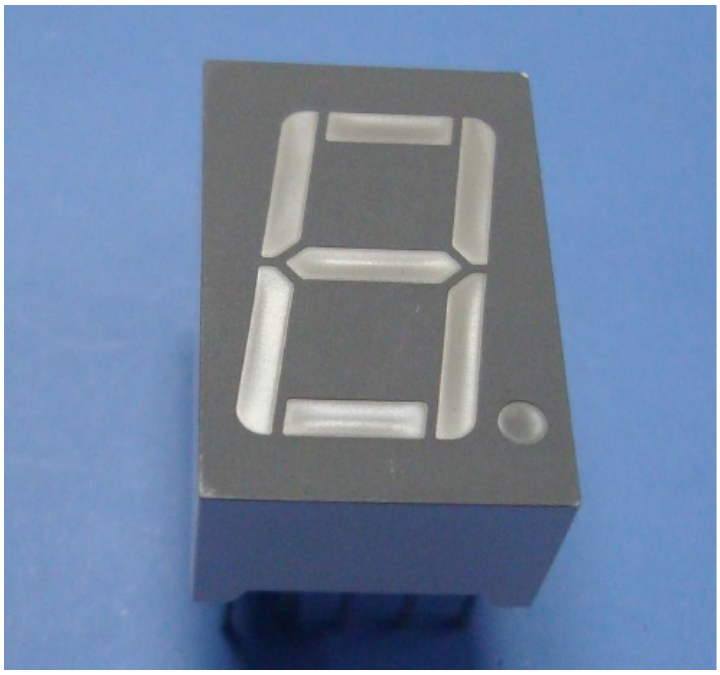
Display de 7 Segmentos e Portas Digitais dos Arduino

Displays

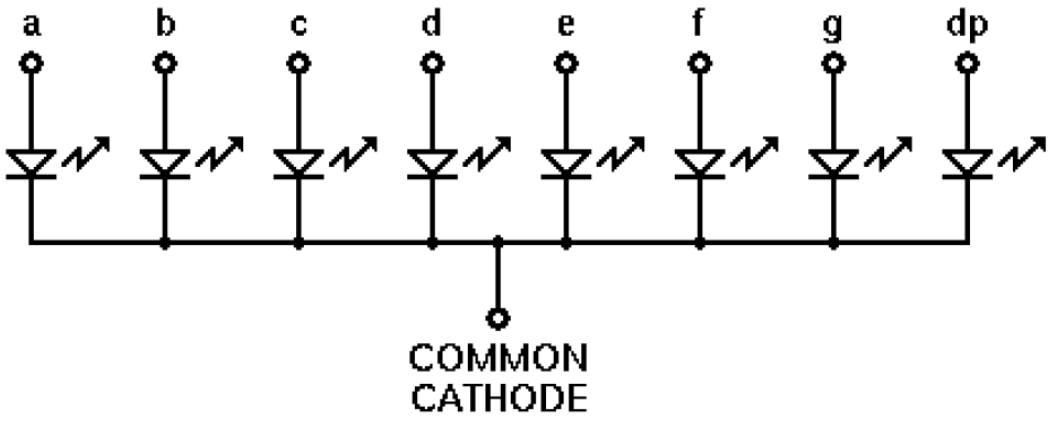
Para muitos aplicativos, não há necessidade de usar um *display* de cristal líquido mais caro para exibir dados. Um simples um display de sete segmentos é suficiente.

O *display* de sete segmentos tem sete LEDs dispostos em forma de número oito. Eles são fáceis de usar e têm o custo baixo. A figura abaixo mostra um *display* típico de sete segmentos.



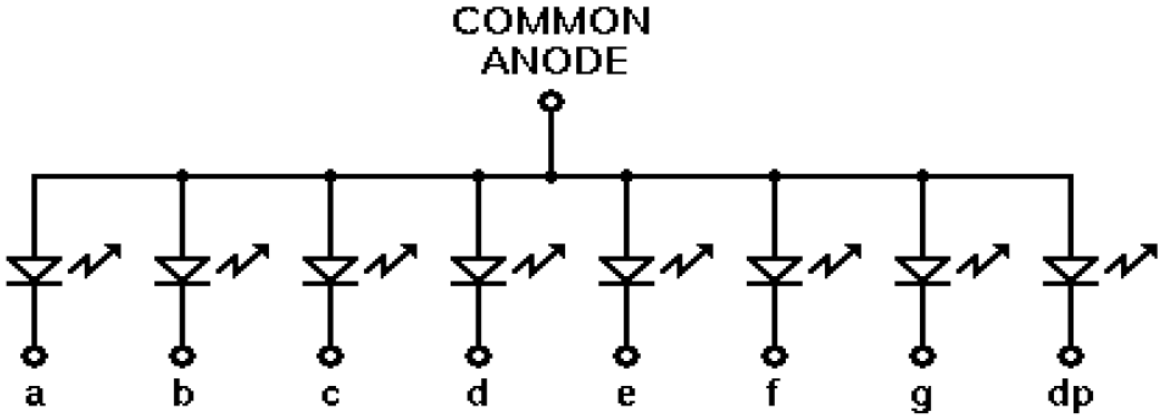
Display de sete segmentos

Displays de sete segmentos são de dois tipos: anodo comum e catodo comum. A estrutura interna de ambos os tipos é quase a mesma. A diferença é a polaridade dos LEDs e do terminal comum. Em um display catodo comum (o que usamos nos experimentos), todos sete LEDs mais o LED ponto têm os catodos conectados aos pinos 3 e pino 8. Para usar este display, precisamos conectar o terra (GND) aos pinos 3 e 8 e conectar + 5V através de resistores aos outros pinos para fazer os segmentos individuais acenderem. A figura a seguir mostra a estrutura interna do display de sete segmentos catodo comum onde dp é o ponto.



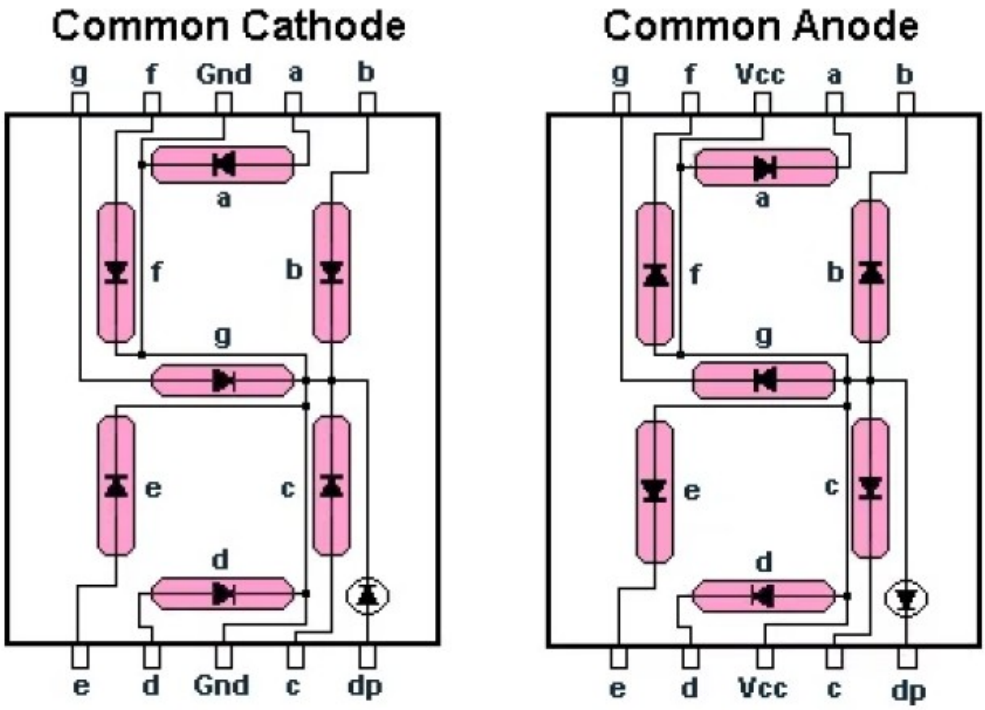
Disposição dos LEDs de um display de catodo comum

Em um display anodo comum, o terminal positivo de todos os oito LEDs são conectados juntos e, em seguida, conectado aos pinos 3 e 8. Para ligar um segmento individual, deve-se ligar o pino 3 e/ou 8 em 5V e aterrar (colocar 0) através de resistores em um dos pinos individuais. O diagrama a seguir mostra a estrutura do display de sete segmentos anodo comum.



Disposição dos LEDs de um display de anodo comum

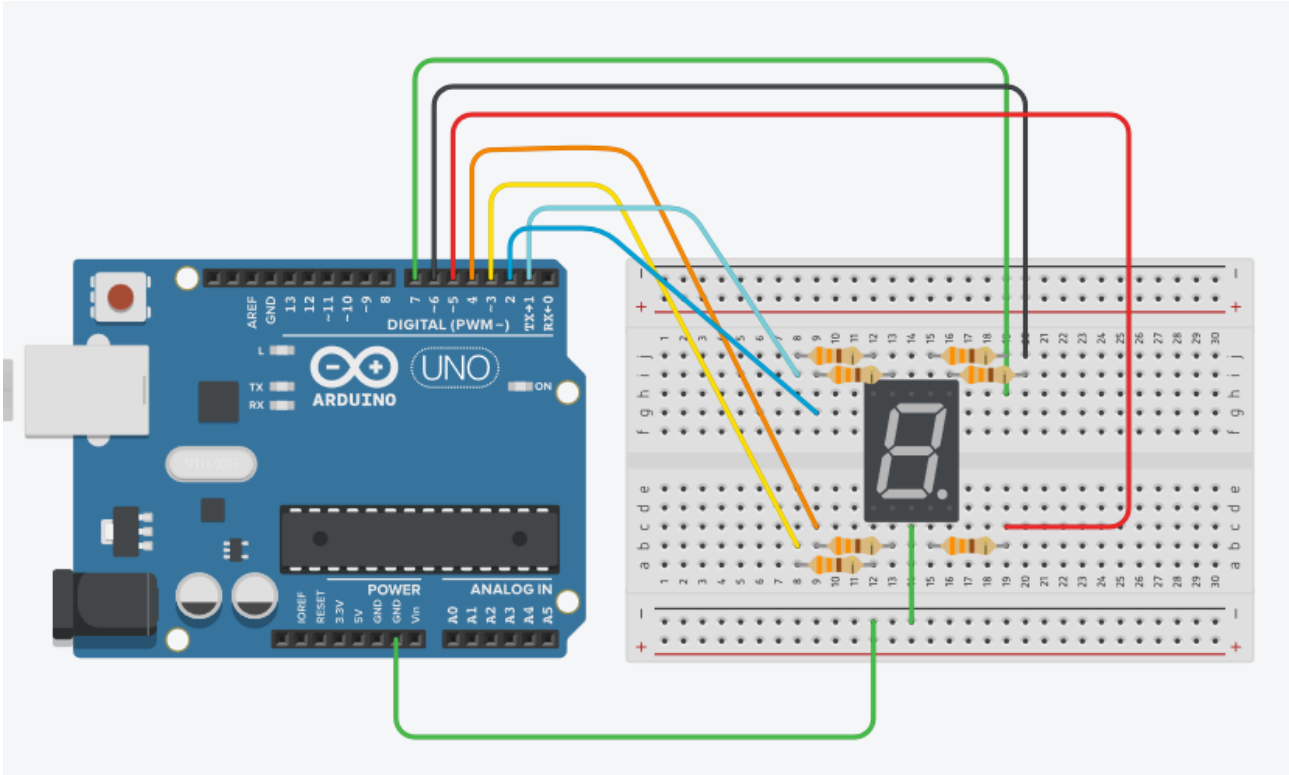
Os 7 segmentos mais o ponto estão rotulados como na figura abaixo.



Pinagem e rotulagem dos segmentos.

O Exemplo a seguir mostra uma ligação do display de 7 segmentos catodo comum com o Arduino. Como temos uma diversidade de possibilidades de ligação dos pinos, esse tipo de ligação pode ser alterada. Neste exemplo temos o pino 3 ligado ao GND, o **a** ao 7, **b** ao 6, **c** ao 5, **d** ao 4, **e** ao 3, **f** ao 2 e **g** ao 1. Os pinos 1 e especialmente o 0 são particularmente problemáticos pois servem para a comunicação entre o Arduino e o computador. Podem ser usados mas com certo cuidado, sendo que algumas vezes deve-se liberar os pinos 0 e 1 para programação e logo depois fazer as ligações do hardware.

Exemplo 1 – Ligação de um display catodo comum ao Arduino
Hardware:



Ligação elétrica de um display catodo comum com o Arduino

Programa:

Neste programa, colocamos todos os pinos ligados como saída e escrevemos simplesmente o número 1 no display que fica fixo, colocando os pinos 6 e 5 (**a** e **b**)em *HIGH*

```
void setup() {
  pinMode (7, OUTPUT);
  pinMode (6, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode (4, OUTPUT);
  pinMode (3, OUTPUT);
  pinMode (2, OUTPUT);
  pinMode (1, OUTPUT);
}

void loop() {
  digitalWrite (6, HIGH);
  digitalWrite (5, HIGH);
}
```

Não foram usados os #defines pois olharemos com mais atenção a outro tipo de programa, usando a manipulação das portas do Arduino.

Portas digitais do Arduino

As portas digitais permitem um acesso de baixo nível para ter acesso mais rapidamente aos pinos de E/S do Arduino. O ATmega328 presente no Arduino Uno tem as seguintes distribuições por 3 portas:

- B – Pinos 8 a 13
- C – Pinos analógicos
- D – Pinos 0 a 7

Existem 3 registradores associados a cada porta:

- DDR – São registradores que indicam o sentido de cada pino sendo 0 para entrada e 1 para saída. Por exemplo o registrador DDRD pode ter um valor atribuído a ele: DDRD = B0100101 – isto indica que os pinos 0, 2 e 6 são definidos como saída e os outros (1, 3, 4, 5 e 7) são definidos como entrada. Veja que a porta D se refere aos pinos digitais 0 a 7 e a correspondência se faz com o 0 correspondendo ao bit menos significativo do DDRD e o 7 ao bit mais significativo do DDRD.
- PORT – São registradores que controlam se a saída é alta (5V) ou baixa (GND). Os valores podem ser alterados se os pinos forem configurados como saída. Como exemplo imagine que os pinos 4 a 7 são definidos como saídas pelo DDRD ou função pinMode(). Se fizermos uma atribuição do PORTD de B11000000, teremos os pinos 6 e 7 com 1 portanto 5V e o restante em GND.
- PIN – São registradores que podem fazer a leitura dos pinos que forem configurados como entradas. Enquanto podemos escrever nos registradores DDR e PORT, os registradores PIN só podem ser lidos. Como exemplo, suponhamos que os pinos 0 a 3 estão configurados como entrada. Se lermos o valor binário 00001100 no registrador PIND significa que os pinos 2 e 3 estão com valores de entrada de 5V o os pinos 0 e 1 com valores de entrada 0V (GND)

Exemplo 2

Usando o mesmo circuito para o *display* do exemplo 1, usaremos a porta D e os registradores DDRD e PORTD associados aos pinos ligados ao display.

Programa:

```
// Este programa utiliza a manipulação da porta D para
// escrever rapidamente nos LEDs do display

byte disp[2] = {B11111100, B01100000};
// esta variável global vetor será usado para preencher
// o PORTD com valores que acendem os LEDs do display
// para exibir 0 ou 1

void setup() {
  DDRD |= B11111110; //Máscara que seta de 1 a 6 e mantém
                     //o último valor do pino 0
}

void loop() {
  PORTD = disp[0]; // exibe o número 0
  delay (1000);    // aguarda 1s
  PORTD = disp[1]; // exite o número 1
  delay (1000);    // aguarda 1s
}
```

Vale ressaltar que todo programa deve ser indentado para uma melhor visualização. Programas mal indentados são difíceis de se entender o código e muitos erros podem ser difíceis de se encontrar devido a falta desta prática de programação. Uma indentação básica de 2 espaços depois de uma chave aberta (ou *if, else, while, for*, mesmo sem chave aberta) é suficiente para que o programa fique legível. Após fechar a chave ou (acabar algum daqueles comandos citados), deve se voltar os dois espaços anteriormente dados. Note que a grande maioria dos IDEs de programas de alto nível já ajuda na prática da indentação, mantendo a coluna após um *enter*.

A primeira declaração é de uma variável global *disp*. Esta variável é um vetor de duas posições e essas posições são inicializadas com a atribuição dos valores entre colchetes. Portanto *disp[0]* equivale a B11111100 e *disp[1]* equivale a B01100000 neste programa. Vejam a relação imediata destes valores com os segmentos do display: 11111100 em binário corresponde a **abcdefg_** do display (sendo que o pino 0 não está ligado a nenhum segmento). Isto significa que se fizermos uma atribuição de valor para a PORTD, **a**, **b**, **c**, **d**, **e**, e **f** serão 1, e isto significa exibir o número 0 no *display*. A mesma justificativa serve para *disp[1]* exibir o número 1 no *display*.

Em *setup()* temos a atribuição DDRD |= B11111110. Lembrando da contração possível dos operadores em C, temos que equivale a DDRD = DDRD | B11111110. Como aprendemos em operador bit a bit o byte 11111110 é uma máscara e com um OR (operador |) sabemos que ele seta os bits de DDRD onde a máscara tem 1 e mantém o valor anterior onde tem 0 na máscara. Isto é especialmente importante aqui pois queremos que os pinos 1 a 7 sejam saída e que o pino 0 se mantenha sem alteração pois é um pino de comunicação do Arduino.

Na função loop() fazemos a atribuição de DDRD ao disp[0] o que já mostramos que faz com que o número 0 seja exibido no display. Em seguida um delay de 1000ms, então exibe o número 1 no display fazendo a atribuição do PORTD ao *disp[1]*. Vejam que a atribuição do PORTD faz com

que os pinos configurados como saída tenham seus valores alterados e possam ligar ou apagar os segmentos do *display*.

Este tipo de abordagem por manipulação por portas é um artifício mais rápido e existem inúmeros exemplos nos sites dedicados à programação de Arduino que não usam manipulação de portas. Experimente estes exemplos destes sites.

É importante também ressaltar que este tipo de abordagem por porta tem problema de portabilidade com os inúmeros Arduinos e seus microcontroladores. Para tornar portável pode ser que uma quantidade de trabalho extra seja necessário.