador) barato usado como alternativa a displays de matriz de pontos, mais complexos e caros. Displays de sete segmentos são comumente usados em eletrônica como forma de exibir uma informação numérica sobre as operações internas de um dispositivo.

Uma evolução dos displays de 7 segmentos são os displays alfanuméricos, que podem exibir números e letras. Atualmente, os displays de cristal líquido alfa-numérico apresentam o melhor custo-benefício como interface de exibição de saída. Existem ainda os displays gráficos, que além de letras, número e símbolos, podem exibir praticamente qualquer imagem, de acordo com a resolução do display.

8.1 Displays de 7 segmentos

Um display de sete segmentos é composto de sete elementos (LEDs) que podem ser ligados ou desligados individualmente. Eles podem ser combinados para produzir representações simplificadas de algarismos arábicos.

Frequentemente, os sete segmentos são dispostos de forma oblíqua ou itálica, o que melhora a legibilidade. Os sete segmentos são dispostos num retângulo com dois segmentos verticais em cada lado e um segmento horizontal em cima e outro em baixo. Em acréscimo, o sétimo segmento bissecta o retângulo horizontalmente. Também há displays de catorze e de dezesseis segmentos para exibição plena de caracteres alfanuméricos. Todavia, estes têm sido substituídos por displays de matriz de pontos ou LCDs alfanuméricos.

Para simplificar as conexões, os anodos, ou catodos, de todos os segmentos são conectados a um terminal comum. Assim, dependendo de quais terminais dos LEDs são conectados em comum, podemos ter displays de anodo comum e displays de catodo comum.

Os segmentos do display são definidos pelas letras de "*a*" a "*g*", conforme indicado na figura 8.1, onde o ponto decimal opcional *dp* (um "oitavo segmento") é usado para a exibição de números não inteiros.

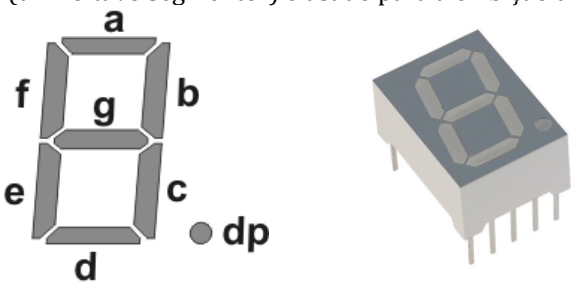

Figura 8.1 - Display de 7 segmentos e sua representação.

Um fato que indica que o microcontrolador é apenas um tipo miniaturizado de computador, projetado para só entender a linguagem de zeros e uns, é expresso plenamente ao tentar exibir qualquer dígito. Ou seja, o microcontrolador não sabe o que são unidades, dezenas ou centenas, nem tampouco conhece a aparência dos dez dígitos que estamos acostumados a usar no sistema arábico. A formação e exibição dos caracteres são feitas acionando-se determinados segmentos e apagando-se outros. Por esta razão, para que um número possa ser apresentado, o programador deve seguir certos critérios.

Antes de tudo, um número de múltiplos dígitos deve ser dividido em unidades, dezenas, centenas, etc. Então, cada um destes dígitos deve ser armazenado em bytes específicos na memória do microcontrolador. Os dígitos obtêm aparência reconhecível através da aplicação de um processo chamado "mascaramento". Em outras palavras, o formato binário de cada dígito é substituído por uma combinação diferente de bits. Por exemplo, o número de 8 bits 0b00001000, que representa 8 em decimal, é substituído pelo número binário 0b01111111, a fim de ativar todos os LEDs, exibindo o dígito 8. O único bit remanescente inativo aqui é reservado para o ponto decimal.

Se uma porta do microcontrolador é conectado ao display de tal forma que o bit 0 ativa o segmento "*a*", o bit 1 ativa o segmento "*b*", e assim sucessivamente, então a tabela abaixo mostra a máscara para cada dígito hexadecimal, considerando que o display seja do tipo catodo comum. Para displays do tipo anodo comum, todos os "uns" (1) da tabela devem ser substituídos por "zeros" (0), e vice-versa. O termo PD é o ponto decimal.

Dígitos Decimais	Segmentos do display								Máscara (g f e d c b a)
	PD	a	b	c	d	e	f	g	
0	0	1	1	1	1	1	1	0	0b00111111
1	0	0	1	1	0	0	0	0	0b00000110
2	0	1	1	0	1	1	0	1	0b01011011
3	0	1	1	1	1	0	0	1	0b01001111
4	0	0	1	1	0	0	1	1	0b01100110
5	0	1	0	1	1	0	1	1	0b01101101
6	0	1	0	1	1	1	1	1	0b01111101
7	0	1	1	1	0	0	0	0	0b00000111
8	0	1	1	1	1	1	1	1	0b01111111
9	0	1	1	1	1	0	1	1	0b01111011
A	0	1	1	1	0	1	1	1	0b01110111
B	0	0	0	1	1	1	1	1	0b00011111
C	0	1	0	0	1	1	1	0	0b01001110
D	0	0	1	1	1	1	0	1	0b00111101
E	0	1	0	0	1	1	1	1	0b01001111
F	0	1	0	0	0	1	1	1	0b01000111

Um exemplo de aplicação é mostrado na figura 8.2, onde o número 5, em binário, é mascarado com o valor 0b01101101 e em seguida é exibido no display de 7 segmentos, no qual cada LED possui um resistor limitador de corrente de 330 Ω.

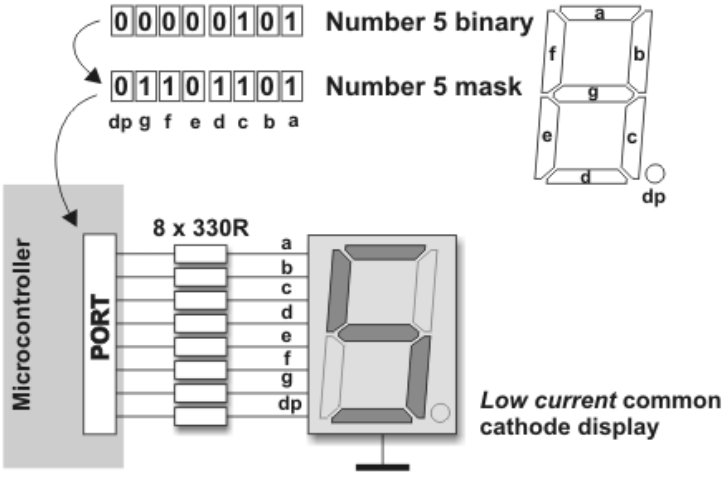

Figura 8.2 – Mascaramento e exibição do dígito 5 no display de sete segmentos.

No quadro abaixo, temos um exemplo simples de decodificação do display de sete segmentos. O programa exibe, continuamente, os dígitos decimais de 0 a 9, sequencialmente, espaçados por um intervalo de 1 segundo, criando uma espécie de contador crescente.

Nesse programa, a função setup( ) configura os pinos 22 a 29 do Arduino Mega 2560 (porta A) como saídas. Um display de 7 segmentos é ligado à porta, com o bit menos significativo conectado ao segmento "*a*". A função loop( ) então envia sequencialmente as máscaras dos dígitos para a porta. O comando delay( ) insere o intervalo de um segundo entre os envios das máscaras.

```
// Decodificação de display de 7 segmentos

void setup()
{
  DDRA = 0xFF;
}

void loop()
{
  PORTA = 0b00111111;
  delay(1000);
  PORTA = 0b00000110;
  delay(1000);
  PORTA = 0b01011011;
  delay(1000);
  PORTA = 0b01001111;
  delay(1000);
  PORTA = 0b01100110;
  delay(1000);
  PORTA = 0b01101101;
  delay(1000);
  PORTA = 0b01111101;
  delay(1000);
  PORTA = 0b00000111;
  delay(1000);
  PORTA = 0b01111111;
  delay(1000);
  PORTA = 0b01101111;
  delay(1000);
}
```

Pela figura 8.2, percebe-se que são necessários pelo menos 7 terminais do microcontrolador para fazer a correta exibição dos dígitos em um display. Imaginemos então a situação em que é necessário fazer o controle de 4 displays simultaneamente. Seriam então necessários 28 terminais para essa tarefa. Entretanto, existe uma técnica de multiplexação de displays que consiste em usar apenas 7 terminais, comuns a todos os displays, além de quatro terminais extras para selecionar um dos displays por vez. Esta é uma técnica que se beneficia de uma propriedade do olho humano, chamada persistência à luz, para criar uma ilusão de ótica. Assim, se a seleção do display for suficientemente rápida, o efeito obtido é como se os 4 displays estivessem ligados ao mesmo tempo, embora apenas um esteja ativo por vez.

Um circuito que executa tal tarefa é mostrado na figura 8.3. Inicialmente, um byte representando as unidades deve ser enviado à porta (PORT2) ao mesmo tempo em que o transistor T1 é ativado pelo pino correspondente na outra porta (PORT1). Após um curto intervalo de tempo, o transistor T1 deve ser desligado, um segundo byte, representando as dezenas, deve ser enviado ao PORT2 e o transistor T2 ativado. Esse processo é repetido ciclicamente, em alta velocidade, para todos os dígitos e transistores correspondentes.

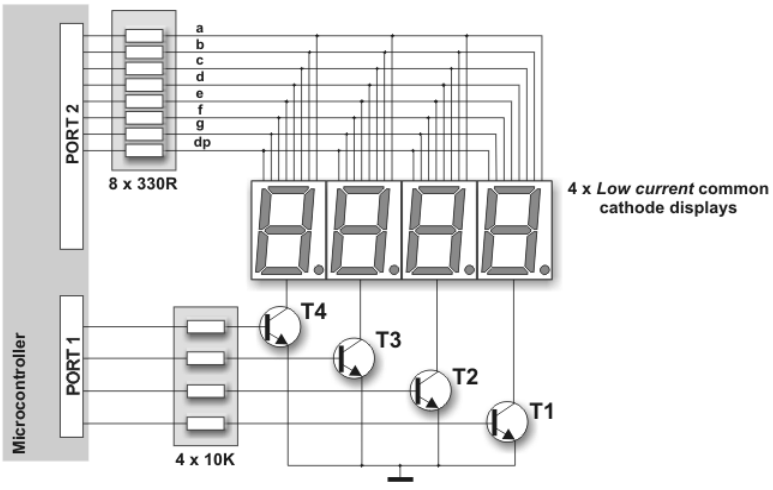

Figura 8.3 – Multiplexação de quatro displays de sete segmentos.

6.2 Matrizes de dados

Uma matriz, ou array, é um conjunto de dados homogêneos referenciado por um identificador. Por dados homogêneos entendem-se variáveis ou dados do mesmo tipo. Assim, uma matriz é um conjunto de variáveis ou dados do mesmo tipo e referenciados por um único identificador. A linguagem Arduino suporta a criação de matrizes multidimensionais. No nosso cotidiano, estamos acostumados a trabalhar com diversos tipos de matrizes unidimensionais, como, por exemplo, listas (uma lista de nomes), bidimensionais (como um tabuleiro de xadrez, ou uma planilha de dados) e as multidimensionais com três ou mais dimensões.

Uma matriz é declara da seguinte forma:

```
tipo_de_dado nome_da_matriz[tamanho];
```

`tipo_de_dado` é um especificador do tipo de dado utilizado em cada um dos elementos da matriz. Podemos criar matrizes com qualquer dos tipos de dados disponíveis na linguagem Arduino;
`nome_da_matriz` é um identificador da matriz. Todas as referências à matriz serão realizadas pelo uso deste identificador;
`tamanho` é o número de elementos que pode ser armazenado na matriz.

Vejamos o exemplo em seguida:

```
int nota[40];          //define uma matriz de 40 números inteiros
char nome[20];         //define uma matriz de 20 caracteres
unsigned int outra[5]; //define uma matriz de 5 números inteiros sem sinal
```

Ao declarar uma variável do tipo matriz, o compilador reserva espaço na memória RAM do microcontrolador para armazenar aquelas variáveis ou dados. Vejamos a representação gráfica da matriz "Matrix[5]":

Matrix [0]
Matrix [1]
Matrix [2]
Matrix [3]
Matrix [4]

O primeiro elemento da matriz é acessado pela declaração "Matrix[0]" e o último elemento pela declaração "Matrix[4]". O primeiro elemento de uma matriz é sempre o de índice zero.

O programa anterior poderia ser reescrito com o uso de matrizes, como mostrado no quadro da página seguinte.

Inicialmente, temos uma declaração de uma matriz de *constantes* cujo nome é tabela. Constantes são valores (numéricos ou não) fixos e que não podem ser alterados pelo programa durante a sua execução. Uma constante é declarada precedida da palavra reservada **const**. A diferença entre uma variável e uma constante é que nas variáveis, o conteúdo pode ser alterado durante a execução do programa, enquanto que nas constantes isso não ocorre. Outra diferença prática é que as variáveis são armazenadas na memória RAM e as constantes são armazenadas na memória de programa do microcontrolador. Uma vez que a memória RAM é de pequeno tamanho, o uso de constantes ajuda o programador a economizar espaço em memória.

A tabela de constantes é do tipo **byte** e possui 10 elementos, referenciados como tabela[0] até tabela[9]. Além da declaração da matriz, esse comando também faz a sua *inicialização*, ou seja, define os valores armazenados em cada elemento da matriz. Por exemplo, o elemento tabela[0] é iniciado com o valor 0b00111111. Os dez valores armazenadas na matriz são as máscaras dos dígitos decimais para o display de sete segmentos.

A função setup( ) configura os pinos 22 a 29 do Arduino Mega 2560 (porta A) como saídas, como feito anteriormente.

Na função loop( ), inicialmente é declarada uma variável do tipo byte chamada *value*. Essa variável é inicializada com o valor 0. O bloco de comandos do laço **while** será executado indefinidamente, visto que a condição a ser avaliada é a palavra reservada "*true*", que sempre retorna verdadeiro. O bloco então escreve na porta A o conteúdo do elemento da tabela referenciado pela variável *value*. Ou seja, inicialmente, o comando `PORTA = tabela[value];` escreve na porta A o valor tabela[0], que no caso é a máscara do dígito 0. Em seguida, espera-se um segundo. A variável *value* é incrementada, assumindo o valor 1 e um teste **if** é feito para verificar se a variável *value* é maior que 9. Como a variável *value* não é maior que 9, o programa volta a escrever na porta A o valor tabela[value], que agora é tabela[1], ou seja, a máscara do dígito 1.

Isso se repete até que sejam escritas as 9 máscaras dos dígitos decimais na porta A. Após isso, o teste **if** resulta em verdadeiro, uma vez que após o último incremento da variável *value* ela assume o valor 10, fazendo com que o comando `value=0`; seja executado e então todo o processo é repetido.

```
// Decodificação de display de 7 segmentos com uso de matrizes

const byte tabela[10]={0b00111111,    //0
                       0b00000110,    //1
                       0b01011011,    //2
                       0b01001111,    //3
                       0b01100110,    //4
                       0b01101101,    //5
                       0b01111101,    //6
                       0b00000111,    //7
                       0b01111111,    //8
                       0b01101111};   //9

void setup()
{
  DDRA = 0xFF;
}

loop()
{
  byte value=0;

  while(true)
  {
    PORTA = tabela[value];
    delay_ms(1000);
    ++value;
    if(value>9)
      value=0;
  }
}
```

6.3 Display de cristal líquido (*Liquid Crystal Display* - LCD)

Este componente é fabricado especificamente para ser usado com microcontroladores, o que significa que não pode ser ativado por circuitos discretos comuns. O modelo aqui descrito é de baixo custo e grande aplicabilidade. É baseado no controlador de displays HD44780 (Hitachi), ou equivalente, e pode exibir mensagens em duas linhas com 16 caracteres cada (display 2x16), onde um caractere é composto por uma matriz de 5x8 ou 5x11 pontos, como mostrado na figura 8.4. Ele pode exibir todas as letras do alfabeto, letras gregas, sinais de pontuação, símbolos matemáticos, etc. Também é possível exibir símbolos personalizados, criados pelo usuário. Outras características úteis incluem mudança automática de mensagem (esquerda e direita), diodo emissor de luz de fundo (backlight), etc.

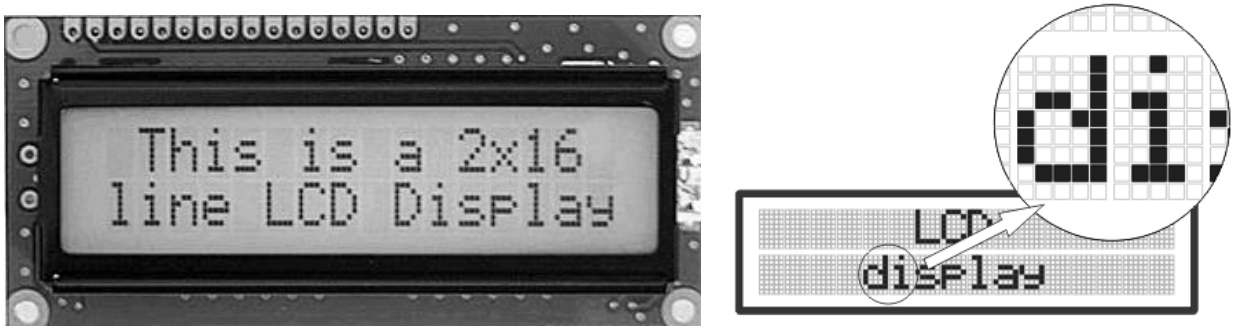

Figura 8.4 – Display 2x16 e detalhe de composição do caractere.

O display é alimentado por uma tensão de 5 V aplicada aos pinos VCC e GND e possui um terminal para ajuste do constraste. Variando a tensão de 0 a VCC aplicada ao terminal *VEE*, podemos variar o contraste da tela do display. Um trimpot é normalmente usado para este fim. Em aplicações mais elaboradas, um sinal de PWM com um filtro passa-baixas é usado para fazer o ajuste de contraste mediante uso de software.

Alguns desses displays têm embutido o backlight (LEDs de cor azul ou verde). Quando usado, um resistor limitador de corrente deve ser conectado em série a um dos pinos do backlight.

Um circuito básico de alimentação e controle de constraste do display é mostrado na figura 8.5.

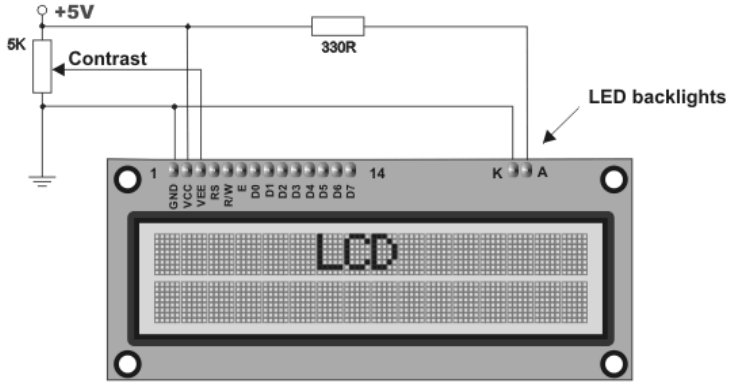

Figura 8.5 – Circuito básico de alimentação e ajuste do contraste do display.

O microcontrolador se comunica com o display, enviando ou recebendo informações, através de um barramento de controle e outro de dados. O barramento de controle é formado pelos terminais *RS, R/W* e *E* que controlam a escrita e leitura no display. O barramento de dados é formado pelos terminais D0 a D7.

Há dois modos de uso do barramento de dados: modo de comunicação de 8 bits e modo de 4-bits. O modo apropriado é selecionado no início da operação do dispositivo, em um processo chamado de "inicialização do display".

O modo de 8 bits usa as linhas D0-D7 para transferir dados de e para o LCD. O objetivo principal do modo de 4 bits é economizar o uso de pinos do microcontrolador. Nesse modo, apenas os 4 bits superiores (D4-D7) são usados para a comunicação, enquanto os outros podem ser deixados desligados. Em contrapartida, isso torna a comunicação um pouco mais lenta.

Outras interfaces com o display de cristal líquido ainda podem ser usadas, como uma interface serial TWI, que exige apenas dois pinos do microcontrolador. Entretanto, trataremos dessas interface no capítulo sobre comunicação serial com o Arduino.

O circuito completo de controle de um LCD é mostrado na figura 8.6.

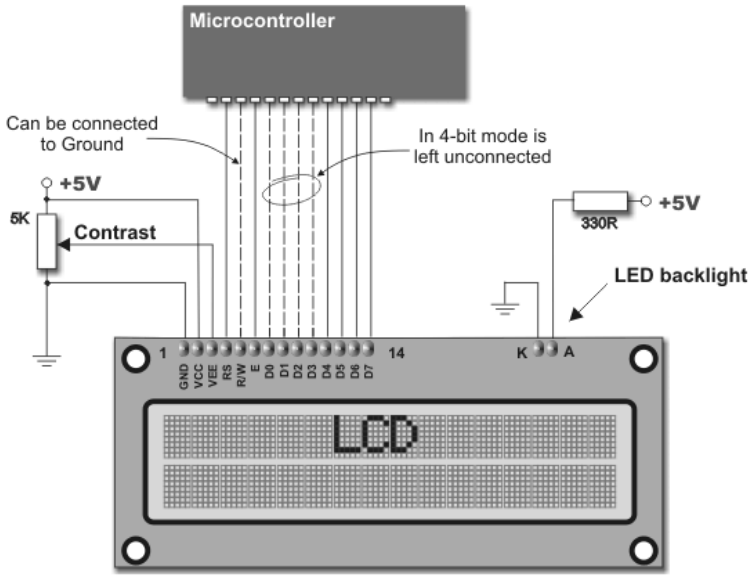

Figura 8.6 – Circuito completo de controle de um LCD.

Para a utilização do display, é necessário usarmos funções especiais que tornam transparente a inicialização e o controle da comunicação com o display, além de permitir a escrita no LCD. Essas funções especiais normalmente são fornecidas em um arquivo de biblioteca.

Uma biblioteca é um trecho de software que fornece funcionalidades específicas a um programa, como por exemplo controlar a posição de um servomotor. O uso de uma biblioteca simplifica o desenvolvimento de uma aplicação, pois o código da biblioteca já está pronto e só precisa ser incorporado ao programa em desenvolvimento para que suas funções possam ser acessadas e utilizadas pelo desenvolvedor. Assim, podemos estender o uso do Arduino, incorporando bibliotecas específicas durante o desenvolvimento de um sketch.

No Arduino existem três tipos diferentes de bibliotecas de software disponíveis:

* Core (biblioteca essencial)
* Padrão
* Adicionais (de terceiros)

A biblioteca essencial vem instalada no IDE do Arduino e é imprescindível para o desenvolvimento de programas, desde os mais simples (como piscar um LED) até projetos complexos. Desta forma, a programação do Arduino fica muito simplificada, pois o programador não tem a necessidade de entender como o código da biblioteca funciona internamente - basta saber como usá-la. Algumas funções comuns fornecidas pela biblioteca core são as funções digitalRead( ), digitalWrite( ), delay( ), entre outras.

As bibliotecas padrão são incluídas na instalação do IDE do Arduino, porém não são incluídas por padrão nos projetos que você cria, pois o Arduino possui recursos de memória limitados, e assim essas bibliotecas somente são incluídas de forma explícita quando você necessita delas. A inclusão de uma biblioteca padrão é feita por meio de uma diretiva #include no início do código do seu sketch. Por exemplo, se você quiser incluir em um sketch a biblioteca padrão Stepper, usada para permitir o controle de motores de passo, você deve incluir a seguinte declaração no início de seu código: #include <Stepper.h>.

Note que o nome da biblioteca deve estar envolvido entre os caracteres **<** e **>**, finaliza com a extensão **.h** e não se deve usar o ponto-e-vírgula no final desta linha. Após a inclusão da biblioteca, pode-se usar suas funções no desenvolvimento de um programa. Para saber quais são essas funções e como utilizá-las, deve-se consultar a documentação específica da biblioteca.

Algumas bibliotecas padrão do Arduino são:

EEPROM - Usada para ler e gravar dados na memória EEPROM do Arduino. No Arduino Mega 2560, a EEPROM tem o tamanho de 4096 bytes.
Ethernet - Permite conectar o Arduino à Internet ou à rede local usando um shield Ethernet.
GSM - Conectar a uma rede GSM/GPRS usando um shield GSM.
LiquidCrystal - Com essa biblioteca podemos controlar displays de cristal líquido (LCD).
SD - Usada para que seja possível escrever e ler dados em cartões de memória SD/SDHC.
Servo - Controlar motores servo
SPI - Comunicação com dispositivos usando o barramento SPI (Serial Peripheral Interface)
SoftwareSerial - Permite a comunicação serial usando os pinos digitais da placa.
Stepper - Controlar motores de passo
WiFi - Permite conexão à rede local e Internet por meio de um shield WiFi.
Wire - Permite enviar/receber dados em uma interface TWI/I2C em uma rede de dispositivos ou sensores.

Existem algumas bibliotecas padrão que somente estão disponíveis em alguns modelos específicos de Arduino, como por exemplo a biblioteca **AudioZero**, que permite reproduzir arquivos de áudio de um cartão SD em uma placa Arduino Zero.

As bibliotecas adicionais (de terceiros) são disponibilizadas por desenvolvedores que contribuem voluntariamente com software para a plataforma e não são distribuídas por padrão com o IDE do Arduino. Para usá-las, deve-se baixá-las e então efetuar sua instalação por meio do IDE. Elas oferecem funções adicionais a bibliotecas existentes ou novas funcionalidades não presentes em nenhuma biblioteca padrão, permitindo estender o uso do Arduino de forma praticamente ilimitada.

No exemplo mostrado a seguir, um Arduino UNO e um LCD são usados para exibir as mensagens “Curso de Arduino” e “Escrita no LCD”, alternadamente. A interligação dos terminais do display com o Arduino deve obedecer à seguinte ordem:
- pino RS do LCD no pino digital 12 do Arduino
- pino E do LCD no pino digital 11 do Arduino
- pino D4 do LCD no pino digital 5 do Arduino
- pino D5 do LCD no pino digital 4 do Arduino
- pino D6 do LCD no pino digital 3 do Arduino
- pino D7 do LCD no pino digital 2 do Arduino

O circuito com as conexões entre o Arduino e o display é mostrado na figura 8.7 e o código é mostrado em seguida.

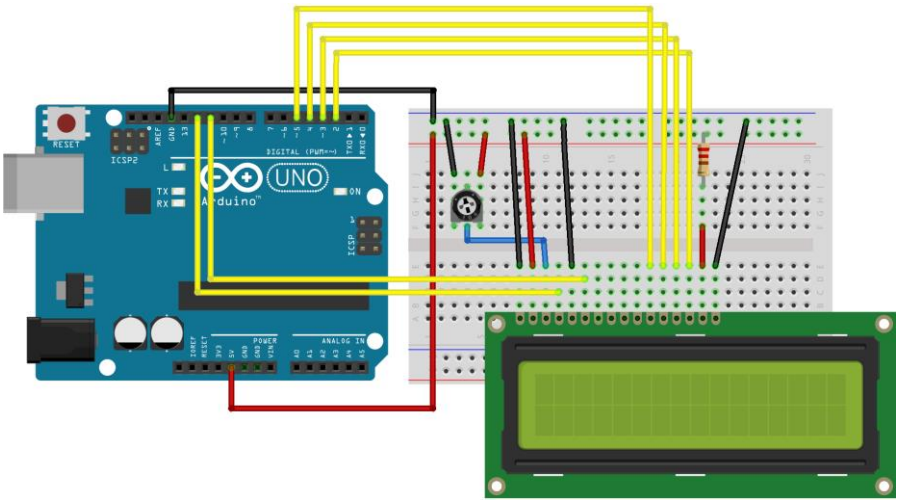

Figura 8.7 – Arduino UNO controlando um LCD.

```
#include <LiquidCrystal.h>

#define  rs  12
#define  en  11
#define  d4  5
#define  d5  4
#define  d6  3
#define  d7  2

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  lcd.begin(16, 2);
}

void loop()
{
  lcd.print("Curso de Arduino");
  delay(1000);
  lcd.clear();
  lcd.print("Escrita no LCD");
  delay(1000);
  lcd.clear();
}
```

As funções referentes ao controle do LCD estão dentro do arquivo chamado LiquidCrystal.h, e foram incluídas no programa com a diretiva **#include** (falaremos mais sobre funções nos próximos capítulos).

A diretiva **#define** é utilizada para definir os números dos pinos que serão usados nos barramentos de controle e dados para o controle do LCD. A linha `LiquidCrystal lcd(rs, en, d4, d5, d6, d7);` é uma instanciação de um objeto da classe LyquidCrystal, definido na biblioteca incluída, ao qual é dado o nome de lcd, além de ser feita a atribuição dos pinos de controle a partir das definições anteriormente declaradas.

Na função setup( ), a linha `lcd.begin(16, 2)` é um método que inicializa o display de 16 caracteres e duas linhas. Esse método deve ser chamado antes de usar o display, pois é nele que é estabelecido o modo de comunicação de 4 bits além de outras configurações iniciais. Na função loop( ), a linha `lcd.print("Curso de Arduino");` também é um método que, nesse caso, imprime a sequência de caracteres "Curso de Arduino", sem as aspas, na primeira linha do display. O texto permanece em exibição por um segundo e, em seguida, o texto é limpo pelo método lcd.clear( ). A linha seguinte imprime a outra sequência de caracteres "Escrita no LCD", também permanecendo por um segundo na tela e, finalmente, a tela é mais uma vez limpa. O processo se repete indefinidamente com as novas chamadas da função loop( ).

A biblioteca LiquidCrystal.h possui cerca de 20 métodos, com diversas finalidades como iniciar a impressão de textos a partir de qualquer posição na tela, exibir e piscar um cursor, deslocar o texto do LCD para a direita ou esquerda, criar caracteres especiais, entre outros. Isso garante uma enorme flexibilidade no uso do display para aplicações de interface homem-máquina. ■