Arquitetura de Computadores
3 – Nível Lógico Digital

Prof. Dr. David Alain do Nascimento

Slides baseados no Livro Organização estruturada de computadores, 6a Ed, Tanenbaum

O nível lógico digital

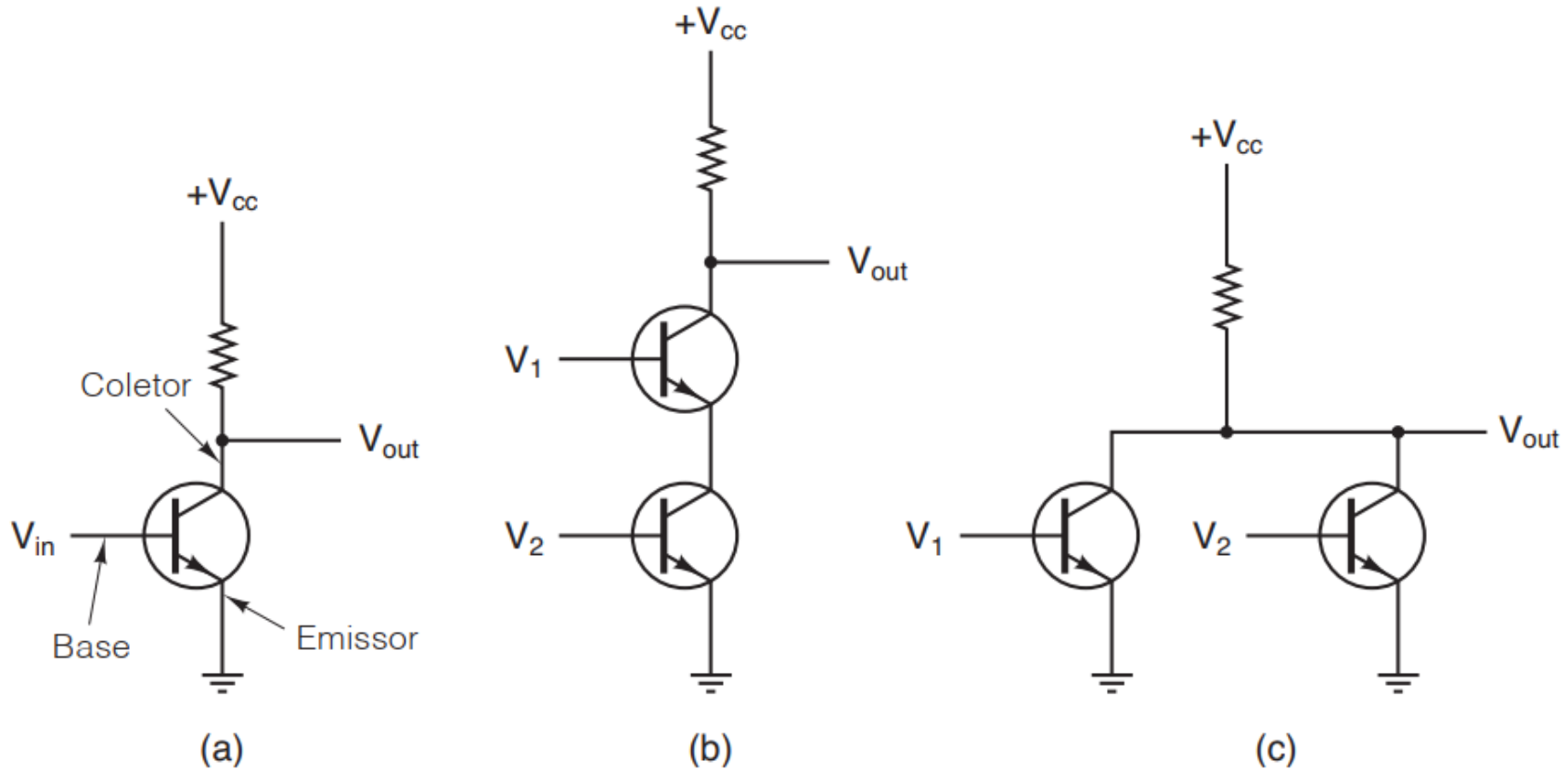
- Os elementos básicos que fazem parte de todos os computadores digitais são surpreendentemente simples.
- Iniciaremos nosso estudo examinando esses elementos básicos e também a álgebra especial de dois valores (álgebra booleana) usada para analisá-los.
- Em seguida, examinaremos alguns circuitos fundamentais 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.

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.
- Toda a lógica digital moderna se apoia no fato de que um transistor pode funcionar como um comutador binário muito rápido.
- Por exemplo, dependendo da função, o tempo necessário para passar de um estado para outro é tipicamente de um nanossegundo ou menos.

Portas

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



Portas

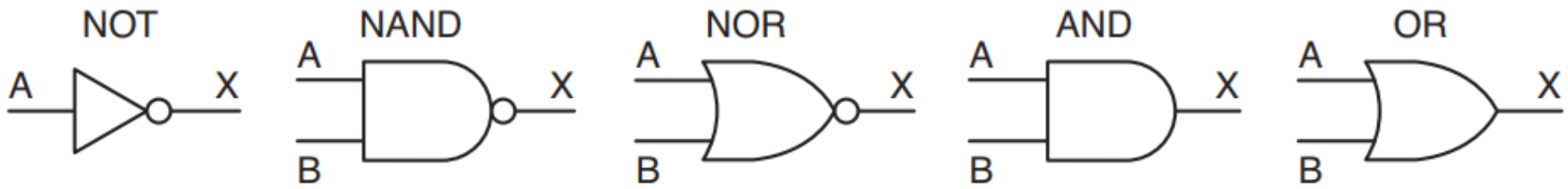
- Embora a questão do modo como são construídas as portas pertença ao nível do dispositivo, as principais famílias de tecnologia de fabricação porque elas são citadas com muita frequência. As duas tecnologias principais são bipolar e **MOS (Metal Oxide Semiconductor – semicondutor de óxido metálico)**.
- 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.

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).
- 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**.

Álgebra booleana

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)

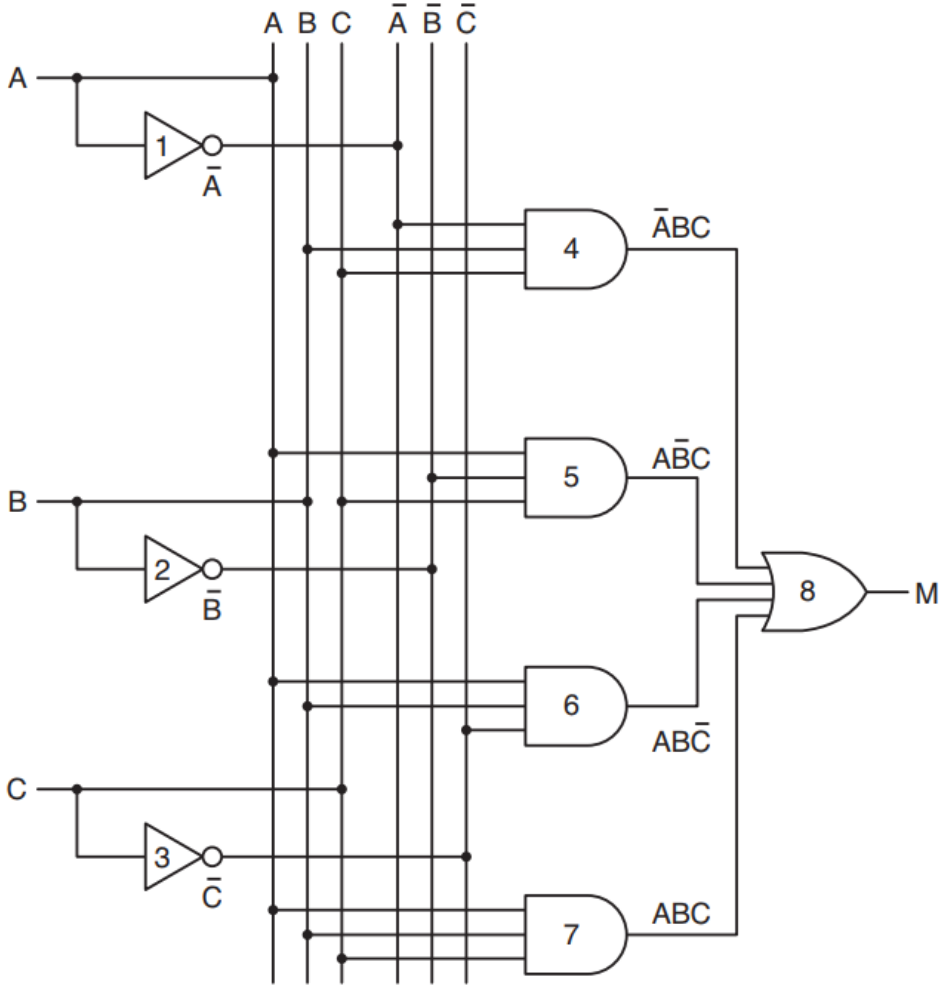
Álgebra booleana

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

- A Figura (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.

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)

Álgebra booleana

- 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.
- 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**.

Álgebra booleana

- Assim, por exemplo, $A\bar{B}C$ assume o valor 1 somente quando A = 1 e B = 0 e C = 1.
- Além disso, $A\bar{B} + B\bar{C}$ é 1 somente quando (A = 1 e B = 0) ou (B = 1 e C = 0).

- O equivalente em Java para as lógicas acima é:
  - $A\bar{B}C$ A && !B && C
  - $A\bar{B} + B\bar{C}$ A && !B || B && !C

Álgebra booleana

- As quatro linhas da Figura 3.3(a) que produzem bits 1 na saída são: $\bar{A}BC$, $A\bar{B}C$, $AB\bar{C}$ e ABC.
- A função, M, é verdadeira (isto é, 1) se qualquer uma dessas quatro condições for verdadeira; daí, podemos escrever

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

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, pois leva diretamente a uma execução da função que usa portas padronizadas.

$M = f(A, B, C)$

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

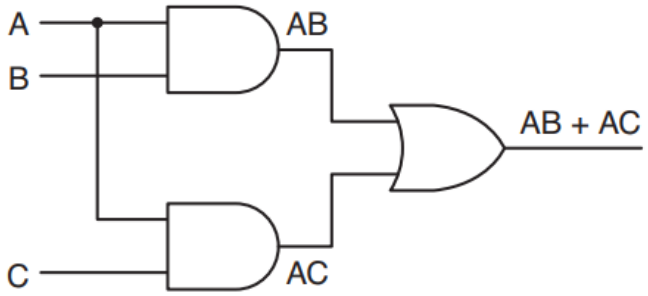
(a)

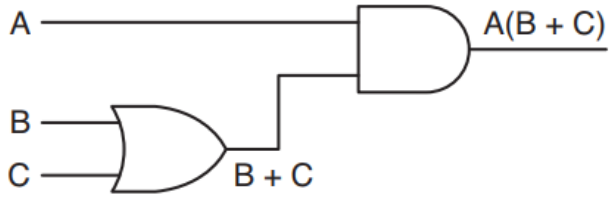
Equivalência de circuito

- 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.

Equivalência de circuito

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)

Circuitos combinatórios

Circuitos combinatórios

- Muitas aplicações de lógica digital requerem um circuito com múltiplas entradas e múltiplas saídas, no qual as saídas são determinadas exclusivamente pelas entradas em questão. Esses circuitos são denominados **circuitos combinatórios**.
- Nem todos os circuitos têm essa propriedade. Por exemplo, um circuito que contenha elementos de memória pode perfeitamente gerar saídas que dependem de valores armazenados, bem como de variáveis de entrada.
- Um circuito que esteja executando uma tabela verdade como a da Figura 3.3(a) é um exemplo típico de um circuito combinatório.

$M = f(A, B, C)$

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

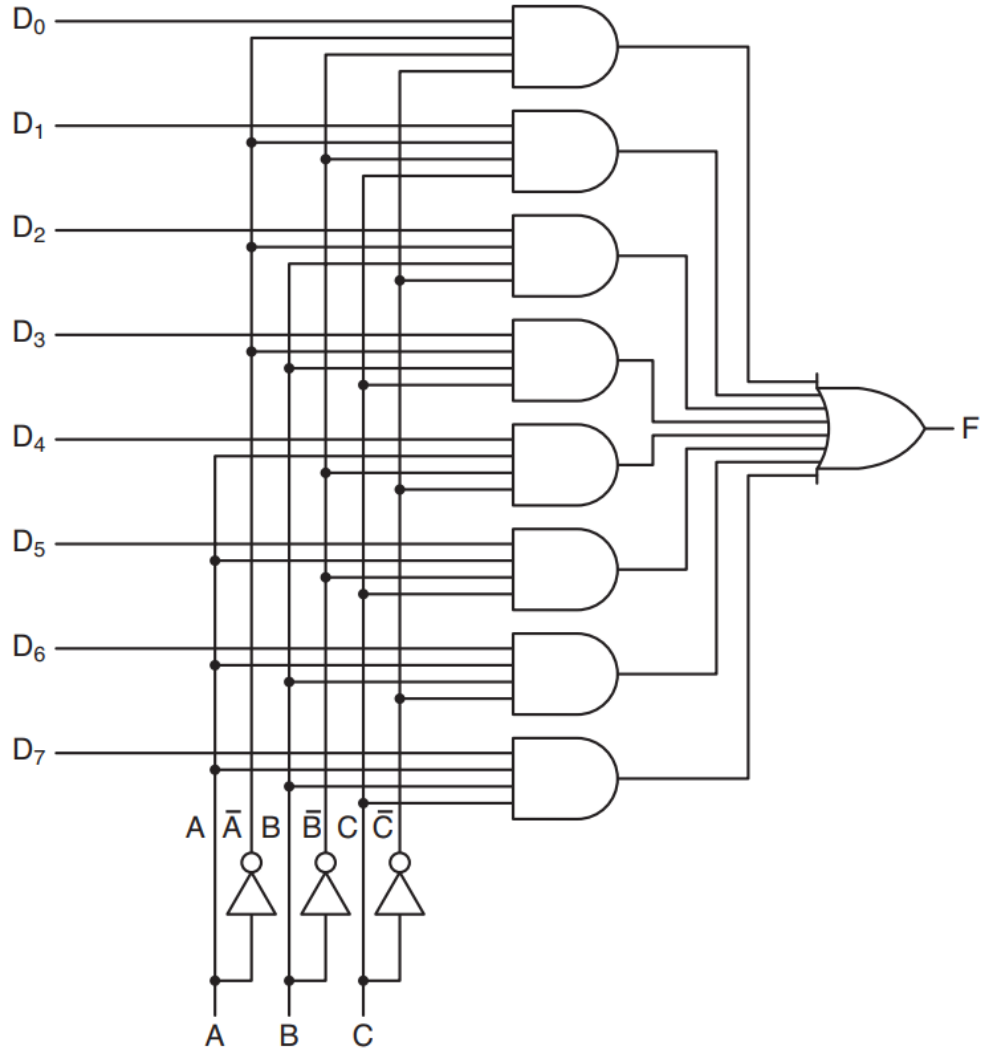
(a)

Multiplexadores (Mux)

- No nível lógico, um multiplexador é um circuito com 2^n entradas de dados, uma saída de dados e n entradas de controle que selecionam uma das entradas de dados.
- Essa entrada selecionada é dirigida (isto é, roteada) até a saída.

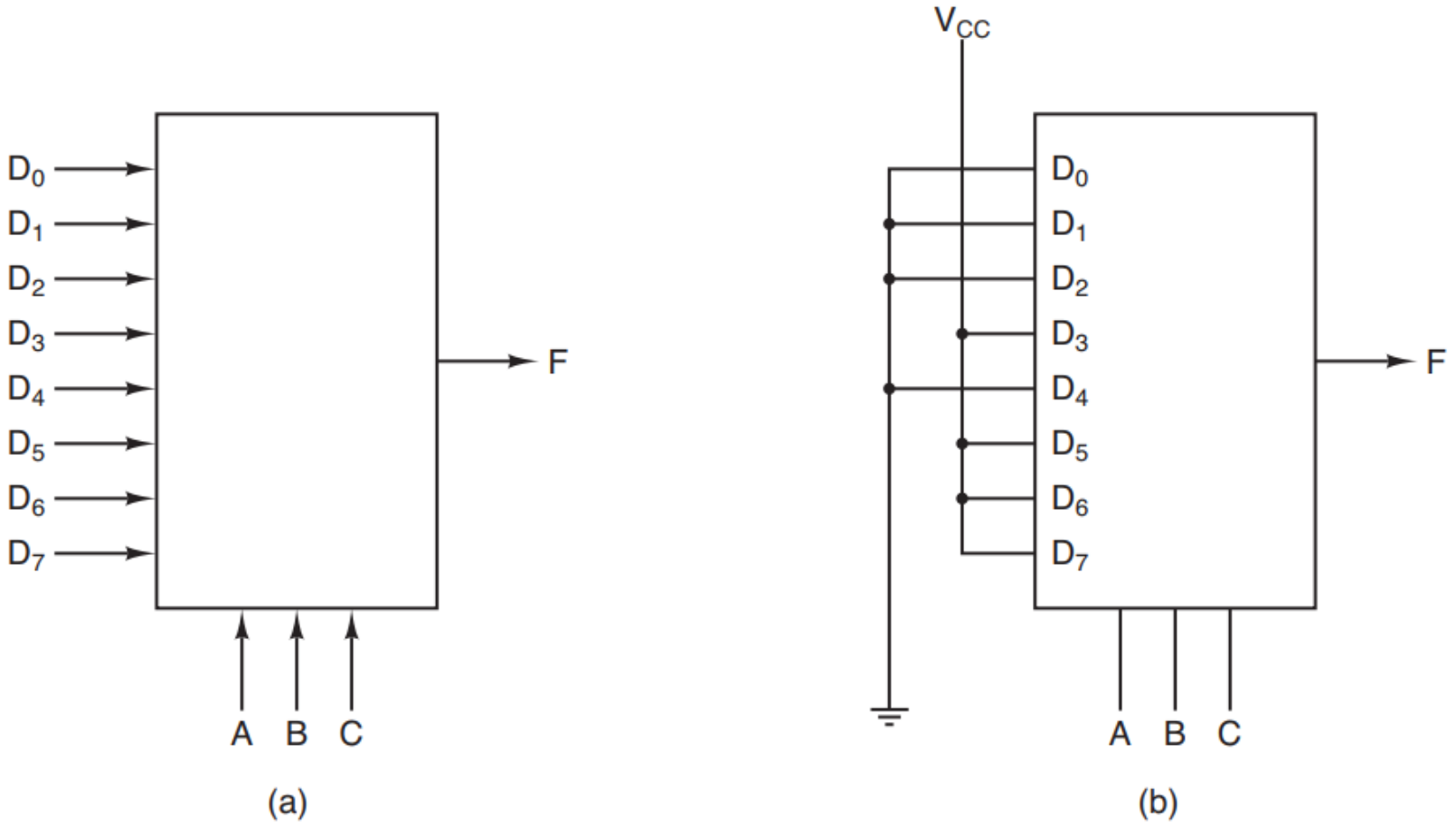
Multiplexadores (Mux)

Figura 3.11 Circuito multiplexador de oito entradas.



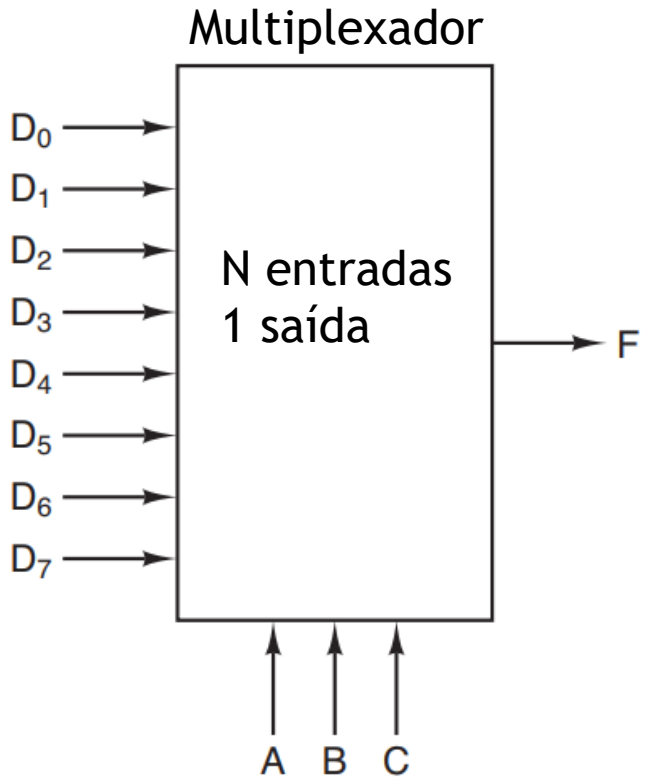
Multiplexadores (Mux)

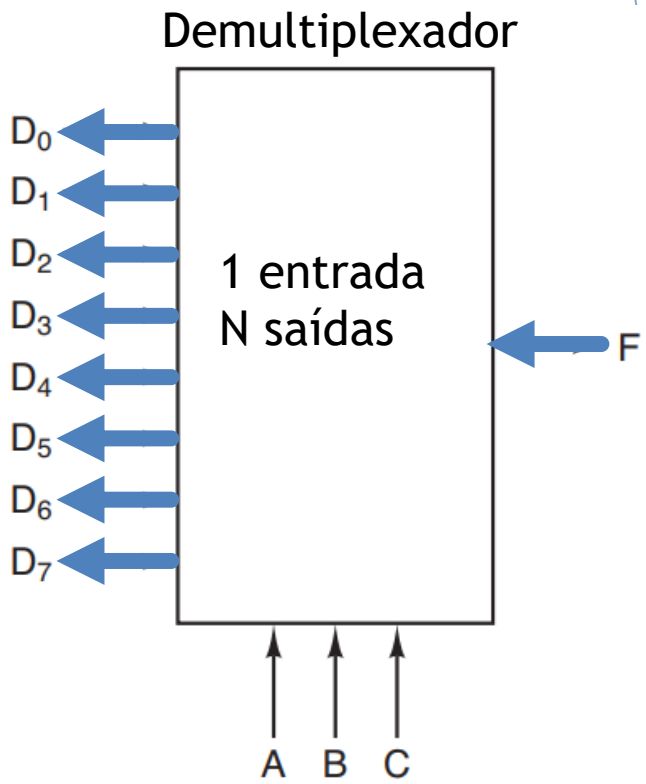
Figura 3.12 (a) Multiplexador com oito entradas. (b) O mesmo multiplexador ligado para calcular a função majoritária.



Demultiplexadores (Demux)

- **Demultiplexadores**
  - O inverso dos circuitos multiplexadores





Decodificadores

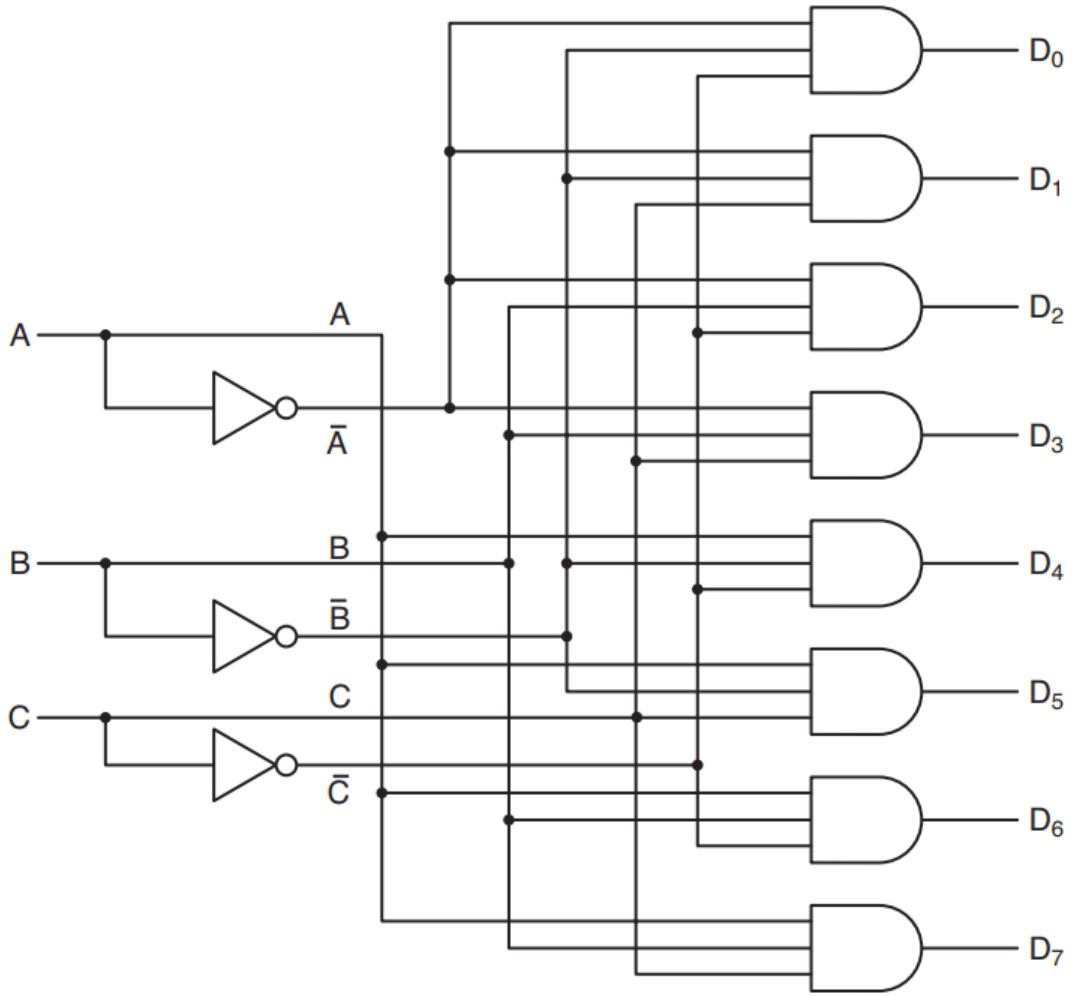
- Um circuito que toma um número de n bits como entrada e o usa para selecionar (isto é, definir em 1) exatamente uma das 2^n linhas de saída.
- Exemplo para $n = 3$:

Decodificadores

Figura 3.13 Circuito decodificador 3 para 8.

n entradas
2^n saídas

Exemplo para $n = 3$



Comparadores

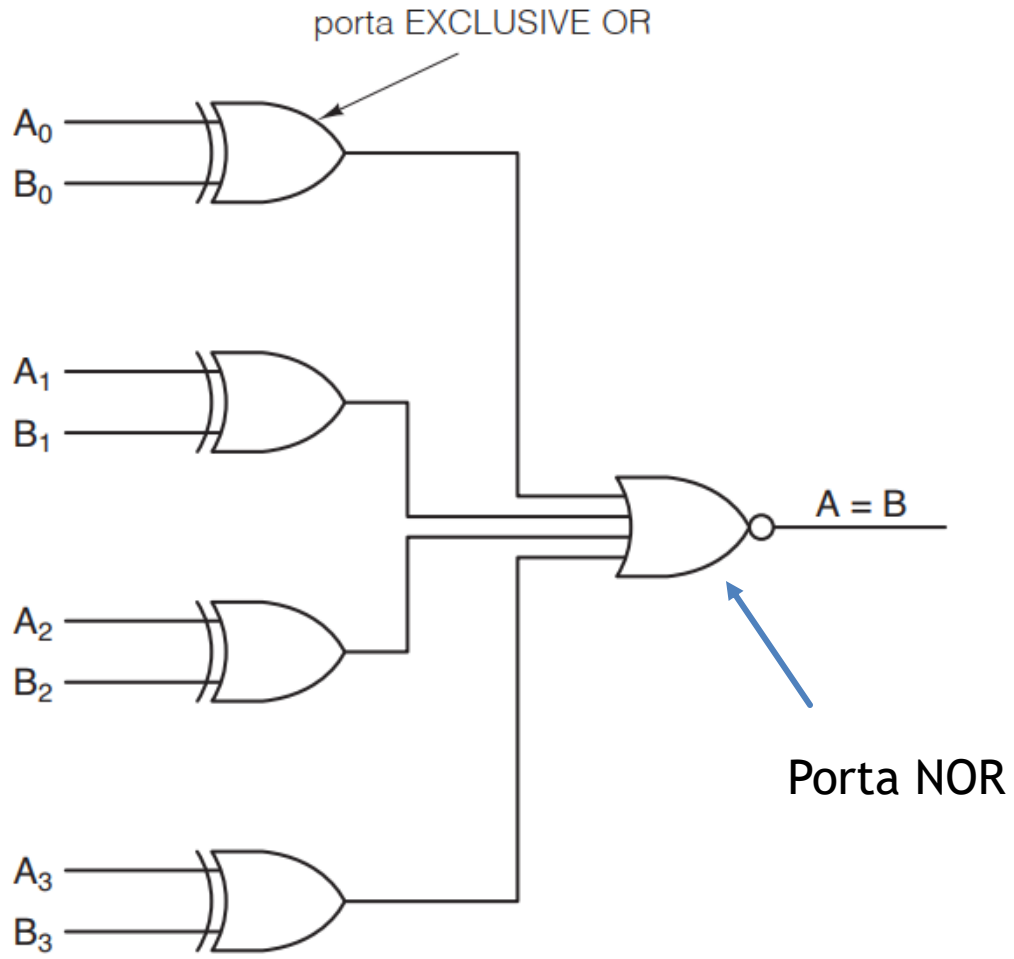
- Compara duas palavras de entrada.
- Um comparador simples toma duas entradas, A e B, cada uma de n bits de comprimento, e produz um **1** se elas forem iguais e um **0** se elas não o forem.
- O circuito é baseado na porta XOR (Exclusive OR), que produz um 0 se suas entradas forem iguais e um 1 se elas forem diferentes.
- Se as duas palavras de entrada forem iguais, todas as quatro portas XOR devem produzir 0.
- Então, pode-se efetuar uma operação XOR em todos os n sinais; se o resultado for 0, as palavras de entrada são iguais; caso contrário, não.

Comparadores

Figura 3.14 Comparador simples de 4 bits.

$A = A_3A_2A_1A_0$

$B = B_3B_2B_1B_0$



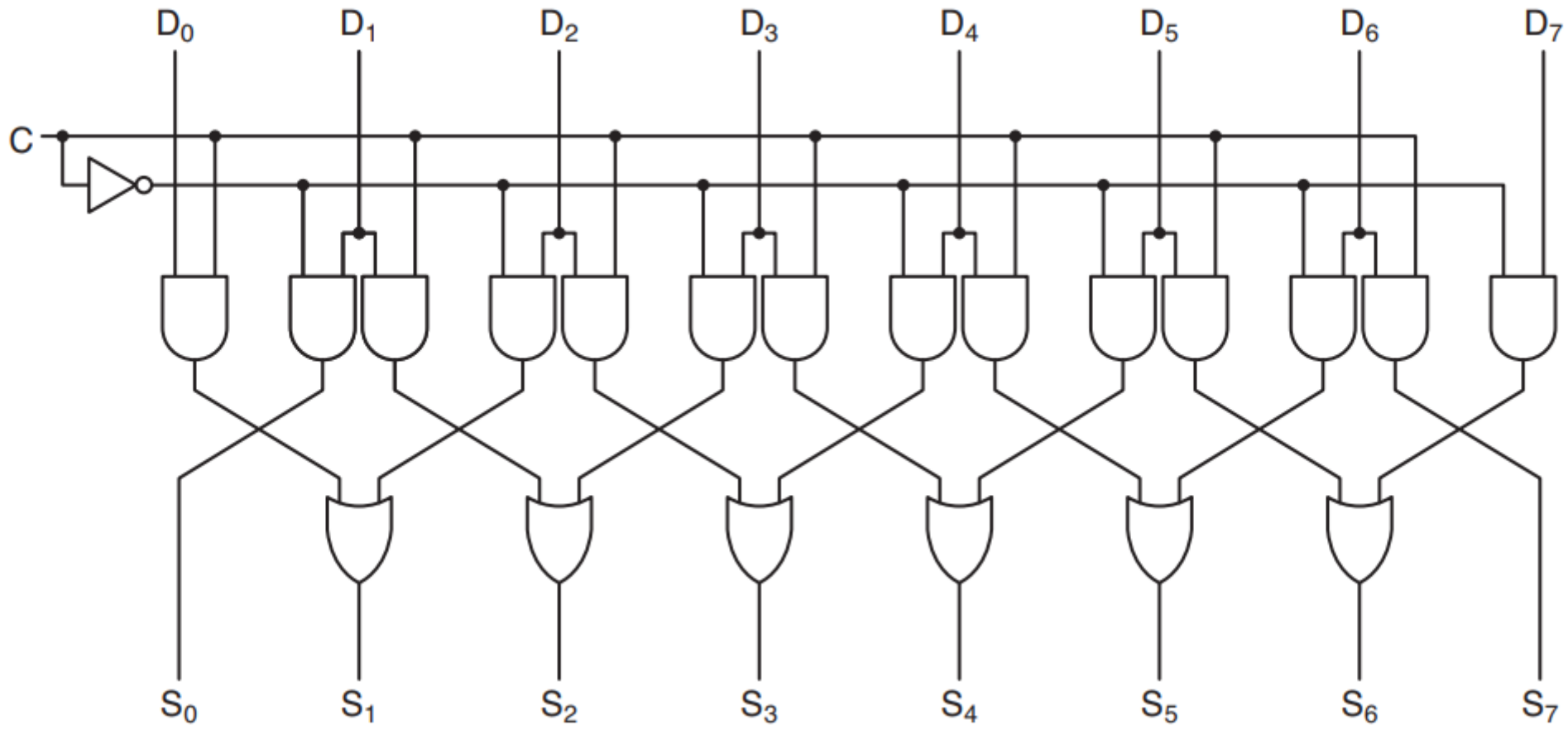
Deslocadores

- Um deslocador é um circuito de n entradas (D_0 a D_{n-1}) e n saídas (S_0 a S_{n-1}).
- A saída é apenas a entrada deslocada de 1 bit.
- Uma linha de controle, **C**, determina a direção do deslocamento, **0** para a esquerda e **1** para a direita.
- Quando o deslocamento for para a esquerda, um 0 é inserido no bit $n-1$.
- De modo semelhante, quando o deslocamento for para a direita, um 0 é inserido no bit 0.

Deslocadores

Exemplo de circuito deslocador com 8 bits

Figura 3.15 Deslocador esquerda/direita de 1 bit.



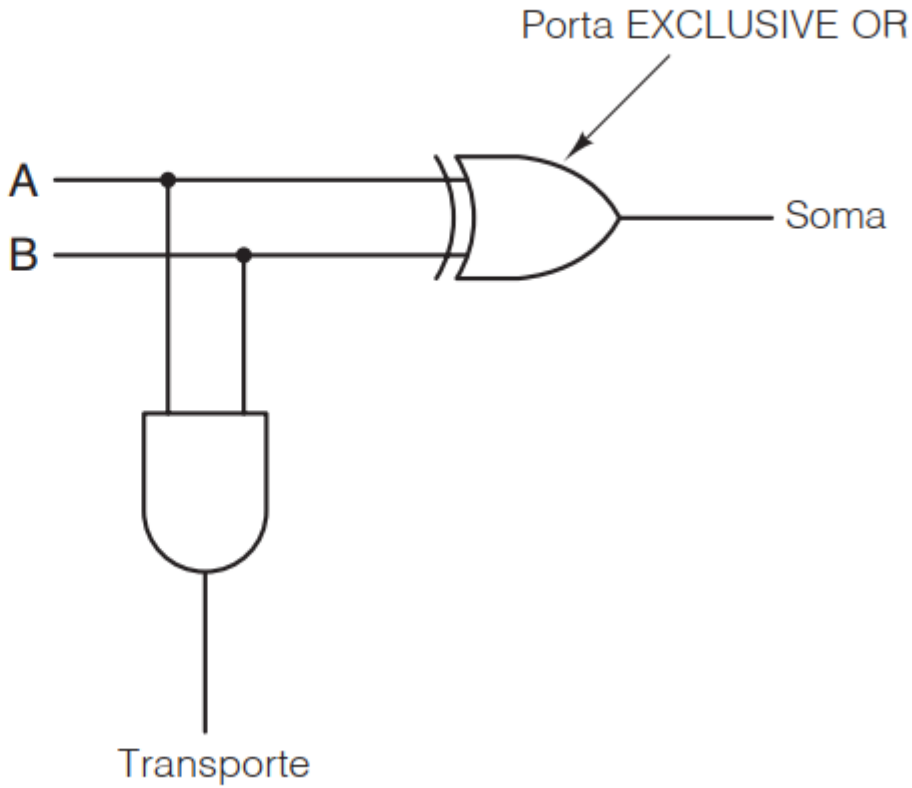
Somadores

- Um computador que não possa somar números inteiros é quase inimaginável.
- Por consequência, um circuito de hardware para efetuar adição é uma parte essencial de toda CPU.
- Há duas saídas presentes: a **soma** das entradas, A e B, e o **transporte** (vai-um) para a posição seguinte (à esquerda).
- Esse circuito simples é conhecido como um meio-somador.

Somadores

Figura 3.16 (a) Tabela verdade para adição de 1 bit. (b) Circuito para um meio-somador.

A	B	Soma	Transporte
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Somadores

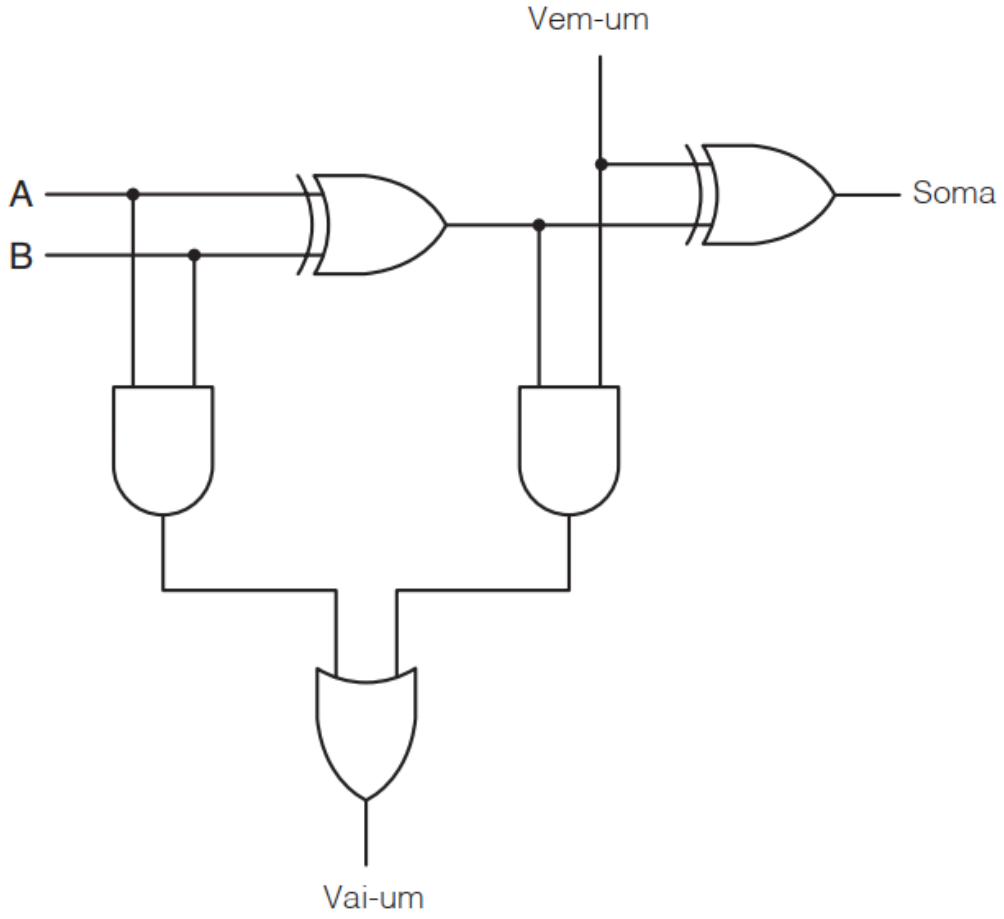
- Embora um meio-somador seja adequado para somar os bits de ordem baixa de duas palavras de entrada de múltiplos bits, ele não servirá para uma posição de bit no meio da palavra porque não trata o transporte de bit da posição à direita (vem-um).
- Em seu lugar, precisamos do somador completo.

Somadores

Figura 3.17 (a) Tabela verdade para somador completo. (b) Circuito para um somador completo.

A	B	Vem-um	Soma	Vai-um
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



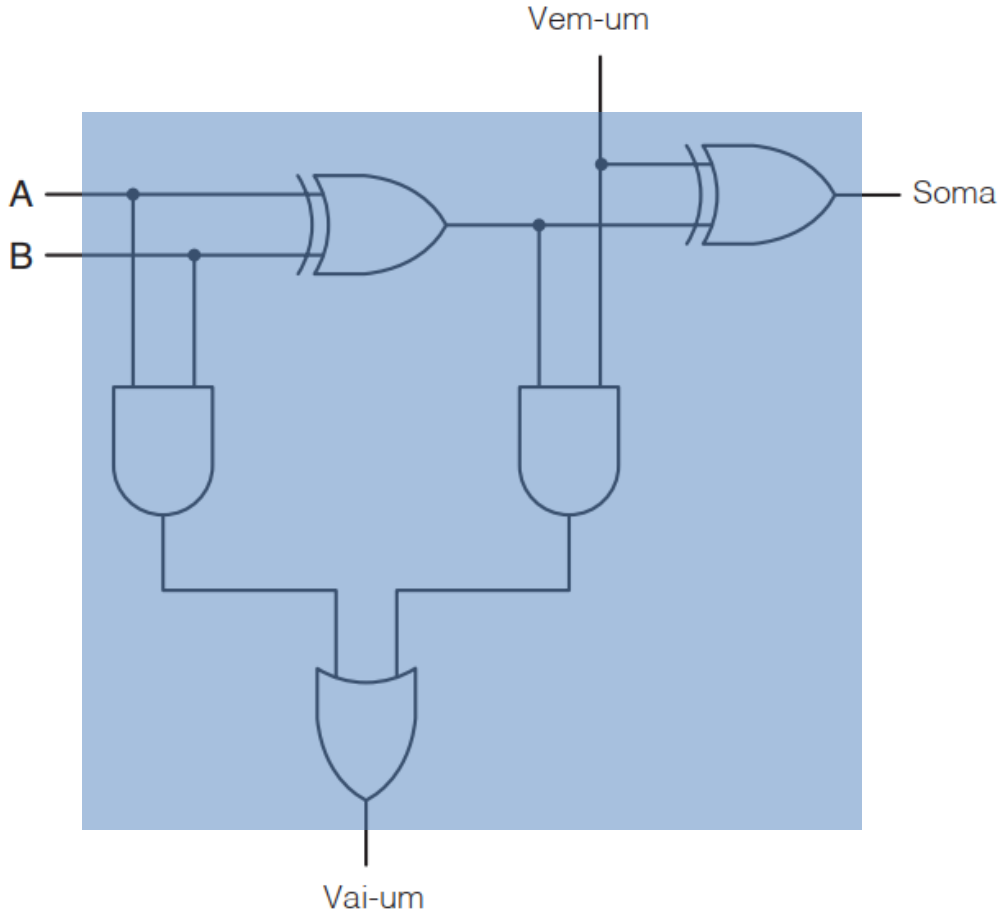
(b)

Somadores

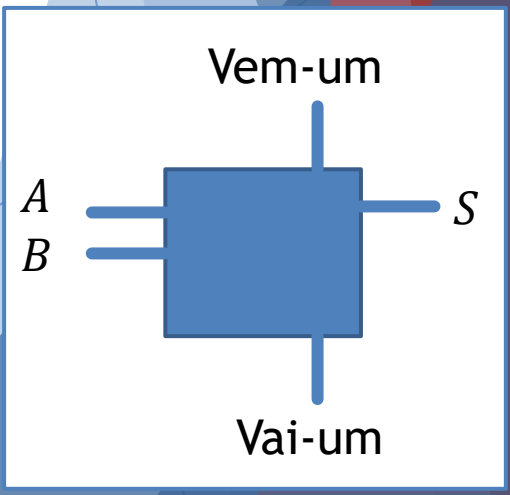
Figura 3.17 (a) Tabela verdade para somador completo. (b) Circuito para um somador completo.

A	B	Vem-um	Soma	Vai-um
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

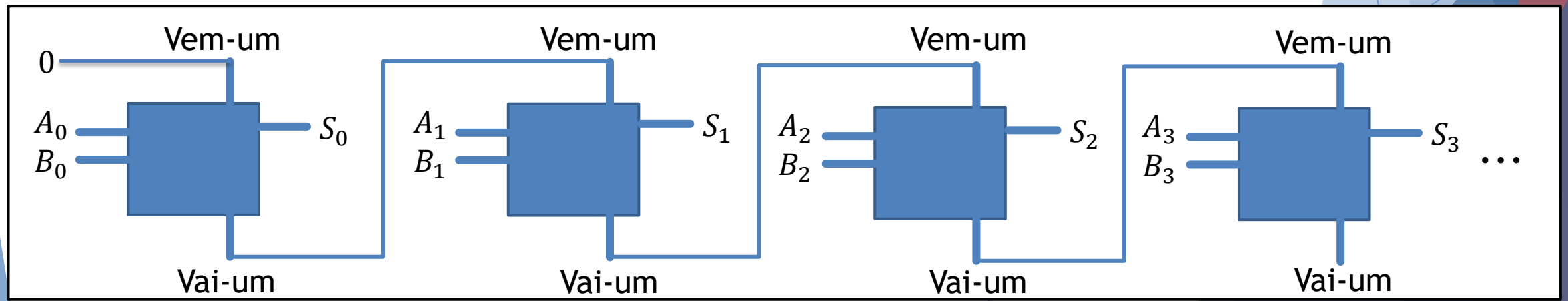


(b)



Somadores

- Para construir um somador para palavras de 16 bits, por exemplo, basta repetir o circuito somador completo 16 vezes.
- O vai-um de um bit é usado como vem-um para seu vizinho da esquerda. O vem-um do bit da extrema direita está ligado a 0.
- Esse tipo de somador é denominado somador de transporte encadeado porque, na pior das hipóteses, somando 1 a 111...111 (binário), a adição não pode ser concluída até que o vai-um tenha percorrido todo o caminho do bit da extrema direita até o da extrema esquerda.



Unidades lógica e aritmética

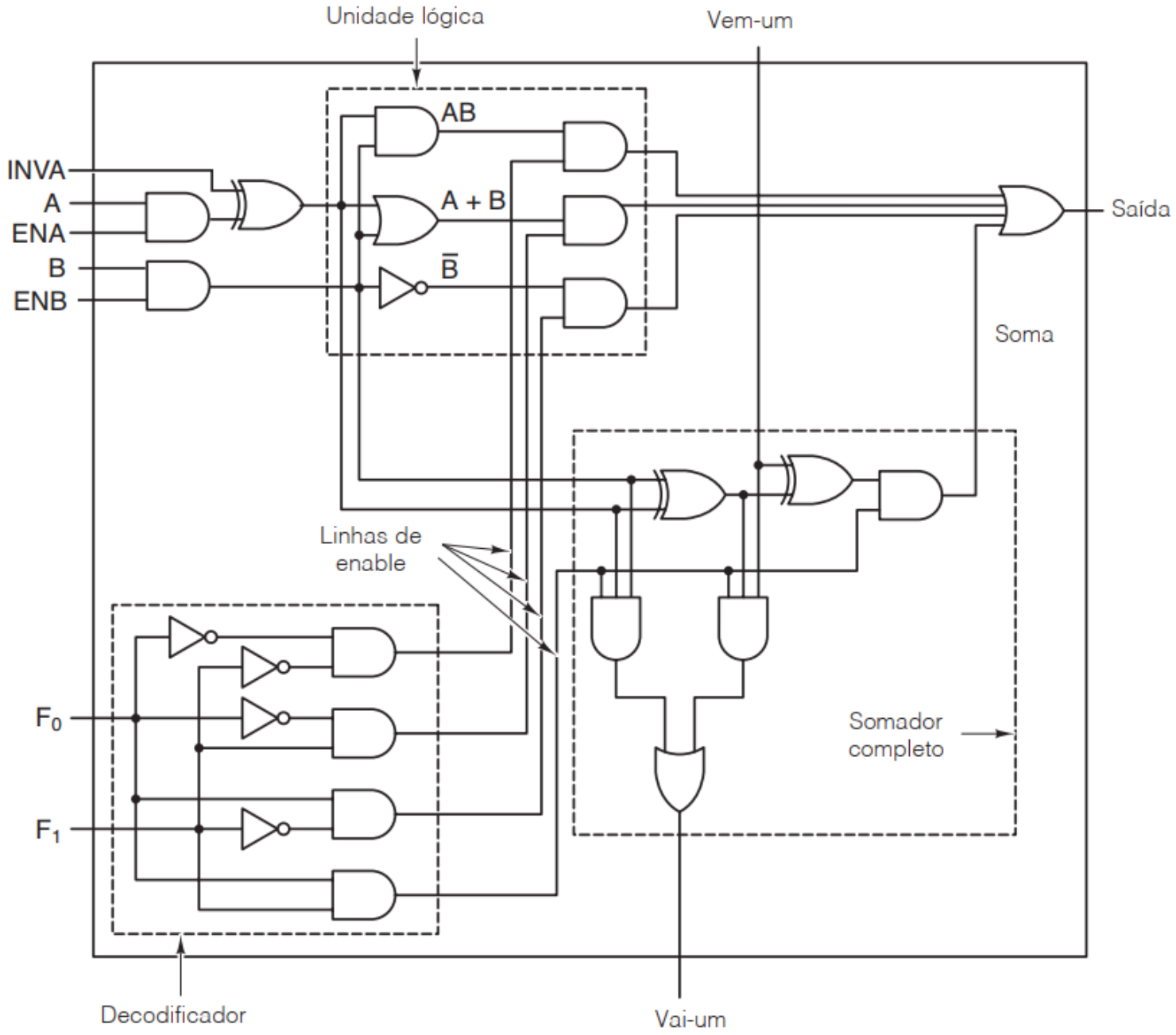
- Grande parte dos computadores contém um único circuito para efetuar AND, OR e soma de duas palavras de máquina. No caso típico, tal circuito para palavras de n bits é composto de n circuitos idênticos para as posições individuais de bits.
- Ela pode calcular qualquer uma das quatro funções – a saber:
  - $A\ AND\ B$
  - $A\ OR\ B$
  - $\bar{B}$
  - $A + B$

  dependendo de as linhas de entrada de seleção de função F_0 e F_1 conterem 00, 01, 10 ou 11 (binário).
- Note que, aqui, A + B significa a soma aritmética de A e B, e não a operação booleana OR.

Unidades lógica e aritmética

- O canto inferior esquerdo de nossa ULA contém um decodificador de 2 bits para gerar sinais de ***enable*** (habilitação) para as quatro operações, com base nos sinais de controle F_0 e F_1 .
- Dependendo dos valores de F_0 e F_1 , exatamente uma das quatro linhas de habilitação é selecionada. Ativar essa linha permite que a saída para a função selecionada passe por ela até a porta OR final, para saída.

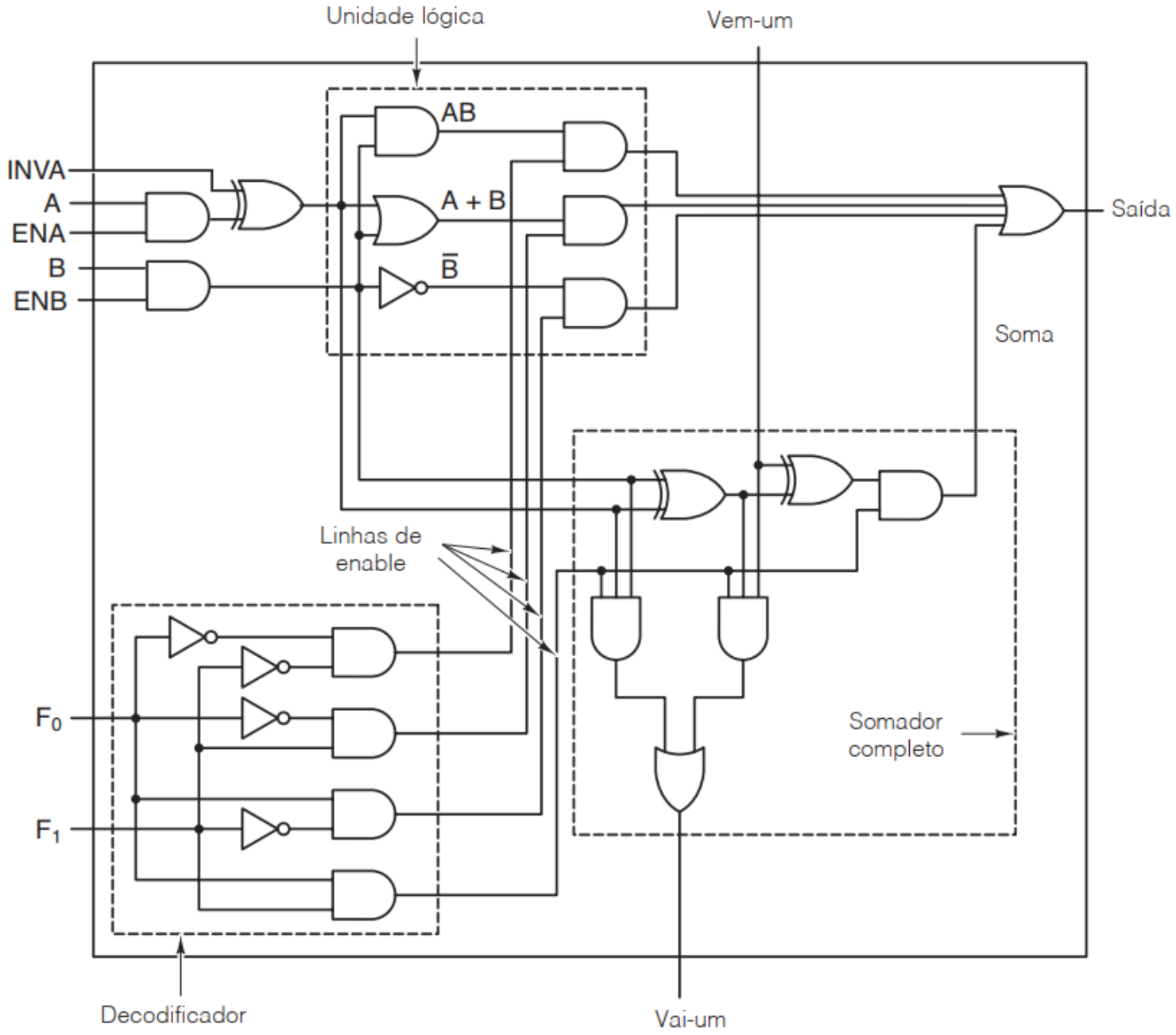
Figura 3.18 ULA de 1 bit.



Unidades lógica e aritmética

- O canto superior esquerdo contém a lógica para calcular $A\ AND\ B$, $A\ OR\ B$ e $\bar{B}$, mas no máximo um desses resultados é passado para a porta or final, dependendo das linhas de habilitação que saem do decodificador.
- Como exatamente uma das saídas do decodificador será 1, exatamente uma das quatro portas AND que comandam a porta OR será habilitada; as outras três resultarão em 0, independente de A e B.

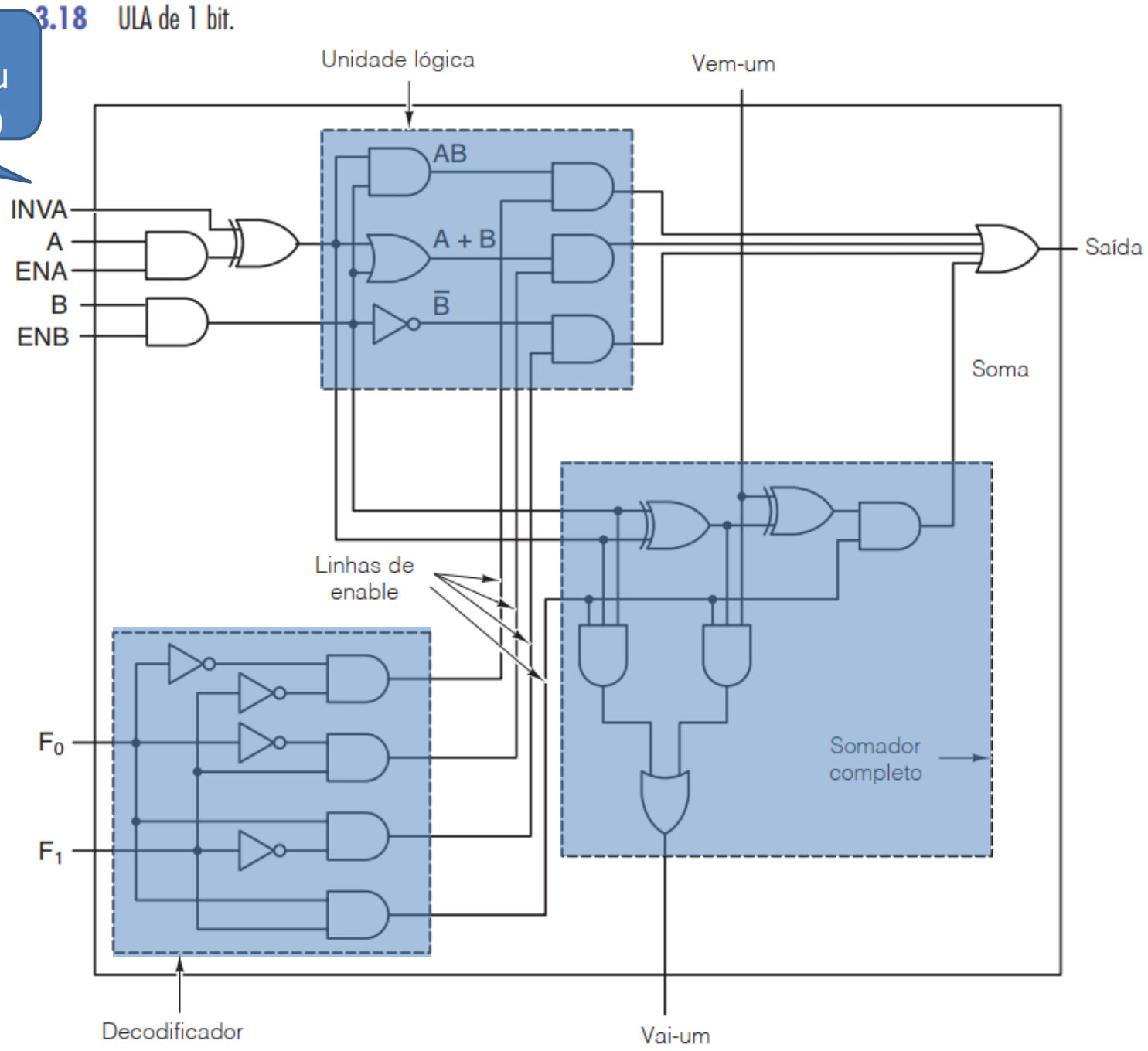
Figura 3.18 ULA de 1 bit.



INVA é uma chave que escolhe se será utilizado o valor de A ou valor invertido de A (ou seja $\bar{A}$)

Unidades lógica e aritmética

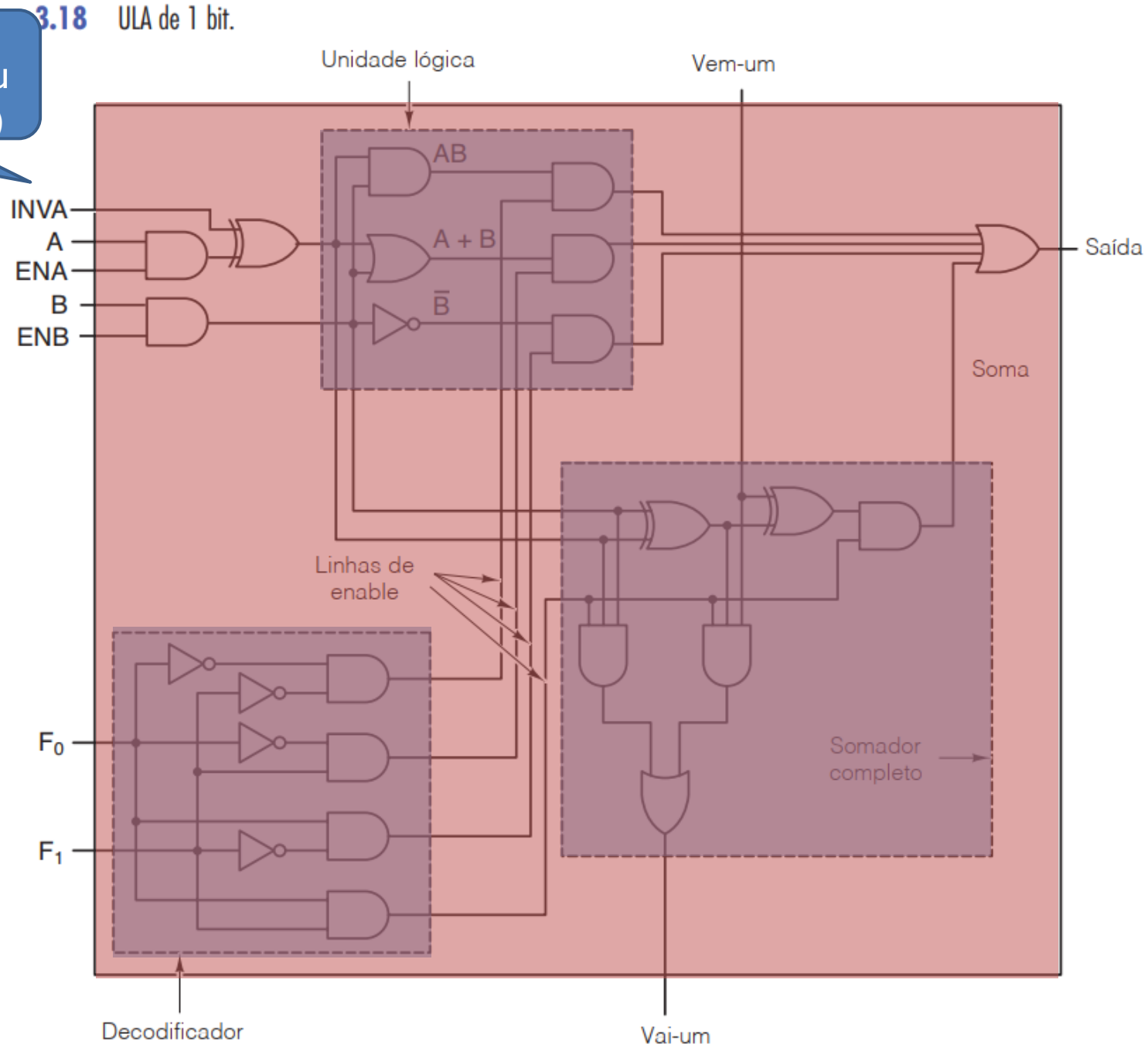
- O canto superior esquerdo contém a lógica para calcular $A\ AND\ B$, $A\ OR\ B$ e $\bar{B}$, mas no máximo um desses resultados é passado para a porta or final, dependendo das linhas de habilitação que saem do decodificador.
- Como exatamente uma das saídas do decodificador será 1, exatamente uma das quatro portas AND que comandam a porta OR será habilitada; as outras três resultarão em 0, independente de A e B.



INVA é uma chave que escolhe se será utilizado o valor de A ou valor invertido de A (ou seja $\bar{A}$)

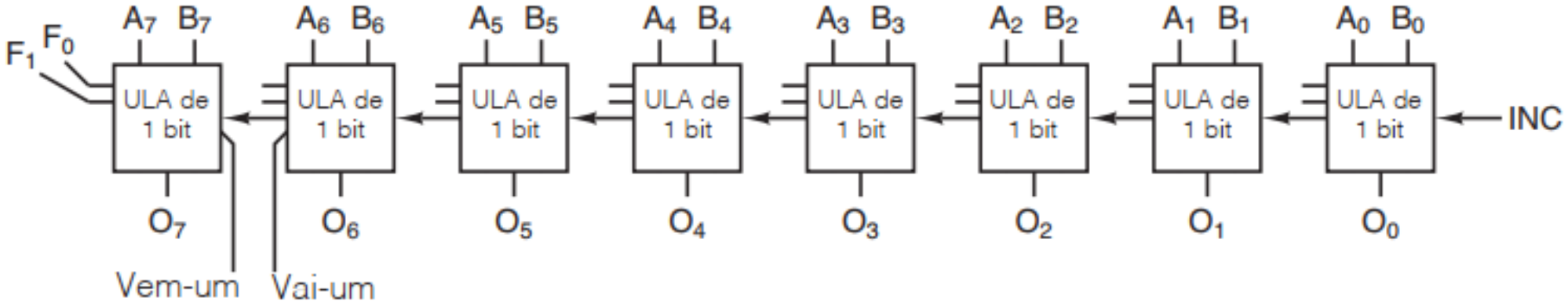
Unidades lógica e aritmética

- O canto superior esquerdo contém a lógica para calcular $A\ AND\ B$, $A\ OR\ B$ e $\bar{B}$, mas no máximo um desses resultados é passado para a porta or final, dependendo das linhas de habilitação que saem do decodificador.
- Como exatamente uma das saídas do decodificador será 1, exatamente uma das quatro portas AND que comandam a porta OR será habilitada; as outras três resultarão em 0, independente de A e B.



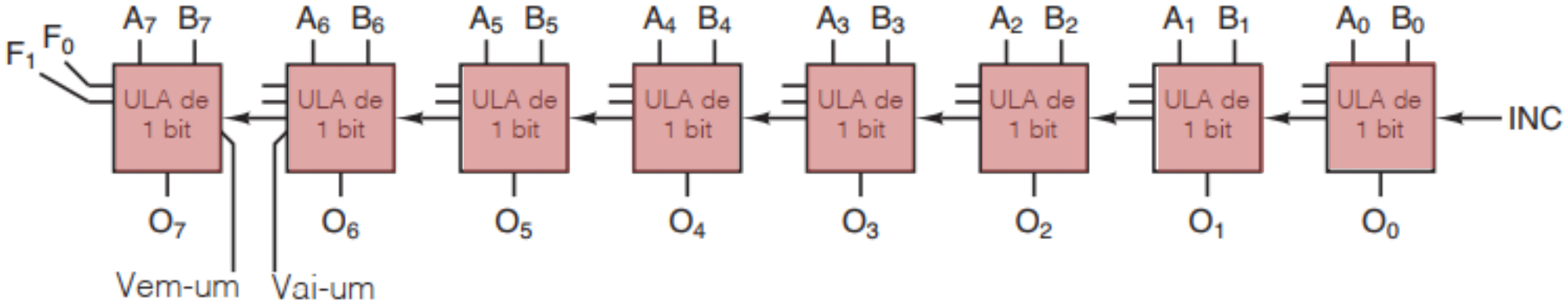
Unidades lógica e aritmética

Figura 3.19 Oito segmentos (*slices*) de ULA de 1 bit conectados para formar uma ULA de 8 bits. Os sinais de habilitação e inversão não são mostrados por simplicidade.



Unidades lógica e aritmética

Figura 3.19 Oito segmentos (*slices*) de ULA de 1 bit conectados para formar uma ULA de 8 bits. Os sinais de habilitação e inversão não são mostrados por simplicidade.



Clocks

- Em muitos circuitos digitais, a ordem em que os eventos ocorrem é crítica. Às vezes um evento deve preceder outro, às vezes dois eventos devem ocorrer simultaneamente.
- Para permitir que os projetistas consigam as relações de temporização requeridas, muitos circuitos digitais usam *clocks* para prover sincronização.
- Nesse contexto, um *clock* é um circuito que emite uma série de pulsos com uma largura de pulso precisa e intervalos precisos entre pulsos consecutivos. O intervalo de tempo entre as arestas correspondentes de dois pulsos consecutivos é denominado **tempo de ciclo de clock**.

Clocks

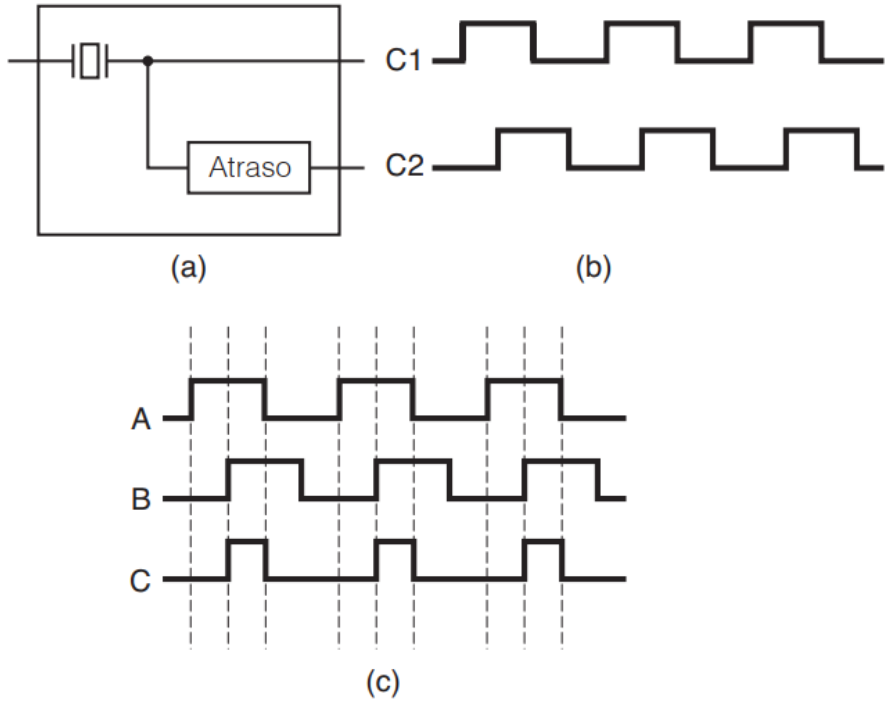
- Em geral, as frequências de pulso estão entre 100 MHz e 4 GHz, correspondendo a ciclos de *clock* de 10 nanossegundos a 250 picossegundos. Para conseguir alta precisão, a frequência de *clock* normalmente é controlada por um oscilador de cristal.
- Muitos eventos podem ocorrer dentro de um computador durante um único ciclo de *clock*. Se eles devem ocorrer em uma ordem específica, o ciclo de *clock* deve ser dividido em subciclos.

Clocks

- Uma maneira comum de prover resolução superior à do *clock* básico é aproveitar a linha de *clock* primária e inserir um circuito com um atraso conhecido, gerando assim um sinal de *clock* secundário deslocado em certa fase em relação ao primeiro, conforme mostra a Figura 3.20(a).
- O diagrama de temporização da Figura 3.20(b) dá quatro referências de tempo para eventos discretos:

  1. Fase ascendente de C1.
  2. Fase descendente de C1.
  3. Fase ascendente de C2.
  4. Fase descendente de C2.

Figura 3.20 (a) Um *clock*. (b) Diagrama de temporização para o *clock*. (c) Geração de um *clock* assimétrico.



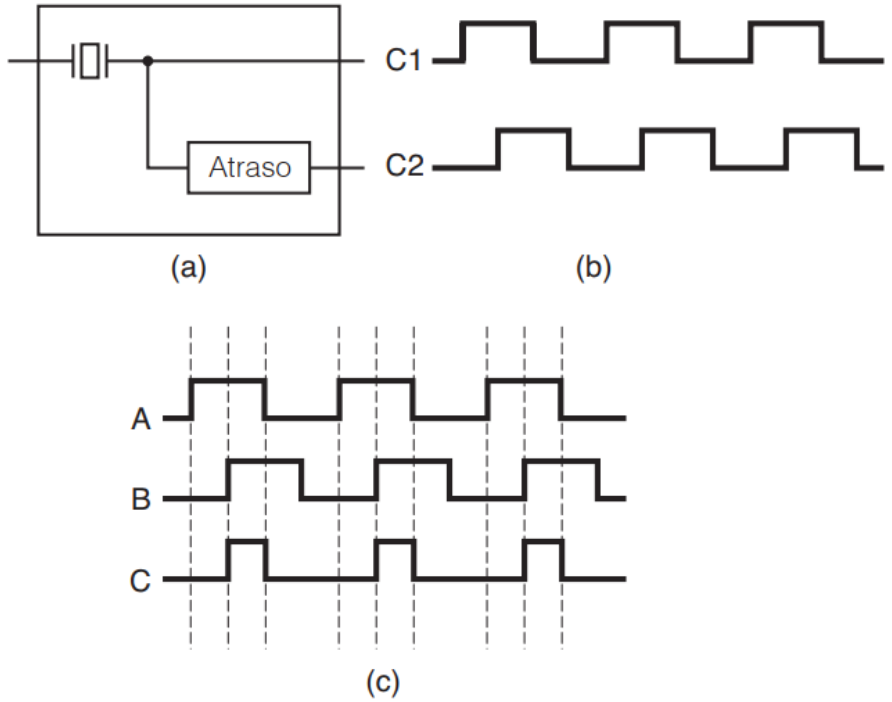
Clocks

- Vinculando diferentes eventos às várias fases, pode-se conseguir a sequência requerida. Se forem necessárias mais do que quatro referências de tempo dentro de um ciclo de clock, podem-se puxar mais linhas da linha primária, com diferentes atrasos, se for preciso.
- Em alguns circuitos, estamos interessados em intervalos de tempo em vez de instantes discretos de tempo.

Clocks

- Por exemplo, pode-se permitir que algum evento aconteça toda vez que C1 estiver alto, em vez de exatamente na fase ascendente. Outro evento só poderá acontecer quando C2 estiver alto.
- Se forem necessários mais de dois intervalos diferentes, podem ser instaladas mais linhas de *clock* ou pode-se fazer com que os estados altos dos dois *clocks* se sobreponham parcialmente no tempo.
- No último caso, podem-se distinguir quatro intervalos distintos:
  - $\overline{C1}$ AND $\overline{C2}$, $\overline{C1}$ AND $C2$, C1 AND $\overline{C2}$ e $C1$ AND $C2$.

Figura 3.20 (a) Um *clock*. (b) Diagrama de temporização para o *clock*. (c) Geração de um *clock* assimétrico.

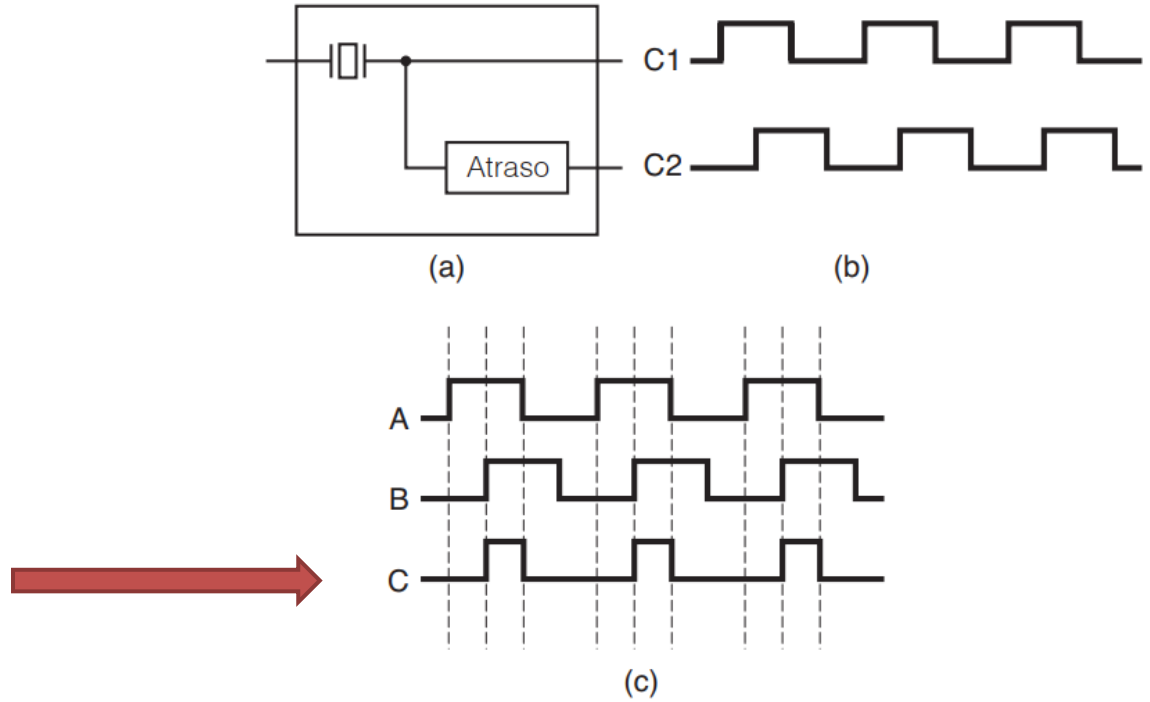


Clocks

- A propósito, clocks são simétricos, com o tempo gasto no estado alto igual ao tempo gasto no estado baixo, como mostra a Figura 3.20(b).
- Para gerar um trem de pulsos assimétrico, o clock básico é deslocado usando um circuito de atraso e efetuando uma operação AND com o sinal original, como mostra a Figura 3.20(c) como C.

$$C = A \text{ AND } B$$

Figura 3.20 (a) Um *clock*. (b) Diagrama de temporização para o *clock*. (c) Geração de um *clock* assimétrico.



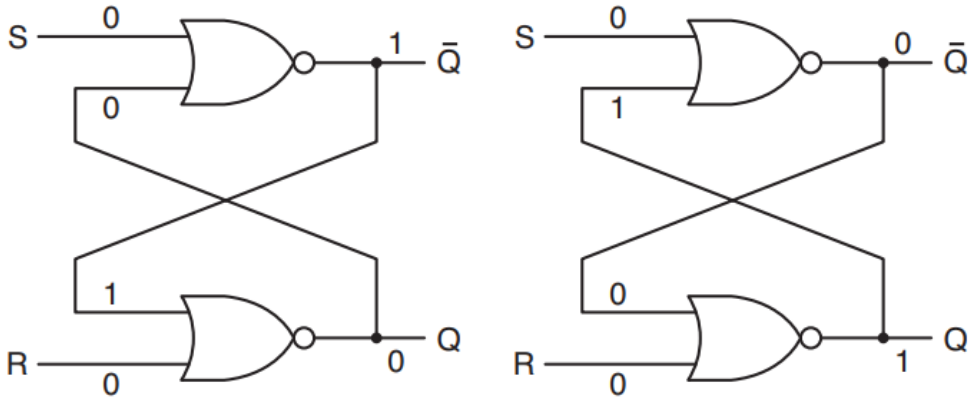
Memória

- Um componente essencial de todo computador é sua memória. Sem ela não poderiam existir os computadores que conhecemos.
- A memória é usada para armazenar instruções a serem executadas e dados.
- Examinaremos os componentes básicos de um sistema de memória começando no nível da porta para ver como eles funcionam e como são combinados para produzir memórias de grande porte.

Memórias de 1 bit
Latch SR

- Para criar uma memória de 1 bit (“latch”), precisamos de um circuito que “se lembre”, de algum modo, de valores de entrada anteriores.
- Tal circuito pode ser construído com base em duas portas NOR, como ilustrado na Figura 3.21(a).
- Circuitos semelhantes podem ser construídos com portas NAND, porém, não vamos utilizá-los porque são conceitualmente idênticos às versões NOR.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



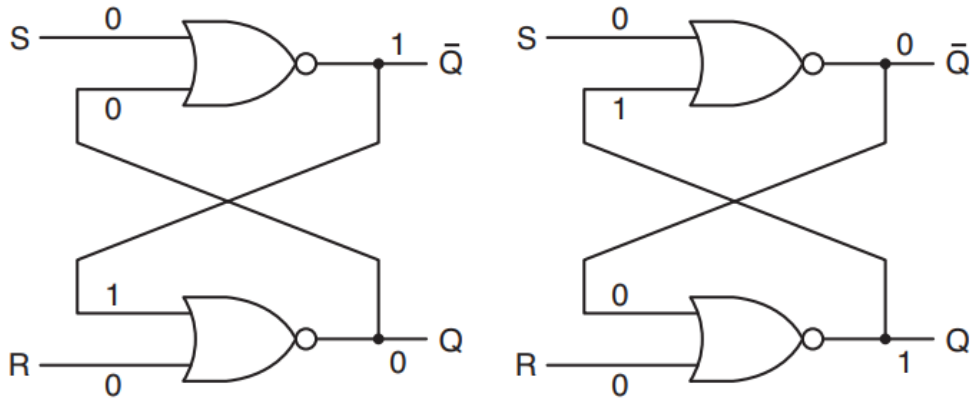
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- O circuito da Figura 3.21(a) é denominado **latch SR**.
- Ele tem duas entradas, S, para ativar (*setting*) o latch, e R, para restaurá-lo (*resetting*), isto é, liberá-lo.
- O circuito também tem duas saídas, Q e $\bar{Q}$, que são complementares, como veremos em breve.
- Ao contrário de um circuito combinacional, as saídas do latch não são exclusivamente determinadas pelas entradas atuais.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



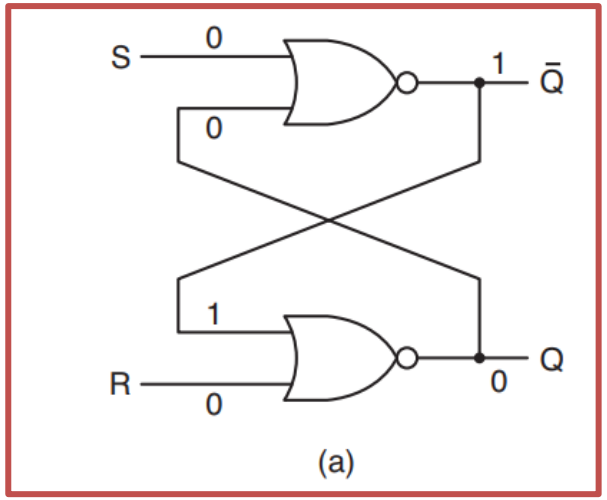
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- Para ver como isso ocorre, vamos supor que ambos, S e R, sejam 0, o que é verdade na maior parte do tempo.
- Apenas para polemizar, vamos supor que $Q = 0$. Como Q é realimentado para a porta NOR superior, ambas as suas entradas são 0, portanto, sua saída, $\bar{Q}$, é 1.
- O 1 é realimentado para a porta inferior que, então, tem entradas 1 e 0, resultando em $Q = 0$.
- Esse estado é no mínimo coerente e está retratado na Figura 3.21(a).

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



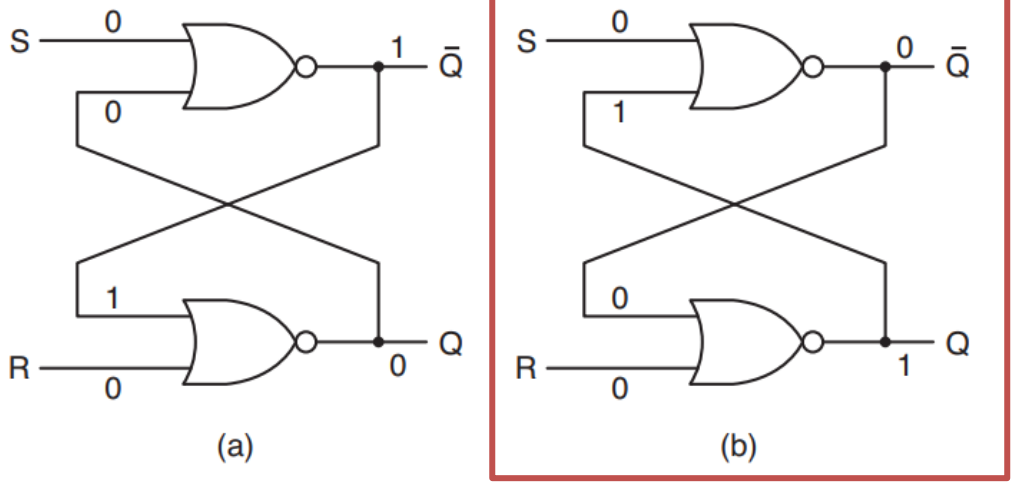
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- Agora, vamos imaginar que Q não seja 0, mas 1, com R e S ainda 0. A porta superior tem entradas de 0 e 1, e uma saída, $\bar{Q}$, de 0, que é realimentada para a porta inferior.
- Esse estado, mostrado na Figura 3.21(b), também é coerente.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



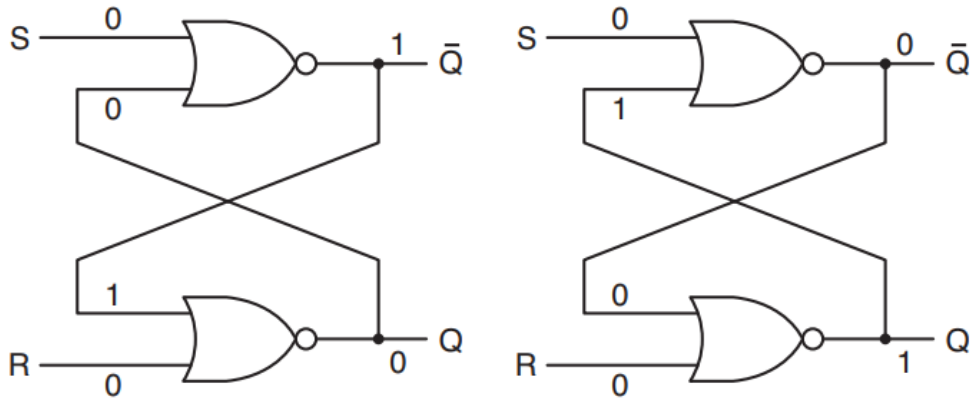
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- Um estado com as duas saídas iguais a 0 é incoerente, porque força ambas as portas a ter dois 0 como entrada, o que, se fosse verdade, produziria 1, não 0, como saída. De modo semelhante, é impossível ter ambas as saídas iguais a 1, porque isso forçaria as entradas a 0 e 1, o que resultaria 0, não 1.
- Nossa conclusão é simples:
  - para $R = S = 0$, o *latch* tem dois estados estáveis, que denominaremos 0 e 1, dependendo de Q.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



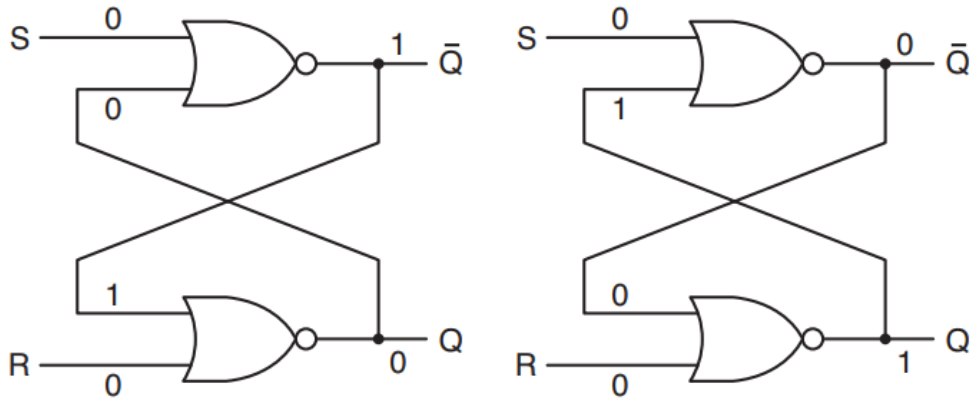
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- Agora, vamos examinar o efeito das entradas sobre o estado do *latch*.
- Suponha que S se torna 1 enquanto Q = 0. Então, as entradas para a porta superior são 1 e 0, forçando a saída Q a 0.
- Essa mudança faz ambas as entradas para a porta inferior serem 0, forçando a saída para 1.
- Portanto, ativar S (isto é, fazer com que seja 1) muda o estado de 0 para 1.
- Definir R em 1 quando o latch está no estado 0 não tem efeito algum porque a saída da porta NOR inferior é 0 para entradas de 10 e entradas de 11.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



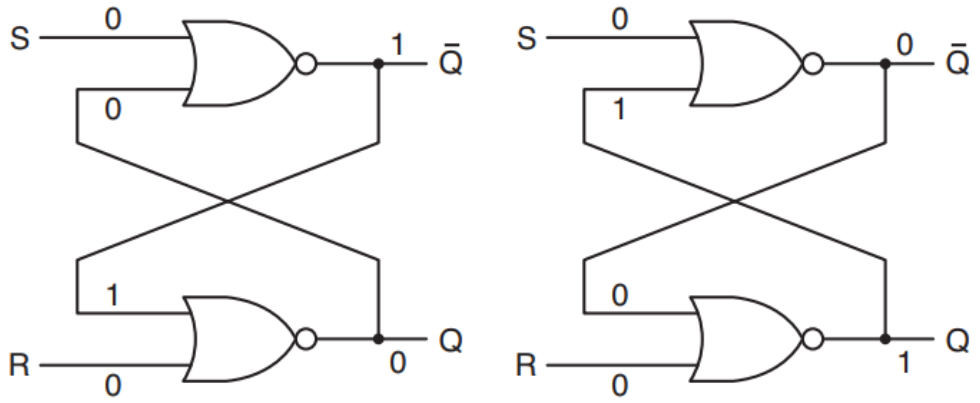
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR

- Usando raciocínio semelhante, é fácil ver que definir S em 1 quando em estado Q = 1 não tem efeito algum, mas definir R leva o *latch* ao estado Q = 0.
- Resumindo, quando S é definido em 1 momentaneamente, o *latch* acaba no estado Q = 1, pouco importando seu estado anterior.
- Da mesma maneira, definir R em 1 momentaneamente força o *latch* ao estado Q = 0. O circuito “se lembra” se foi S ou R definido por último. Usando essa propriedade podemos construir memórias de computadores.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



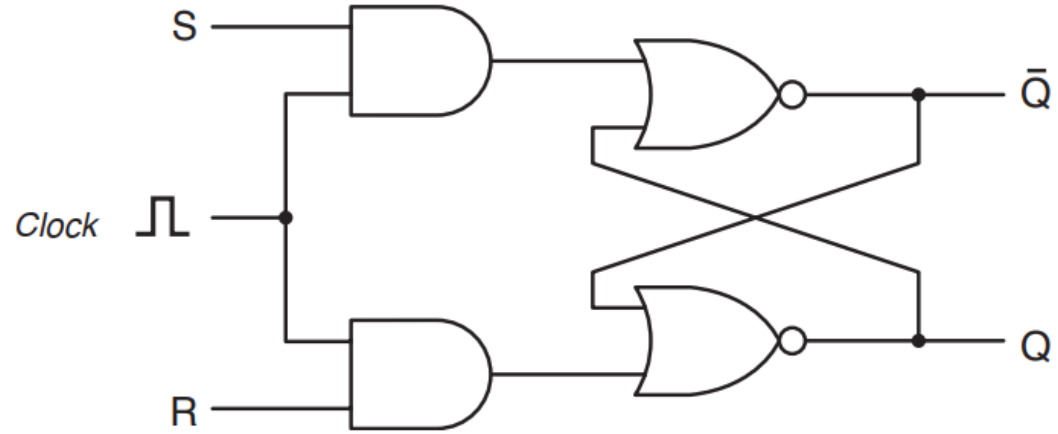
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Memórias de 1 bit
Latch SR com clock

- Muitas vezes é conveniente impedir que o *latch* mude de estado, a não ser em certos momentos especificados.
- Esse circuito tem uma entrada adicional, o clock, que em geral é 0. Com o clock em 0, ambas as portas AND geram saída 0, independentemente de S e R, e o *latch* não muda de estado. Quando o clock é 1, o efeito das portas AND desaparece e o *latch* se torna sensível a S e R. Apesar de seu nome, o sinal do clock não precisa ser gerado por um clock.

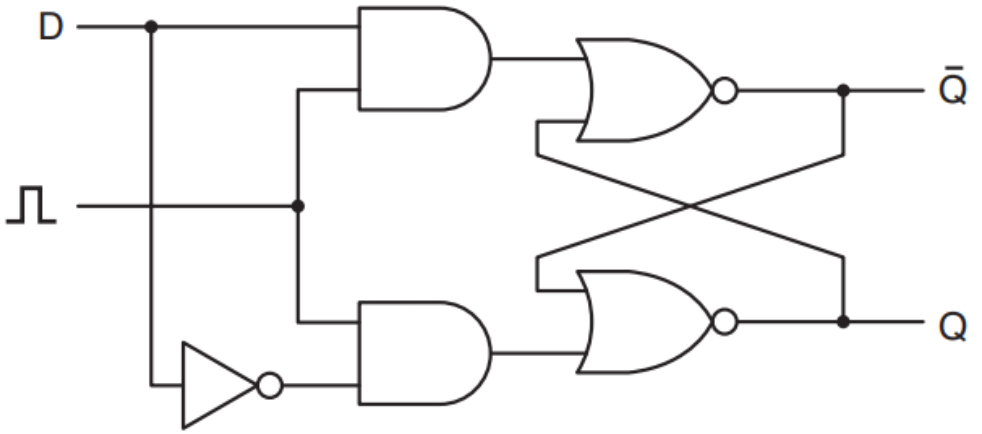
Figura 3.22 *Latch* SR com *clock*.



Memórias de 1 bit
Latches D com clock

- Uma boa maneira de resolver a instabilidade do latch SR (causada quando S = R = 1) é evitar que ela ocorra.
- A Figura 3.23 apresenta um circuito de latch com somente uma entrada, D.
- Como a entrada para a porta AND inferior é sempre o complemento da entrada para a superior, nunca ocorre o problema de ambas as entradas serem 1. Quando D = 1 e o clock for 1, o latch é levado ao estado Q = 1. Quando D = 0 e o clock for 1, ele é levado ao estado Q = 0.

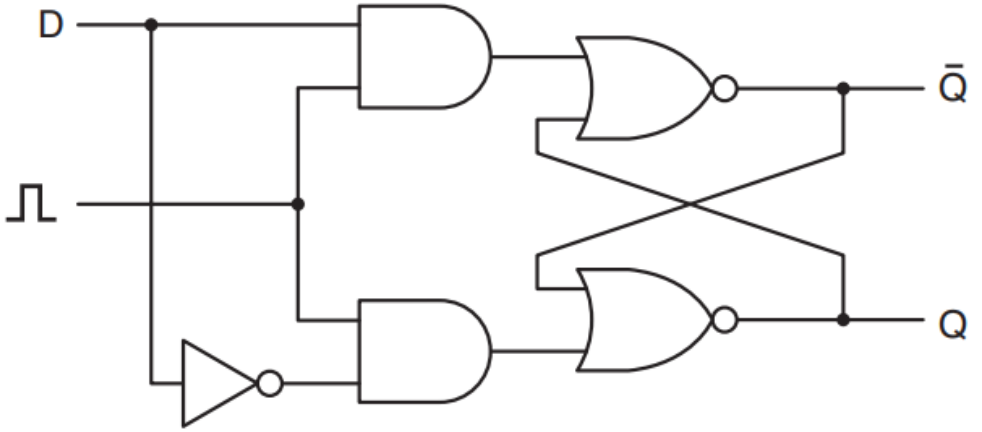
Figura 3.23 *Latch* D com *clock*.



Memórias de 1 bit
Latches D com clock

- Em outras palavras, quando o clock for 1, o valor corrente de D é lido e armazenado no *latch*. Esse circuito, denominado *latch* D com clock, é uma verdadeira memória de 1 bit.
- O valor armazenado sempre estará disponível em Q. Para carregar o valor atual de D na memória, um pulso positivo é colocado na linha do clock.

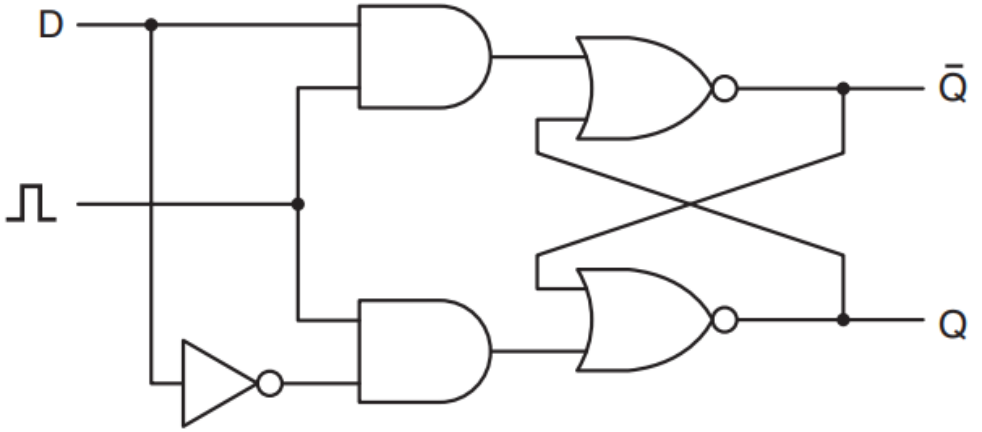
Figura 3.23 *Latch* D com *clock*.



Memórias de 1 bit
Latches D com clock

- Esse circuito requer 11 transistores. Circuitos mais sofisticados (porém, menos óbvios) podem armazenar 1 bit com até seis transistores.
- Esses projetos costumam ser usados na prática. Esse circuito pode permanecer estável indefinidamente, desde que seja aplicada energia (não mostrado).
- Em breve, veremos os circuitos de memória que se esquecem rápido do estado em que estão, a menos que, de alguma forma, sejam “relembrados” constantemente.

Figura 3.23 *Latch* D com *clock*.



Flip-flops

- Em muitos circuitos é necessário ler o valor em determinada linha em dado instante, e armazená-lo.
- Nessa variante, denominada *flip-flop*, a transição de estado não ocorre quando o clock é 1, mas durante a transição de 0 para 1 (borda ascendente), ou de 1 para 0 (borda descendente). Assim, o comprimento do pulso do clock não é importante, contanto que as transições ocorram rapidamente.
- Para dar ênfase, vamos repetir qual é a diferença entre um flip-flop e um latch. Um flip-flop é **disparado pela borda**, enquanto um **latch é disparado pelo nível**. Contudo, fique atento, porque esses termos são muito confundidos na literatura. Muitos autores usam “flip-flop” quando estão se referindo a um latch, e vice-versa.

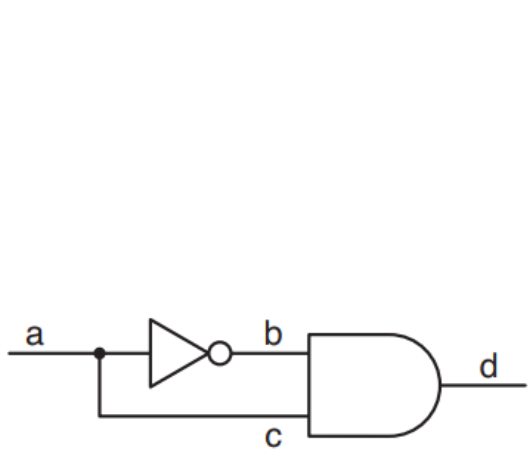
Flip-flops

- Há várias formas de projetar um flip-flop. Por exemplo, se houvesse alguma maneira de gerar um pulso muito curto na borda ascendente do sinal de clock, esse pulso poderia ser alimentado para um latch D. Na verdade, essa maneira existe, e o circuito para ela é mostrado na Figura 3.24(a).
- À primeira vista, poderia parecer que a saída da porta AND seria sempre zero, uma vez que a operação AND de qualquer sinal com seu inverso é zero, mas a situação é um pouco diferente disso.

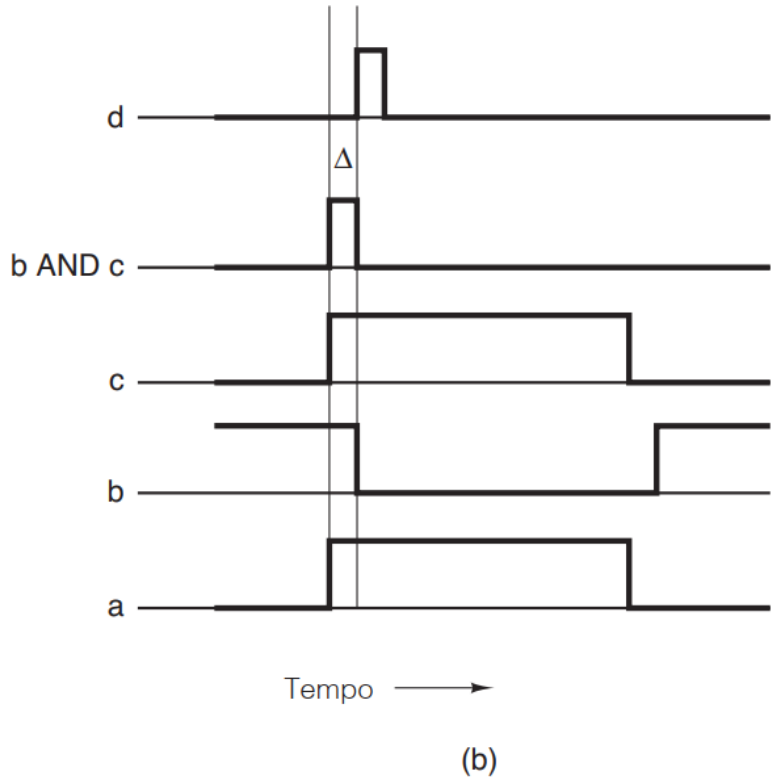
Flip-flops

- O inversor tem um atraso de propagação pequeno, mas não zero, e é esse atraso que faz o circuito funcionar. Suponha que meçamos a tensão nos quatro pontos de medição **a**, **b**, **c** e **d**. O sinal de entrada, medido em **a**, é um pulso de clock longo, como mostrado na parte inferior da Figura 3.24(b).
- O sinal em **b** é mostrado acima dele. Observe que ele está invertido e também ligeiramente atrasado, quase sempre de alguns nanossegundos, dependendo do tipo de inversor utilizado.

Figura 3.24 (a) Gerador de pulso. (b) Temporização em quatro pontos do circuito.



(a)

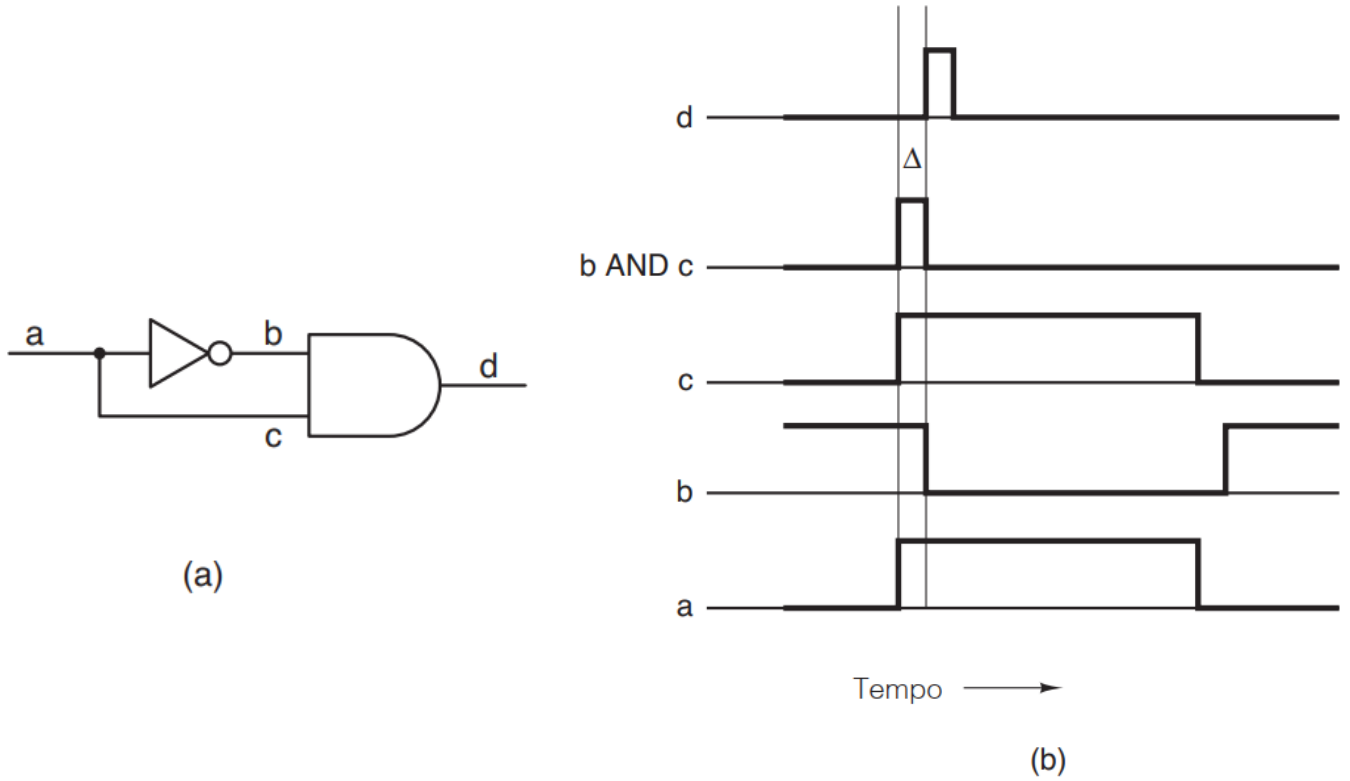


(b)

Flip-flops

- O sinal em **c** também está atrasado, mas apenas pelo tempo correspondente à propagação (à velocidade da luz) do sinal. Se a distância física entre **a** e **c** for, por exemplo, 20 micra (micrômetros), então o atraso de propagação é 0,0001 ns, que é desprezível em comparação com o tempo que o sinal leva para se propagar pelo inversor.
- Assim, para todos os efeitos e propósitos, o sinal em **c** é praticamente idêntico ao sinal em **a**.

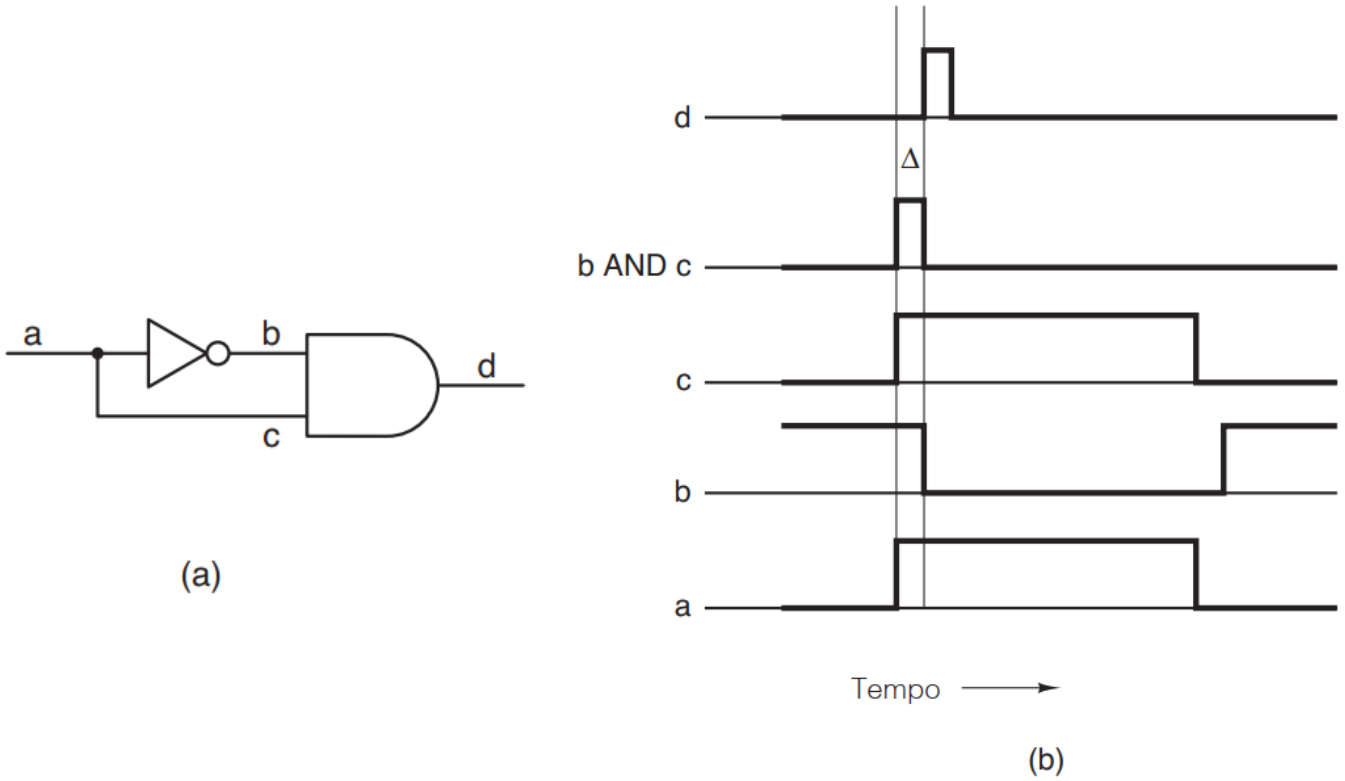
Figura 3.24 (a) Gerador de pulso. (b) Temporização em quatro pontos do circuito.



Flip-flops

- Quando se efetua uma operação and com as entradas para a porta AND, **b** e **c**, o resultado é um pulso curto, como mostra a Figura 3.24(b), onde a largura do pulso, Δ, é igual ao atraso da porta do inversor, em geral 5 ns ou menos.
- A saída da porta AND é exatamente esse pulso deslocado pelo atraso da porta AND, como mostrado na parte superior da Figura 3.24(b). Esse deslocamento de tempo significa apenas que o *latch* D será ativado com um atraso fixo após a fase ascendente do clock, mas não tem efeito sobre a largura do pulso.

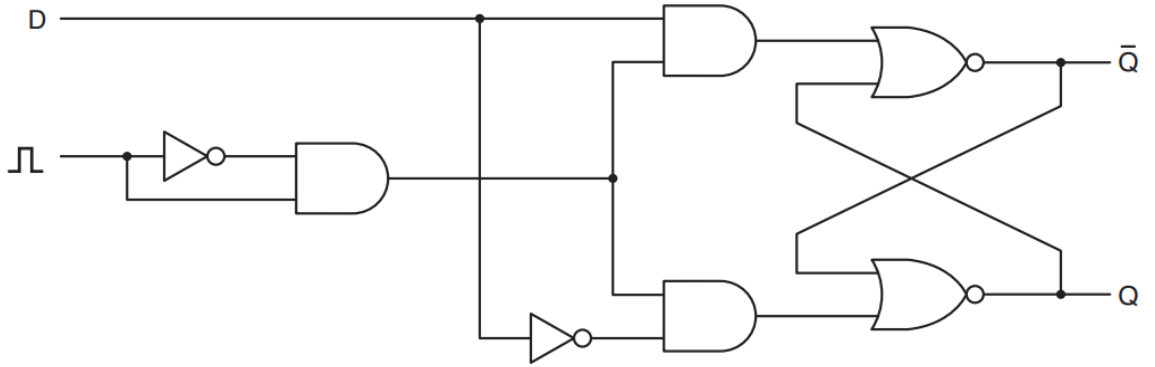
Figura 3.24 (a) Gerador de pulso. (b) Temporização em quatro pontos do circuito.



Flip-flops

- Em uma memória com tempo de ciclo de 10 ns, um pulso de 1 ns para informar quando ler a linha D pode ser curto o bastante, caso em que o circuito completo pode ser o da Figura 3.25. vale a pena observar que esse projeto de *flip-flop* é atraente porque é fácil de entender, embora, na prática, sejam usados *flip-flops* mais sofisticados.

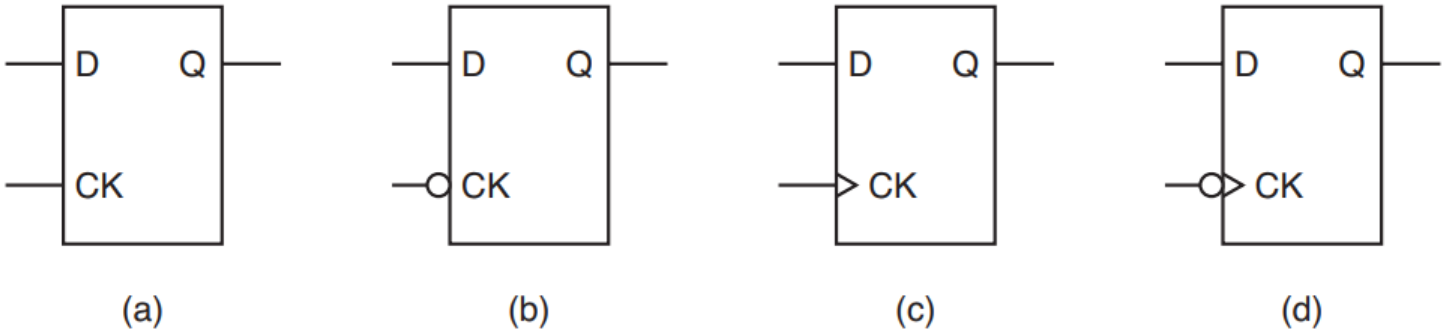
Figura 3.25 *Flip-flop* D.



Flip-flops

- Os símbolos padronizados para latches e flip-flops são mostrados na Figura 3.26. A Figura 3.26(a) é um latch cujo estado é carregado quando o clock, CK, é 1, ao contrário da Figura 3.26(b), que é um latch cujo clock costuma ser 1, mas cai para 0 momentaneamente para carregar o estado a partir de D.

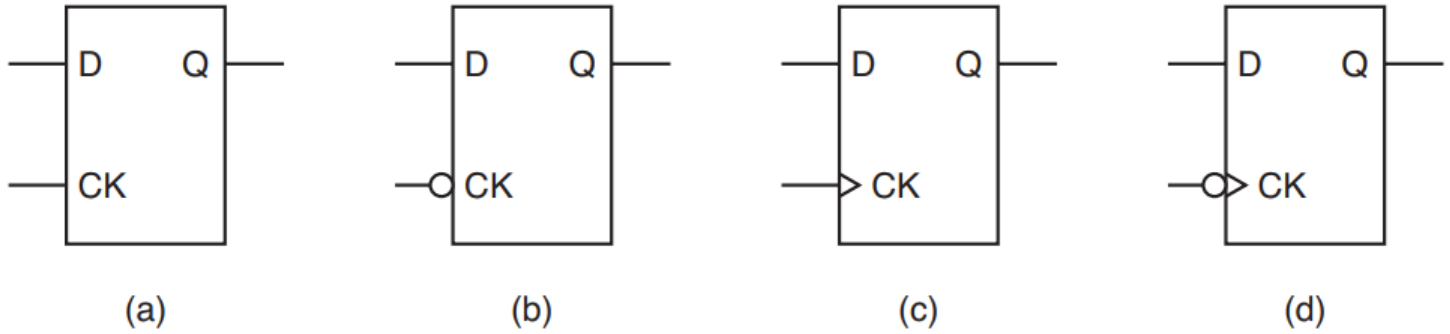
Figura 3.26 *Latches e flip-flops D.*



Flip-flops

- As figuras 3.26(c) e (d) são flip-flops em vez de latches, o que é indicado pelo símbolo em ângulo nas entradas do clock. A Figura 3.26(c) muda de estado na borda ascendente do pulso do clock (transição de 0 para 1), enquanto a Figura 3.26(d) muda de estado na borda descendente (transição de 1 para 0).
- Muitos *latches* e *flip-flops* (mas não todos) também têm $\bar{Q}$ como uma saída, e alguns têm duas entradas adicionais *Set* ou *Preset* (que forçam o estado para Q = 1) e *Reset* ou *Clear* (que forçam o estado para Q = 0).

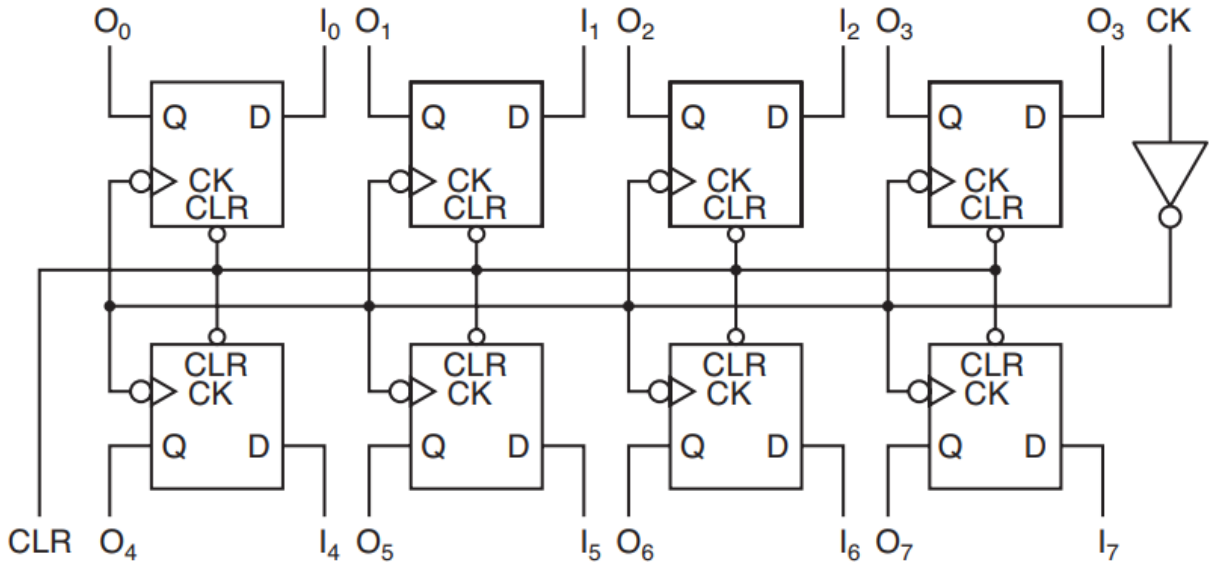
Figura 3.26 *Latches* e *flip-flops* D.



Registradores

- Flip-flops podem ser combinados em grupos para criar registradores, que mantêm tipos de dados com comprimentos maiores do que 1 bit.
- O registrador na Figura 3.27 mostra como oito flip-flops podem ser ligados para formar um registrador armazenador de 8 bits.
- O registrador aceita um valor de entrada de 8 bits (I_0 a I_7) quando o *clock CK* fizer uma transição.
- Para implementar um registrador, todas as linhas de clock são conectadas ao mesmo sinal de entrada CK, de modo que, quando o clock fizer uma transição, cada registrador aceitará o novo valor de dados de 8 bits no barramento de entrada.

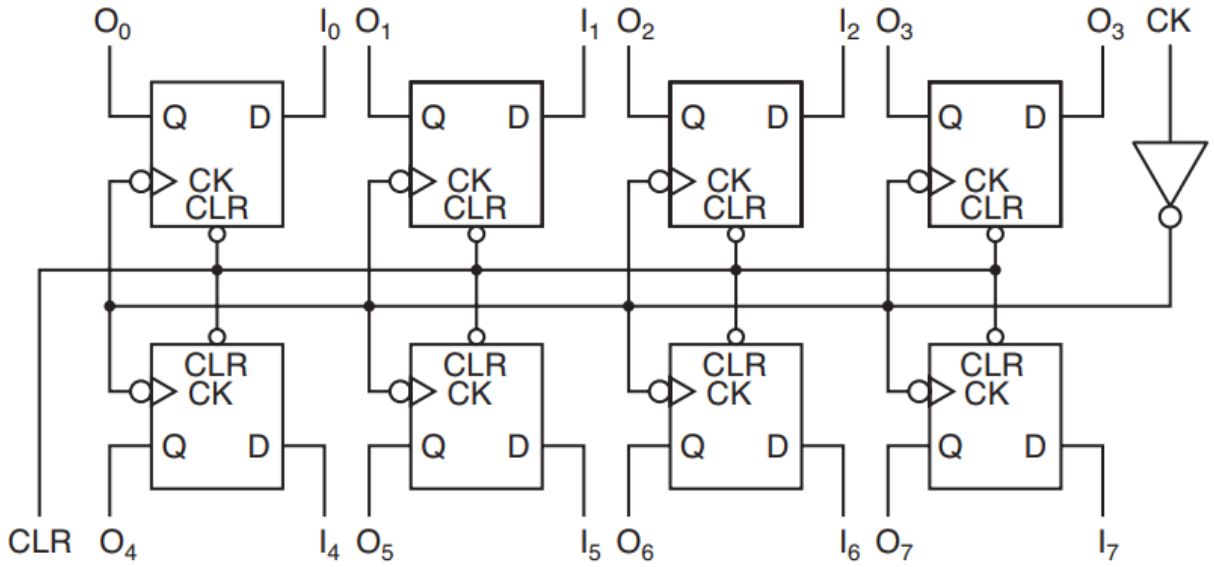
Figura 3.27 Um registrador de 8 bits construído a partir de *flip-flops* de único bit.



Registradores

- Os próprios flip-flops são do tipo da Figura 3.26(d), mas as bolhas de inversão nos flip-flops são canceladas pelo inversor ligado ao sinal de *clock CK*, de modo que os flip-flops são carregados na transição ascendente do clock.
- Todos os oito sinais clear também são ligados, de modo que, quando o sinal *clear CLR* passar para 0, todos os flip-flops serão forçados a passar para o seu estado 0.
- Caso você queira saber por que o sinal de *clock CK* é invertido na entrada e depois invertido novamente em cada flip-flop, um sinal de entrada pode não ter corrente suficiente para alimentar todos os oito flip-flops; o inversor da entrada, na realidade, está sendo usado como um amplificador.

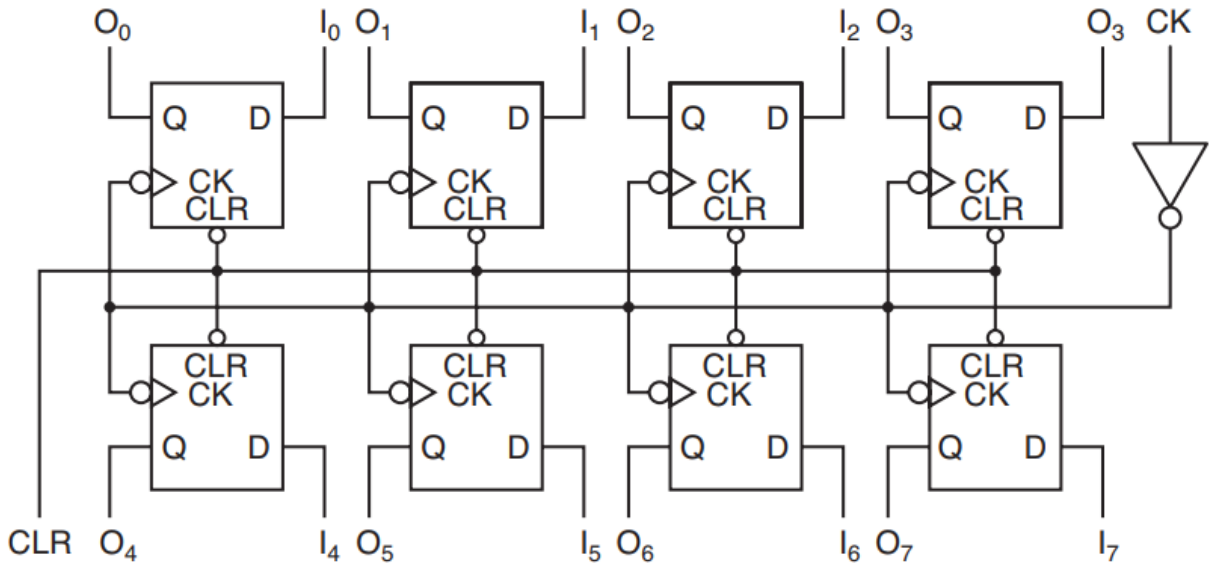
Figura 3.27 Um registrador de 8 bits construído a partir de *flip-flops* de único bit.



Registradores

- Quando tivermos projetado um registrador de 8 bits, poderemos usá-lo como um bloco de montagem para criar registradores maiores.
- Por exemplo, um registrador de 32 bits poderia ser criado pela combinação de dois registradores de 16 bits, unindo seus sinais de *clock CK* e sinais de *clear CLR*.

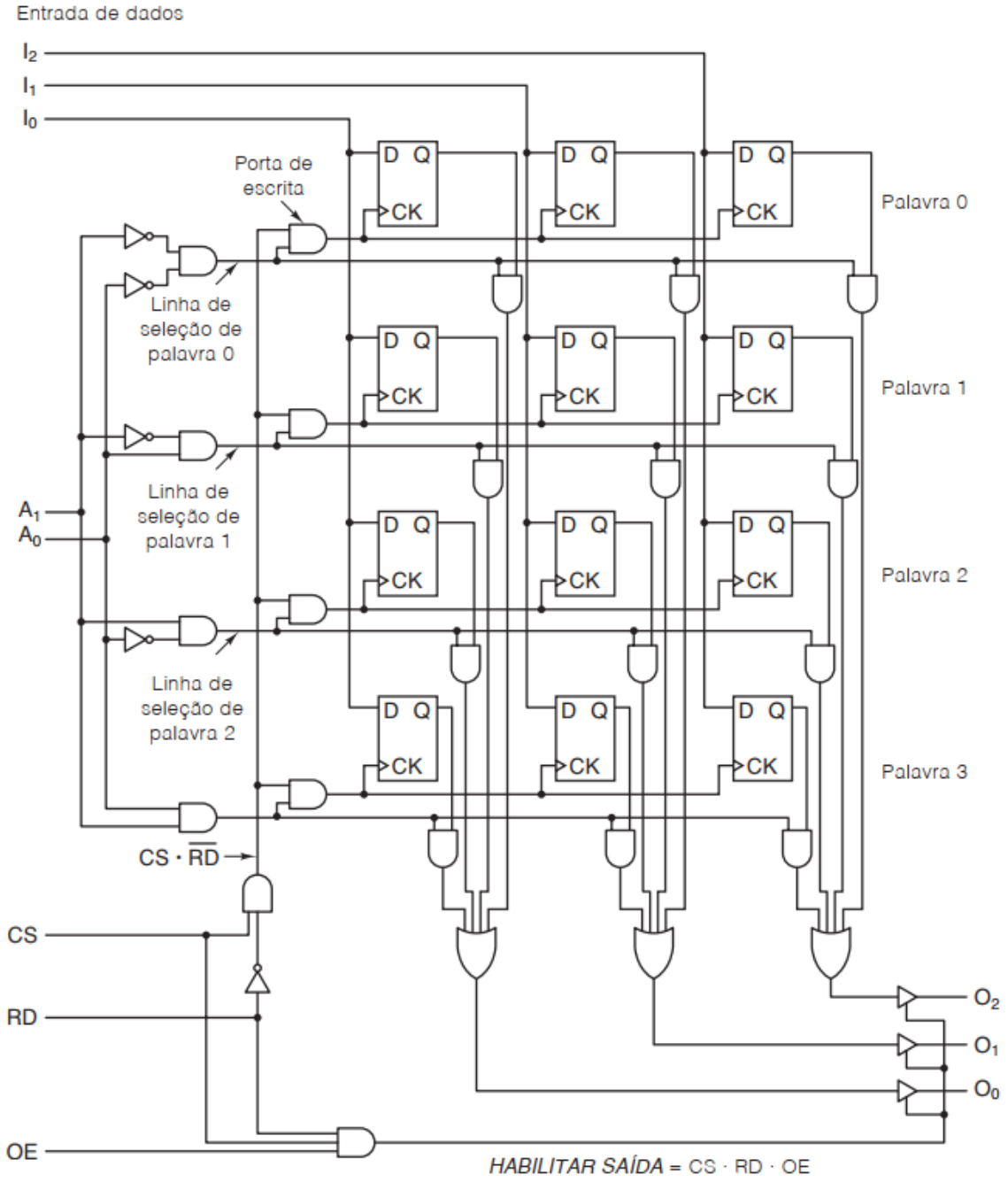
Figura 3.27 Um registrador de 8 bits construído a partir de *flip-flops* de único bit.



Organização da memória

- Embora agora tenhamos progredido de uma simples memória de 1 bit da Figura 3.23 para a de 8 bits da Figura 3.27, para construir memórias grandes é preciso uma organização diferente, na qual palavras individuais podem ser endereçadas.
- Uma organização de memória muito utilizada e que obedece a esse critério é mostrada na Figura 3.28. Esse exemplo ilustra uma memória com quatro palavras de 3 bits.
- Cada operação lê ou escreve uma palavra completa de 3 bits. Embora uma capacidade total de memória de 12 bits seja pouco mais do que nosso flip-flop octal, ela requer um número menor de pinos e, mais importante, o projeto pode ser estendido com facilidade para memórias grandes.
- Observe que o número de palavras é sempre uma potência de 2.

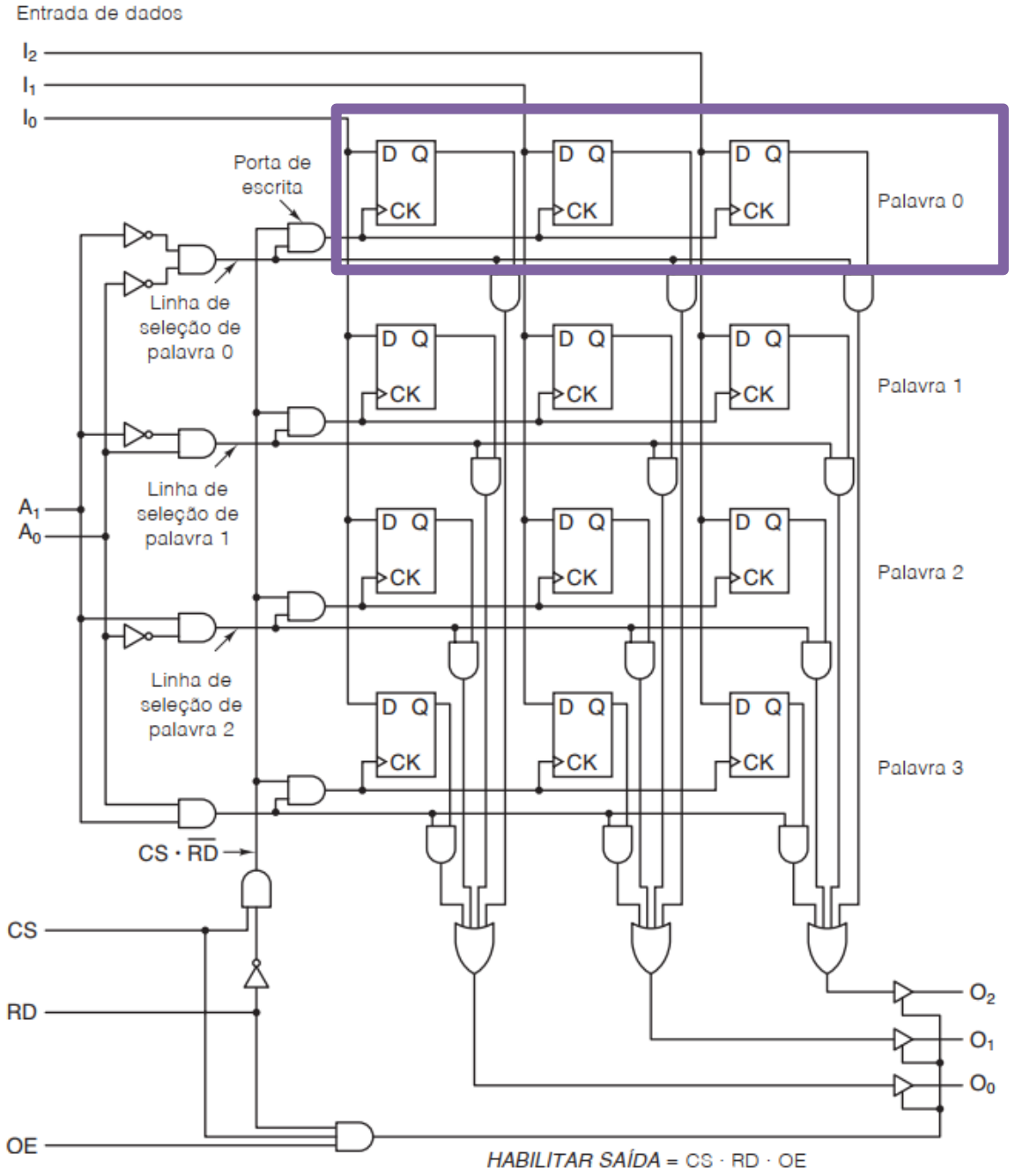
Figura 3.28 Diagrama lógico para uma memória 4 x 3. Cada linha é uma das quatro palavras de 3 bits. Uma operação de leitura ou escrita sempre lê ou escreve uma palavra completa.



Organização da memória

- Embora à primeira vista talvez pareça complicada, a memória da Figura 3.28 na verdade é bastante simples devido à sua estrutura regular.
- Ela tem oito linhas de entrada e três de saída.
- Três entradas são de dados: I_0, I_1 e I_2; duas são para o endereço: A_0 e A_1; e três são para controle: CS para *chip select* (selecionar chip), RD para distinguir entre ler e escrever e OE para *output enable* (habilitar saída).
- As três saídas são para dados: O0, O1 e O2. É interessante notar que essa memória de 12 bits requer menos sinais que o registrador de 8 bits anterior. Este requer 20 sinais, incluindo alimentação e terra, enquanto a memória de 12 bits requer apenas 13 sinais. O bloco de memória requer menos sinais porque, diferente do registrador, os bits de memória compartilham um sinal de saída.
- Nessa memória, cada um dos 4 bits de memória compartilha um sinal de saída. O valor das linhas de endereço determina quais dos 4 bits de memória pode receber ou enviar um valor.

Figura 3.28 Diagrama lógico para uma memória 4 x 3. Cada linha é uma das quatro palavras de 3 bits. Uma operação de leitura ou escrita sempre lê ou escreve uma palavra completa.



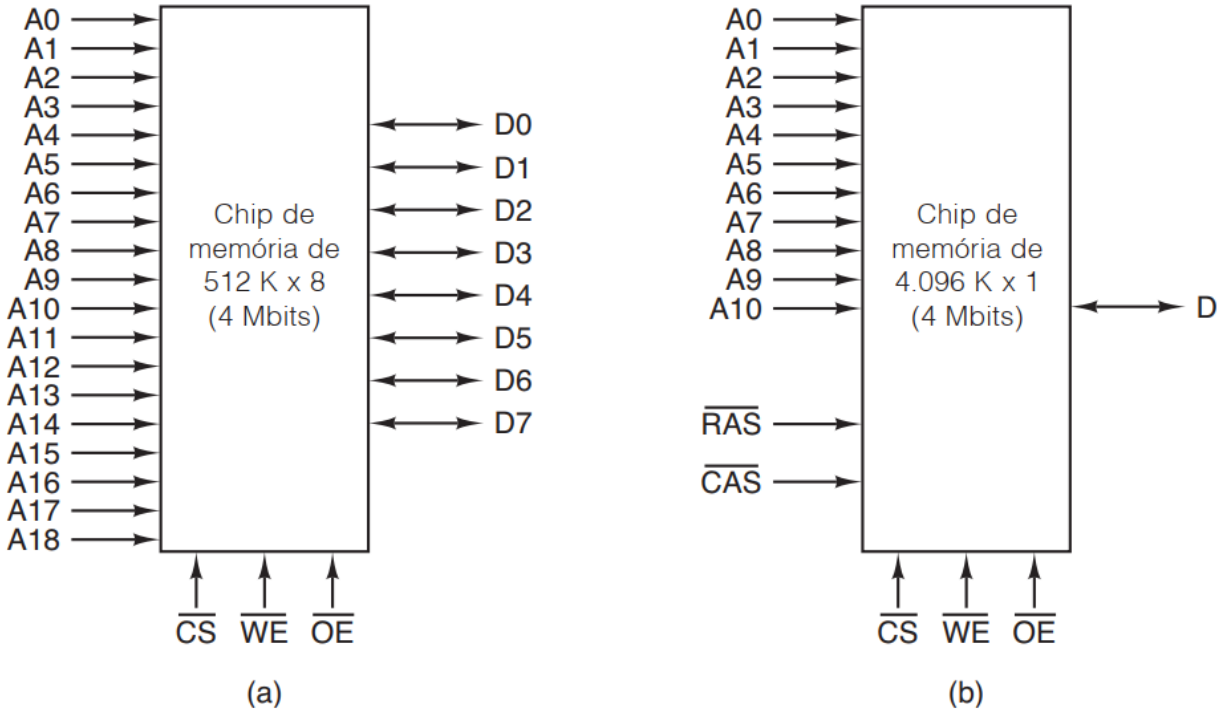
Chips de memória

- O bom da memória da Figura 3.28 é que ela pode ser ampliada com facilidade para tamanhos maiores.
- Na figura 3.28, a memória é 4 × 3, isto é, quatro palavras de 3 bits cada. Para ampliá-la para 4 × 8, basta adicionar cinco colunas de quatro flip-flops cada, bem como adicionar cinco linhas de entrada e cinco linhas de saída. Para passar de 4 × 3 para 8 × 3, devemos acrescentar quatro linhas de três flip-flops cada, bem como uma linha de endereço A2.
- Com esse tipo de estrutura, o número de palavras na memória deve ser uma potência de 2 para que haja o máximo de eficiência, mas o número de bits em uma palavra pode ser qualquer um.

Chips de memória

- Há vários modos de organizar o chip para qualquer tamanho de memória dado.
- A Figura 3.30 mostra duas organizações possíveis para um chip de memória mais antigo de 4 Mbits de tamanho: 512 K × 8 e 4.096 K × 1.
- *Nota: Os tamanhos de chips de memória costumam ser citados em bits em vez de bytes.*

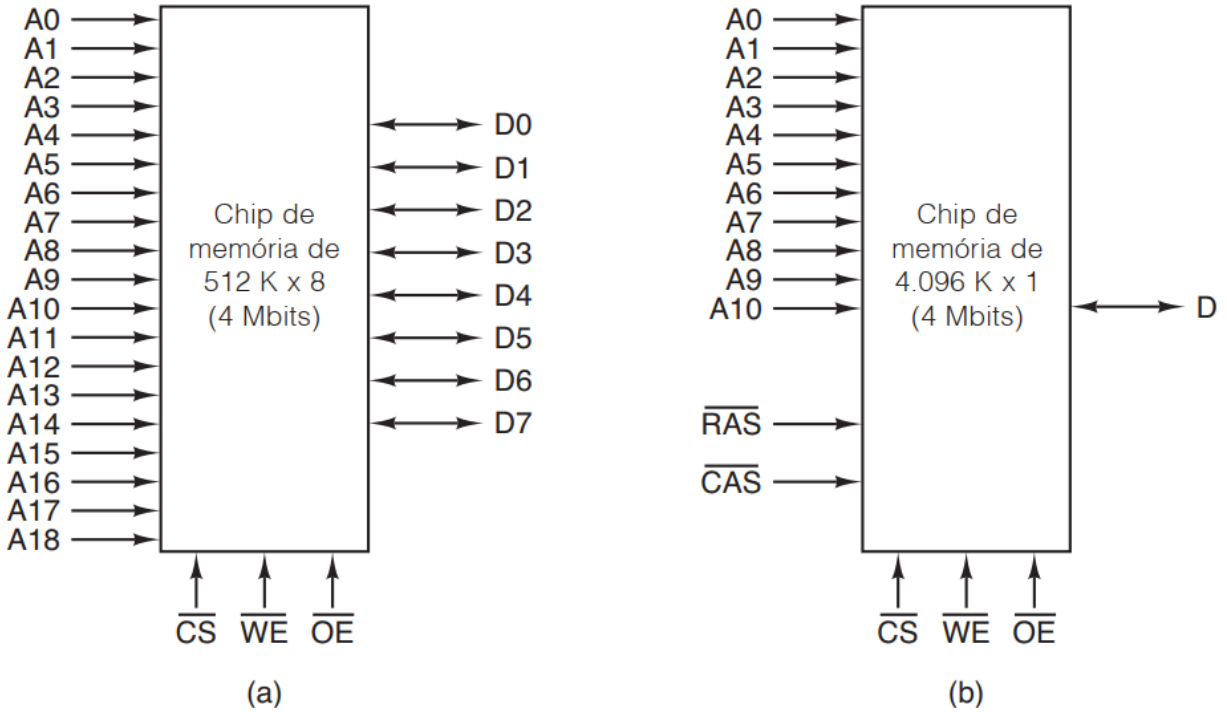
Figura 3.30 Dois modos de organizar um chip de memória de 4 Mbits.



Chips de memória

- Na Figura 3.30(a), são necessárias 19 linhas de endereço para endereçar um dos 219 bytes e oito linhas de dados para carregar e armazenar o byte selecionado.

Figura 3.30 Dois modos de organizar um chip de memória de 4 Mbits.

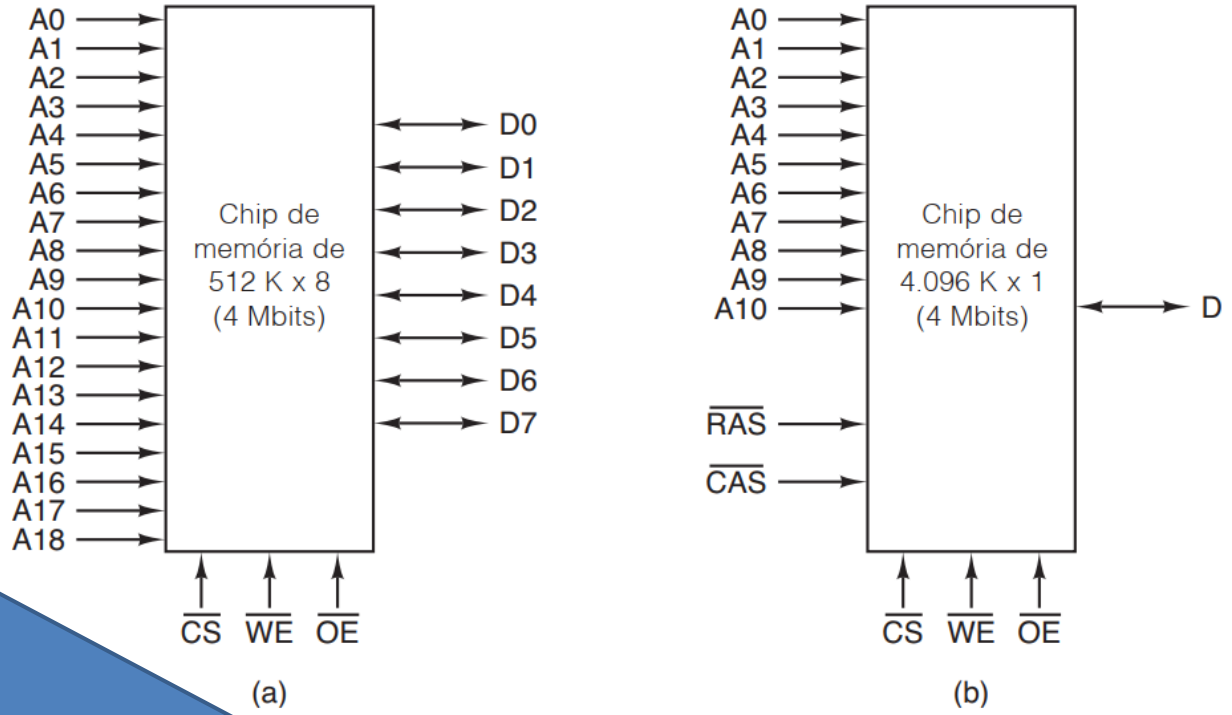


Chips de memória

Strobe é um termo semelhante à flash de luz.

- Na Figura 3.30(b), é usado um esquema de endereçamento diferente. Esse chip é organizado internamente como uma matriz 2.048 × 2.048 de células de 1 bit, o que dá 4 Mbits.
- Para endereçar o chip, em primeiro lugar uma linha é selecionada ao se colocar seu número de 11 bits nos pinos de endereço. Então o $\overline{\text{RAS}}$ (*Row Address Strobe* – strobe de endereço de linha) é afirmado.
- Em seguida, um número de coluna é colocado nos pinos de endereço e o $\overline{\text{CAS}}$ (*Column Address Strobe* – strobe de endereço de coluna) é afirmado.
- O chip responde aceitando ou entregando um bit de dados.

Figura 3.30 Dois modos de organizar um chip de memória de 4 Mbits.



O termo afirmado é utilizado para designar uma ação. Por exemplo, uma entrada A é afirmada em nível lógico 1 e $\bar{A}$ é afirmada em nível lógico 0.

Chips de memória

- Chips de memória de grande porte costumam ser construídos como matrizes $n \times n$ endereçadas por linha e coluna.
- Essa organização reduz o número de pinos requerido, mas também torna mais lento o endereçamento do chip, já que são necessários dois ciclos, um para a linha e outro para a coluna.
- Para recuperar um pouco da velocidade perdida por esse projeto, alguns chips de memória podem receber um endereço de linha acompanhado por uma sequência de endereços de coluna para acessar bits consecutivos em uma linha.

Chips de memória

- Anos atrás, os maiores chips de memória costumavam ser organizados como os da Figura 3.30(b).
- À medida que as palavras de memória cresciam de 8 bits até 32 bits e mais, os chips de 1 bit começaram a ser inconvenientes. Construir uma memória com uma palavra de 32 bits usando chips de 4.096 K × 1 requer 32 chips em paralelo.
- Esses 32 chips têm capacidade total de no mínimo 16 MB, ao passo que usar chips de 512 K × 8 requer somente quatro chips em paralelo e permite memórias pequenas, de até 2 MB. Para evitar ter 32 chips para memória, grande parte dos fabricantes lançou famílias com 4, 8 e 16 bits de largura.
- A situação com as palavras de 64 bits é pior ainda, é claro.

RAMs e ROMs

- Todas as memórias que vimos até agora podem ser escritas e lidas.
- Elas são denominadas memórias RAM (***Random Access Memory*** – memória de acesso aleatório).
- RAMs podem ser de duas variedades, estáticas e dinâmicas.
- Nas estáticas (**Static RAMs – SRAMs**), a construção interna usa circuitos similares ao nosso flip-flop D básico.
- Uma das propriedades dessas memórias é que seus conteúdos são conservados enquanto houver fornecimento de energia: segundos, minutos, horas e até mesmo dias. As RAMs estáticas são muito rápidas.
- Um tempo de acesso típico é da ordem de **um nanossegundo ou menos**. Por essa razão, elas são muito usadas como memória cache.

RAMs e ROMs

- RAMs dinâmicas (***Dynamic RAMs* - DRAMs**), ao contrário, não usam flip-flops.
- Em vez disso, uma RAM dinâmica é um arranjo de células, cada uma contendo um transistor e um pequenino capacitor.
- Os capacitores podem ser carregados ou descarregados, permitindo que 0s e 1s sejam armazenados.
- Como a carga elétrica tende a vazar, cada bit em uma RAM dinâmica deve ser renovado (recarregado) com alguns milissegundos de intervalo para evitar que os dados desapareçam.
- Como a lógica externa é que tem de cuidar da renovação, as RAMs dinâmicas precisam de uma interface mais complexa do que as estáticas, embora em muitas aplicações essa desvantagem seja compensada por suas maiores capacidades.

RAMs e ROMs

- As RAMs dinâmicas precisam de apenas um transistor e um capacitor por bit, em comparação com os seis transistores por bit para a melhor RAM estática, elas têm densidade muito alta (muitos bits por chip).
- Por essa razão, as memórias principais quase sempre são construídas com RAMs dinâmicas, mas são mais lentas (**dezenas de nanossegundos**).
- Dessa maneira, a combinação de uma cache de RAM estática e uma memória principal de RAM dinâmica tenta combinar as boas propriedades de cada uma.

RAMs e ROMs

- RAMs não são o único tipo de chip de memória.
- Em muitas aplicações, como brinquedos, eletrodomésticos e carros, o programa e alguns dos dados devem permanecer armazenados mesmo quando o fornecimento de energia for interrompido.
- Uma vez instalados, nem o programa nem os dados são alterados.
- Esses requisitos levaram ao desenvolvimento de ROMs (**Read-Only Memories - memórias somente de leitura**), que não podem ser alteradas nem apagadas, seja intencionalmente ou não.
- Os dados de uma ROM são inseridos durante sua fabricação por um processo que expõe um material fotossensível por meio de uma máscara que contém o padrão de bits desejado e então grava o padrão sobre a superfície exposta (ou não exposta).
- A única maneira de mudar o programa em uma ROM é substituir o chip inteiro.

RAMs e ROMs

Figura 3.32 Comparação entre vários tipos de memórias (Arranjo de portas programável em campo).

Tipo	Categoria	Modo de apagar	Byte alterável	Volátil	Utilização típica
SRAM	Leitura/escrita	Elétrico	Sim	Sim	*Cache* de nível 2
DRAM	Leitura/escrita	Elétrico	Sim	Sim	Memória principal (antiga)
SDRAM	Leitura/escrita	Elétrico	Sim	Sim	Memória principal (nova)
ROM	Somente de leitura	Não é possível	Não	Não	Equipamentos de grande volume
PROM	Somente de leitura	Não é possível	Não	Não	Equipamentos de pequeno volume
EPROM	Principalmente leitura	Luz UV	Não	Não	Prototipagem de dispositivos
EEPROM	Principalmente leitura	Elétrico	Sim	Não	Prototipagem de dispositivos
Flash	Leitura/escrita	Elétrico	Não	Não	Filme para câmera digital

Chips de CPU

- Todas as CPUs modernas são contidas em um único chip, o que faz sua interação com o resto do sistema ser bem definida.
- Cada chip de CPU tem um conjunto de pinos por meio dos quais deve ocorrer toda sua comunicação com o mundo exterior.
- Alguns pinos produzem sinais da CPU para o mundo exterior (pinos de saída); outros aceitam sinais do mundo exterior (pinos de entrada); alguns podem fazer as duas coisas (pinos de entrada e saída E/S).
- Entendendo a função de todos esses pinos, podemos aprender como a CPU interage com a memória e os dispositivos de E/S no nível lógico digital.

Chips de CPU

- Os pinos de um chip de CPU podem ser divididos em três tipos: de **endereço**, de **dados** e de **controle**.
- Eles são conectados a pinos similares na memória e a chips de E/S por meio de um conjunto de fios paralelos, denominado **barramento**.
- Para buscar uma instrução, primeiro a CPU coloca o endereço de memória daquela instrução em seus pinos de endereço.
- Então, ela ativa uma ou mais linhas de controle para informar à memória que ela quer ler uma palavra, por exemplo.
- A memória responde colocando a palavra requisitada nos pinos de dados da CPU e ativando um sinal que informa o que acabou de fazer.
- Quando percebe esse sinal, a CPU aceita a palavra e executa a instrução.

Chips de CPU

- Dois dos parâmetros fundamentais que determinam o desempenho de uma CPU são o número de pinos de endereço e o número de pinos de dados.
- Um chip com m pinos de endereço pode endereçar até 2^m localizações de memória.
- Valores comuns de m são 16, 32 e 64.
- De modo semelhante, um chip com n pinos de dados pode ler ou escrever uma palavra de n bits em uma única operação.
- Valores comuns de n são 8, 32 e 64.

Chips de CPU

- Uma CPU com 8 pinos de dados efetuará quatro operações para ler uma palavra de 32 bits, enquanto uma CPU com 32 pinos de dados pode executar a mesma tarefa em uma única operação.
- Assim, o chip com 32 pinos de dados é muito mais rápido; porém, invariavelmente, também é mais caro.

Chips de CPU

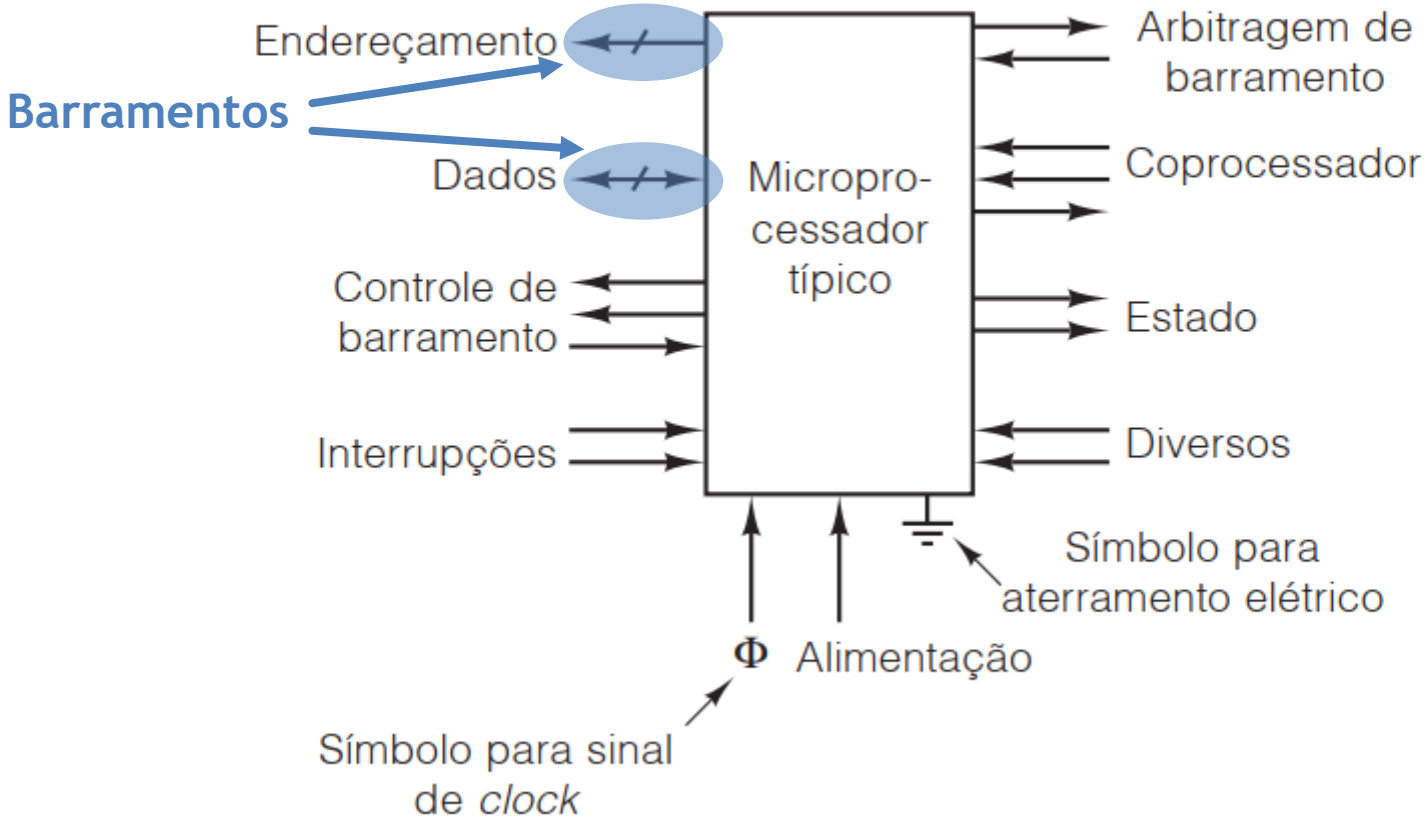
- Além dos pinos de endereço e de dados, cada CPU tem alguns pinos de controle.
- Os pinos de controle regulam o fluxo e a temporização de dados que vêm da CPU e vão para ela, além de ter outras utilizações diversas.
- Todas as CPUs têm pinos para energia elétrica (geralmente $+1{,}2$ volt a $+\ 1{,}5$ volt), para terra e para um sinal de *clock* (uma onda quadrada com uma frequência bem definida), mas os outros pinos variam muito de um chip para outro.

Chips de CPU

- Não obstante, os pinos de controle podem ser agrupados aproximadamente nas seguintes categorias principais:
  - Controle de barramento.
  - Interrupções.
  - Arbitragem de barramento.
  - Sinalização de coprocessador.
  - Estado.
  - Diversos.

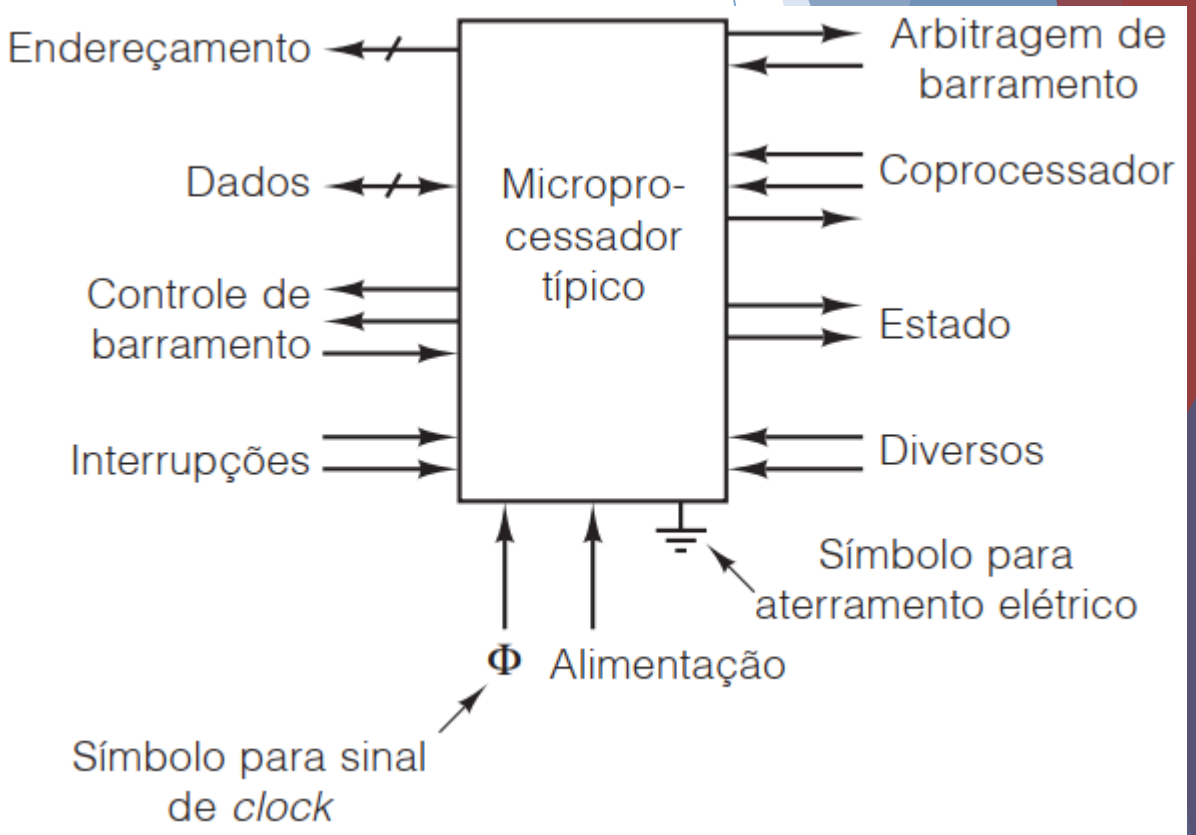
Chips de CPU

Figura 3.34 Pinagem lógica de uma CPU genérica. As setas indicam sinais de entrada e sinais de saída. Os segmentos de reta diagonais indicam que são utilizados vários pinos. Há um número que indica quantos são os pinos para uma CPU específica.



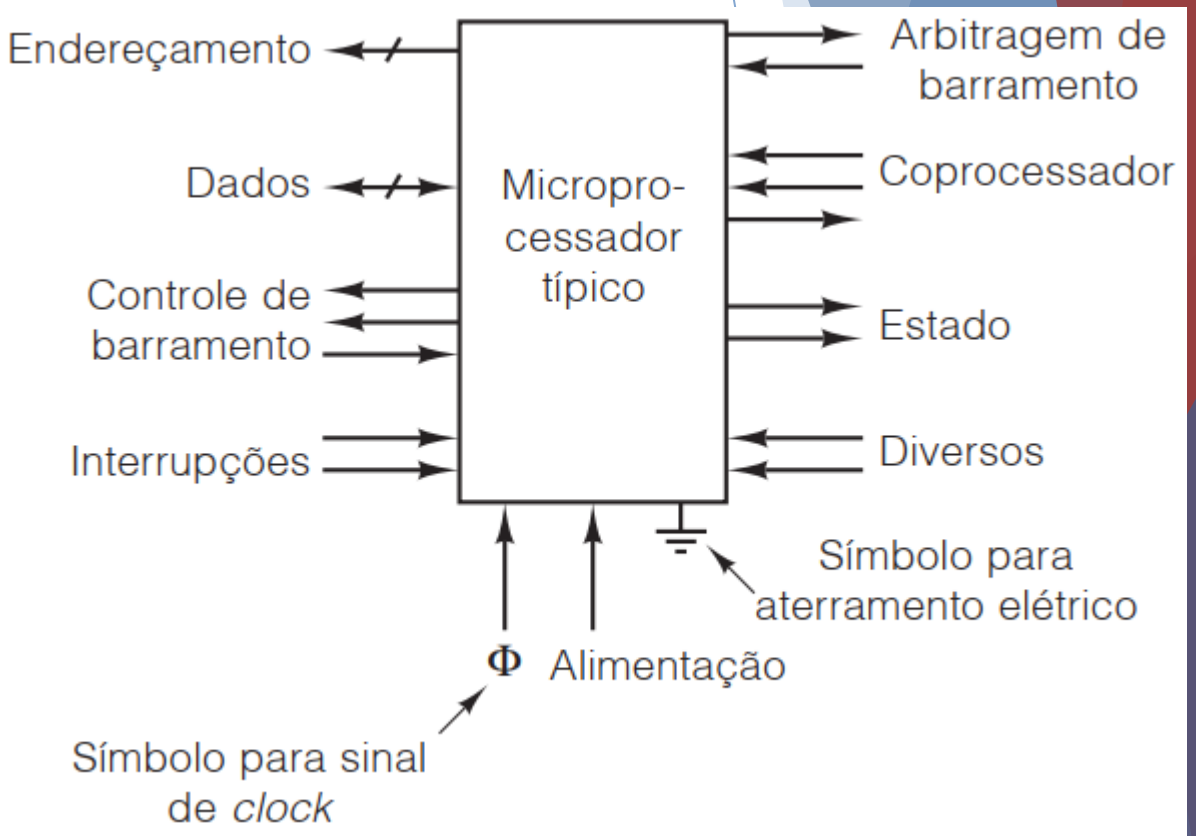
Chips de CPU

- Em grande parte dos sistemas, a CPU pode dizer a um dispositivo de E/S que inicie uma operação e então continuar e fazer outra coisa qualquer enquanto o dispositivo de E/S está realizando seu trabalho.
- Quando a E/S estiver concluída, o chip controlador de E/S ativa um sinal em um desses pinos para interromper a CPU e fazê-la prestar algum serviço ao dispositivo de E/S, por exemplo, verificar se ocorreram erros de E/S.
- Algumas CPUs têm um pino de saída para confirmar o sinal de interrupção.



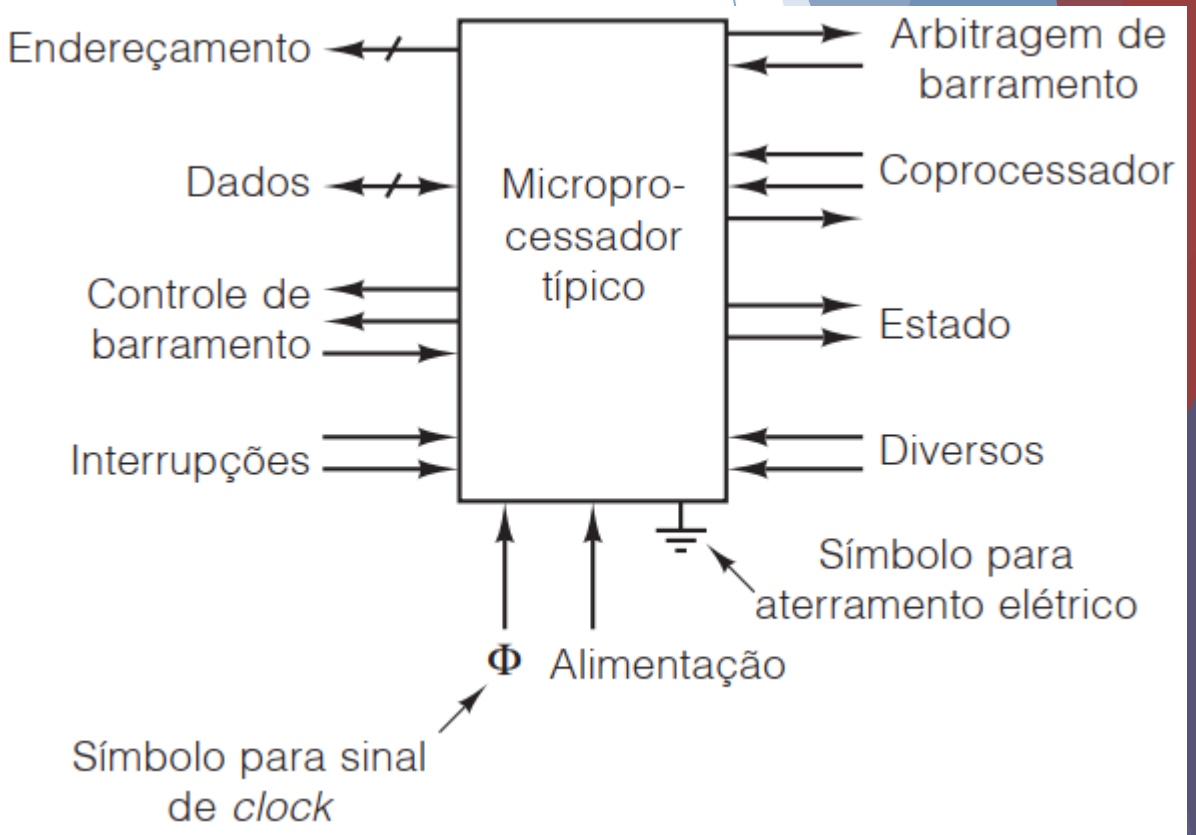
Chips de CPU

- A maioria dos pinos de controle do barramento são saídas da CPU para o barramento (e, portanto, entradas para a memória e chips de E/S) que informam se a CPU quer ler ou escrever na memória ou fazer outra coisa qualquer.
- A CPU usa esses pinos para controlar o resto do sistema e informar o que ela quer fazer.
- Os pinos de interrupção são entradas que vêm de dispositivos de E/S para a CPU.



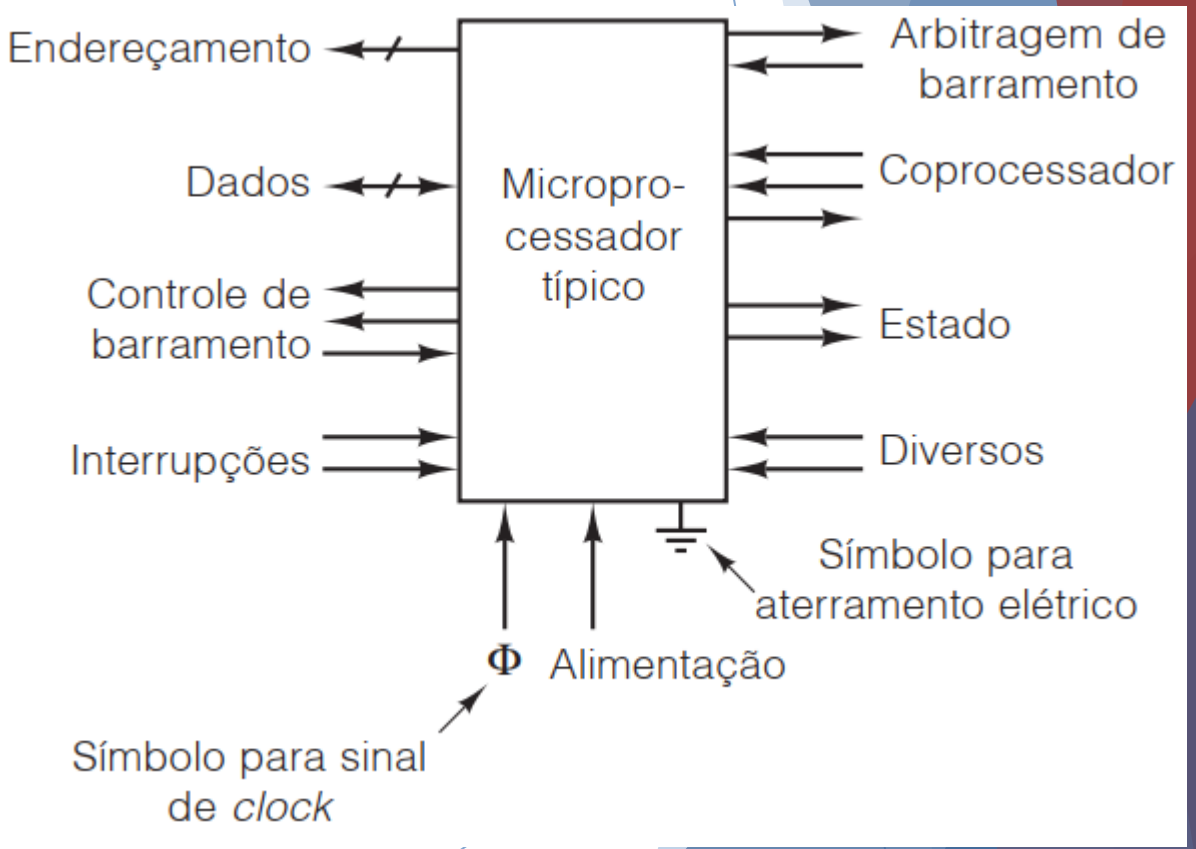
Chips de CPU

- Os pinos de arbitragem de barramento são necessários para disciplinar o tráfego no barramento de modo a impedir que dois dispositivos tentem usá-lo ao mesmo tempo.
- Do ponto de vista da arbitragem, a CPU é um dispositivo e tem de requisitar o barramento como qualquer outro.
- Alguns chips de CPUs são projetados para funcionar com coprocessadores, como chips de ponto flutuante, mas às vezes também com chips gráficos ou outros chips.



Chips de CPU

- Para facilitar a comunicação entre CPU e coprocessador, há pinos especiais dedicados a fazer e aceitar requisições.
- Além desses sinais, há outros pinos diversos presentes em algumas CPUs.
- Alguns deles fornecem ou aceitam informações de estado, outros são úteis para depuração ou para reiniciar o computador, e outros mais estão presentes para garantir a compatibilidade com chips de E/S mais antigos.



Barramentos de computador

- Um barramento é um caminho elétrico comum entre vários dispositivos.
- Os barramentos podem ser categorizados por sua função.
- Podem ser usados no interior da CPU para transportar dados de e para a ULA ou ser externos à CPU, para conectá-la à memória ou a dispositivos de E/S.
- Cada tipo tem seus próprios requisitos e propriedades.

Barramentos de computador

- Os primeiros computadores pessoais tinham somente um barramento externo, ou barramento do sistema, que consistia em 50 a 100 fios de cobre paralelos gravados na placa-mãe, com conectores a intervalos regulares para ligação com a memória e placas de E/S.
- Os computadores pessoais modernos em geral têm um barramento de uso especial entre a CPU e a memória e (pelo menos) outro barramento para os dispositivos de E/S.
- Um sistema mínimo, com um barramento de memória e um barramento de E/S, é ilustrado na Figura 3.35.

Barramentos de computador

- Na literatura, às vezes os barramentos são representados por setas largas e sombreadas, como nesta figura.
- A distinção entre essas setas e uma linha reta cortada por um pequeno segmento de reta inclinado acompanhado de um número de bits é sutil.
- Quando todos os bits são do mesmo tipo, por exemplo, todos são bits de endereço ou todos são bits de dados, então costuma ser usada a representação pelo segmento de reta diagonal.
- Quando estão envolvidas linhas de endereço, de dados e de controle, a seta larga sombreada é a mais comum.

Barramentos de computador

Figura 3.35 Sistema de computador com vários barramentos.