Capítulo 3

O nível lógico digital

Na parte inferior da hierarquia da Figura 1.2 encontramos o nível lógico digital, o real hardware do computador. Neste capítulo, examinaremos muitos aspectos da lógica digital, como um fundamento para o estudo de níveis mais altos em capítulos subsequentes. Esse assunto está no limiar entre a ciência da computação e a engenharia elétrica, mas o material é independente, portanto, não há necessidade de experiência prévia de hardware nem de engenharia para entendê-lo.

Os elementos básicos que fazem parte de todos os computadores digitais são surpreendentemente simples. Iniciaremos nosso estudo examinando esses elementos básicos e também a álgebra especial de dois valores (álgebra booleana) usada para analisá-los. Em seguida, examinaremos alguns circuitos fundamentais que podem ser construídos usando simples combinações de portas, entre eles os circuitos que efetuam a aritmética. O tópico que vem depois desse é o modo como essas portas podem ser combinadas para armazenar informações, isto é, como as memórias são organizadas. Logo após, chegamos à questão das CPUs e, em especial, de como é a interface entre CPUs de um só chip, a memória e os dispositivos periféricos. Mais adiante neste capítulo serão estudados diversos exemplos da indústria de computadores.

3.1 Portas e álgebra booleana

Circuitos digitais podem ser construídos com um pequeno número de elementos primitivos combinando-os de inúmeras maneiras. Nas seções seguintes, descreveremos tais elementos, mostraremos como eles podem ser combinados e introduziremos uma poderosa técnica matemática que pode ser usada para analisar seu comportamento.

3.1.1 Portas

Um circuito digital é aquele em que estão presentes somente dois valores lógicos. O normal é que um sinal entre 0 e 0,5 volt represente um valor (por exemplo, 0 binário) e um sinal entre 1 e 1,5 volt represente o outro valor (por exemplo, 1 binário). Não são permitidas tensões fora dessas duas faixas. Minúsculos dispositivos eletrônicos, denominados **portas** (*gates*), podem calcular várias funções desses sinais de dois valores. Essas portas formam a base do hardware sobre a qual todos os computadores digitais são construídos.

Os detalhes do funcionamento interno das portas estão fora do escopo deste livro, pois pertencem ao **nível de dispositivo**, que está abaixo do nível 0. Não obstante, agora vamos divagar um pouco e examinar rapidamente a ideia básica, que não é difícil. No fundo, toda a lógica digital moderna se apoia no fato de que um transistor pode funcionar como um comutador binário muito rápido. Na Figura 3.1(a), mostramos um transistor bipolar (representado pelo círculo) inserido em um circuito simples. Esse transistor tem três conexões com o mundo exterior: o **coletor**, a **base** e o **emissor**. Quando a voltagem de entrada, V_{in}, está abaixo de certo valor crítico, o transistor desliga e age como uma resistência infinita. Isso faz com que a saída do circuito, V_{out}, assuma um valor próximo a V_{cc}, uma voltagem regulada externamente, em geral +1,5 volt para esse tipo de transistor. Quando V_{in} excede o valor crítico, o transistor liga e age como um fio, fazendo V_{out} ficar conectado com a terra (por convenção, 0 volt).

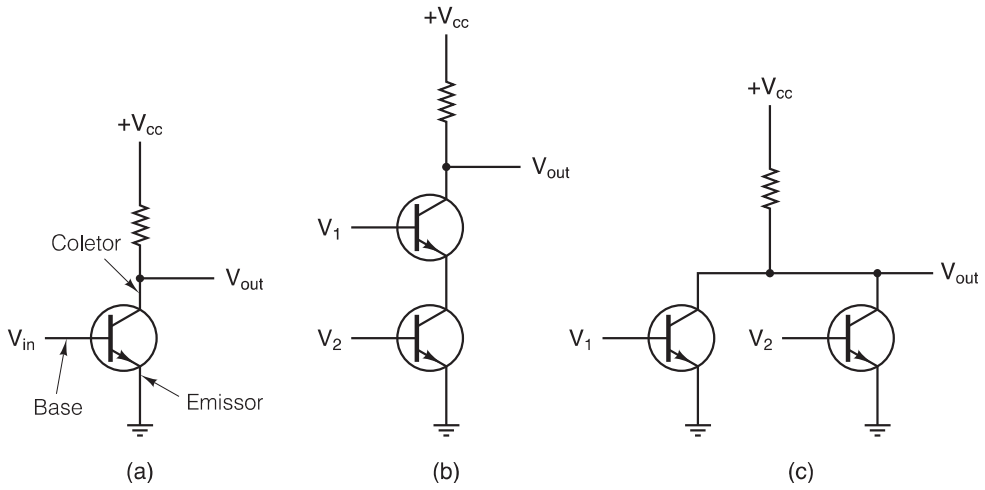
O importante é notar que, quando V_{in} é baixa, V_{out} é alta, e vice-versa. Assim, esse circuito é um inversor, que converte um 0 lógico em um 1 lógico e um 1 lógico em um 0 lógico. O resistor (linha serrilhada) é necessário para limitar a quantidade de corrente drenada pelo transistor, de modo que ele não queime. O tempo típico exigido para passar de um estado para outro é tipicamente de um nanossegundo ou menos.

Na Figura 3.1(b), dois transistores estão ligados em série. Se ambas, V_1 e V_2, forem altas, ambos os transistores conduzirão e V_{out} cairá. Se qualquer das entradas for baixa, o transistor correspondente se desligará e a saída será alta. Em outras palavras, V_{out} será baixa se, e somente se, ambas, V_1 e V_2, forem altas.

Na Figura 3.1(c), os dois transistores estão ligados em paralelo em vez de em série. Nessa configuração, se qualquer das entradas for alta, o transistor correspondente ligará e conectará a saída com a terra. Se ambas as entradas forem baixas, a saída permanecerá alta.

Esses três circuitos, ou seus equivalentes, formam as três portas mais simples e são denominadas portas NOT, NAND e NOR, respectivamente. Portas NOT costumam ser denominadas **inversoras**; usaremos os dois termos indiferentemente. Se agora adotarmos a convenção de que "alta" (V_{cc} volts) é um 1 lógico e "baixa" (terra) é um 0 lógico, podemos expressar o valor de saída como uma função dos valores de entrada. Os símbolos usados para representar essas portas são mostrados nas figuras 3.2(a)-(c) junto com o comportamento funcional de cada circuito. Nessas figuras, A e B são entradas e X é a saída. Cada linha especifica a saída para uma combinação diferente das entradas.

Figura 3.1 (a) Inversor de transistor. (b) Porta NAND. (c) Porta NOR.



Se o sinal de saída da Figura 3.1(b) for alimentado em um circuito inversor, obtemos outro circuito com o inverso exato da porta NAND, a saber, um cuja saída é 1 se, e somente se, ambas as entradas forem 1. Esse circuito é denominado uma porta AND; seu símbolo e descrição funcional são dados na Figura 3.2(d). De modo semelhante, a porta NOR pode ser conectada a um inversor para produzir um circuito cuja saída é 1 se quaisquer das saídas, ou ambas, for um 1, mas 0 se ambas as entradas forem 0. O símbolo e a descrição funcional desse circuito, denominado uma porta OR, são dados na Figura 3.2(e). Os pequenos círculos usados como parte dos símbolos para o inversor, porta NAND e porta NOR, são denominados **bolhas de inversão**. Também são usadas em outros contextos para indicar um sinal invertido.

As cinco portas da Figura 3.2 são os principais elementos de construção do nível lógico digital. A discussão precedente deve ter deixado claro que as portas NAND e NOR requerem dois transistores cada, ao passo que as portas AND e OR requerem três cada. Por essa razão, muitos computadores são baseados em portas NAND e NOR em vez das portas mais conhecidas, AND e OR. (Na prática, todas as portas são executadas de modo um pouco diferente, mas as NAND e NOR ainda são mais simples do que as AND e OR.) A propósito, vale a pena observar que as portas podem perfeitamente ter mais de duas entradas. Em princípio, uma porta NAND, por exemplo, pode ter, arbitrariamente, muitas entradas, mas na prática não é comum encontrar mais de oito.

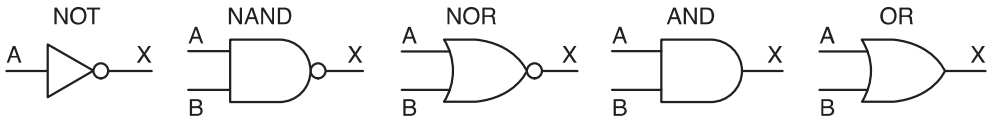
Embora a questão do modo como são construídas as portas pertença ao nível do dispositivo, gostaríamos de mencionar as principais famílias de tecnologia de fabricação porque elas são citadas com muita frequência. As duas tecnologias principais são **bipolar** e **MOS** (**Metal Oxide Semiconductor – semicondutor de óxido metálico**). Os dois principais tipos bipolares são a **TTL** (**Transistor-Transistor Logic – lógica transistor-transistor**), que há muitos anos é o burro de carga da eletrônica digital, e a **ECL** (**Emitter-Coupled Logic – lógica de emissor acoplado**), que era usada quando se requeria uma operação de velocidade muito alta. Para circuitos de computador, o que predomina agora é a tecnologia MOS.

Portas MOS são mais lentas do que as TTL e ECL, mas exigem bem menos energia elétrica e ocupam um espaço muito menor, portanto, um grande número delas pode ser compactado e empacotado. Há muitas variedades de MOS, entre as quais PMOS, NMOS e CMOS. Embora os modos de construção dos transistores MOS e dos transistores bipolares sejam diferentes, sua capacidade de funcionar como comutadores eletrônicos é a mesma. A maioria das CPUs e memórias modernas usa tecnologia CMOS, que funciona a +1,5 volt. E isso é tudo o que diremos sobre o nível de dispositivo. O leitor interessado em continuar o estudo desse nível deve consultar as leituras sugeridas na Sala Virtual.

3.1.2 Álgebra booleana

Para descrever os circuitos que podem ser construídos combinando portas, é necessário um novo tipo de álgebra, no qual variáveis e funções podem assumir somente os valores 0 e 1. Essa álgebra é denominada **álgebra booleana**, nome que se deve a seu descobridor, o matemático inglês George Boole (1815–1864). Em termos

Figura 3.2 Símbolos e comportamento funcional das cinco portas básicas.



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

estritos, estamos nos referindo a um tipo específico de álgebra booleana, uma **álgebra de comutação**, mas o termo "álgebra booleana" é tão utilizado no lugar de "álgebra de comutação" que não faremos a distinção.

Assim como há funções na álgebra "ordinária" (isto é, a álgebra do colegial), também há funções na álgebra booleana. Uma função booleana tem uma ou mais variáveis de entrada e produz um resultado que depende somente dos valores dessas variáveis. Uma função simples, f, pode ser definida ao se dizer que $f(A)$ é 1 se A for 0 e $f(A)$ é 0 se A for 1. Essa função é a função NOT da Figura 3.2(a).

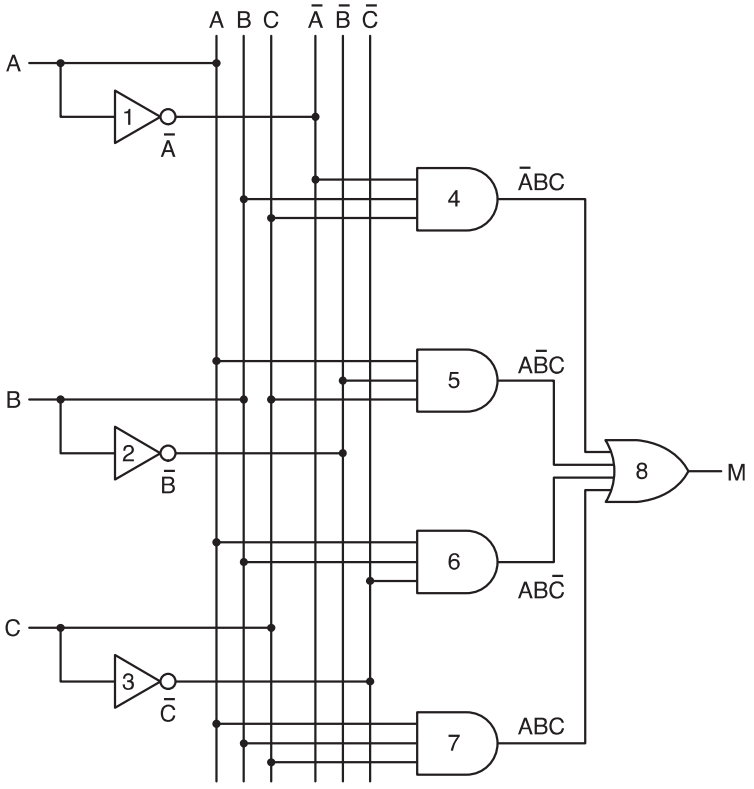
Como uma função booleana de n variáveis só tem 2^n combinações possíveis de valores de entrada, ela pode ser completamente descrita por uma tabela com 2^n linhas, na qual cada linha informa o valor da função para uma combinação diferente de valores de entrada. Ela é denominada **tabela verdade**. As tabelas da Figura 3.2 são todas exemplos de tabelas verdade. Se concordarmos em sempre listar as linhas de uma tabela verdade em ordem numérica (base 2), isto é, para duas variáveis na ordem 00, 01, 10 e 11, a função pode ser completamente descrita pelo número binário de 2^n bits obtido pela leitura vertical da coluna de resultado da tabela verdade. Assim, NAND é 1110, NOR é 1000, AND é 0001 e OR é 0111. É óbvio que só existem 16 funções booleanas de duas variáveis, correspondentes às 16 possíveis sequências de 4 bits resultantes. Por outro lado, a álgebra ordinária tem um número infinito de funções de duas variáveis, nenhuma das quais pode ser descrita por meio de uma tabela de saídas para todas as entradas possíveis, porque cada variável pode assumir qualquer valor de um número infinito de valores possíveis.

A Figura 3.3(a) mostra a tabela verdade para uma função booleana de três variáveis: $M = f(A, B, C)$. Essa função é a de lógica majoritária, isto é, ela é 0 se a maioria de suas entradas for 0, e 1 se a maioria de suas entradas for 1. Embora qualquer função booleana possa ser completamente especificada dada sua tabela verdade, à medida que aumenta o número de variáveis, essa notação fica cada vez mais trabalhosa. Portanto, costuma-se usar outra notação no lugar dela.

Figura 3.3 (a) Tabela verdade para a função majoritária de três variáveis. (b) Circuito para (a).

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)

Para ver como ocorre essa outra notação, observe que qualquer função booleana pode ser especificada ao se dizer quais combinações de variáveis de entrada dão um valor de saída igual a 1. Para a função da Figura 3.3(a), há quatro combinações de variáveis de entrada que fazem com que M seja 1. Por convenção, marcaremos a variável de entrada com uma barra para indicar que seu valor é invertido. A ausência de uma barra significa que o valor não é invertido. Além disso, usaremos a multiplicação implícita ou um ponto para representar a função booleana AND e + para representar a função booleana OR. Assim, por exemplo, $A\overline{B}C$ assume o valor 1 somente quando $A = 1$ e $B = 0$ e $C = 1$. Além disso, $A\overline{B} + B\overline{C}$ é 1 somente quando ($A = 1$ e $B = 0$) ou ($B = 1$ e $C = 0$). As quatro linhas da Figura 3.3(a) que produzem bits 1 na saída são: $\overline{A}BC$, $A\overline{B}C$, $AB\overline{C}$ e ABC. A função, M, é verdadeira (isto é, 1) se qualquer uma dessas quatro condições for verdadeira; daí, podemos escrever

$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

como um modo compacto de dar a tabela verdade. Assim, uma função de n variáveis pode ser descrita como se desse uma "soma" de no máximo 2^n termos de "produtos" de n variáveis. Essa formulação é de especial importância, como veremos em breve, pois leva diretamente a uma execução da função que usa portas padronizadas.

É importante ter em mente a distinção entre uma função booleana abstrata e sua execução por um circuito eletrônico. Uma função booleana consiste em variáveis, como A, B e C, e operadores booleanos, como AND, OR e NOT. Ela é descrita por uma tabela verdade ou por uma função booleana como

$$F = A\overline{B}C + AB\overline{C}$$

Uma função booleana pode ser executada por um circuito eletrônico (muitas vezes de vários modos diferentes) usando sinais que representam as variáveis de entrada e saída e portas como AND, OR e NOT. Em geral, empregaremos a notação AND, OR e NOT quando nos referirmos aos operadores booleanos, e AND, OR e NOT quando nos referirmos a portas, embora essa notação quase sempre seja ambígua em se tratando de indicar funções ou portas.

3.1.3 Execução de funções booleanas

Como já mencionamos, a formulação de uma função booleana como uma soma de até 2^n termos produtos leva a uma possível implementação. Usando a Figura 3.3 como exemplo, podemos ver como essa implementação é efetuada. Na Figura 3.3(b), as entradas, A, B e C, aparecem na extremidade esquerda, e a função de saída, M, na extremidade direita. Como são necessários complementos (inversos) das variáveis de entrada, eles são gerados tomando as entradas e passando-as pelos inversores rotulados 1, 2 e 3. Para evitar atravancar a figura, desenhamos seis linhas verticais, três das quais conectadas às variáveis de entrada e três aos complementos dessas variáveis. Tais linhas oferecem uma fonte conveniente para as entradas das portas subsequentes. Por exemplo, as portas 5, 6 e 7 usam A como uma entrada. Em um circuito real, essas portas provavelmente estariam ligadas direto a A sem usar nenhum fio "vertical" intermediário.

O circuito contém quatro portas AND, uma para cada termo da equação para M (isto é, uma para cada linha da tabela verdade que tenha um bit 1 na coluna de resultado). Cada porta AND calcula uma linha da tabela verdade, como indicado. Por fim, todos os termos produtos alimentam a porta lógica OR para obter o resultado final.

O circuito da Figura 3.3(b) usa uma convenção que utilizaremos repetidas vezes neste livro: quando duas linhas se cruzam, não há nenhuma ligação implícita a menos que haja um ponto negro bem visível na intersecção. Por exemplo, a saída da porta 3 cruza todas as seis linhas verticais, mas está ligada apenas a $\overline{C}$. É bom lembrar que alguns autores usam outras convenções.

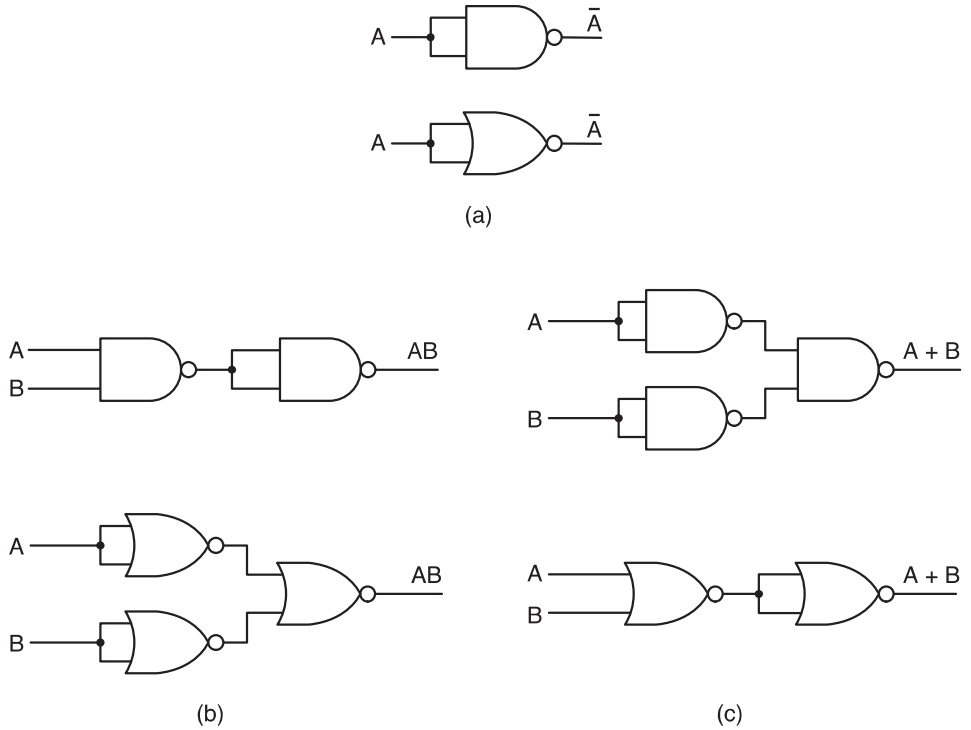
Pelo exemplo da Figura 3.3 deve ficar claro como colocar em prática um circuito para qualquer função booleana:

1. Escreva a tabela verdade para a função.
2. Providencie inversores para gerar o complemento de cada entrada.
3. Desenhe uma porta AND para cada termo que tenha um 1 na coluna de resultado.
4. Ligue as portas AND às entradas adequadas.
5. Alimente a saída de todas as portas AND a uma porta OR.

Embora tenhamos mostrado como qualquer função booleana pode ser executada usando portas NOT, AND e OR, muitas vezes é conveniente realizar circuitos usando só um tipo de porta. Felizmente, converter circuitos gerados pelo algoritmo precedente à forma NAND pura ou NOR pura é uma operação direta. Para fazer essa conversão, basta que tenhamos um modo de implementar NOT, AND e OR usando um único tipo de porta. A linha superior da Figura 3.4 mostra como todas essas três podem ser implementadas usando apenas portas NAND; a fileira de baixo mostra como isso pode ser feito usando apenas portas NOR. (Essas operações são diretas, mas também há outras maneiras.)

Um modo de implementar uma função booleana usando somente portas NAND ou somente portas NOR é primeiro seguir o procedimento dado anteriormente para construí-la com NOT, AND e OR. Em seguida, substituir as portas de múltiplas entradas por circuitos equivalentes usando portas de duas entradas. Por exemplo, $A + B + C + D$ pode ser computada como $(A + B) + (C + D)$, empregando três portas OR de duas entradas. Por fim, as portas NOT, AND e OR são substituídas pelos circuitos da Figura 3.4.

Figura 3.4 Construção de portas (a) NOT, (b) AND e (c) OR usando somente portas NAND ou somente portas NOR.



Embora esse procedimento não resulte em circuitos ótimos, no sentido do número mínimo de portas, ele mostra que sempre há uma solução viável. Ambas as portas, NAND e NOR, são denominadas **completas** porque qualquer função booleana pode ser calculada usando quaisquer das duas. Nenhuma outra porta tem essa propriedade, o que é outra razão para elas serem preferidas como blocos de construção de circuitos.

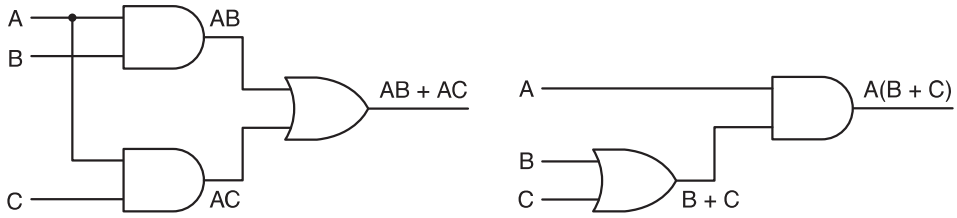
3.1.4 Equivalência de circuito

Projetistas de circuitos muitas vezes tentam reduzir o número de portas em seus produtos para reduzir a área da placa de circuito interno necessária para executá-las, diminuir o consumo de potência e aumentar a velocidade. Para reduzir a complexidade de um circuito, o projetista tem de encontrar outro circuito que calcule a mesma função que o original, mas efetue essa operação com um número menor de portas (ou talvez com portas mais

simples, por exemplo, com duas em vez de com quatro entradas). A álgebra booleana pode ser uma ferramenta valiosa na busca de circuitos equivalentes.

Como exemplo de como a álgebra booleana pode ser usada, considere o circuito e a tabela verdade para $AB + AC$ mostrados na Figura 3.5(a). Embora ainda não as tenhamos discutido, muitas das regras da álgebra comum também são válidas para a booleana. Em particular, a expressão $AB + AC$ pode ser fatorada para $A(B + C)$ usando a lei distributiva. A Figura 3.5(b) mostra o circuito e a tabela verdade para $A(B + C)$. Como duas funções são equivalentes se, e somente se, elas tiverem a mesma saída para todas as entradas possíveis, é fácil ver pelas tabelas verdade da Figura 3.5 que $A(B + C)$ é equivalente a $AB + AC$. Apesar dessa equivalência, o circuito da Figura 3.5(b) é claramente melhor do que o da Figura 3.5(a), pois contém menos portas.

Figura 3.5 Duas funções equivalentes. (a) $AB + AC$. (b) $A(B + C)$.



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

Em geral, um projetista de circuitos começa com uma função booleana e depois aplica a ela as leis da álgebra booleana na tentativa de achar uma função mais simples, porém equivalente. Um circuito pode ser construído com base na função final.

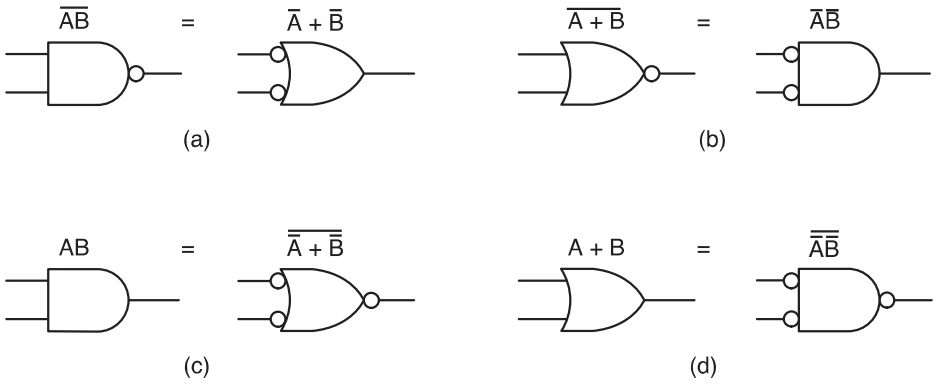
Para usar essa abordagem, precisamos de algumas identidades da álgebra booleana. A Figura 3.6 mostra algumas das mais importantes. É interessante notar que cada lei tem duas formas que são **duais** uma da outra. Permutando AND e OR e também 0 e 1, quaisquer das formas pode ser produzida com base na outra. Todas as leis podem ser provadas com facilidade construindo suas tabelas verdade. Com exceção da lei de De Morgan, a lei da absorção, e da forma AND da lei distributiva, os resultados são razoavelmente intuitivos. A lei de De Morgan pode ser estendida para mais de duas variáveis, por exemplo, $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$.

Figura 3.6 Algumas identidades da álgebra booleana.

Nome	Forma AND	Forma OR
Lei da identidade	$1A = A$	$0 + A = A$
Lei do elemento nulo	$0A = 0$	$1 + A = 1$
Lei idempotente	$AA = A$	$A + A = A$
Lei do inverso	$A\overline{A} = 0$	$A + \overline{A} = 1$
Lei comutativa	$AB = BA$	$A + B = B + A$
Lei associativa	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Lei distributiva	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Lei da absorção	$A(A + B) = A$	$A + AB = A$
Lei de De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A} + \overline{B} = \overline{AB}$

A lei de De Morgan sugere uma notação alternativa. Na Figura 3.7(a), a forma AND é mostrada com negação indicada por bolhas de inversão tanto para entrada quanto para saída. Assim, uma porta OR com entradas invertidas é equivalente a uma porta NAND. Pela Figura 3.7(b), a forma dual da lei de De Morgan, deve ficar claro que uma porta NOR pode ser desenhada como uma porta AND com entradas invertidas. Negando ambas as formas da lei de De Morgan, chegamos às figuras 3.7(c) e (d), que mostram representações equivalentes das portas AND e OR. Existem símbolos análogos para as formas de múltiplas variáveis da lei de De Morgan (por exemplo, uma porta NAND com n entradas se torna uma porta OR com entradas invertidas).

Figura 3.7 Símbolos alternativos para algumas portas: (a) NAND. (b) NOR. (c) AND. (d) OR.

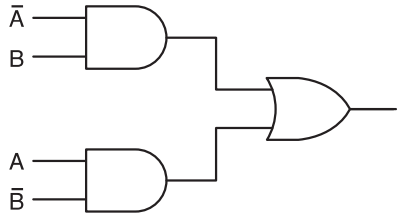


Usando as identidades da Figura 3.7 e as análogas para portas de múltiplas entradas é fácil converter a representação de soma de produtos de uma tabela verdade para a forma NAND pura ou NOR pura. Como exemplo, considere a função EXCLUSIVE OR da Figura 3.8(a). O circuito padrão da soma de produtos é mostrado na Figura 3.8(b). Para converter para a forma NAND, as linhas que conectam a saída das portas AND à entrada da porta OR devem ser redesenhadas com duas bolhas de inversão, conforme mostra a Figura 3.8(c). Por fim, usando a Figura 3.7(a), chegamos à Figura 3.8(d). As variáveis $\overline{A}$ e $\overline{B}$ podem ser geradas de A e B usando portas NAND ou NOR com suas entradas interligadas. Note que as bolhas de inversão podem ser deslocadas à vontade ao longo da linha, por exemplo, desde as saídas das portas de entrada na Figura 3.8(d) até as entradas da porta de saída.

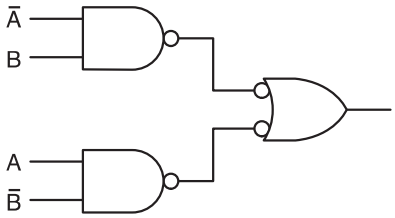
Figura 3.8 (a) Tabela verdade para a função XOR. (b)–(d) Três circuitos para calcular essa tabela.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

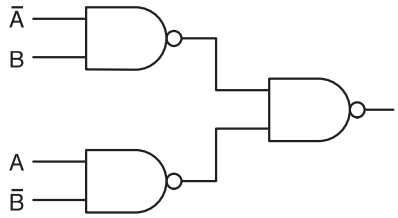
(a)



(b)



(c)



(d)

Como observação final em relação à equivalência de circuitos, demonstraremos agora o surpreendente resultado, isto é, a mesma porta física pode calcular funções diferentes dependendo das convenções usadas. Na Figura 3.9(a), mostramos a saída de certa porta, *F*, para diferentes combinações de entrada. Tanto entradas quanto saídas são representadas por volts. Se adotarmos a convenção de que 0 volt é 0 lógico e 1,5 volt é 1 lógico, denominada **lógica positiva**, obtemos a tabela verdade da Figura 3.9(b), a função AND. Contudo, se adotarmos a **lógica negativa**, na qual 0 volt é 1 lógico e 1,5 volt é 0 lógico, obtemos a tabela verdade da Figura 3.9(c), a função OR.

Figura 3.9 (a) Características elétricas de um dispositivo. (b) Lógica positiva. (c) Lógica negativa.

A	B	F
0V	0V	0V
0V	5V	0V
5V	0V	0V
5V	5V	5V

(a)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

(b)

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

(c)

Assim, a convenção escolhida para mapear voltagens para valores lógicos é crítica. A menos que especifiquemos outra coisa, daqui em diante usaremos lógica positiva, portanto, os termos 1 lógico, verdade e tensão alta são sinônimos, assim como 0 lógico, falso e tensão baixa.

3.2 Circuitos lógicos digitais básicos

Nas seções anteriores vimos como executar tabelas verdade e outros circuitos simples usando portas individuais. Na prática, poucos circuitos são construídos porta por porta, embora tenha havido uma época em que isso era comum. Hoje, os blocos de construção mais comuns são módulos que contêm várias portas. Nas próximas seções, examinaremos esses blocos de construção mais de perto e veremos como eles podem ser construídos com base em portas individuais.