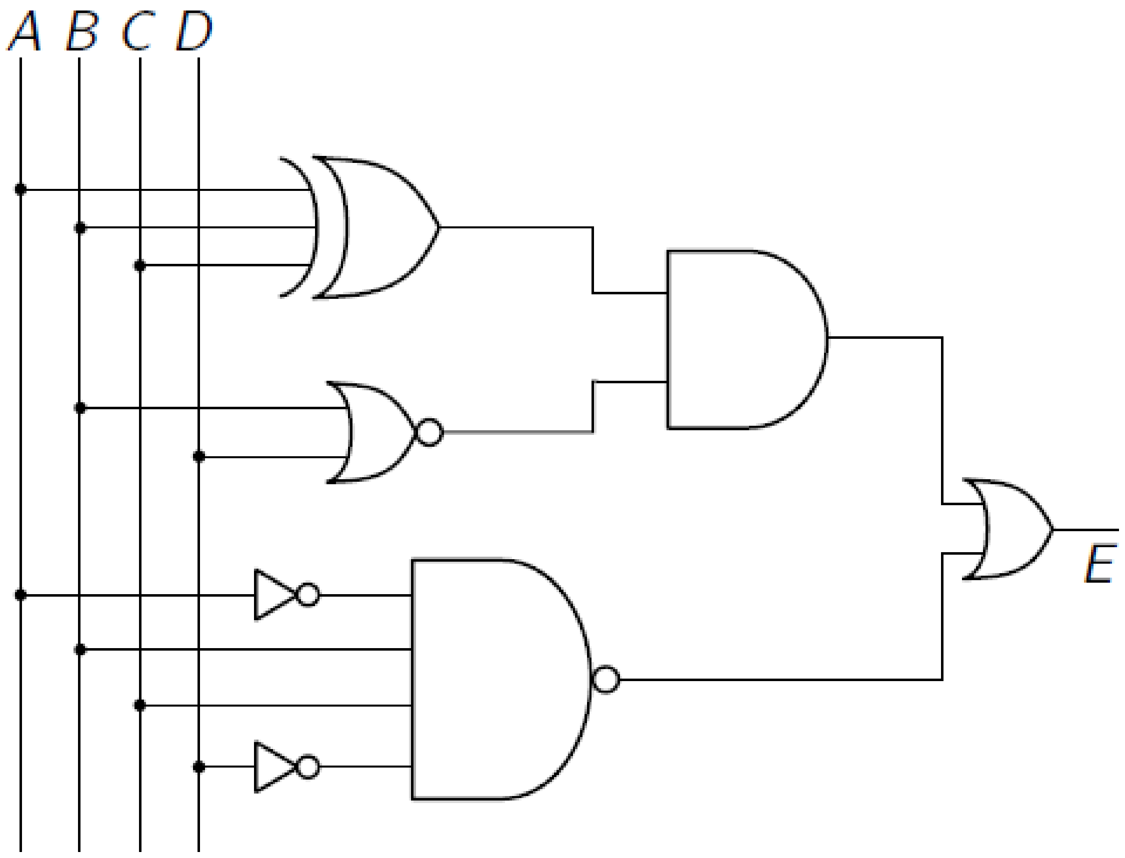
Aula 06 - Circuitos combinatórios

Sumário

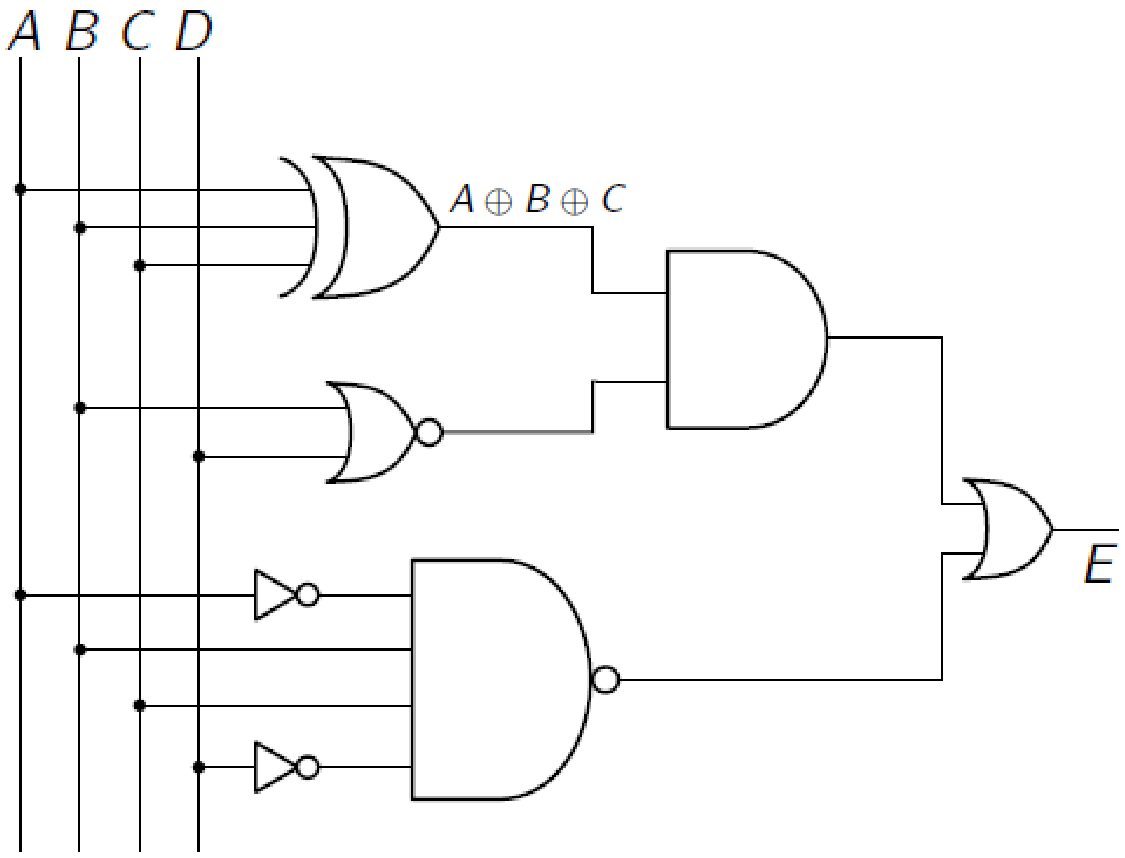
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



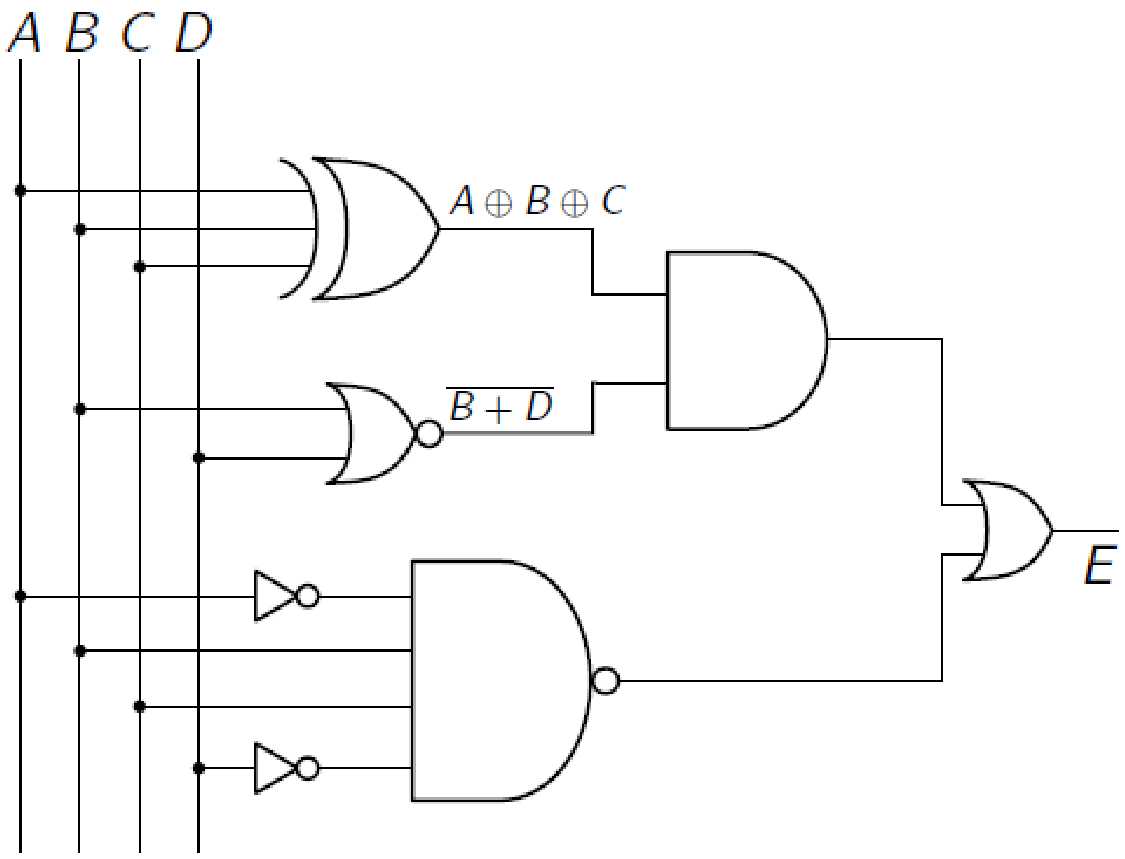
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



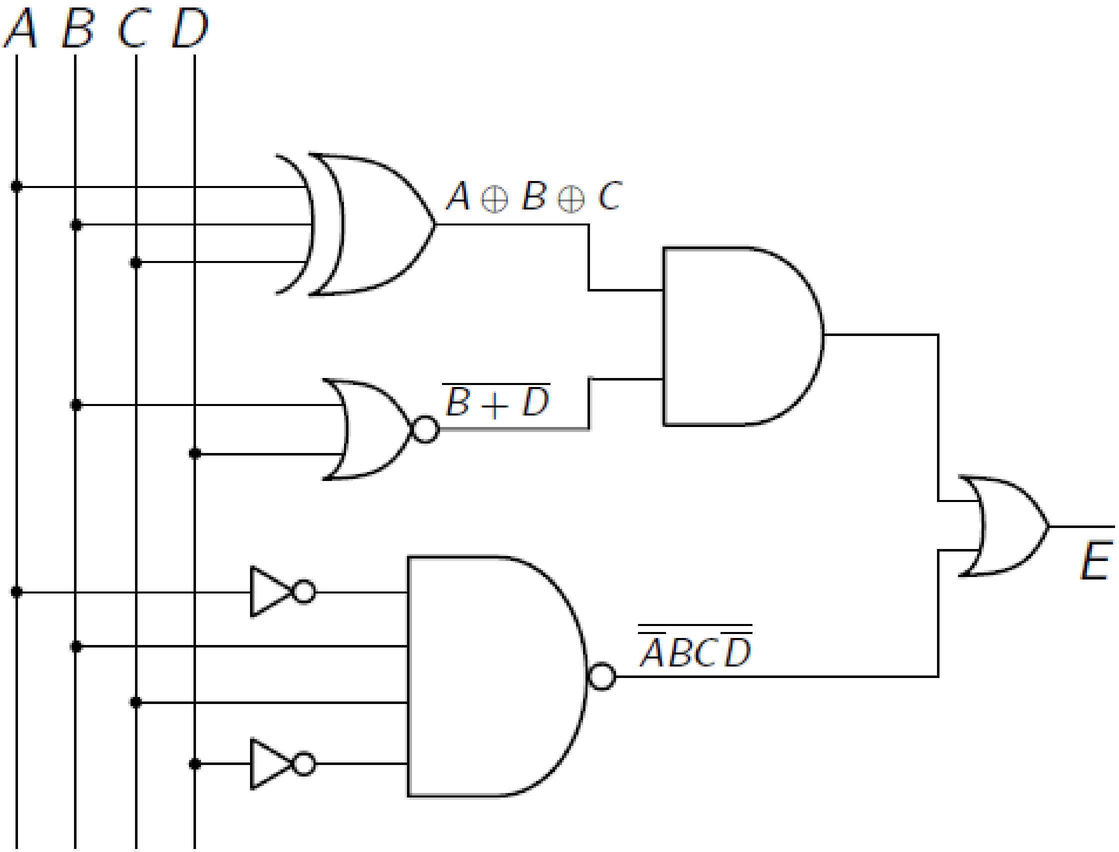
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



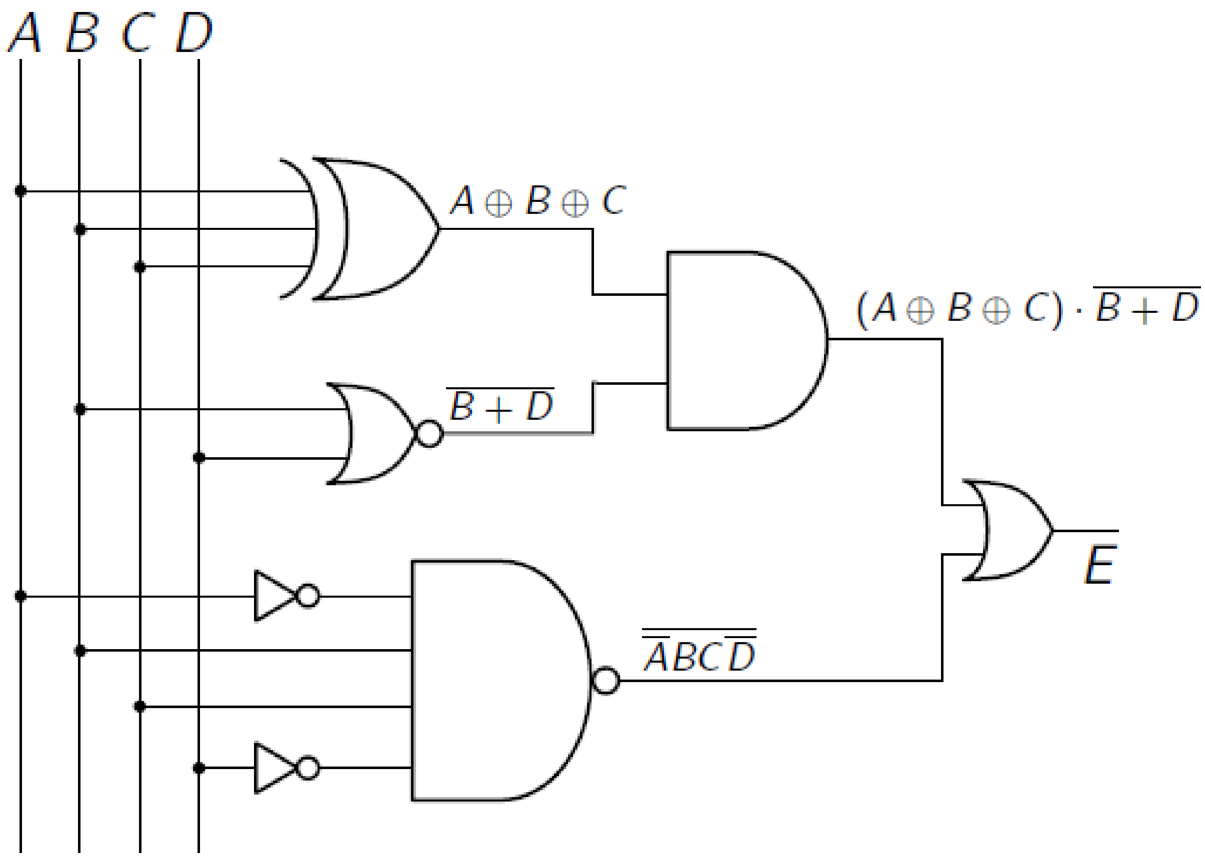
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



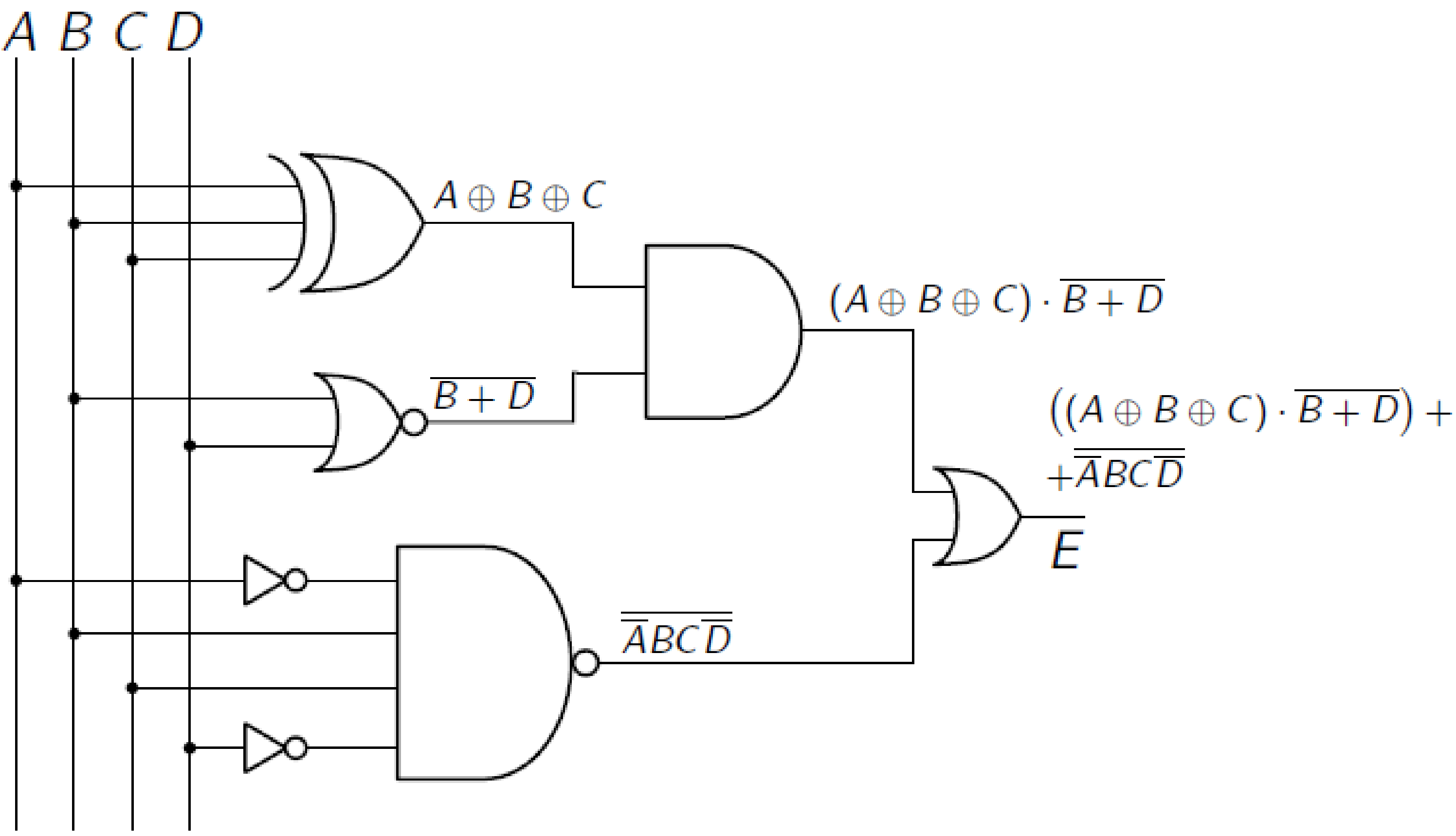
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



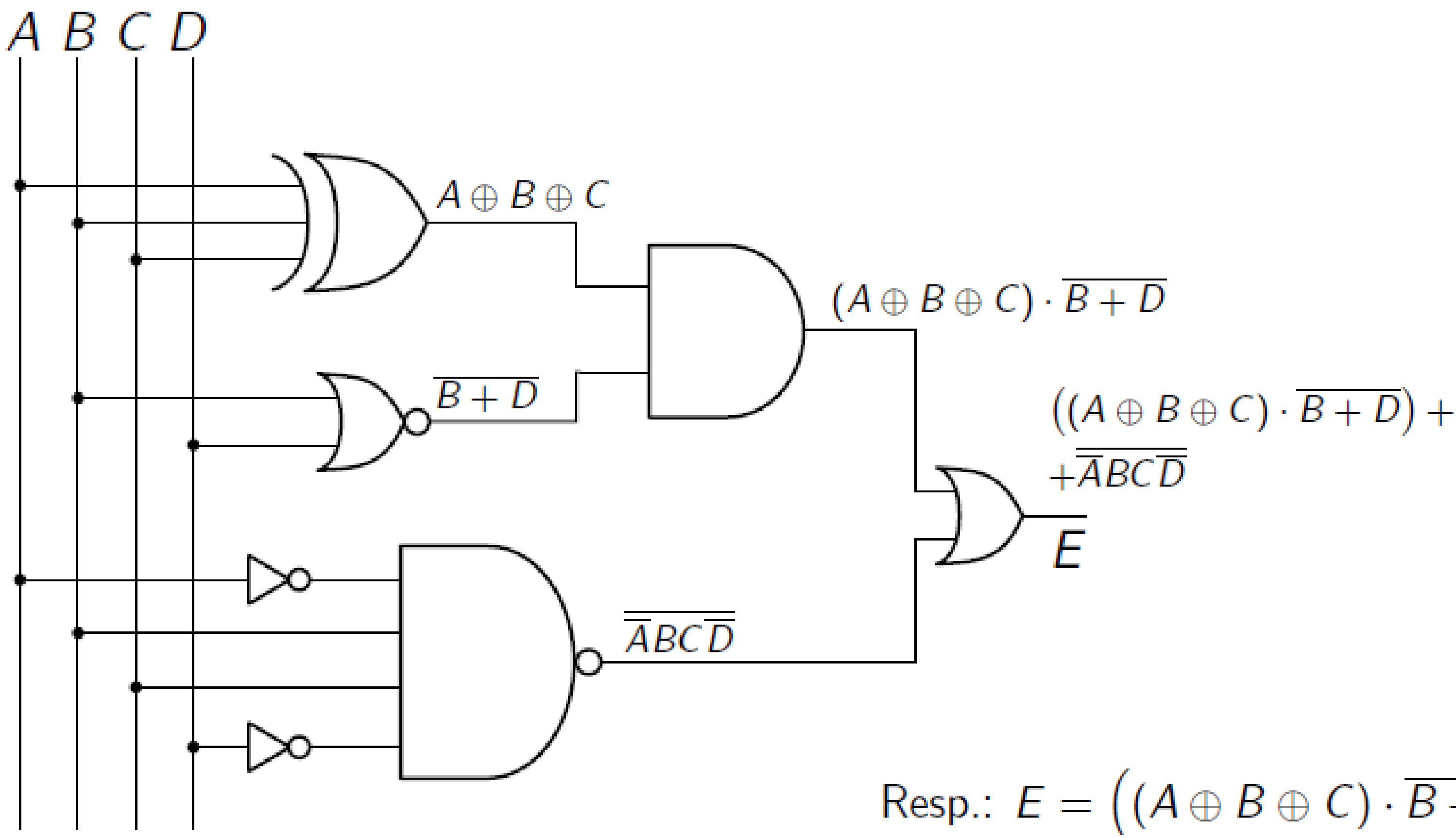
Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



Simplificação algébrica

- Dado o circuito abaixo, encontre uma expressão lógica para E em função de (A, B, C, D):



Resp.: $E = \left((A \oplus B \oplus C) \cdot \overline{B+D}\right) + \overline{\overline{A}BC\overline{D}}$

Simplificação algébrica

- Aplicar as identidades da álgebra booleana para gerar uma função equivalente com menos variáveis
  - Apenas para expressões **MUITO simples**.

- Para expressões mais complexas
  - Utilizar os **Mapas de Karnaugh**

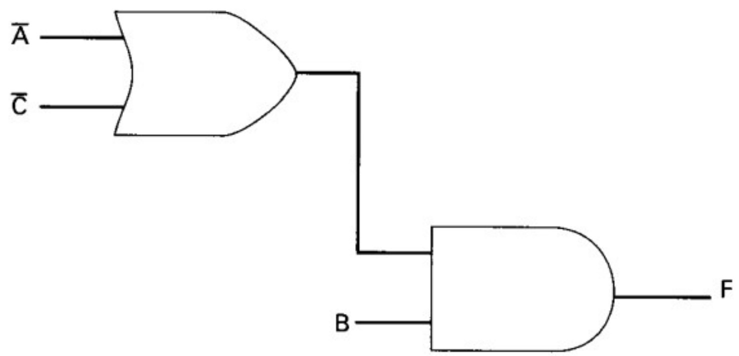
- EXEMPLO:
  - $F = A'B + BC'$ → 5 operadores
  - Simplificada: $F = B(A' + C')$ → 4 operadores

Simplificação algébrica

- Aplicar as identidades da álgebra booleana para gerar uma função equivalente com menos variáveis
  - Apenas para expressões **MUITO simples**.

- Para expressões mais complexas
  - Utilizar os **Mapas de Karnaugh**

- EXEMPLO:
  - F = A'B + BC' → 5 operadores
  - Simplificada: F = B(A' + C') → 4 operadores



Projeto de circuitos lógicos

- Geralmente não utilizamos todos os tipos de portas lógicas em uma implementação de circuito lógico.
  - Um número menor de portas lógicas torna o **projeto mais simples**.

Projeto de circuitos lógicos

- Geralmente não utilizamos todos os tipos de portas lógicas em uma implementação de circuito lógico.
  - Um número menor de portas lógicas torna o **projeto mais simples**.

- Conjunto de portas lógicas **funcionalmente completos:**
  - Qualquer função booleana pode ser implementada usando apenas as portas de um conjunto

    AND, OR, NOT

    AND, NOT

    OR, NOT

    NAND

    NOR

Projeto de circuitos lógicos

- **AND, OR, NOT:**
  - Conjunto funcionalmente completo pois representam as três operações básicas da álgebra booleana.

Projeto de circuitos lógicos

- **AND, OR, NOT:**
  - Conjunto funcionalmente completo pois representam as três operações básicas da álgebra booleana.

- **AND, NOT:**
  - Para ser funcionalmente completo:

    É necessário representar a função OR usando AND e NOT

    Pelas leis de De Morgan: $A + B = (A' \cdot B')'$

    A OR B = NOT ((NOT A) AND (NOT B))

Projeto de circuitos lógicos

- **AND, OR, NOT:**
  - Conjunto funcionalmente completo pois representam as três operações básicas da álgebra booleana.

- **AND, NOT:**
  - Para ser funcionalmente completo:

    É necessário representar a função OR usando AND e NOT

    Pelas leis de De Morgan: $A + B = (A' \cdot B')'$

    A OR B = NOT ((NOT A) AND (NOT B))

- **OR, NOT:**
  - Analogamente ao conjunto AND, NOT

    Pelas leis de De Morgan: $A \cdot B = (A' + B')'$

    A AND B = NOT ((NOT A) OR (NOT B))

Projeto de circuitos lógicos

- Elabore um circuito com portas lógicas NOT, AND e OR cuja saída corresponda à expressão A XOR B:

Projeto de circuitos lógicos

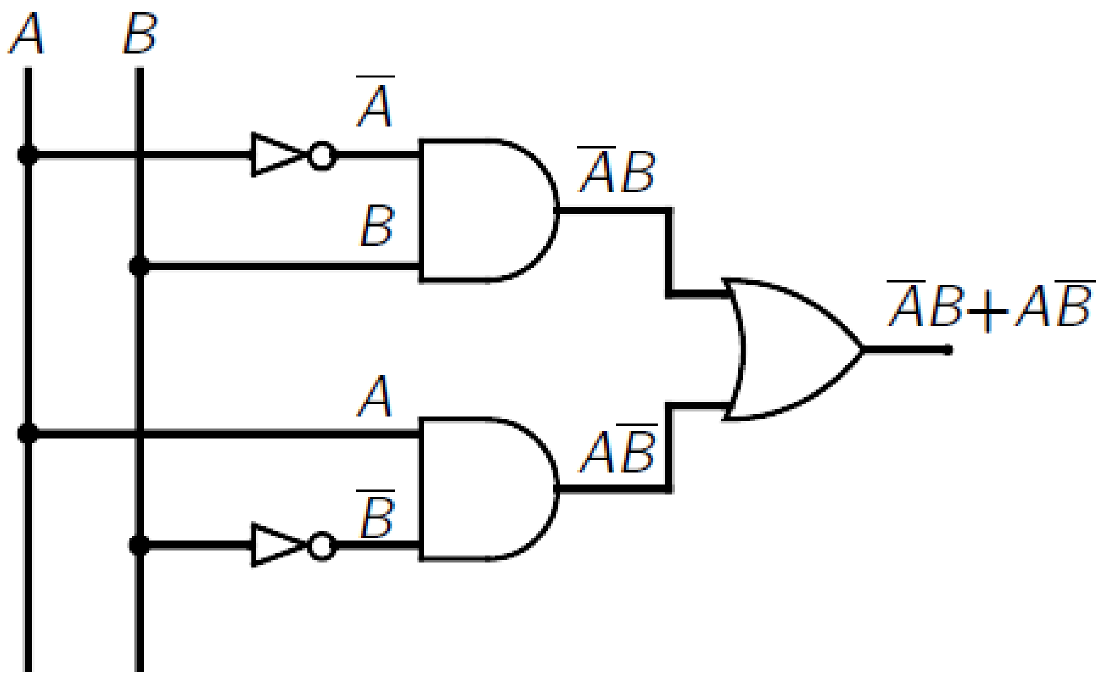
- Elabore um circuito com portas lógicas NOT, AND e OR cuja saída corresponda à expressão A XOR B:

Sabemos que A XOR B = A’B OR AB’

Projeto de circuitos lógicos

- Elabore um circuito com portas lógicas NOT, AND e OR cuja saída corresponda à expressão A XOR B:

Sabemos que A XOR B = A’B OR AB’

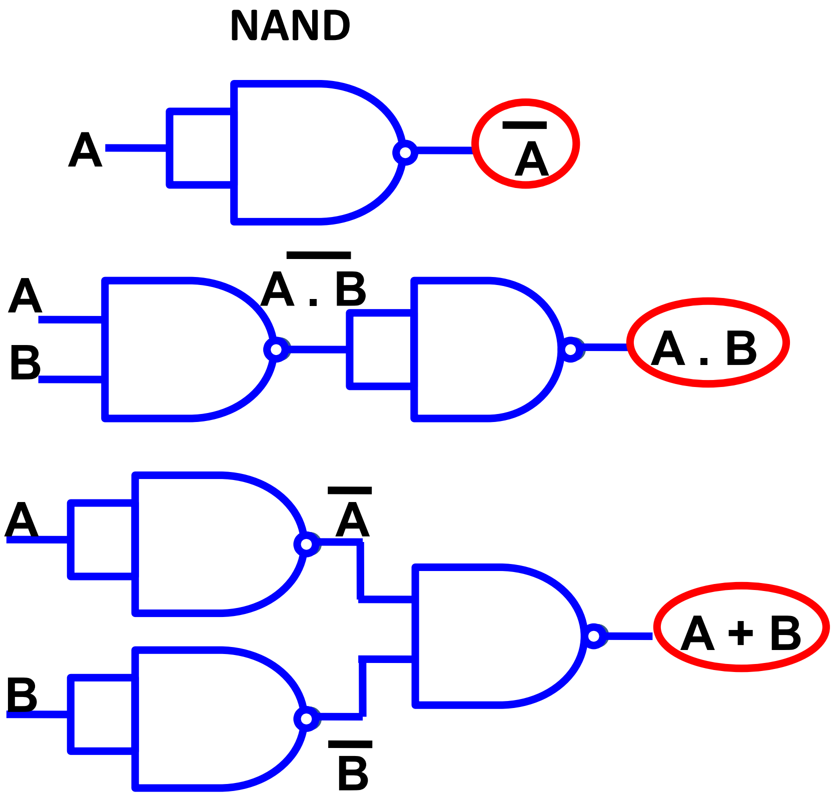


NAND e NOR

- As funções **AND, OR e NOT** podem ser implementadas usando apenas a porta **NAND** ou apenas a porta **NOR**.

NAND e NOR

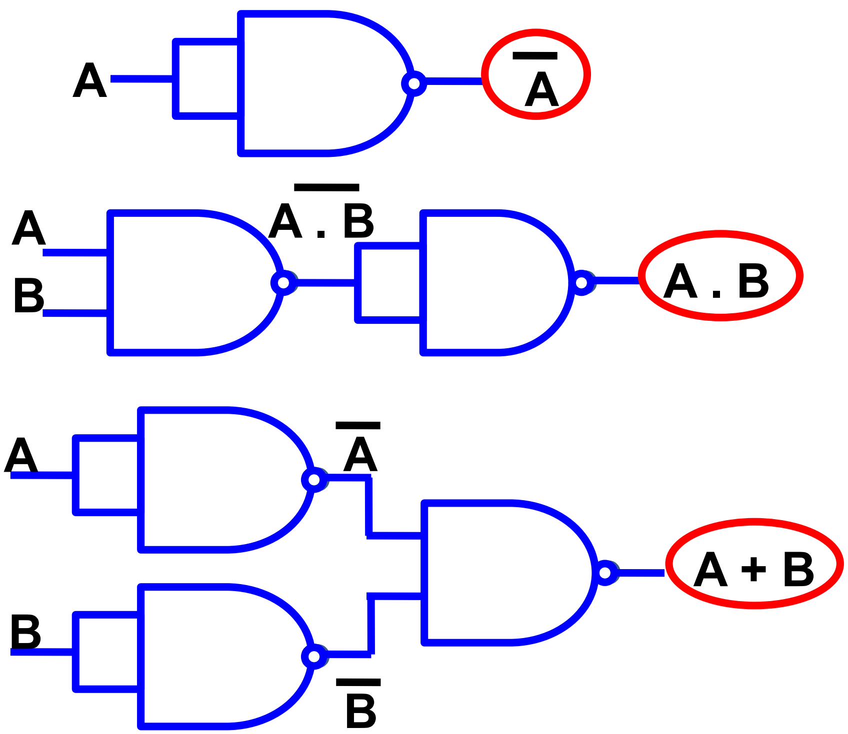
- As funções **AND, OR e NOT** podem ser implementadas usando apenas a porta **NAND** ou apenas a porta **NOR**.



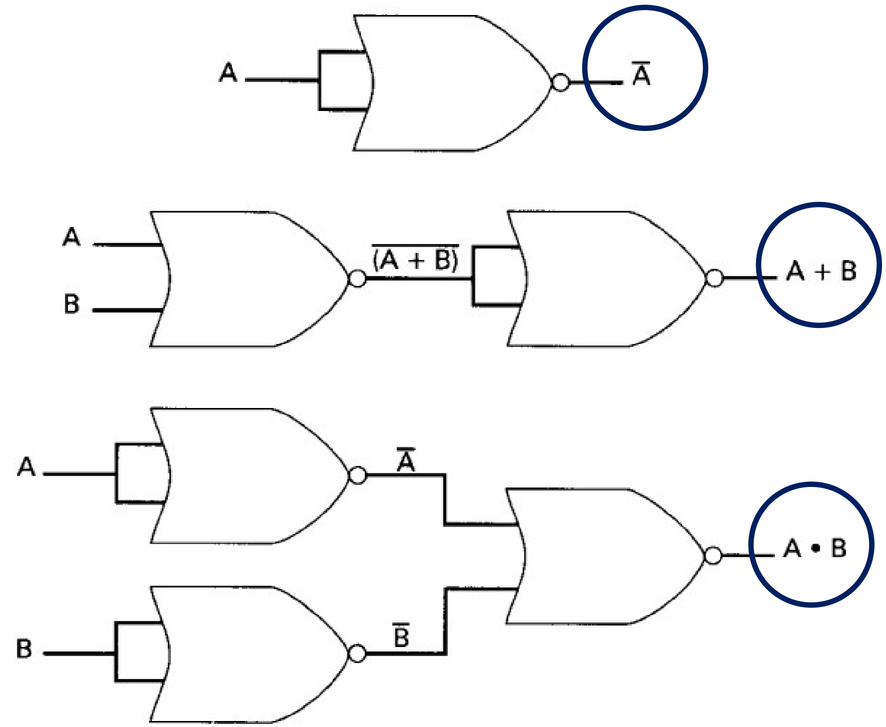
NAND e NOR

- As funções **AND, OR e NOT** podem ser implementadas usando apenas a porta **NAND** ou apenas a porta **NOR**.

NAND



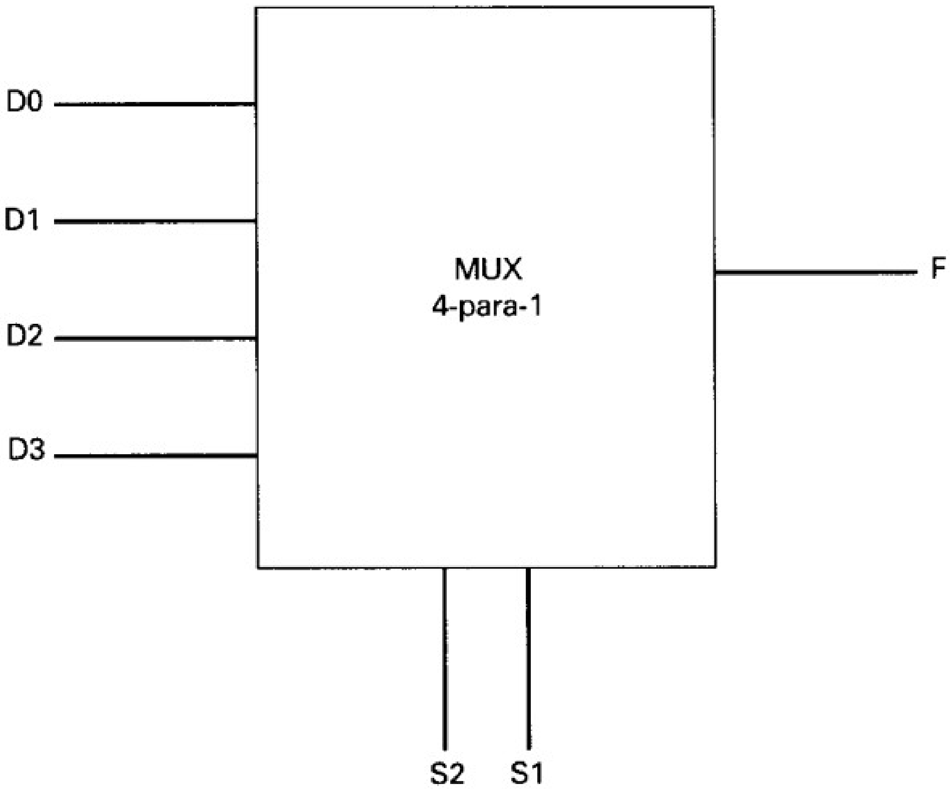
NOR



Aplicações

Multiplexadores

- Várias entradas.
- Uma única saída.
- A cada instante uma única entrada passa para a saída.
- Determinado por um conjunto de linhas de seleção.
- Para n linhas de controle:
  - 2^n entradas



Multiplexadores

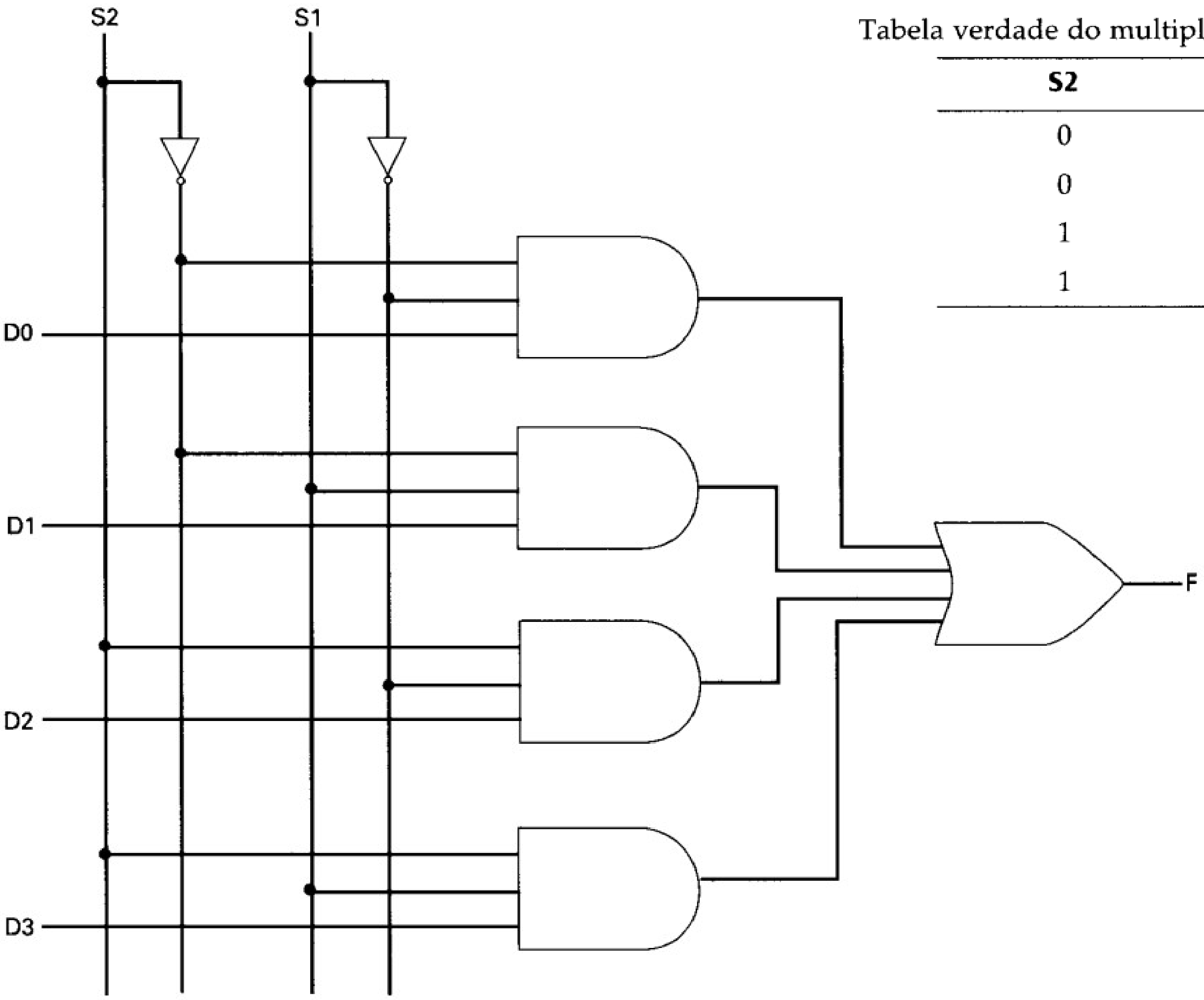


Tabela verdade do multiplexador 4-para-1

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Decodificadores

- Circuito combinatório:
  - Um certo número de linhas de saída.
  - Apenas uma é ativada em cada instante
  - dependendo do padrão de sinais de entrada.

Decodificadores

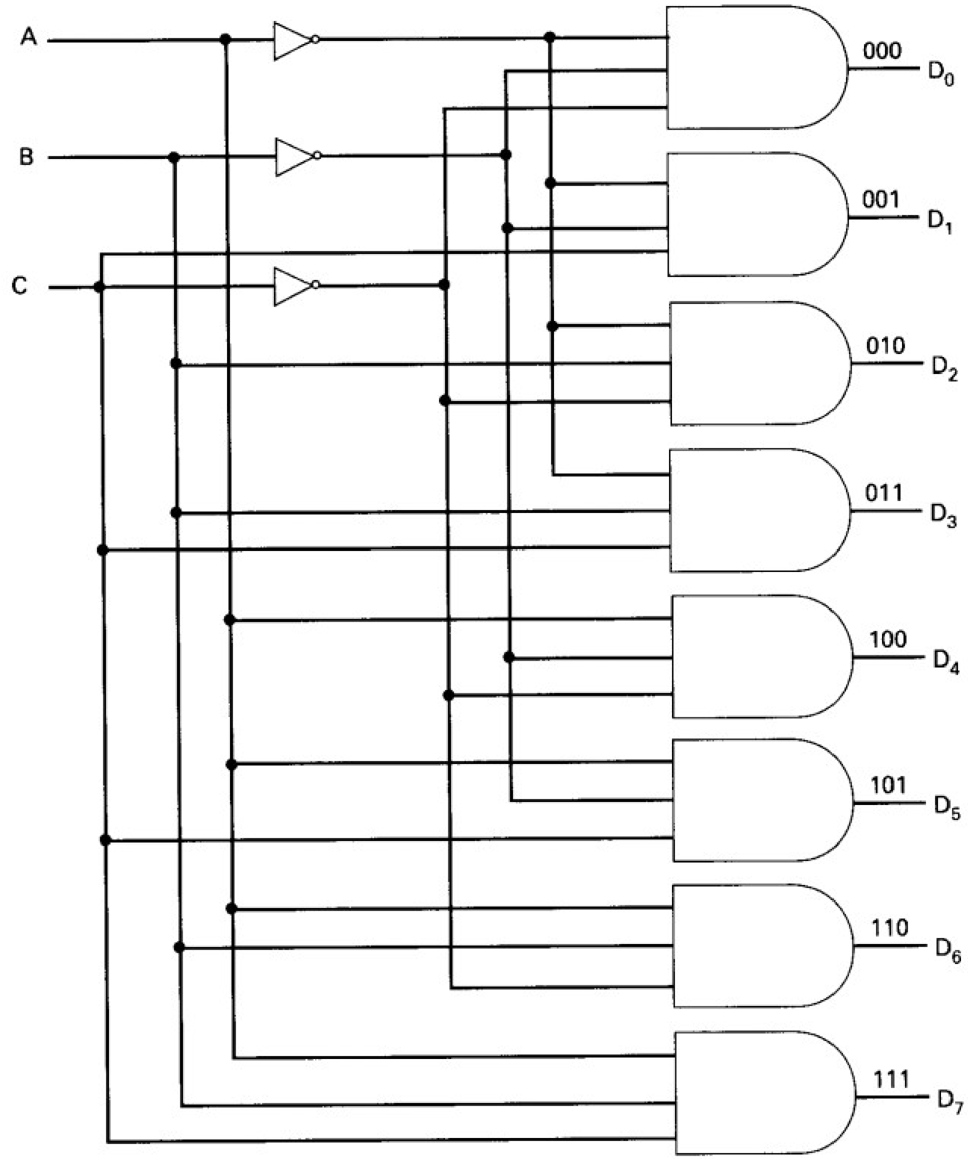


Figura A.15 Decodificador com 3 entradas e $2^3 = 8$ saídas.

Decodificação de Endereços

- Memória de 1 Kbyte
  - 4 pastilhas de memória RAM de 256 bytes (256 palavras de 8 bits)

Endereço	Pastilha
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Decodificação de Endereços

- Memória de 1 Kbyte
  - 4 pastilhas de memória RAM de 256 bytes (256 palavras de 8 bits)

Endereço	Pastilha
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

- Endereços de 10 bits
  - 8 bits menos significativos (2^8 = 256 palavras)
  - 2 bits mais significativos (2^2 = 4 pastilhas)
  - Selecionar uma das quatro pastilhas (decodificador)

Decodificação de Endereços

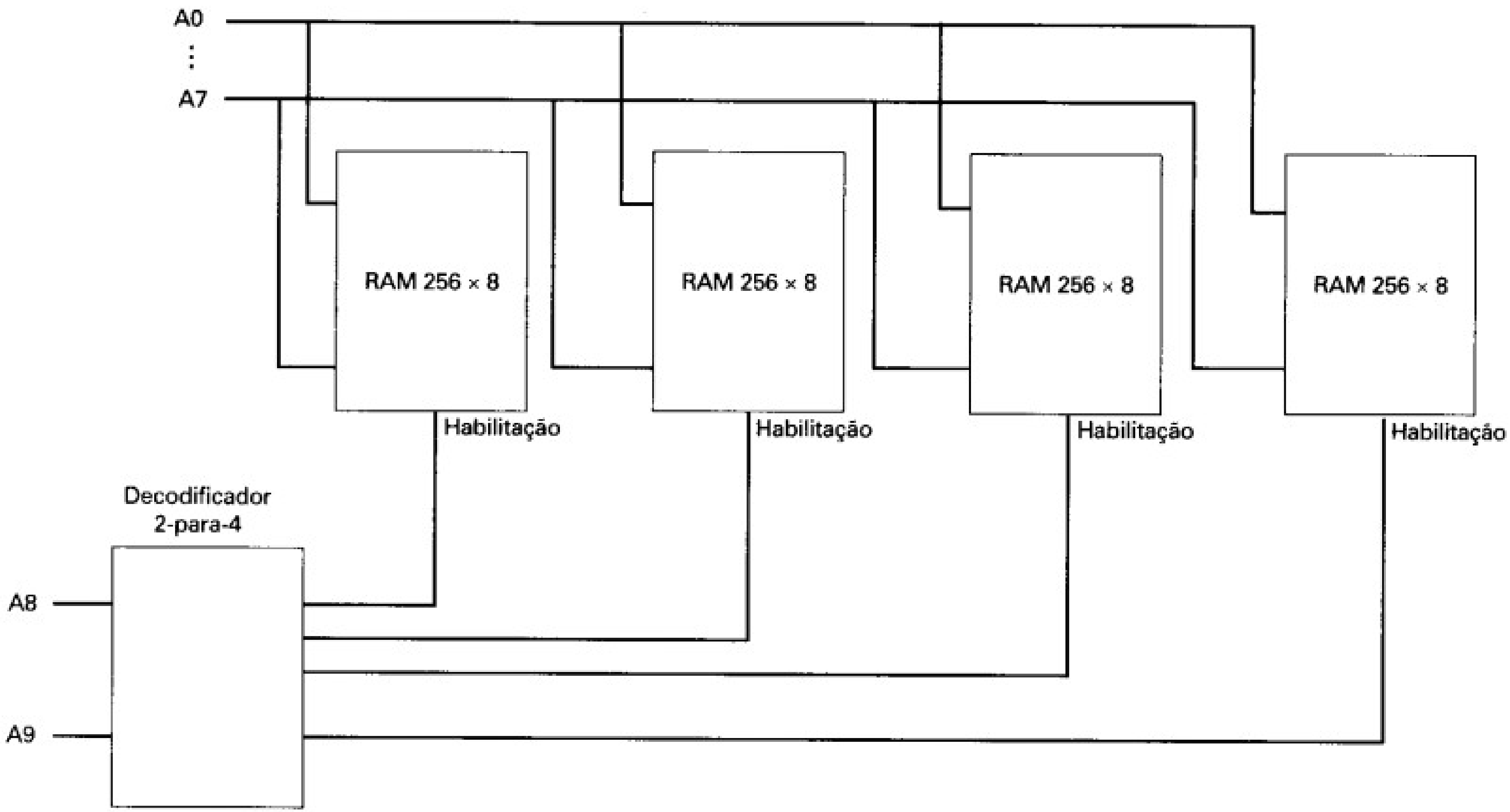


Figura A.16 Decodificação de endereços.

Memórias apenas de leitura

- ROM (*Read-Only-Memory*)
  - Memória implementada usando circuitos combinatórios.
  - Um **decodificador** e um **conjunto de portas OR**.
- Memória ROM de 64 bits
  - 16 palavras de 4 bits

Memórias apenas de leitura

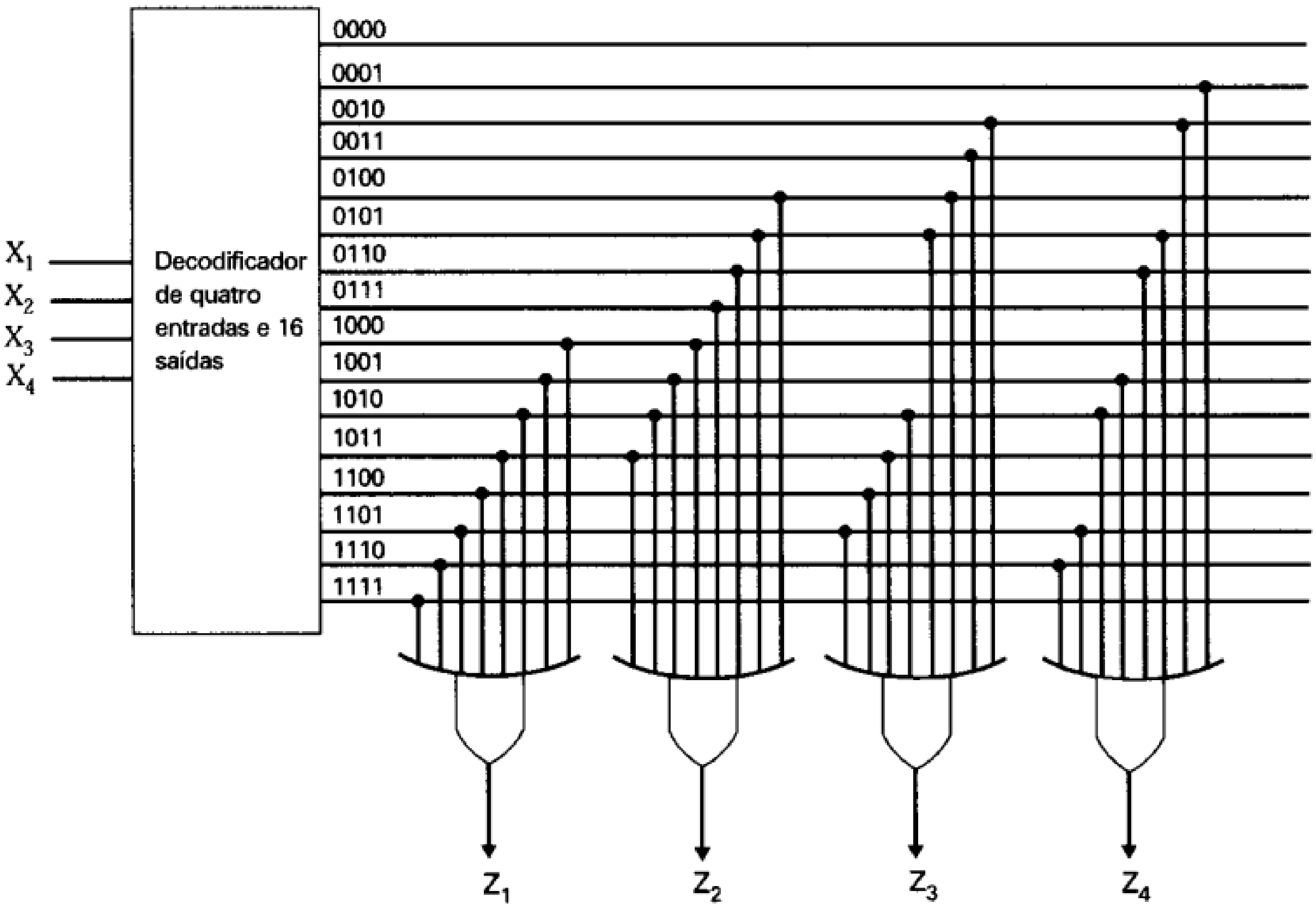
Tabela A.8 Tabela verdade para uma ROM

Entrada				Saída			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

São as posições da memória ROM.

Os dados que você quer acessar.

ROM de 64 bits



Circuitos somadores

- Adição binária
  - Vai-um (*Carry*)

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

Adição binária

(a) Adição de um único bit

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

Adição binária

(a) Adição de um único bit

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

(b) Adição com uma entrada de bit de 'vai-um'

C_{in}	A	B	Soma	C_{out}
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

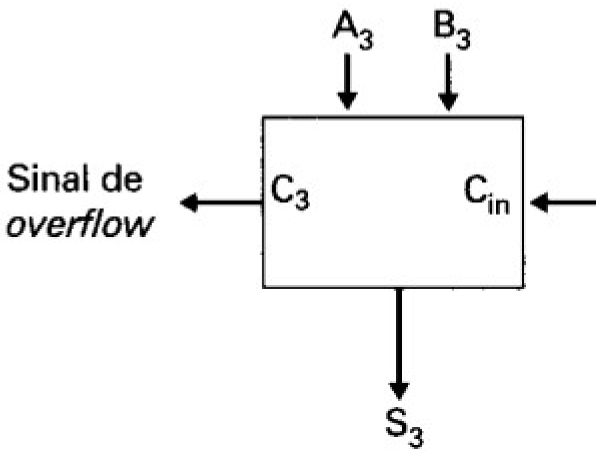
Adição binária

(a) Adição de um único bit

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

(b) Adição com uma entrada de bit de 'vai-um'

C_{in}	A	B	Soma	C_{out}
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



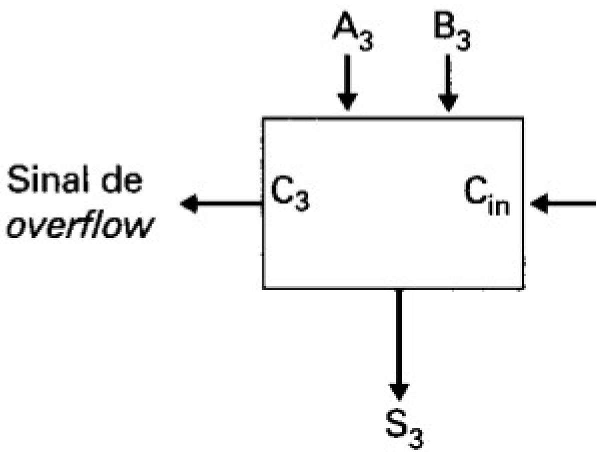
Circuitos somadores

(a) Adição de um único bit

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

(b) Adição com uma entrada de bit de 'vai-um'

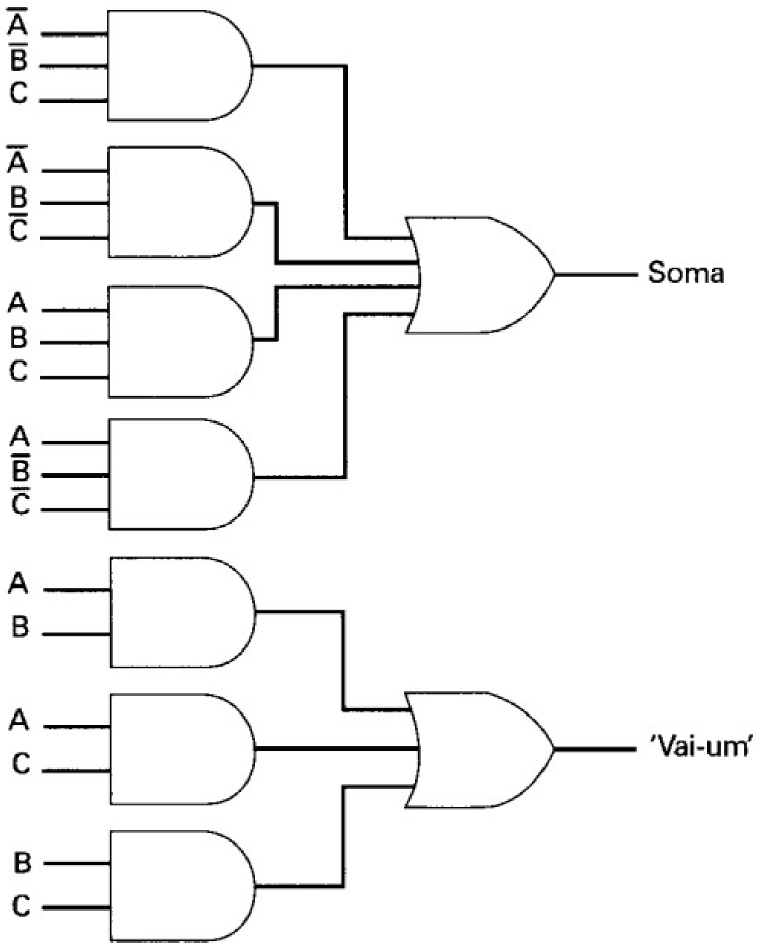
C_{in}	A	B	Soma	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$\text{Soma} = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\,\overline{C}$$

$$\text{Vai-um} = AB + AC + BC$$

Adição binária



$$\text{Soma} = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\,\overline{C}$$

$$\text{Vai-um} = AB + AC + BC$$

Somador de 4 bits