Universidade Federal de Viçosa

Aula 06 – Circuitos Combinatórios

Prof. João Fernando Mari

joaof.mari@ufv.br

Roteiro

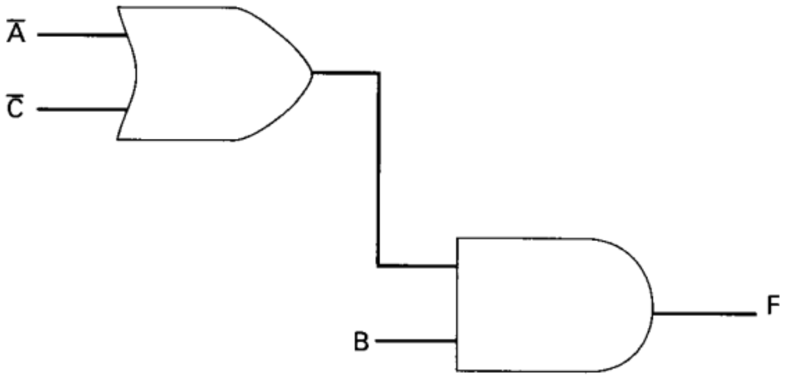
- Simplificação algébrica (sem Mapas de Karnaugh)
- Projeto de circuitos lógicos
- Universalidade das portas lógicas: AND, OR, NOT / AND, NOT / OR, NOT
- Universalidade das portas NAND e NOR
- [EXEMPLO] Universalidade das portas NAND e NOR
- Multiplexadores (MUX)
- Decodificadores
- [EXEMPLO] Decodificação de Endereços
- Memórias apenas de leitura (Memória ROM)
- Adição binária
- Adição binária – Usando XOR
- Somador de 4 bits
- “Vai um” antecipado (*carry lookahead*)
- Somador de 32 bits usando somadores de 8 bits

Simplificação algébrica (sem Mapas de Karnaugh)

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

- 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

Universalidade das portas lógicas: AND, OR, NOT / AND, NOT / OR, NOT

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

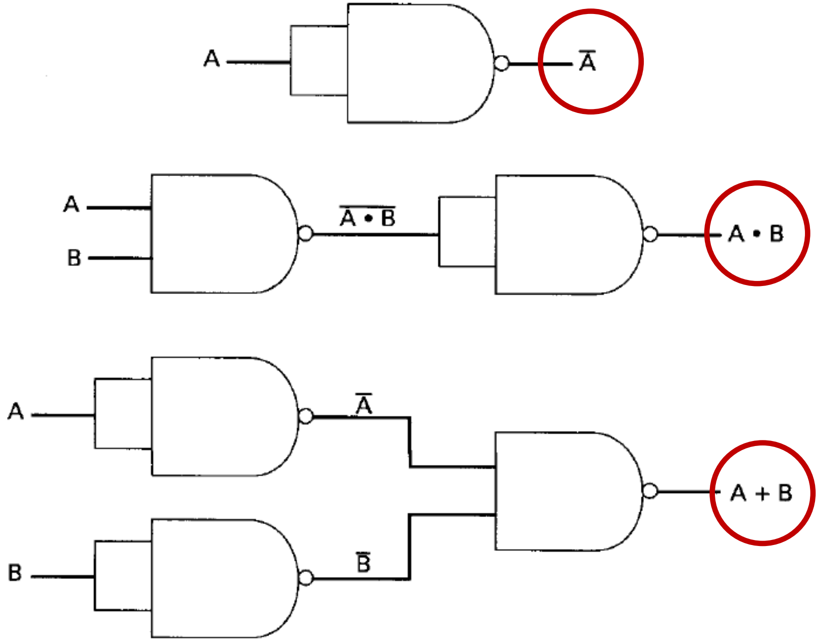
- **AND, NOT:**
  - Para ser funcionalmente completo é necessário representar a função OR usando AND e NOT
    - Pelas leis de De Morgan: $A + B = (A' \cdot B')'$
    - A OR B = NOT ((NOT A) AND (NOT B))

- **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))

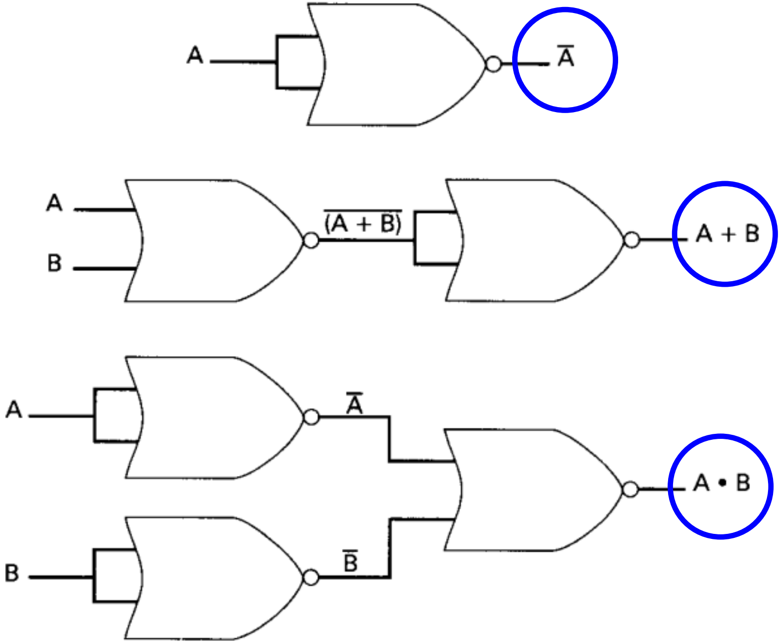
Universalidade das portas NAND e NOR

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

NAND



NOR



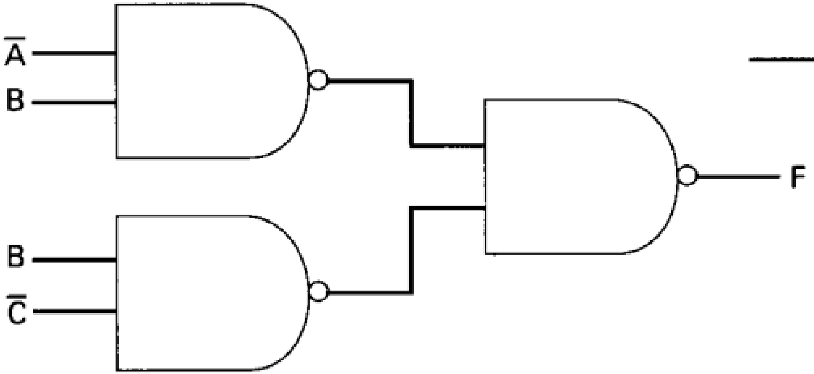
[EXEMPLO] Universalidade das portas NAND e NOR

$$F = B(\overline{A} + \overline{C})$$

$$F = B(\overline{A} + \overline{C}) = (\overline{A}B) + (B\overline{C})$$

Por De Morgan:

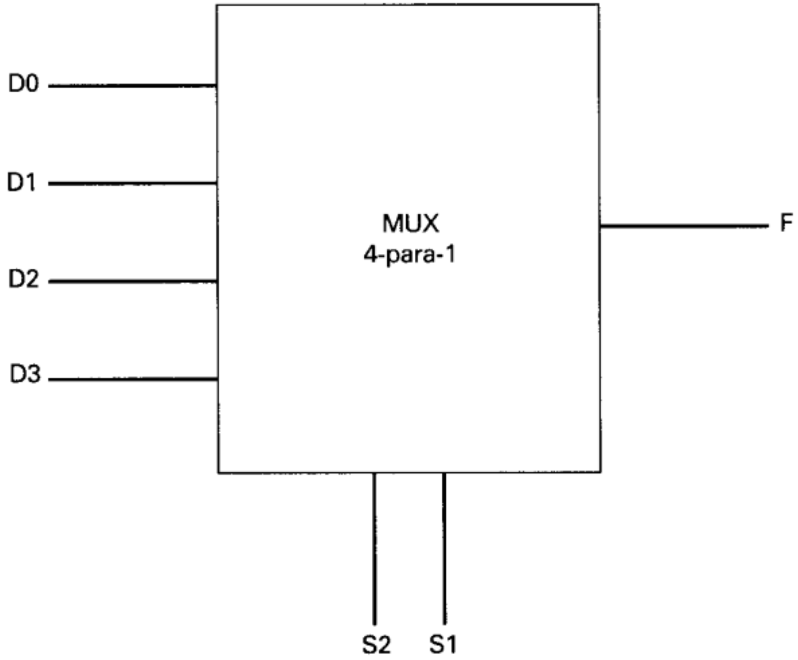
$$F = \overline{\overline{(\overline{A}B)} \bullet \overline{(B\overline{C})}}$$



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

Multiplexadores (MUX)

- 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

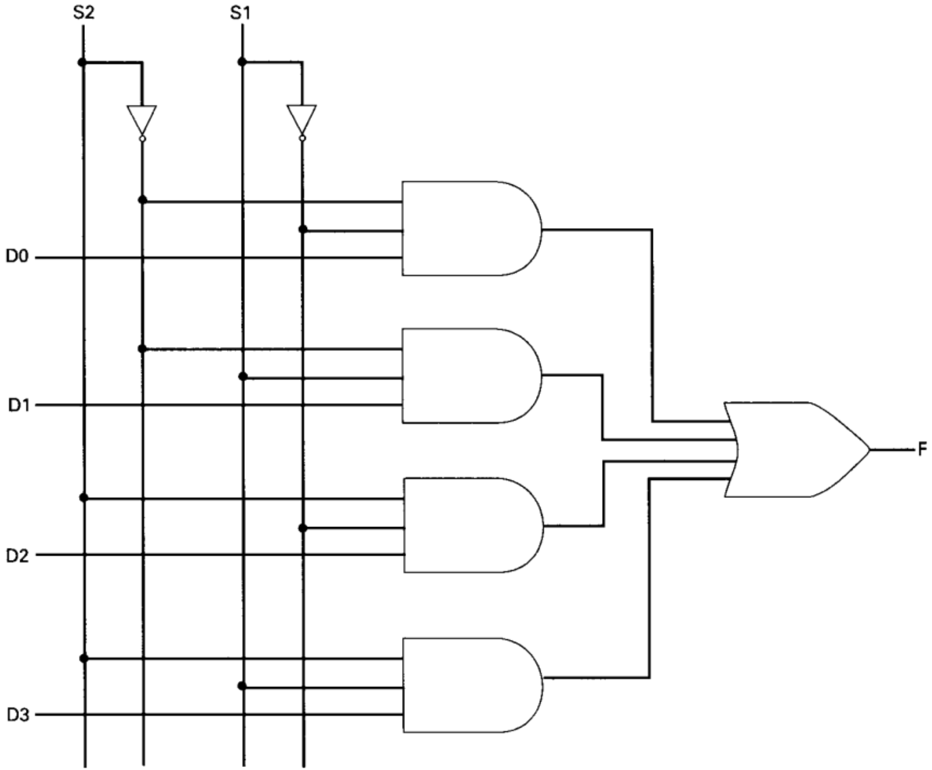


Multiplexadores (MUX)

UFV Universidade Federal de Viçosa

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.

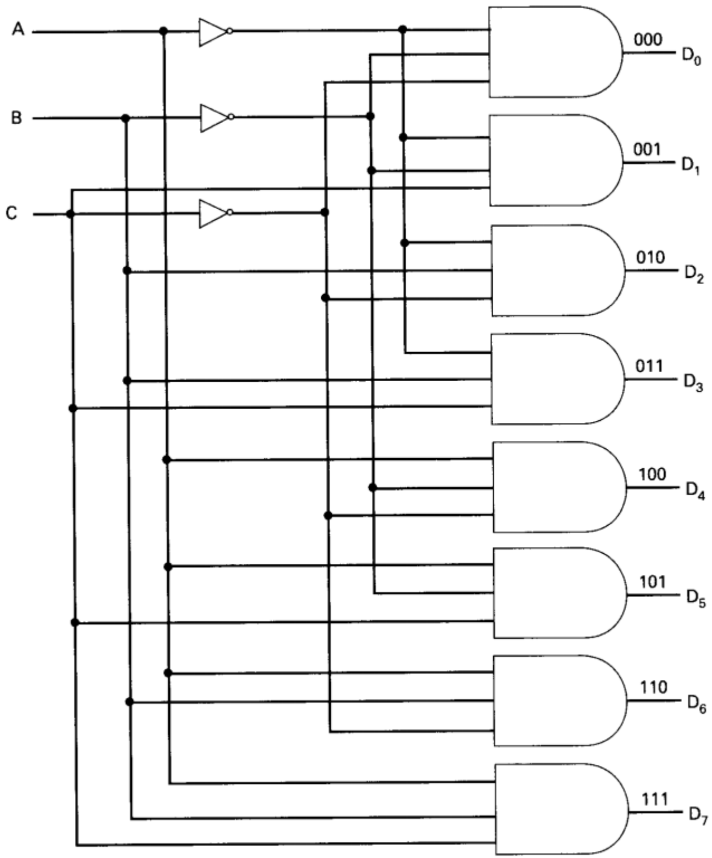


Figura A.15 Decodificador com 3 entradas e $2^3 = 8$ saídas.

[EXEMPLO] 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)

[EXEMPLO] Decodificação de Endereços

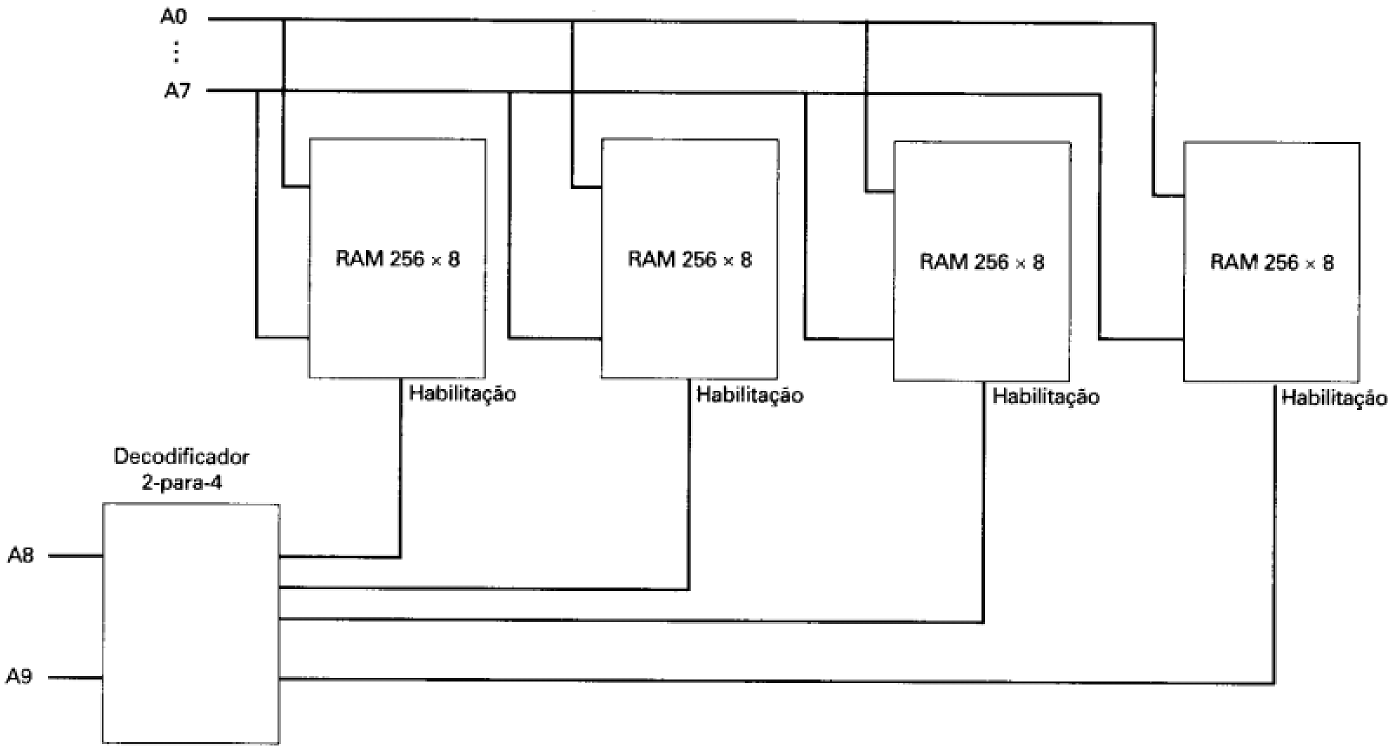


Figura A.16 Decodificação de endereços.

Memórias apenas de leitura (Memória ROM)

- Circuitos combinatórios:
  - São circuitos “sem memória”.

- 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 (Memória ROM)

UFV Universidade Federal de Viçosa

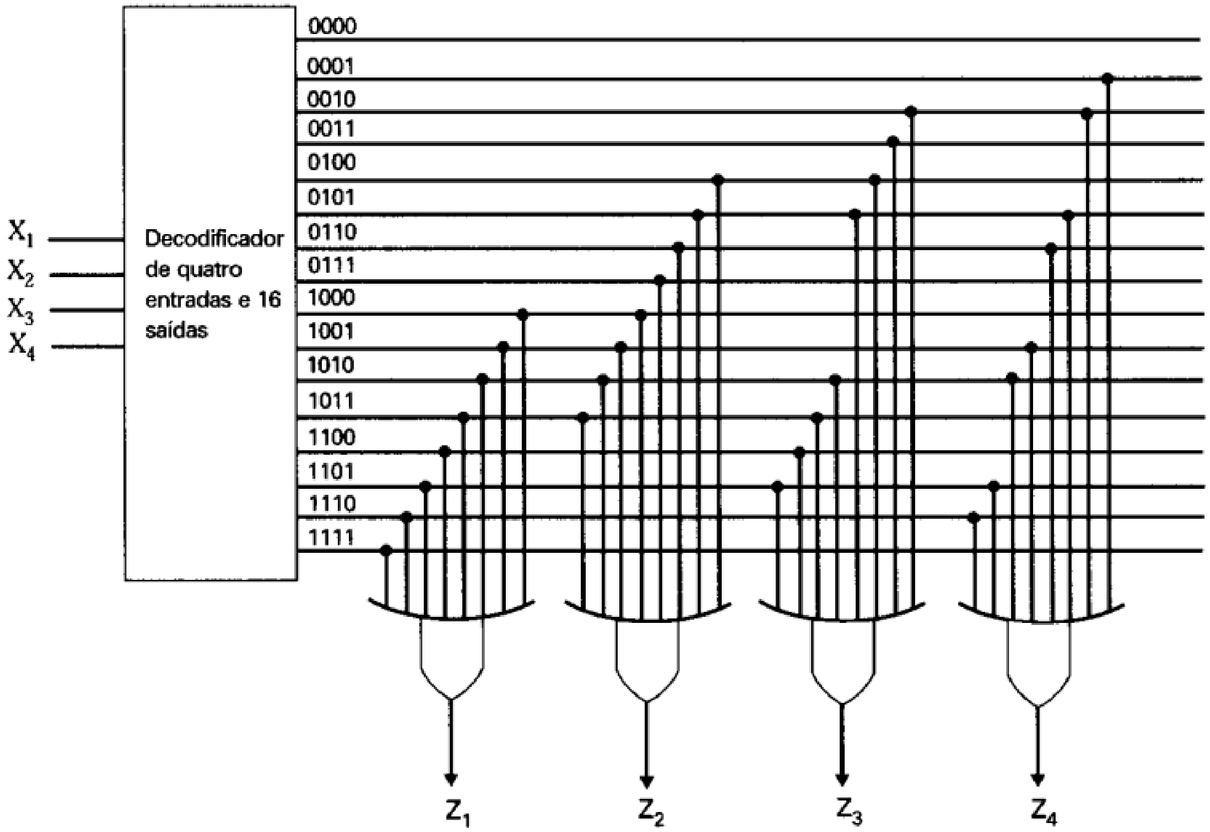
Tabela A.8 Tabela verdade para uma ROM

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 armazenar.

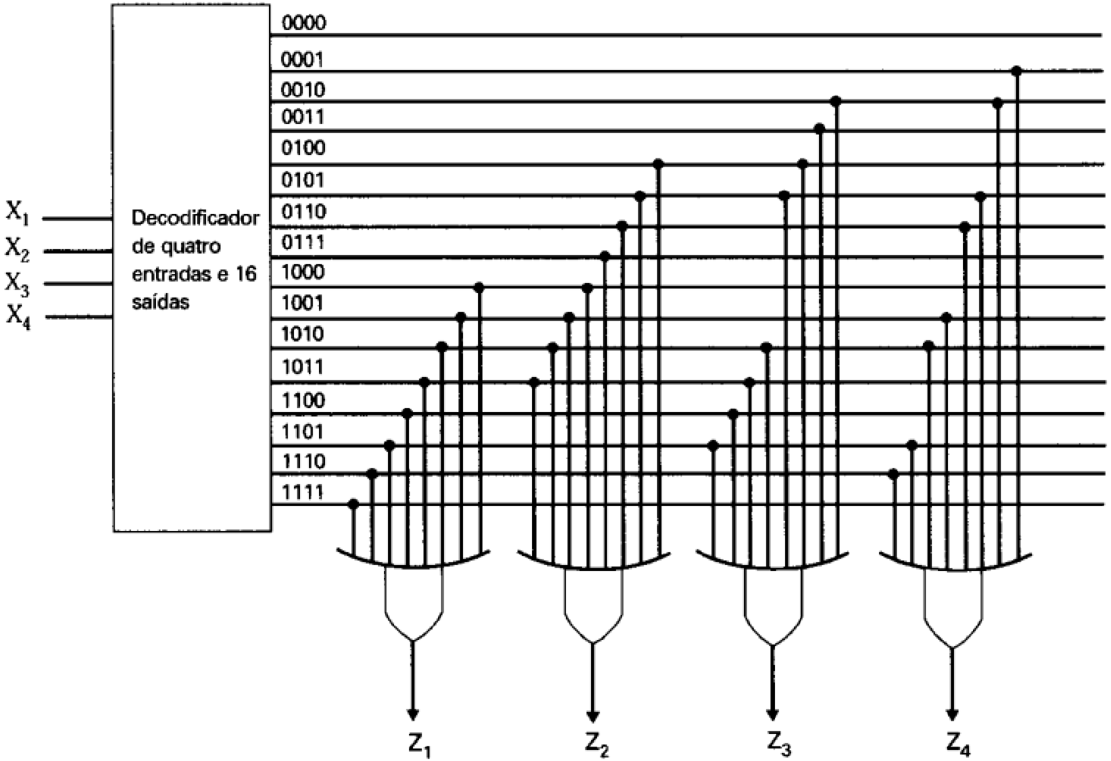
ROM de 64 bits



ROM de 64 bits

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



Circuitos Somadores

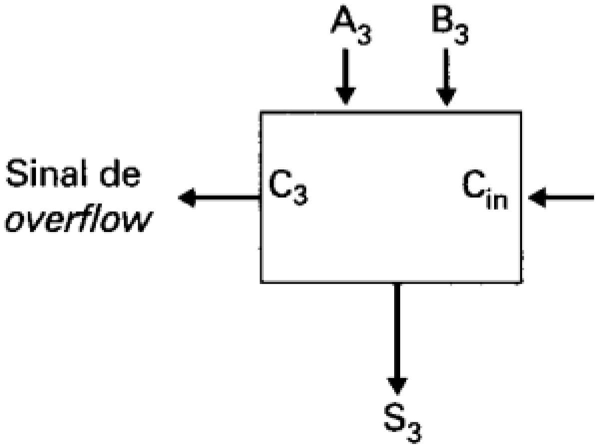
- Circuito combinatório
  - Operações aritmética
- Adição binária
  - Vai-um (Carry)

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

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



(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

Soma = A'BC' + AB'C' + A'B'C + ABC

Vai-um = ABC' + A'BC + AB'C + ABC

Obs.: C = C_{in}.

Adição Binária

Soma = A’BC’ + AB’C’ + A’B’C + ABC
Vai-um = ABC’ + A’BC + AB’C + ABC

Já está simplificado!

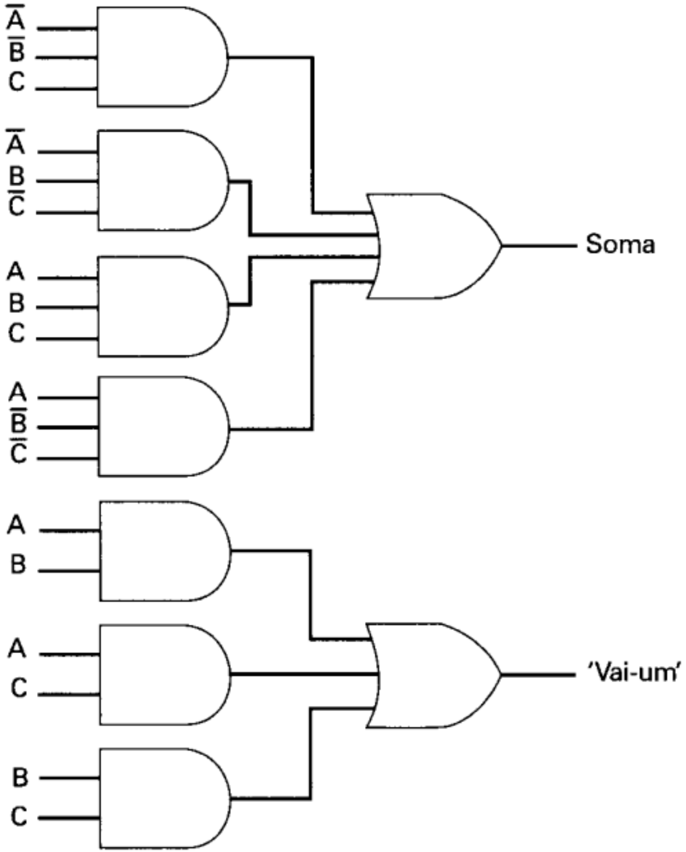
Soma:

A \ BC	00	01	11	10
0		1		1
1	1		1	

Vai-um:

A \ BC	00	01	11	10
0			1	
1		1	1	1

Soma = A’BC’ + AB’C’ + A’B’C + ABC
Vai-um = AB + AC + BC



Adição binária – Usando XOR

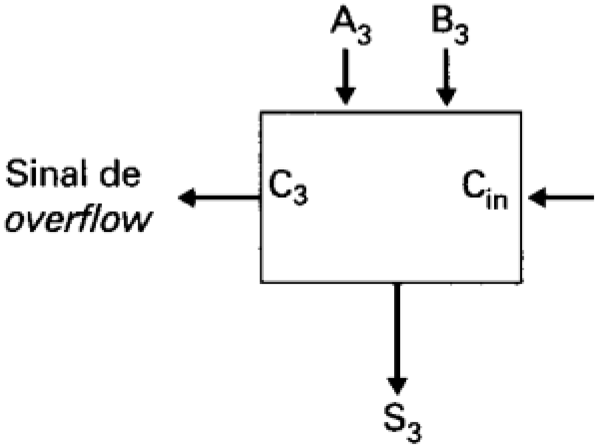
(a) Adição de um único bit

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

$A \oplus B$



$\boxed{\textbf{Soma} = \mathbf{A \oplus B \oplus Cin}}$

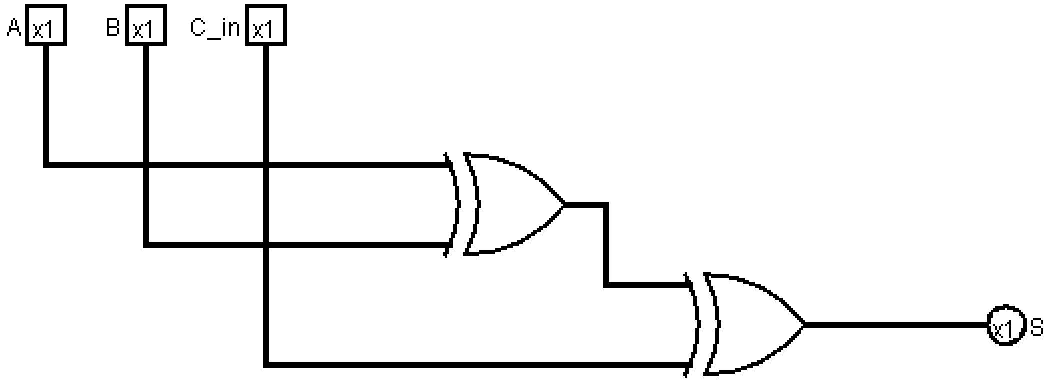
Vai-um = AB + AC + BC

Obs.: $C = C_{in}$.

Adição binária – Usando XOR

(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

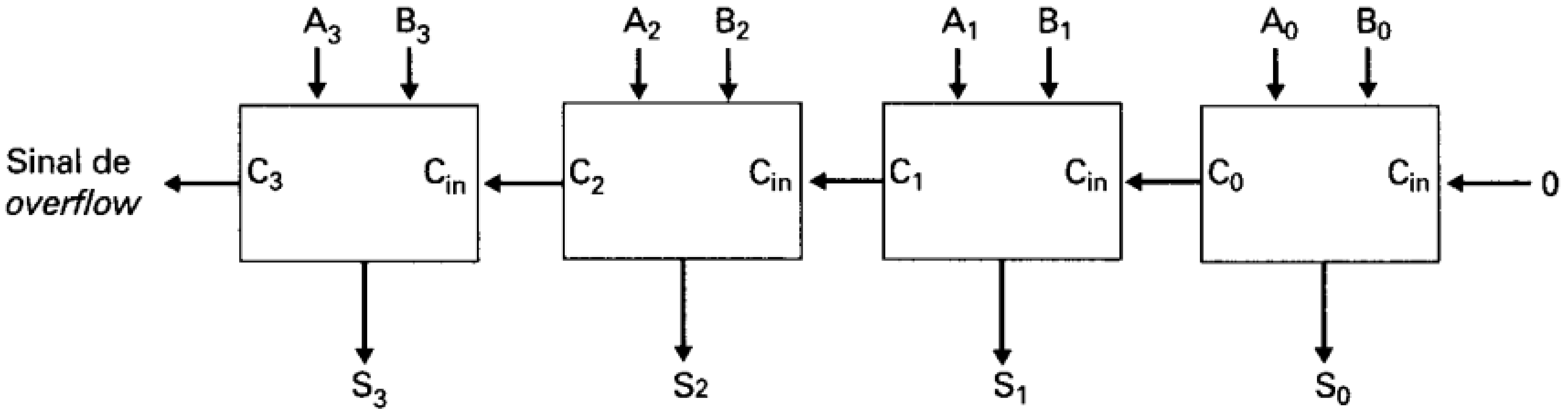


Soma = A ⊕ B ⊕ Cin

Vai-um = AB + AC + BC

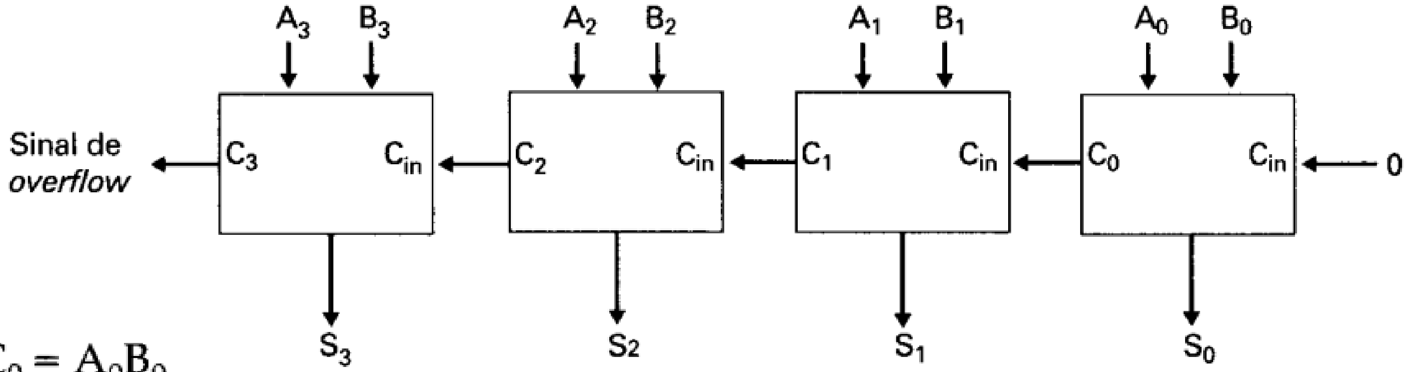
Obs.: C = C_{in}.

Somador de 4 bits



"Vai um" antecipado (*carry lookahead*) – 4 bits

- Cada somador depende do "vai um" do somador anterior.
  - Atraso crescente do bit menos significativo par o mais significativo.
  - Para somadores grandes o atraso acumulado é inaceitável
  - **SOLUÇÃO**: Calcular o "vai um" **antecipadamente**.



$$C_0 = A_0B_0$$

$$C_1 = A_1B_1 + A_1A_0B_0 + B_1A_0B_0$$

$$C_2 = A_2B_2 + \boxed{A_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0} + \boxed{B_2A_1B_1 + B_2A_1A_0B_0 + B_2B_1A_0B_0}$$

“Vai um” antecipado (*carry lookahead*) – 4 bits

$C_0 = A_0B_0$

$C_1 = A_1B_1 + \mathbf{A_1}A_0B_0 + \mathbf{B_1}A_0B_0$

$C_2 = A_2B_2 +$
$\mathbf{A_2}A_1B_1 + \mathbf{A_2}A_1A_0B_0 + \mathbf{A_2}B_1A_0B_0 +$
$\mathbf{B_2}A_1B_1 + \mathbf{B_2}A_1A_0B_0 + \mathbf{B_2}B_1A_0B_0$

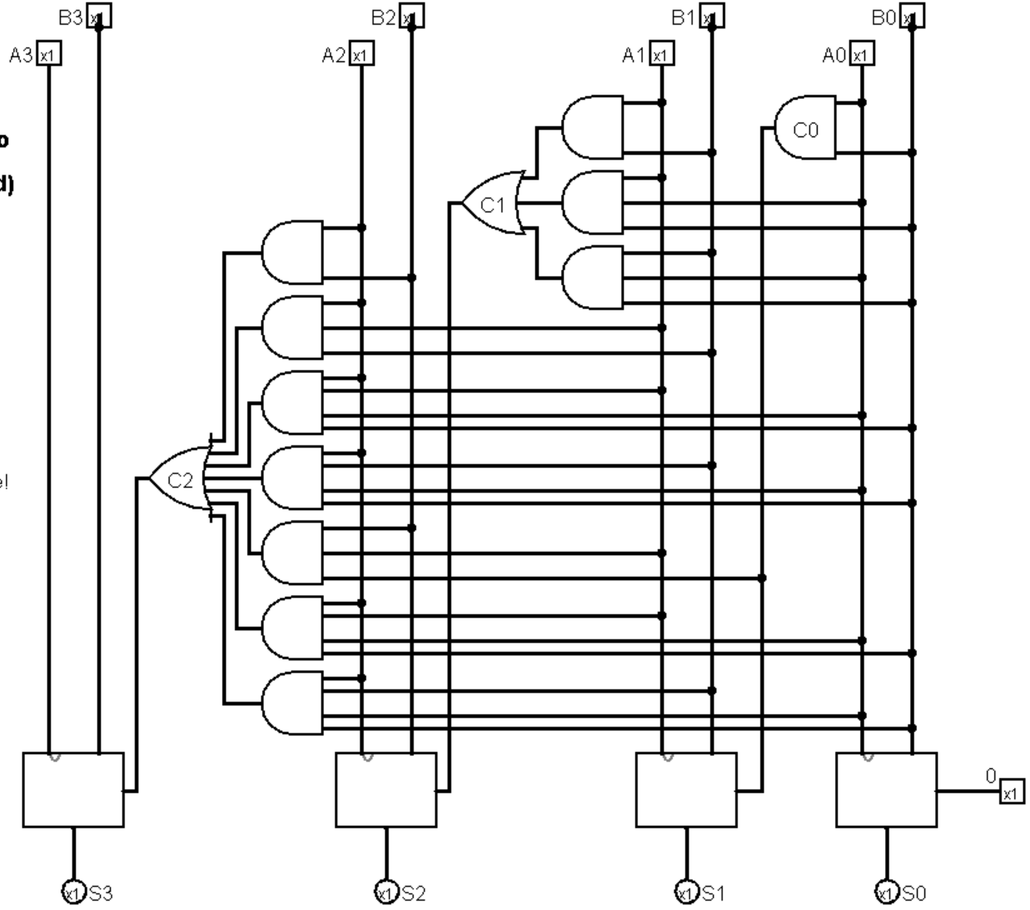
$C_3 = A_3B_3 +$
$\mathbf{A_3}A_2B_2 + \mathbf{A_3}A_2A_1B_1 + \mathbf{A_3}A_2A_1A_0B_0 + \mathbf{A_3}A_2B_1A_0B_0 + \mathbf{A_3}B_2A_1B_1 + \mathbf{A_3}B_2A_1A_0B_0 + \mathbf{A_3}B_2B_1A_0B_0 +$
$\mathbf{B_3}A_2B_2 + \mathbf{B_3}A_2A_1B_1 + \mathbf{B_3}A_2A_1A_0B_0 + \mathbf{B_3}A_2B_1A_0B_0 + \mathbf{B_3}B_2A_1B_1 + \mathbf{B_3}B_2A_1A_0B_0 + \mathbf{B_3}B_2B_1A_0B_0$

“Vai um” antecipado (*carry lookahead*) – 4 bits

Somador de 4 bits com vai-um antecipado

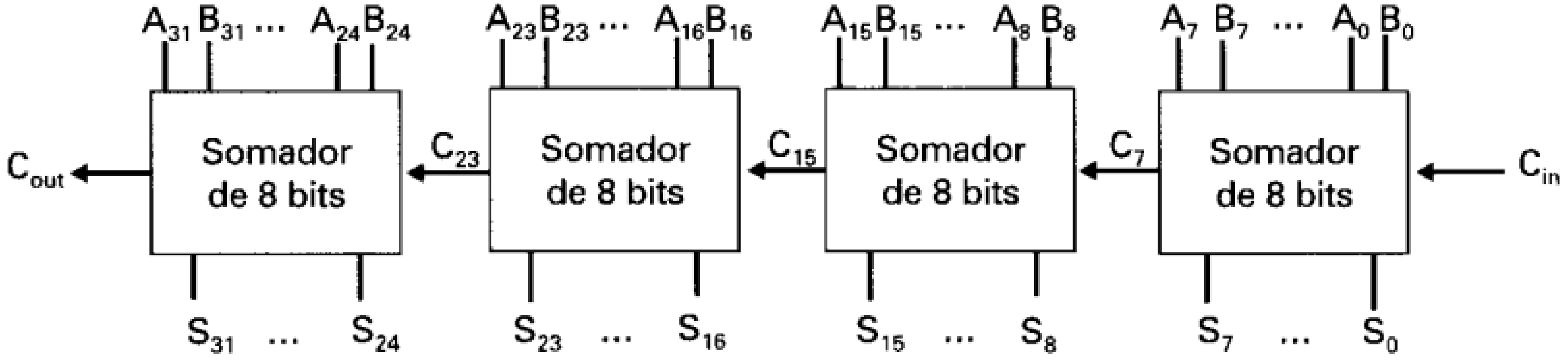
(carry lookahead)

C3 omitido por ser muito grande!



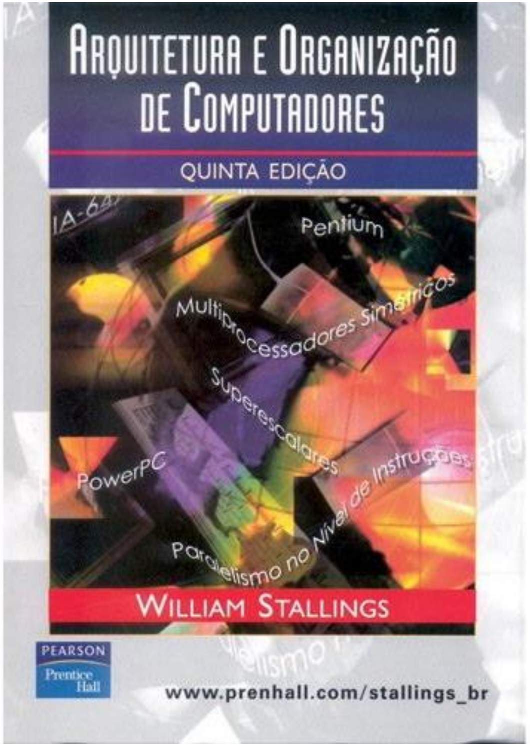
Somador de 32 bits usando somadores de 8 bits

- O **“vai um” antecipado** se torna complexo para somadores grandes.
  - **SOLUÇÃO**:
    - “Vai-um” antecipado para blocos pequenos
      - 4 a 8 bits, no máximo.
    - Propagação do “vai-um” entre esses blocos “grandes”.



Referências

- STALLINGS, W. **Arquitetura e Organização de Computadores**, 5. Ed., Pearson, 2010.
  - Apêndice A



Material complementar

Universidade Federal de Viçosa

- How Computers Calculate - the ALU: Crash Course Computer Science #5
  - https://www.youtube.com/watch?v=1I5ZMmrOfnA

FIM